



Quartus II Version 6.1 Handbook

Volume 2: Design Implementation & Optimization



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

QI15V2-6.1

Copyright © 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001



Chapter Revision Dates	xiii
About this Handbook	xv
How to Contact Altera	xv
Third-Party Software Product Information	xv
Typographic Conventions	xvi

Section I. Scripting & Constraint Entry

Chapter 1. Assignment Editor

Introduction	1-1
Using the Assignment Editor	1-1
Category, Node Filter, Information & Edit Bars	1-2
Viewing & Saving Assignments in the Assignment Editor	1-6
Assignment Editor Features	1-7
Using the Enhanced Spreadsheet Interface	1-8
Dynamic Syntax Checking	1-9
Node Filter Bar	1-10
Using Assignment Groups	1-11
Customizable Columns	1-12
Tcl Interface	1-13
Assigning Pin Locations Using the Assignment Editor	1-14
Creating Timing Constraints Using the Assignment Editor	1-14
Exporting & Importing Assignments	1-15
Exporting Assignments	1-16
Exporting Pin Assignments	1-16
Importing Assignments	1-18
Conclusion	1-20
Document Revision History	1-20

Chapter 2. Command-Line Scripting

Introduction	2-1
The Benefits of Command-Line Executables	2-1
Introductory Example	2-2
Command-Line Executables	2-3
Command-Line Scripting Help	2-6
Command-Line Option Details	2-7
Option Precedence	2-8

Design Flow	2-11
Compilation with <code>quartus_sh --flow</code>	2-11
Text-Based Report Files	2-12
Makefile Implementation	2-13
Command-Line Scripting Examples	2-16
Create a Project & Apply Constraints	2-16
Check Design File Syntax	2-18
Create a Project & Synthesize a Netlist Using Netlist Optimizations	2-19
Archive & Restore Projects	2-19
Perform I/O Assignment Analysis	2-20
Update Memory Contents without Recompiling	2-20
Fit a Design as Quickly as Possible	2-21
Fit a Design Using Multiple Seeds	2-21
The QFlow Script	2-23

Chapter 3. Tcl Scripting

Introduction	3-1
What is Tcl?	3-2
Quartus II Tcl Packages	3-3
Loading Packages	3-5
Executables Supporting Tcl	3-9
Command-Line Options: <code>-t</code> , <code>-s</code> & <code>--tcl_eval</code>	3-10
Using the Quartus II Tcl Console Window	3-11
End-to-End Design Flows	3-12
Creating Projects & Making Assignments	3-13
EDA Tool Assignments	3-14
Using LogicLock Regions	3-18
Compiling Designs	3-21
Reporting	3-22
Creating CSV Files for Excel	3-24
Timing Analysis	3-25
Classic Timing Analysis	3-26
TimeQuest Timing Analysis	3-29
Automating Script Execution	3-30
Making the Assignment	3-31
Script Execution	3-31
Execution Example	3-32
Controlling Processing	3-33
Displaying Messages	3-33
Other Scripting Features	3-33
Natural Bus Naming	3-33
Using Collection Commands	3-34
Using the <code>post_message</code> Command	3-35
Accessing Command-Line Arguments	3-36
Using the Quartus II Tcl Shell in Interactive Mode	3-39
Quartus II Legacy Tcl Support	3-42
Tcl Scripting Basics	3-42

Hello World Example	3-42
Variables	3-43
Substitutions	3-43
Arithmetic	3-44
Lists	3-44
Arrays	3-45
Control Structures	3-46
Procedures	3-47
File I/O	3-48
Syntax & Comments	3-49
References	3-50

Chapter 4. Managing Quartus II Projects

Introduction	4-1
Creating a New Project	4-2
Using Revisions With Your Design	4-3
Creating & Deleting Revisions	4-3
Comparing Revisions	4-6
Creating Different Versions of Your Design	4-7
Archiving Projects with the Quartus II Archive Project Feature	4-8
Version-Compatible Databases	4-10
Quartus II Project Platform Migration	4-11
Filenames & Hierarchy	4-11
Search Path Precedence Rules	4-15
Quartus II-Generated Files for Third-Party EDA Tools	4-15
Migrating Database Files	4-15
Working with Messages	4-16
Messages Window	4-17
Hiding Messages	4-18
Message Suppression	4-19
Message Suppression Methods	4-21
Details & Limitations	4-21
Message Suppression Manager	4-22
Quartus II Settings File	4-25
Format Preservation	4-25
Quartus II Default Settings File	4-26
Scripting Support	4-26
Managing Revisions	4-27
Archiving Projects with a Tcl Command or at the Command Prompt	4-28
Restoring Archived Projects	4-28
Importing & Exporting Version-Compatible Databases	4-28
Specifying Libraries Using Scripts	4-29
Conclusion	4-30

Section II. I/O & PCB Tools

Chapter 5. I/O Management

I/O Planning Overview	5-1
Understanding Altera FPGA Pin Terminology	5-4
Package Pins	5-5
Pads	5-6
I/O Banks	5-6
VREF Groups	5-8
Importing & Exporting Pin Assignments	5-8
Comma Separated Value File	5-8
Quartus II Settings Files	5-9
Tcl Script	5-9
FPGA Xchange File	5-10
Pin-Out File	5-11
Creating Pin-Related Assignments	5-12
Assignment Editor	5-13
Tcl Scripts	5-17
Timing Closure Floorplan	5-18
Synthesis Attributes	5-19
Using the Pin Planner	5-21
Early I/O Planning Using the Pin Planner	5-52
Create a Megafunction or IP MegaCore Variation from the Pin Planner	5-53
Import a Megafunction or IP MegaCore Variation from the Pin Planner	5-54
Create a Top-Level Netlist for I/O Analysis	5-54
Using I/O Assignment Analysis to Validate Pin Assignments	5-58
I/O Assignment Analysis Design Flows	5-59
Inputs for I/O Assignment Analysis	5-66
Understanding the I/O Assignment Analysis Report & Messages	5-68
Scripting Support	5-76
Incorporating PCB Design Tools	5-78
Advanced I/O Timing	5-78
Default I/O Timing & Power with Capacitive Loading	5-79
Enabling & Configuring Advanced I/O Timing	5-81
Conclusion	5-90
Document Revision History	5-90

Chapter 6. Mentor Graphics PCB Design Tools Support

Introduction	6-1
FPGA-to-PCB Design Flow	6-2
Setting Up the Quartus II Software	6-5
Generating Pin-Out Files	6-7
Generating FPGA Xchange Files	6-7
Creating a Backup Quartus II Settings File	6-8
FPGA-to-Board Integration with the I/O Designer Software	6-8
I/O Designer Database Wizard	6-10
Updating Pin Assignments from the Quartus II Software	6-18
Sending Pin Assignment Changes to the Quartus II Software	6-21

Generating Symbols for the DxDesigner Software	6–23
Scripting Support	6–29
FPGA-to-Board Integration with the DxDesigner Software	6–31
DxDesigner Project Settings	6–31
DxDesigner Symbol Wizard	6–33
Conclusion	6–36

Chapter 7. Cadence PCB Design Tools Support

Introduction	7–1
Product Comparison	7–2
FPGA-to-PCB Design Flow	7–3
Setting Up the Quartus II Software	7–5
Generating Pin-Out Files	7–6
FPGA-to-Board Integration with the Cadence Allegro Design Entry HDL Software	7–6
Symbol Creation	7–6
Instantiating the Symbol in the Cadence Allegro Design Entry HDL Software	7–17
FPGA-to-Board Integration with Allegro Design Entry CIS	7–18
Allegro Design Entry CIS Project Creation	7–19
Generate Part	7–19
Split Part	7–22
Instantiate Symbol in Design Entry CIS Schematic	7–24
Altera Libraries for Design Entry CIS	7–25
Conclusion	7–27

Section III. Area, Timing & Power Optimization

Chapter 8. Area & Timing Optimization

Introduction	8–1
Optimization Process Stages	8–1
Design Space Explorer	8–2
Optimization Advisors	8–3
Initial Compilation	8–5
Device Setting	8–5
Smart Compilation Setting	8–6
Partitions & Floorplan Assignments for Incremental Compilation	8–6
Timing Requirement Settings	8–6
Optimize Hold Timing	8–9
Optimize Fast Corner Timing	8–10
Asynchronous Control Signal Recovery/Removal Analysis	8–10
Fitter Effort Setting	8–11
I/O Assignments	8–12
Early Timing Estimation	8–12
Design Assistant	8–13
Design Analysis	8–14
Error & Warning Messages	8–14

Ignored Timing Assignments	8-14
Resource Utilization	8-14
I/O Timing (Including t_{PD})	8-16
f_{MAX} Timing	8-18
Global Routing Resources	8-22
Compilation Time	8-23
Resource Utilization Optimization Techniques (LUT-Based Devices)	8-23
Resolving Resource Utilization Issues Summary	8-23
I/O Pin Utilization or Placement	8-24
Logic Utilization or Placement	8-25
Routing	8-36
I/O Timing Optimization Techniques (LUT-Based Devices)	8-40
Improving Setup & Clock-to-Output Times Summary	8-40
Timing-Driven Compilation	8-41
Fast Input, Output & Output Enable Registers	8-42
Programmable Delays	8-42
Use PLLs to Shift Clock Edges	8-45
Use Fast Regional Clocks in Stratix Devices & Regional Clocks in Stratix II Devices	8-46
Change How Hold Times are Optimized for MAX II Devices	8-46
f_{MAX} Timing Optimization Techniques (LUT-Based Devices)	8-47
Improving f_{MAX} Summary	8-47
Synthesis Netlist Optimizations & Physical Synthesis Optimizations	8-48
Turn Off Extra-Effort Power Optimization Settings	8-52
Optimize Synthesis for Speed, Not Area	8-52
Flatten the Hierarchy During Synthesis	8-53
Set the Synthesis Effort to High	8-54
Change State Machine Encoding	8-54
Duplicate Logic for Fan-Out Control	8-54
Prevent Shift Register Inference	8-55
Use Other Synthesis Options Available in Your Synthesis Tool	8-56
Fitter Seed	8-56
Optimize Source Code	8-57
LogicLock Assignments	8-58
Location Assignments & Back-Annotation	8-61
Resource Utilization Optimization Techniques (Macrocell-Based CPLDs)	8-65
Use Dedicated Inputs for Global Control Signals	8-65
Reserve Device Resources	8-66
Pin Assignment Guidelines & Procedures	8-66
Resolving Resource Utilization Problems	8-69
Timing Optimization Techniques (Macrocell-Based CPLDs)	8-73
Improving Setup Time	8-74
Improving Clock-to-Output Time	8-74
Improving Propagation Delay (t_{PD})	8-76
Improving Maximum Frequency (f_{MAX})	8-76
Optimizing Source Code—Pipelining for Complex Register Logic	8-77
Compilation-Time Optimization Techniques	8-80
Incremental Compilation	8-80
Reduce Synthesis Time & Synthesis Netlist Optimization Time	8-81

Check Early Timing Estimation before Fitting	8-81
Reduce Placement Time	8-82
Reduce Routing Time	8-84
Use Multiple Processors for Multi-Threaded Compilation	8-85
Scripting Support	8-86
Initial Compilation Settings	8-87
Resource Utilization Optimization Techniques (LUT-Based Devices)	8-87
I/O Timing Optimization Techniques (LUT-Based Devices)	8-88
fMAX Timing Optimization Techniques (LUT-Based Devices)	8-89
Conclusion	8-90
Document Revision History	8-91

Chapter 9. Power Optimization

Introduction	9-1
Power Dissipation	9-2
Design Space Explorer	9-3
Power-Driven Compilation	9-5
Power-Driven Fitter	9-10
Recommended Flow for Power-Driven Compilation	9-14
Area-Driven Synthesis	9-14
Gate-Level Register Retiming	9-16
Design Guidelines	9-19
Clock Power Management	9-19
Reducing Memory Power Consumption	9-22
Pipelining & Retiming	9-25
Architectural Optimization	9-28
I/O Power Guidelines	9-32
Power Optimization Advisor	9-34
Conclusion	9-37
Document Revision History	9-38

Chapter 10. Timing Closure Floorplan

Introduction	10-1
Invoking the Timing Closure Floorplan Editor	10-2
Timing Closure Floorplan Views	10-4
Viewing Assignments	10-6
Viewing Critical Paths	10-8
Physical Timing Estimates	10-12
LogicLock Region Connectivity	10-14
Viewing Routing Congestion	10-16
Conclusion	10-17
Document Revision History	10-18

Chapter 11. Netlist Optimizations & Physical Synthesis

Introduction	11-1
Synthesis Netlist Optimizations	11-3
WYSIWYG Primitive Resynthesis	11-3

Gate-Level Register Retiming	11-5
Preserving Synthesis Netlist Optimization Results	11-10
Physical Synthesis Optimizations	11-11
Automatic Asynchronous Signal Pipelining	11-13
Physical Synthesis for Combinational Logic	11-14
Physical Synthesis for Registers—Register Duplication	11-15
Physical Synthesis for Registers—Register Retiming	11-16
Preserving Your Physical Synthesis Results	11-17
Applying Netlist Optimization Options	11-19
Scripting Support	11-20
Synthesis Netlist Optimizations	11-20
Physical Synthesis Optimizations	11-21
Incremental Compilation	11-22
Back-Annotating Assignments	11-22
Conclusion	11-23
Document Revision History	11-23

Chapter 12. Design Space Explorer

Introduction	12-1
DSE Concepts	12-1
DSE Exploration	12-2
General Description	12-2
Timing Analyzer Support	12-4
DSE Flow	12-5
DSE Support for Altera Device Families	12-6
DSE Project Settings	12-7
Setting Up the DSE Work Environment	12-7
Specifying the Revision	12-7
Setting the Initial Seed	12-7
Restructuring LogicLock Regions	12-7
Quartus II Integrated Synthesis	12-9
Performing an Advanced Search in Design Space Explorer	12-9
Exploration Space	12-10
Optimization Goal	12-13
Quality of Fit (QoF)	12-14
Search Method	12-15
DSE Flow Options	12-15
Create a Revision from a DSE Point	12-15
Stop If Zero Failing Paths are Achieved	12-17
Continue Exploration Even If Base Compilation Fails	12-17
Run Quartus II PowerPlay Power Analyzer During Exploration	12-17
Archive All Compilations	12-17
Stop Flow After Time	12-17
Save Exploration Space to File	12-17
Ignore SignalTap & SignalProbe Settings	12-18
Skip Base Analysis & Compilation If Possible	12-18
Lower Priority of Compilation Threads	12-18

DSE Configuration File	12–18
DSE Advanced Information	12–19
Computer Load Sharing in DSE Using Distributed Exploration	12–19
Concurrent Local Compilations	12–21
Creating Custom Spaces for DSE	12–21
Chapter 13. LogicLock Design Methodology	
Introduction	13–1
Improving Design Performance	13–1
The Quartus II LogicLock Methodology	13–2
Preserving Timing Results Using the LogicLock Flow	13–3
Creating LogicLock Regions	13–4
Timing Closure Floorplan View	13–10
LogicLock Region Properties	13–11
Hierarchical (Parent and/or Child) LogicLock Regions	13–12
Assigning LogicLock Region Content	13–13
Excluded Resources	13–15
Tcl Scripts	13–17
Importing and Exporting LogicLock Regions	13–17
Additional Quartus II LogicLock Design Features	13–22
LogicLock Restrictions	13–31
Constraint Priority	13–31
Placing LogicLock Regions	13–32
Placing Memory, Pins & Other Device Features into LogicLock Regions	13–33
Back-Annotating Routing Information	13–34
Exporting Back-Annotated Routing in LogicLock Regions	13–35
Importing Back-Annotated Routing in LogicLock Regions	13–37
Scripting Support	13–38
Initializing & Uninitializing a LogicLock Region	13–38
Creating or Modifying LogicLock Regions	13–38
Obtaining LogicLock Region Properties	13–39
Assigning LogicLock Region Content	13–39
Prevent Further Netlist Optimization	13–39
Save a Node-level Netlist for the Entire Design into a Persistent Source File (.vqm)	13–40
Exporting LogicLock Regions	13–40
Importing LogicLock Regions	13–41
Setting LogicLock Assignment Priority	13–41
Assigning Virtual Pins	13–41
Back-Annotating LogicLock Regions	13–42
Conclusion	13–42
Chapter 14. Synplicity Amplify Physical Synthesis Support	
Introduction	14–1
Software Requirements	14–1
Amplify Physical Synthesis Concepts	14–2
Amplify-to-Quartus II Flow	14–3
Initial Pass: No Physical Constraints	14–3

Iterative Passes: Optimizing the Critical Paths	14-5
Using the Amplify Physical Optimizer Floorplans	14-6
Multiplexers	14-7
Independent Paths	14-9
Feedback Paths	14-9
Starting & Ending Points	14-9
Utilization	14-11
Detailed Floorplans	14-11
Forward Annotating Amplify Physical Optimizer Constraints into the Quartus II Software	14-12
Altera Megafunctions Using the MegaWizard Plug-In Manager with the Amplify Software	14-13
Conclusion	14-14



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: *November 2006*
 - Part number: *QII52001-6.1.0*

- Chapter 2. Command-Line Scripting
 - Revised: *November 2006*
 - Part number: *QII52002-6.1.0*

- Chapter 3. Tcl Scripting
 - Revised: *November 2006*
 - Part number: *QII52003-6.1.0*

- Chapter 4. Managing Quartus II Projects
 - Revised: *November 2006*
 - Part number: *QII52012-6.1.0*

- Chapter 5. I/O Management
 - Revised: *November 2006*
 - Part number: *QII52013-6.1.0*

- Chapter 6. Mentor Graphics PCB Design Tools Support
 - Revised: *November 2006*
 - Part number: *QII52015-6.1.0*

- Chapter 7. Cadence PCB Design Tools Support
 - Revised: *November 2006*
 - Part number: *QII52014-6.1.0*

- Chapter 8. Area & Timing Optimization
 - Revised: *November 2006*
 - Part number: *QII52005-6.1.0*

- Chapter 9. Power Optimization
 - Revised: *November 2006*
 - Part number: *QII51016-6.1.0*

Chapter 10. Timing Closure Floorplan

Revised: *November 2006*Part number: *QII52006-6.1.0*

Chapter 11. Netlist Optimizations & Physical Synthesis

Revised: *November 2006*Part number: *QII52007-6.1.0*

Chapter 12. Design Space Explorer

Revised: *November 2006*Part number: *QII52008-6.1.0*

Chapter 13. LogicLock Design Methodology

Revised: *November 2006*Part number: *QII52009-6.1.0*

Chapter 14. Synplicity Amplify Physical Synthesis Support

Revised: *November 2006*Part number: *QII52011-6.1.0*



About this Handbook

This handbook provides comprehensive information about the Altera® Quartus®II design software, version 6.1.

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) (7:00 a.m. to 5:00 p.m. Pacific Time)	(408) 544-7000 (1) (7:00 a.m. to 5:00 p.m. Pacific Time)
Product literature	www.altera.com	www.altera.com
Altera literature services	literature@altera.com (1)	literature@altera.com (1)
Non-technical customer service	(800) 767-3753	(408) 544-7000 (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.

Third-Party Software Product Information

Third-party software products described in this handbook are not Altera products, are licensed by Altera from third parties, and are subject to change without notice. Updates to these third-party software products may not be concurrent with Quartus II software releases. Altera has assumed responsibility for the selection of such third-party software products and its use in the Quartus II 6.1 software release. To the extent that the software products described in this handbook are derived from third-party software, no third party warrants the software, assumes any liability regarding use of the software, or undertakes to furnish you any support or information relating to the software. EXCEPT AS EXPRESSLY SET FORTH IN THE APPLICABLE ALTERA PROGRAM LICENSE SUBSCRIPTION AGREEMENT UNDER WHICH THIS SOFTWARE WAS PROVIDED TO YOU, ALTERA AND THIRD-PARTY LICENSORS DISCLAIM ALL WARRANTIES WITH RESPECT TO THE USE OF SUCH THIRD-PARTY SOFTWARE CODE OR DOCUMENTATION IN THE SOFTWARE, INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT. For more information, including the latest available version of specific third-party software products, refer to the documentation for the software in question.

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_{MAX} , lqdesigns directory, d: drive, chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75: High-Speed Board Design</i> .
<i>Italic type</i>	Internal timing parameters and variables are shown in italic type. Examples: <i>t_{PIA}</i> , <i>n + 1</i> . Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <file name>, <project name>.pdf 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. 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 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.
✓, —, N/A	Used in table cells to indicate the following: ✓ indicates a “Yes” or “Applicable” statement; — indicates a “No” or “Not Supported” statement; N/A indicates that the table cell entry is not applicable to the item of interest.
■ ● ●	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.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user's work.
	A warning calls attention to a condition or possible situation that can cause injury to the user.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information about a particular topic.

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 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:

- [Chapter 1, Assignment Editor](#)
- [Chapter 2, Command-Line Scripting](#)
- [Chapter 3, Tcl Scripting](#)
- [Chapter 4, Managing Quartus II Projects](#)



For information about the revision history for chapters in this section, refer to each individual chapter for that chapter's revision history.

Introduction

The complexity of today's FPGA designs is compounded by the increasing density and associated pin counts of current FPGAs. It requires that you make a large number of pin assignments that include the pin locations and I/O standards to successfully implement a complex design in the latest generation of FPGAs.

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
- Assignment Editor Features
- Assigning Pin Locations Using the Assignment Editor
- Creating Timing Constraints Using the Assignment Editor
- Exporting & 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, 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 window. This scalability makes it easy to view or edit your assignments right next to your design files. To open the Assignment Editor, click the **Assignment Editor** icon in the toolbar, or on the Assignments menu, click **Assignment Editor**.

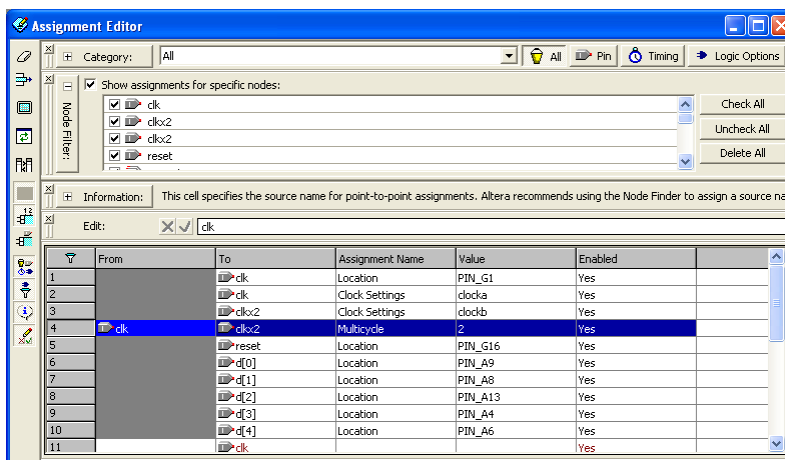


You can also launch the Assignment Editor by pressing Ctrl+Shift+A.

Category, Node Filter, Information & Edit Bars

The Assignment Editor window is divided into four bars and a spreadsheet (Figure 1–1).

Figure 1–1. The Assignment Editor Window



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.

Table 1–1. Assignment Editor Bar Descriptions

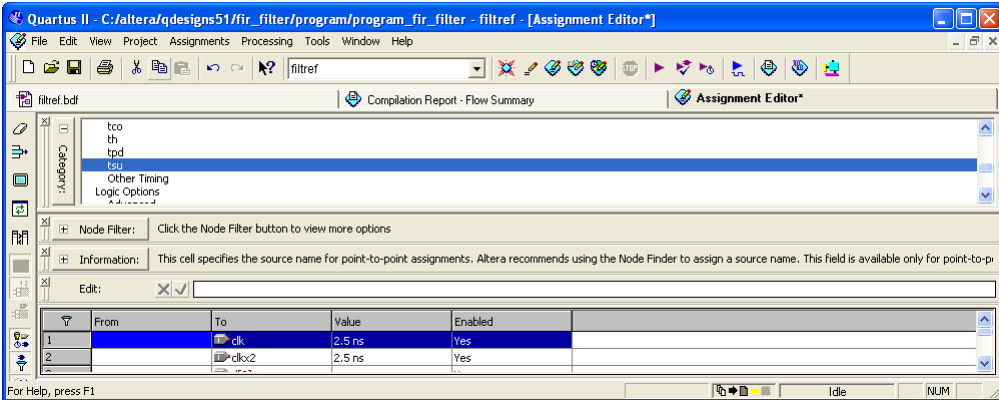
Bar Name	Description
Category	Lists the type of available assignments.
Node Filter	Lists a selection of design nodes to be viewed or assigned.
Information	Displays a description of the currently selected cell.
Edit	Allows you to edit the text in the currently selected cell(s).

Category Bar

The **Category** bar lists all assignment categories available for the selected 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.

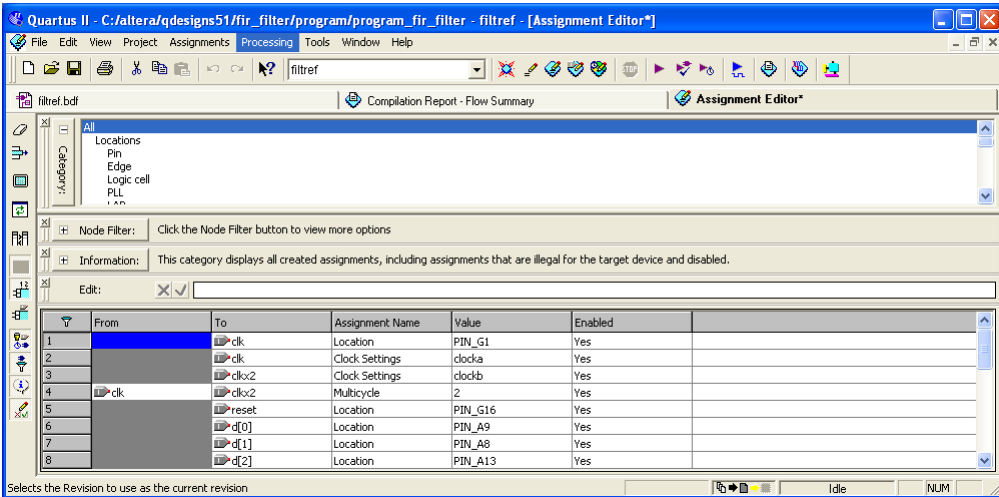
To view all t_{SU} assignments in your project, select **tsu** in the **Category** list (Figure 1–2).

Figure 1–2. t_{SU} Selected in the Category List



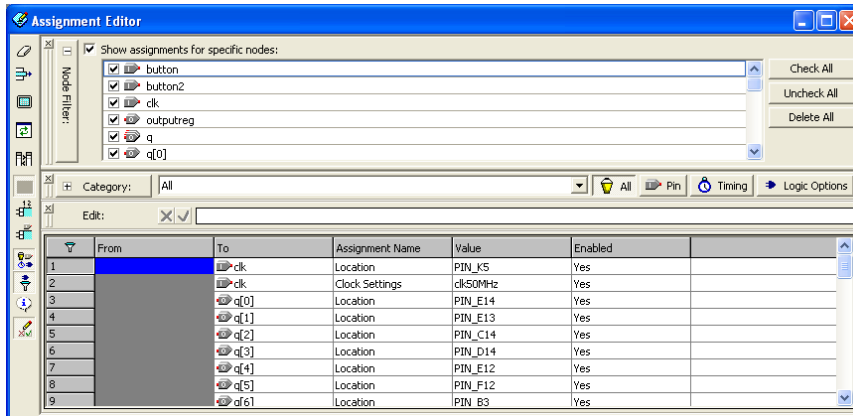
If you select **All** in the **Category** bar (Figure 1–3), the Assignment Editor displays all assignments.

Figure 1–3. All Selected in the Category List



When you collapse the **Category** bar, four shortcut buttons are displayed allowing you to select from various preset categories (Figure 1–4).

Figure 1–4. *Category Bar*



Use the **Pin** category to create pin location assignments. The **Pin** category displays additional information about each FPGA pin including its I/O Bank number, VREF group number, corresponding pad number, and primary and secondary functions.

When entering a pin number, the Assignment Editor auto completes the pin number. For example, instead of typing `Pin_AA3`, you can type `AA3` and let the Assignment Editor auto complete the pin number to `Pin_AA3`. You can also choose a pin location from the pins list by double clicking the cell in the location column. All occupied pin locations are shown in italics.

Node Filter Bar

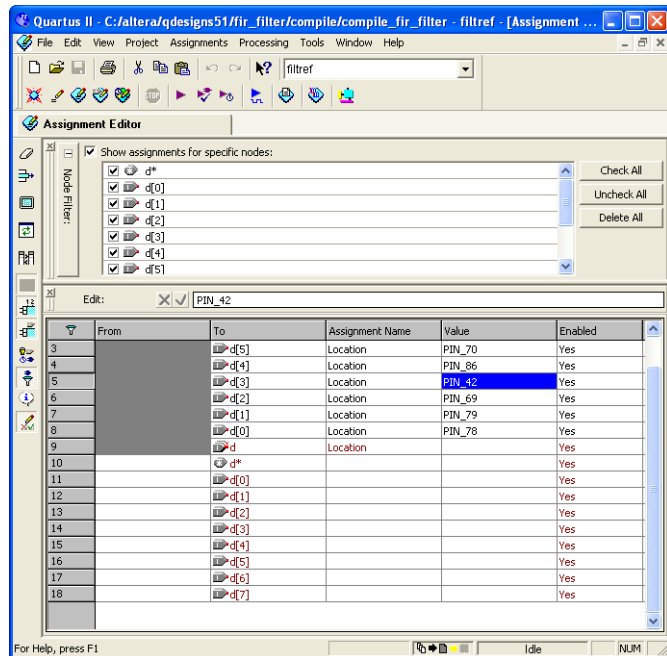
When **Show assignments for specific nodes** is turned on, the spreadsheet shows only assignments for nodes matching the selected node name filters in the **Node Filter** bar. You can selectively enable individual node name filters listed in the **Node Filter** bar. You can create a new node name filter 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 pre-populates 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 are shown in dark red. When you choose **Save** on the File menu, and there are incomplete assignments, a prompt gives you the choice to save and lose incomplete assignments, or cancel the save.

As shown in [Figure 1–5](#), when all the bits of the `d` input bus are enabled in the **Node Filter** bar, all unrelated assignments are filtered out.



In the **Node Filter** bar, selecting a `d` input bus only highlights the row. If you want to enable the bus, you must turn on the bus.

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



Information Bar

The **Information** bar provides a brief description of the currently selected cell and what information you should enter into the cell. For example, the **Information** bar describes if it is correct to enter a node name, or a number value into a cell. If the selected cell is a logic option, then the **Information** bar shows a description of that option.



For more information on logic options, refer to the 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 (Figure 1–6), then type the new value into the Edit box in the Edit bar, and click Accept (Figure 1–7).

Figure 1–6. Edit Bar Selection

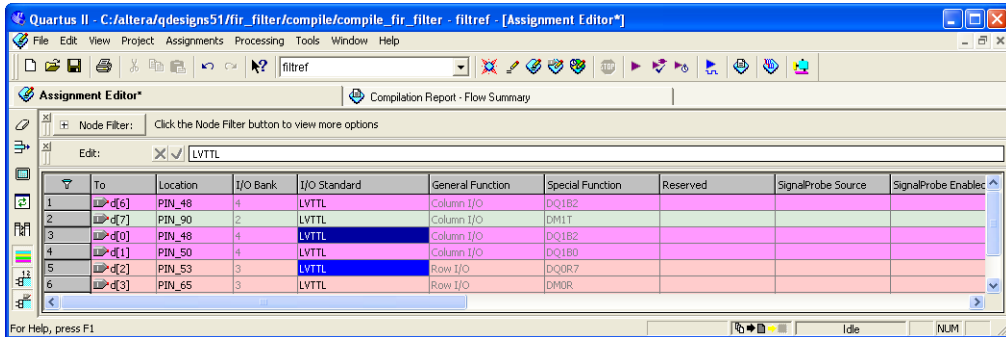
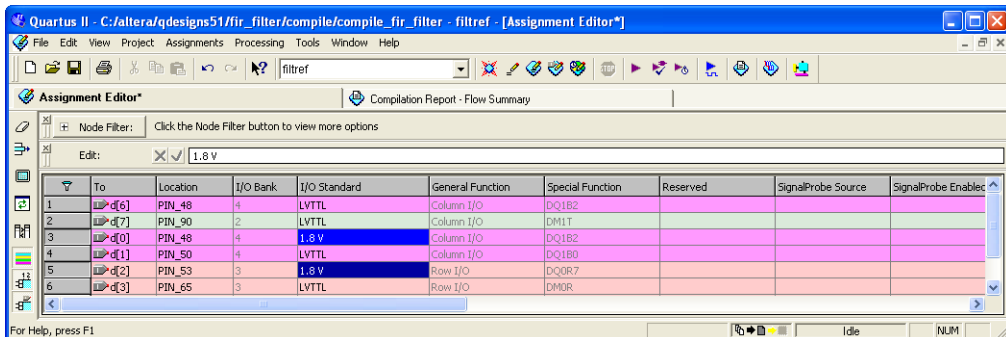


Figure 1–7. Edit Bar Change



Viewing & Saving Assignments in the Assignment Editor

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, you can refresh the Assignment Editor after you add, remove, or change an assignment outside the Assignment Editor.

By default, all assignments made in the Quartus II software are first stored into memory, then to the Quartus II Setting File (**.qsf**) on the disk after you start a processing task, or if you save or close your project. Saving assignments to memory avoids reading and writing to your disk drive and improves the performance of the software.

After making assignments in the Assignment Editor, on the File menu, click **Save** to save your assignments and update the Quartus II Settings File outside the Assignment Editor.

Starting with the Quartus II software version 5.1, you can force all assignments to be written to a disk drive. This is performed by turning off **Update assignments to disk during design processing only** in the **Processing** page of the **Options** settings dialog box on the Tools menu.



For more information on how the Quartus II software writes to the Quartus II Settings File, refer to the *Quartus II Project Management* chapter in volume 2 of the *Quartus II Handbook*.

You can refresh the Assignment Editor window by clicking **Refresh** from the View menu. If you make an assignment in the Quartus II software, such as in the Tcl console or in the Pin Planner, the Assignment Editor reloads the new assignments from memory. If you directly modify the Quartus II Settings File outside the Assignment Editor, click **Refresh** on the View menu to view the assignments.



If the Quartus II Settings File is edited while the project is open, go to the File menu and click **Save Project** to ensure that you are editing the latest Quartus II Settings File.

Each time the Assignment Editor is refreshed, the following message displays in the Message window:

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

Assignment Editor Features

You can open the Assignment Editor from many locations in the Quartus II software, including the Text Editor, the Node Finder, the Timing Closure Floorplan, the Pin Planner, the Compilation Report, and the Messages window. For example, you can highlight a node name in your design file and open the Assignment Editor with the node name populated.








You can also open other windows from the Assignment Editor. From a node listed in the Assignment Editor spreadsheet, you can locate the node in any of the following windows: Pin Planner, Timing Closure Floorplan, Chip Editor, Block Editor, or 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 list boxes, and copy and paste multiple cells into the Assignment Editor. As you enter an assignment, the font color of the row changes to indicate the status of the assignment. Refer to “[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 is shown beside each node name and node name filter to identify its type. The node type icon identifies the entry as an input, output, bidirectional pin, a register, combinational logic, or an assignment group ([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

	To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reserved	SignalProbe Source
1	 follow	PIN_68	3	LVTTTL	Row I/O	VREF1B3		
2	 d[6]	PIN_48	4	LVTTTL	Column I/O	DQ1B2		
3	 d[7]	PIN_90	2	LVTTTL	Column I/O	DM1T		
4	 d*	PIN_99	2	LVTTTL	Column I/O	DEV_OE/DQ1T6		
5	 inst4	PIN_98	2	LVTTTL	Column I/O	DQ1T5		
6	 inst5	PIN_57	3	LVTTTL	Row I/O	VREF2B3		
7	 inst5[0]	PIN_29	4	LVTTTL	Column I/O	DQ1B5		

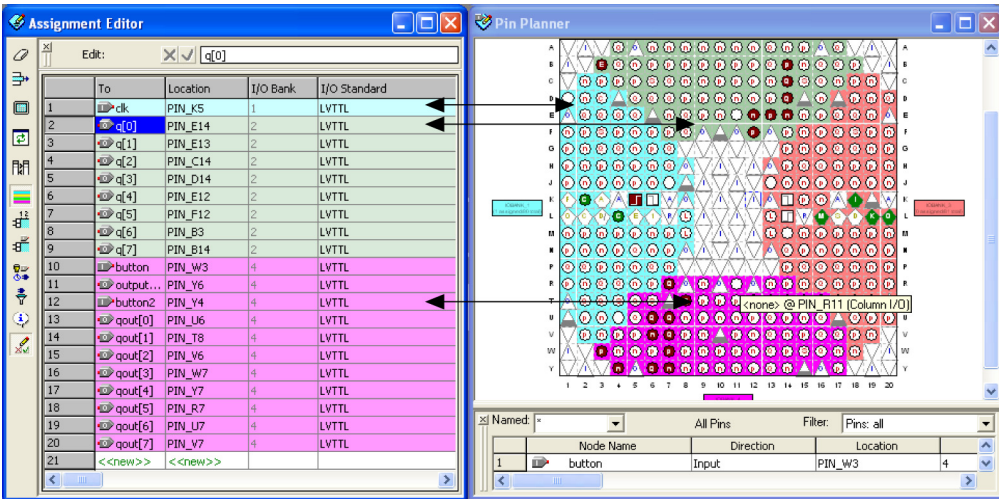
The Assignment Editor supports wildcards in the following types of assignments:

- All timing assignments
- Point-to-point global signal assignments (applicable to Stratix® II and Stratix devices)
- Point-to-point or pad-to-core delay chain assignments
- All assignments that support wild cards are shown in the drop list under the Assignment Name column of the Assignment Editor with “(Accepts wildcards/groups)” displayed beside it

The spreadsheet also supports customizable columns that allow you to show, hide, and arrange columns. For more information, refer to “[Customizable Columns](#)” on page 1–12.

When making pin location assignments, the background color of the cells coordinates with the color of the I/O bank shown in the Pin Planner ([Figure 1–9](#)).

Figure 1–9. Spreadsheet-Like Interface



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 rejects incorrect settings. For example, the Assignment Editor does not allow assignment of a pin name 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 (Table 1–2). To customize the colors in the Assignment Editor, on the Tools menu, click **Options**.

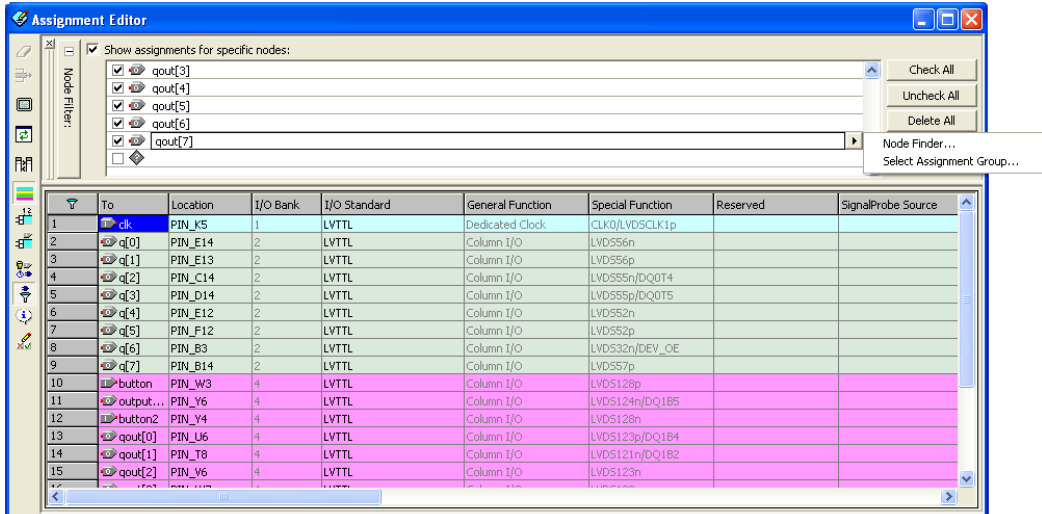
Table 1–2. Description of the Text Color in the Spreadsheet

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

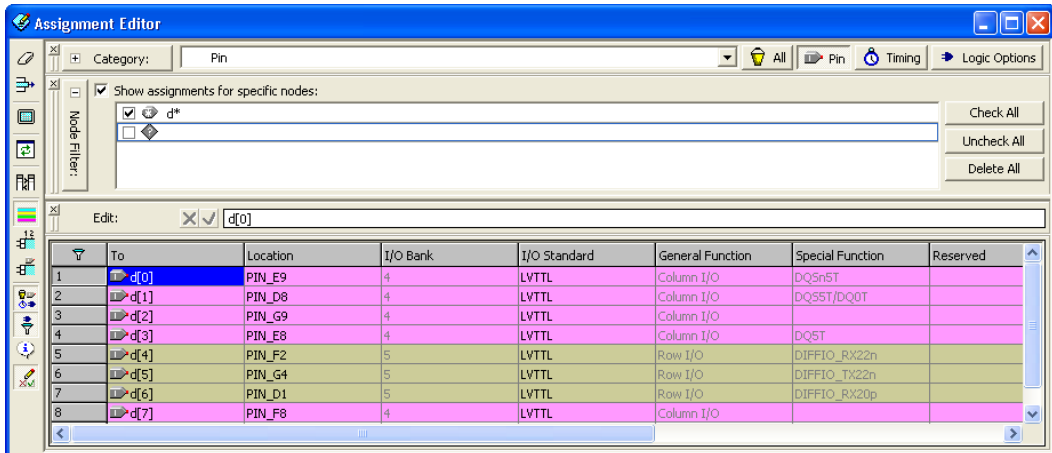
The **Node Filter** bar provides flexibility in how you view and make your settings. The **Node Filter** bar contains a list of node filters. To create a new entry, use the **Node Finder** or manually type the node name. Double-click an empty row in the **Node Filter** list, click on the **arrow**, and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

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 ?) are used to filter for a selection of all the design nodes with one entry in the Node Filter. For example, you can enter `d*` into the **Node Filter** list to view all assignments for `d[0]`, `d[1]`, `d[2]`, and `d[3]` (Figure 1–11).

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



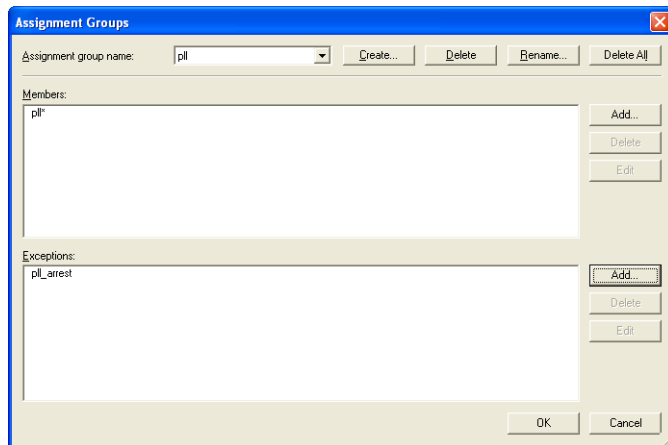
Using Assignment Groups

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

To create an assignment group, on the Assignments menu, click **Assignment (Time) Groups**. The **Assignment Groups** dialog box is shown. You can add or delete members of each assignment group with wild cards in the Node Finder (Figure 1–12).



For more information on using Assignment Groups for timing analysis, refer to the *Classic Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Figure 1–12. Assignment Groups Dialog Box

There are cases when wildcards are not flexible enough to select a large number of nodes that have similar node names. You can use assignment groups to combine wildcards, which select a large number of nodes, and use exceptions to remove nodes that you did not intend to select. Although settings may not always display correctly when you have wildcards or assignment groups, the fitter always recognizes assignments created with wildcards and assignment groups when the design is compiled.

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, and align the content of the column left, center, or right for improved readability.

When the Quartus II software starts for the first time, you see a pre-selected set of columns. For example, when the Quartus II software is first started, the Comment column is hidden. To show or hide any of the available columns, on the View menu, click **Customize Columns**. When you restart the Quartus II software, your column settings are maintained.


Depending on the category selected, there are many different hidden columns you can display. For example, with the Pins category selected, there are many columns that are not shown by default, such as VREF group, pad number, output pin load, toggle rate, timing requirements, and fast input and output register options.

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.

 Even though you can make many pin-related assignments with the **Pin** category selected, only the pin location assignment is disabled when you disable a row using the Enabled column.

Tcl Interface

Whether you use the Assignment Editor or another feature to create your design assignments, you can export them to a Tcl file. You can then use the Tcl file to reapply the settings or to archive your assignments. On the File menu, click **Export** to export your assignments (currently displayed in the spreadsheet of the Assignment Editor) to a Tcl script.

 On the Project menu, click **Generate TCL File for Project** to generate a Tcl script file that sets up your design and applies all the assignments.

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

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"

```

System | Processing | Extra Info | Info | Warning | Critical Warning | Error | Suppressed



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

Assigning Pin Locations Using the Assignment Editor

There are two methods for making pin assignments with the Assignment Editor. The first approach involves choosing a design node name for each device pin location. It is important to understand the properties of each pin on the FPGA device before you assign a design node to the location. For example, when following pin placement guidelines, you need to know which I/O bank or VREF group each pin belongs to.

On the Assignments menu, click **Assignment Editor**. To view all pin numbers in the targeted package, click the **Pin** category. On the View menu, click **Show All Assignable Pin Numbers**. You can customize the columns shown in the Assignment Editor to display property information about each pin including their pad numbers, as well as primary and secondary functions.



For more information on pin placement guidelines, refer to the *Selectable I/O Standards* chapters in the appropriate device handbook.

The second approach involves choosing a pin location for each pin in your design. To view all pin numbers in the targeted package, open the **Assignment Editor**, click the **Pin** category, and on the View menu, click **Show All Known Pin Names**. For each pin name, select a pin location.



For more information about creating pin assignments, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Creating Timing Constraints Using the Assignment Editor

Accurate timing constraints guide the place-and-route engine in the Quartus II software to help optimize your design into the FPGA. After completing a place-and-route, perform a static timing analysis using the classic timing analyzer or the TimeQuest timing analyzer to analyze slack and critical paths in your design.

If you are using the Classic Timing Analyzer, create timing constraints using the Assignment Editor. On the Assignments menu, click **Assignment Editor**. In the **Category** list, select **Timing**, and make timing assignments in the spreadsheet section of the Assignment Editor.



For more information on the Classic Timing Analyzer, refer to the *Classic Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

If you are using the TimeQuest Timing Analyzer, the TimeQuest Timing Analyzer uses timing assignments from a Synopsys Design Constraint (.sdc) file.



For information on converting the timing assignments in your Quartus Settings File to an Synopsys Design Constraint file, refer to the *Switching to the TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Exporting & Importing Assignments

Designs that use the LogicLock™ hierarchal design methodology use the **Import Assignment** command to import assignments into the current project. You can also use the **Export Assignments** command to save all the assignments in your project to a file to be used for archiving or to transfer assignments from one project to another.

On the Assignments menu, click **Export Assignments** or **Import Assignments** to do the following:

- Export your Quartus II assignments to a Quartus II Settings File.
- Import assignments from a Quartus II Entity Settings File (**.esf**), a MAX+PLUS® II Assignment and Configuration File (**.acf**), or a Comma Separated Value (**.csv**) file.

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



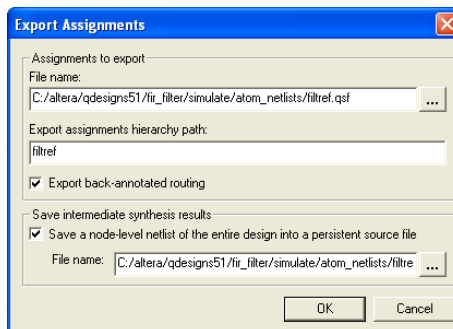
When applicable, the **Export** command exports the contents of the active window in the Quartus II software to another file format.

You can use these file formats for many different aspects of your project. For example, you can use a Comma Separated Value 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. The LogicLock design flow uses the Quartus II Settings File to transfer your LogicLock region settings.

Exporting Assignments

You can use the **Export Assignments** dialog box to export your Quartus II software assignments into a Quartus II Settings File, generate a node-level netlist file, and export back-annotated routing information as a Routing Constraints File (.rcf) (Figure 1–14).

Figure 1–14. Export Assignments Dialog Box



On the Assignments menu, click **Export Assignments** to open the **Export Assignments** dialog box. The LogicLock design flow also uses this dialog box to export LogicLock regions.



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

On the File menu, click **Export** to export all assignments to a Tcl file or export a set of assignments to a Comma Separated Value file. When you export assignments to a Tcl file, only user-created assignments are written to the Tcl script file; default assignments are not exported.

When assignments are exported to a Comma Separated Value file, only the assignments displayed in the current view of the Assignment Editor are exported.

Exporting Pin Assignments

To export your pin assignments to a Comma Separated Value file, you can open the Assignment Editor and select **Pin** from the Category bar. The **Pin** category displays detailed properties about each pin similar to that of the device pin-out files in addition to the pin name and pin number. On the File menu, click **Export**, and select **Comma Separated Value File** from the **Save as type** list.

The first uncommented row of the Comma Separated Value 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 (Figure 1–15). On the View menu, click **Customize Columns** to add and remove columns that are displayed in the spreadsheet. You can view and make edits to the Comma Separated Value file with Excel or other spreadsheet tools. If you intend to import the Comma Separated Value file back into the Quartus II software, the column headings must remain unedited and in the same order.



For more information on exporting pin assignments, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Figure 1–15. Assignment Editor With Category Set to Pin

	To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reserved
1	clk	PIN_N20	1	LVTTTL	Dedicated Clock	CLK3p, Input	
2	clkx2	PIN_M21	2	LVTTTL	Dedicated Clock	CLK1p, Input	
3	d[0]	PIN_E9	4	LVTTTL	Column I/O	DQS5n5T	
4	d[1]	PIN_D8	4	LVTTTL	Column I/O	DQS5T/DQ0T	
5	d[2]	PIN_G9	4	LVTTTL	Column I/O		
6	d[3]	PIN_E8	4	LVTTTL	Column I/O	DQ5T	
7	d[4]	PIN_F2	5	LVTTTL	Row I/O	DIFFIO_RX22n	
8	d[5]	PIN_G4	5	LVTTTL	Row I/O	DIFFIO_TX22n	
9	d[6]	PIN_D1	5	LVTTTL	Row I/O	DIFFIO_RX20p	
10	d[7]	PIN_F8	4	LVTTTL	Column I/O		
11	follow	PIN_F9	4	LVTTTL	Column I/O	DQ5T	
12	newt	PIN_G6	5	LVTTTL	Row I/O	DIFFIO_TX21n	
13	reset	PIN M2	5	LVTTTL	Dedicated Clock	CLK11p, Input	

The following code is an example of an exported Comma Separated Value 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,Enabled
clk,PIN_N20,1,LVTTL,Dedicated Clock,"CLK3p, Input",,Yes
clkx2,PIN_M21,2,LVTTL,Dedicated Clock,"CLK1p, Input",,Yes
d[0],PIN_E9,4,LVTTL,Column I/O,DQSn5T,,Yes
d[1],PIN_D8,4,LVTTL,Column I/O,DQS5T/DQ0T,,Yes
d[2],PIN_G9,4,LVTTL,Column I/O,,Yes
d[3],PIN_E8,4,LVTTL,Column I/O,DQ5T,,Yes
d[4],PIN_F2,5,LVTTL,Row I/O,DIFFIO_RX22n,,Yes
d[5],PIN_G4,5,LVTTL,Row I/O,DIFFIO_TX22n,,Yes
d[6],PIN_D1,5,LVTTL,Row I/O,DIFFIO_RX20p,,Yes
d[7],PIN_F8,4,LVTTL,Column I/O,,Yes
```

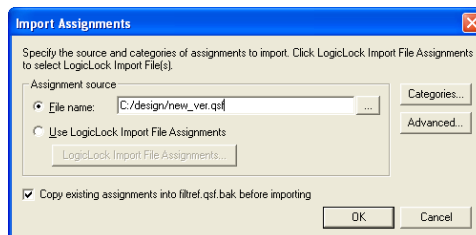
Importing Assignments

The **Import Assignments** dialog box allows you to import Quartus II assignments from a Quartus II Settings File, a Quartus II Entity Settings File, a MAX+PLUS II Assignment Configuration File, or a Comma Separated Value (Figure 1–16).

To import assignments from any of the supported assignment files, perform the following steps:

1. On the Assignments menu, click **Import Assignments**. The **Import Assignments** dialog box is shown (Figure 1–16).

Figure 1–16. Import Assignments Dialog Box



2. In the **File name** text-entry box, type the file name, or browse to the assignment file. The **Select File** dialog box is shown.
3. In the **Select File** dialog box, select the file, and click **Open**.
4. Click **OK**.



When you import a Comma Separated Value file, the first uncommented row of the file must be in the exact format as it was when exported.

When using the LogicLock™ flow methodology to import assignments, perform the following steps:

1. On the Assignments menu, click **Import Assignments**. The **Import Assignments** dialog box appears (Figure 1-16).
2. Turn on **Use LogicLock Import File Assignments**, and click **LogicLock Import File Assignments**.
3. When the **LogicLock Import File Assignments** dialog box opens, select the assignments to import and click **OK**.



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

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

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

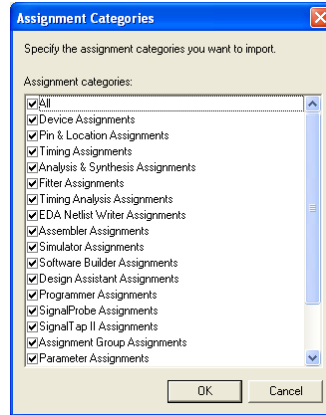
1. Click **Categories** in the **Import Assignments** dialog box.
2. Turn on the categories you want to import from the **Assignment categories** list (Figure 1-17).

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



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

Figure 1–17. Assignment Categories Dialog Box



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**, **Node Filter**, **Information**, and **Edit** bars, the Assignment Editor provides an efficient assignment entry solution for FPGA designers.

Document Revision History

Table 1–3 shows the revision history of this document.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history to document.	
May 2006 v6.0.0	Minor updates for the Quartus II software version 6.0.0. <ul style="list-style-type: none"> Added Classic Timing Analyzer and TimeQuest Timing Analyzer information. 	
October 2005 v5.1.0	Updated for the Quartus II software version 5.1.0.	

May 2005 v5.0.0	<ul style="list-style-type: none">• Updated for the Quartus II software version 5.0.0.• General formatting and editing updates.• Updated 2 graphics and references to reflect changes in the Quartus II software version 5.0.0	
Dec. 2004 v2.1	<ul style="list-style-type: none">• Updated for Quartus II software version 4.2:• General formatting and editing updates.• Updated information about refreshing the Assignment Editor.• Updated figures.• Added information about how to make selections to the Assignment Editor window.• Added Time Groups reference.• Reworded description of Customizable Columns.• Added new section Creating Pin Locations Using the Assignment Editor.• Added new description to Exporting & Importing Assignments.	
June 2004 v2.0	<ul style="list-style-type: none">• Updates to tables, figures.• New functionality in the Quartus II software version 4.1.	
Feb. 2004 v1.0	Initial release.	

Introduction

FPGA design software that easily integrates into your design flow saves time and improves productivity. The Altera® Quartus® II software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

The benefits provided by command-line executables include:

- Command-line control over each step of the design flow
- Easy integration with scripted design flows including makefiles
- Reduced memory requirements
- Improved performance

The command-line executables are also completely compatible with the Quartus II GUI, allowing you to use the exact combination of tools that you prefer.

This chapter describes how to take advantage of Quartus II command-line executables, and provides several examples of scripts that automate different segments of the FPGA design flow.

The Benefits of Command-Line Executables

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

Command-line executables allow for easy integration with scripted design flows. It is simple to create scripts in any language with a series of commands. These scripts can be batch-processed, allowing for integration with distributed computing in server farms. You can also integrate the Quartus II command-line executables 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.

Command-line executables add integration and scripting flexibility without sacrificing the ease-of-use of the Quartus II GUI. You can use the Quartus II GUI and command-line executables at different stages in the design flow. For example, you might use the Quartus II GUI to edit the

floorplan for the design, use the command-line executables to perform place-and-route, and return to the Quartus II GUI to perform debugging with the Chip Editor.

Command-line executables reduce the amount of memory required during each step in the design flow. Because each executable targets only one step in the design flow, it is relatively compact, both in file size and the amount of memory used when running. This memory reduction improves performance, and is particularly beneficial in design environments where computer networks or workstations are heavily used with reduced memory.

Introductory Example

The following introduction to design flow with command-line 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 four commands shown in [Example 2-1](#) at a command prompt in the new project directory:



The *<Quartus II directory>/bin* directory must be in your PATH environment variable.

Example 2-1. Introductory Example

```
quartus_map filtref --source=filtref.bdf --family=CYCLONE ←  
quartus_fit filtref --part=EP1C12Q240C6 --fmax=80MHz --tsu=8ns ←  
quartus_asm filtref ←  
quartus_tan 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. This 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_asm filtref** command creates programming files for the **filtref** project.

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

You can put the four commands from [Example 2-1](#) into a batch file or script file, and run them. For example, you can create a simple UNIX shell script called `compile.sh`, which includes the code shown in [Example 2-2](#).

Example 2-2. UNIX Shell Script: `compile.sh`

```
#!/bin/sh
PROJECT=filtref
TOP_LEVEL_FILE=filtref.bdf
FAMILY=Cyclone
PART=EP1C12Q240C6
FMAX=80MHz
quartus_map $PROJECT --source=$TOP_LEVEL_FILE --family=$FAMILY
quartus_fit $PROJECT --part=$PART --fmax=$FMAX
quartus_asm $PROJECT
quartus_tan $PROJECT
```

Edit the script as necessary and compile your project.

Command-Line Executables

[Table 2-1](#) details the command-line executables and their respective descriptions.

<i>Table 2-1. Quartus II Command-Line Executables & Descriptions (Part 1 of 4)</i>	
Executable	Description
Analysis & Synthesis <code>quartus_map</code>	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 <code>quartus_fit</code>	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. Quartus II Analysis & Synthesis must be run successfully before running the Fitter.

Table 2–1. Quartus II Command-Line Executables & Descriptions (Part 2 of 4)

Executable	Description
Assembler quartus_asm	<p>The Quartus II Assembler generates a device programming image, in the form of one or more of the following from a successful fit (that is, place-and-route).</p> <ul style="list-style-type: none"> ● Programmer Object Files (.pof) ● SRAM Object Files (.sof) ● Hexadecimal (Intel-Format) Output Files (.hexout) ● Tabular Text Files (.tff) ● Raw Binary Files (.rbf) <p>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 Hexadecimal (Intel-Format) Output Files, Tabular Text Files, and Raw Binary Files can be used by other programming hardware manufacturers that provide support for Altera devices.</p> <p>The Quartus II Fitter must be run successfully before running the Assembler.</p>
Classic Timing Analyzer quartus_tan	<p>The Quartus II Classic Timing Analyzer computes delays for the given design and device, and annotates them on the netlist. Then, the Classic Timing Analyzer performs timing analysis, allowing you to analyze the performance of all logic in your design. The quartus_tan executable includes Tcl support.</p> <p>Quartus II Analysis & Synthesis or the Fitter must be run successfully before running the Classic Timing Analyzer.</p>
TimeQuest Timing Analyzer quartus_sta	<p>The Quartus II TimeQuest Timing Analyzer computes delays for the given design and device, and annotates them on the netlist. Then, the TimeQuest Timing Analyzer performs timing analysis, allowing you to analyze the performance of all logic in your design. The quartus_sta executable includes Tcl support and SDC support.</p> <p>Quartus II Analysis & Synthesis or the Fitter must be run successfully before running the TimeQuest Timing Analyzer.</p>
Design Assistant quartus_drc	<p>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. The Design Assistant supports designs that target any Altera device supported by the Quartus II software, except MAX® 3000 and MAX 7000 devices.</p> <p>Quartus II Analysis & Synthesis or the Fitter must be run successfully before running the Design Assistant.</p>
Compiler Database Interface quartus_cdb	<p>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.</p> <p>Analysis & Synthesis must be run successfully before running the Compiler Database Interface.</p>

Table 2–1. Quartus II Command-Line Executables & Descriptions (Part 3 of 4)

Executable	Description
EDA Netlist Writer quartus_eda	<p>The Quartus II EDA Netlist Writer generates netlist and other output files for use with other EDA tools.</p> <p>Analysis & Synthesis, the Fitter, or Timing Analyzer must be run successfully before running the EDA Netlist Writer, depending on the arguments used.</p>
Simulator quartus_sim	<p>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.</p> <p>Quartus II Analysis & Synthesis must be run successfully before running a functional simulation.</p> <p>The Timing Analyzer must be run successfully before running a timing simulation.</p>
Power Analyzer quartus_pow	<p>The Quartus II PowerPlay Power Analyzer estimates the thermal dynamic power and the thermal static power consumed by the design. For newer families such as Stratix® II and MAX II, the power drawn from each power supply is also estimated.</p> <p>Quartus II Analysis & Synthesis or the Fitter must be run successfully before running the PowerPlay Power Analyzer.</p>
Programmer quartus_pgm	<p>The Quartus II Programmer programs Altera devices. The Programmer uses one of the supported file formats:</p> <ul style="list-style-type: none"> ● Programmer Object Files (.pof) ● SRAM Object Files (.sof) ● Jam File (.jam) ● Jam Byte-Code File (.jbc) <p>Make sure you specify a valid programming mode, programming cable, and operation for a specified device.</p>
Convert Programming File quartus_cpf	<p>The Quartus II Convert Programming File module converts one programming file format to a different possible format.</p> <p>Make sure you specify valid options and an input programming file to generate the new requested programming file format.</p>
Quartus Shell quartus_sh	<p>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. 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.</p>

Table 2–1. Quartus II Command-Line Executables & Descriptions (Part 4 of 4)

Executable	Description
TimeQuest Timing Analyzer GUI quartus_staw	This executable opens the TimeQuest Timing Analyzer GUI. This is helpful because you don't have to open the entire Quartus II GUI for certain operations.
Programmer GUI quartus_pgmw	This executable opens up the programmer—a GUI to the quartus_pgm executable. This is helpful because users don't have to open the entire Quartus II GUI for certain operations

Command-Line Scripting Help

Help on command-line executables is available through different methods. You can access help built in to the executables with command-line options. You can use the Quartus II Command-Line and Tcl API Help browser for an easy graphical view of the help information. Additionally, you can refer to the *Scripting Reference Manual* on the Quartus II literature page on Altera's website, which has the same information in PDF format.

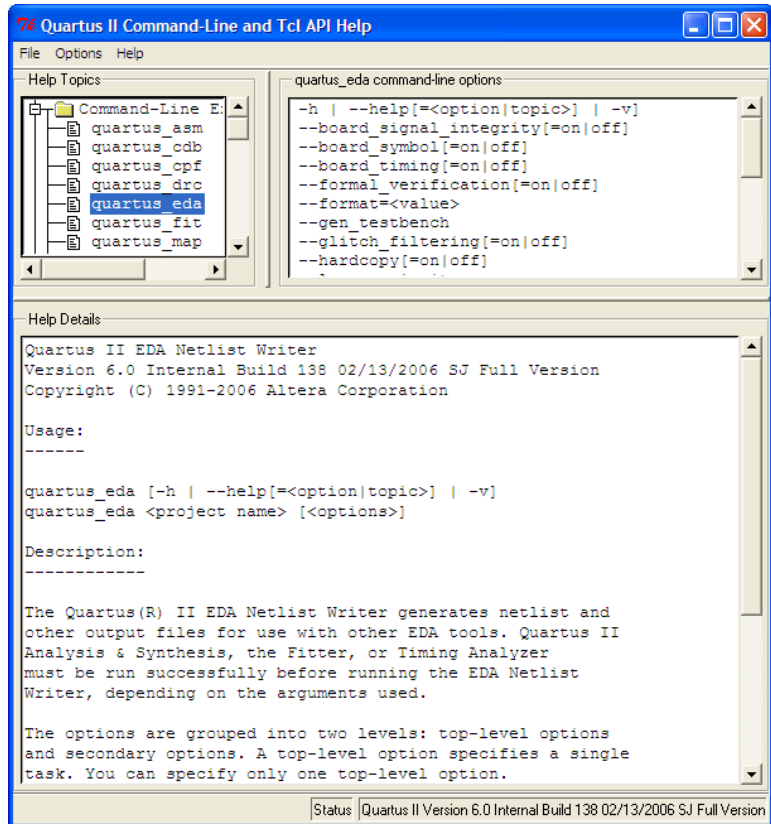
To use the Quartus II Command-Line and Tcl API Help browser, type the following:

```
quartus_sh --qhelp ←
```

This command starts the Quartus II Command-Line and Tcl API Help browser, a viewer for information about the Quartus II Command-Line executables and Tcl API (Figure 2–1).

Use the `-h` option with any of the Quartus II Command-Line executables to get a description and list of supported options. Use the `--help=<option name>` option for detailed information about each option.

Figure 2–1. Quartus II Command-Line & Tcl API Help Browser



Command-Line Option Details

Command-line options are provided for 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 command-line executables that are run after this update use the updated assignment. Refer to “[Option Precedence](#)” on page 2–8 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, choose this method.



Refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. Scripting information for all Quartus II project settings and assignments is located in the *QSF Reference Manual*.

Option Precedence

If you use command-line executables, you should 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 for the project. Assignments for a project can also be made with 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 Quartus II Settings File or the compiler database files. There are two command-line options to specify whether Quartus II Settings File 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 Quartus II Settings File or compiler database file is set to its default value.

The file precedence command-line options are `--read_settings_files` and `--write_settings_files`.

By default, the `--read_settings_files` and `--write_settings_files` options are turned on. Turning on the `--read_settings_files` option causes a command-line executable to read assignments from the Quartus II Settings File instead of from the compiler database files. Turning on the `--write_settings_files` option causes a command-line 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 `--read_settings_files` option.

Option Specified	Precedence for Reading Assignments
<code>--read_settings_files = on</code> (default)	<ol style="list-style-type: none"> 1. Command-line options 2. Quartus II Settings File 3. Project database (db directory, if it exists) 4. Quartus II software defaults
<code>--read_settings_files = off</code>	<ol style="list-style-type: none"> 1. Command-line options 2. Project database (db directory, if it exists) 3. Quartus II software defaults

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

Option Specified	Location for Writing Assignments
<code>--write_settings_files = on</code> (Default)	Quartus II Settings File and compiler database
<code>--write_settings_files = off</code>	Compiler database

Example 2-3 assumes that a project named `fir_filter` exists, and that the analysis and synthesis step has been performed (using the `quartus_map` executable).

Example 2-3. Write Settings Files

```
quartus_fit fir_filter --fmax=80MHz ←
quartus_tan fir_filter ←
quartus_tan fir_filter --fmax=100MHz --tao=timing_result-100.tao
--write_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 `--write_settings_files=off` option, the command-line 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 `--write_settings_files=off` option is not specified, the command-line executable updates the Quartus II Settings File to reflect the 100-MHz global f_{MAX} requirement.

Use the options `--read_settings_files=off` and `--write_settings_files=off` (where appropriate) to optimize the way that the Quartus II software reads and updates settings files.

[Example 2-4](#) shows how to avoid unnecessary reading and writing.

Example 2-4. Avoiding Unnecessary Reading & Writing

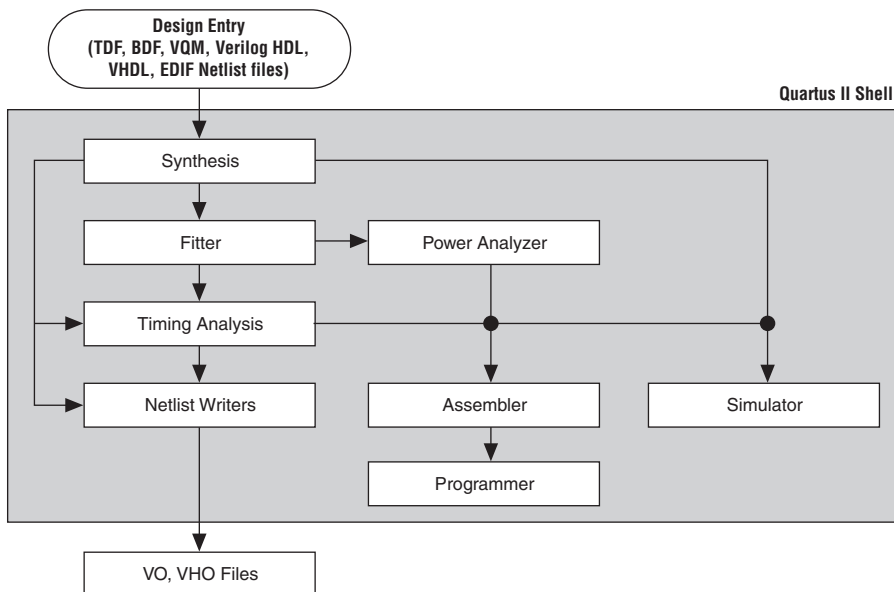
```
quartus_map filtref --source=filtref
  --part=ep1s10f780c5 ←
quartus_fit filtref --fmax=100MHz
  --read_settings_files=off ←
quartus_tan filtref --read_settings_files=off
  --write_settings_files=off ←
quartus_asm filtref --read_settings_files=off
  --write_settings_files=off ←
```

The `quartus_tan` and `quartus_asm` executables do not read or write settings files because they do not change any settings in the project.

Design Flow

Figure 2–2 shows a typical design flow.

Figure 2–2. Typical Design Flow



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 about 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` executable to perform command-line compilations in earlier versions of the Quartus II software, you should use the `quartus_sh --flow` command beginning with the Quartus II software version 3.0.

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

```
quartus_sh --flow compile filtref ◀
```

Text-Based Report Files

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

Report file names contain the revision name and the short-form name of the executable that generated the report file: `<revision>.<executable>.rpt`. For example, using the `quartus_fit` executable to place and route a project with the revision name `design_top` generates a report file named `design_top.fit.rpt`. Similarly, using the `quartus_tan` 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 about this package, refer to [“Command-Line Scripting Help” on page 2–6](#).

You can use Quartus II command-line executables in scripts that control a design flow that uses other software in addition to the Quartus II software. For example, if your design flow uses other synthesis or simulation software, and you can run the other software at a system command prompt, you can include it in a single script. The Quartus II command-line executables include options for common global project settings and operations, but you must use a Tcl script or the Quartus II GUI to set up a new project and apply individual constraints, such as pin location assignments and timing requirements. Command-line executables are very useful for working with existing projects, for making common global settings, and for performing common operations. For more flexibility in a flow, use a Tcl script, which makes it easier to pass data between different stages of the design flow and have more control during the flow.



For more information about Tcl scripts, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*, or the *Quartus II Scripting Reference Manual*.

For example, your script could run other synthesis software, then place-and-route the design in the Quartus II software, then generate output netlists for other simulation software. [Example 2–5](#) shows how to do this with a UNIX shell script for a design that targets a Cyclone II device.

Example 2-5. Script for End-to-End Flow

```
#!/bin/sh
# Run synthesis first.
# This example assumes you use Synplify software
synplify -batch synthesise.tcl

# If your Quartus II project exists already, you can just
# recompile the design.
# You can also use the script described in a later example to
# create a new project from scratch
quartus_sh --flow compile myproject

# Use the quartus_tan executable to do best and worst case
# timing analysis
quartus_tan myproject --tao=worst_case
quartus_tan myproject --fast_model --tao=best_case

# Use the quartus_eda executable to write out a gate-level
# Verilog simulation netlist for ModelSim
quartus_eda my_project --simulation --tool=modelsim
--format=verilog

# Perform the simulation with the ModelSim software
vlib cycloneii_ver
vlog -work cycloneii_ver c:/quartusii/eda/sim_lib/cycloneii_atoms.v
vlib work
vlog -work work my_project.vo
vsim -L cycloneii_ver -t lps work.my_project
```

Makefile Implementation

You can also use the Quartus II command-line 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.

To facilitate easier development of efficient makefiles, the following “smart action” scripting command is provided with the Quartus II software:

```
quartus_sh --determine_smart_action ◀
```

Because assignments for a Quartus II project are stored in the Quartus II Settings File (.qsf), including it in every rule results in unnecessary processing steps. For example, updating a setting related to programming file generation (which requires re-running only **quartus_asm**) modifies the Quartus II Settings File, requiring a complete recompilation if the Quartus II Settings File is included in every rule.

The smart action command determines the earliest command-line executable in the compilation flow that must be run based on the current Quartus II Settings File, and generates a change file corresponding to that executable. For a given command-line executable named **quartus_<executable>**, the change file is named with the format **<executable>.chg**. For example, if **quartus_map** must be re-run, the smart action command creates or updates a file named **map.chg**. Thus, rather than including the Quartus II Settings File in each makefile rule, include only the appropriate change file.

Example 2-6 uses change files and the smart action command. You can copy and modify it for your own use. A copy of this example is included in the help for the makefile option, which is available by typing:

```
quartus_sh --help=makefiles ←
```

Example 2-6. Sample Makefile

```
#####
# Project Configuration:
#
# Specify the name of the design (project), the Quartus II Settings
# File (.qsf), 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
#####

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

clean:
    rm -rf *.rpt *.chg smart.log *.htm *.eqn *.pin *.sof *.pof db

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 that 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 following command:

```
quartus_sh --help=determine_smart_action ←
```

Command-Line Scripting Examples

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

Create a Project & Apply Constraints

The command-line executables include options for common global project settings and commands. To apply constraints such as pin locations and timing assignments, run a Tcl script with the constraints in it. You can write a Tcl constraint file from scratch, or generate one for an existing project. From the Project menu, click **Generate Tcl File for Project**.

Example 2-7 creates a project with a Tcl script and applies project constraints using the tutorial design files in the *<Quartus II installation directory>/qdesigns/tutorial/* directory:

Example 2-7. Tcl Script to Create Project & Apply Constraints

```
project_new filtref -overwrite
# Assign family, device, and top-level file
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C12Q240C6
set_global_assignment -name BDF_FILE filtref.bdf
# Assign pins
set_location_assignment -to clk Pin_28
set_location_assignment -to clkx2 Pin_29
set_location_assignment -to d[0] Pin_139
set_location_assignment -to d[1] Pin_140
# Other pin assignments could follow
# Create timing assignments
create_base_clock -fmax "100 MHz" -target clk clocka
create_relative_clock -base_clock clocka -divide 2 \
    -offset "500 ps" -target clkx2 clockb
set_multicycle_assignment -from clk -to clkx2 2
# Other timing assignments could follow
project_close
```

Save the script in a file called **setup_proj.tcl** and type the commands illustrated in **Example 2-8** at a command prompt to create the design, apply constraints, compile the design, and perform fast-corner and slow-corner timing analysis. Timing analysis results are saved in two files.

Example 2-8. Script to Create & Compile a Project

```
quartus_sh -t setup_proj.tcl ←
quartus_map filtref ←
quartus_fit filtref ←
quartus_asm filtref ←
quartus_tan filtref --fast_model --tao=min.tao
    --export_settings=off ←
quartus_tan filtref --tao=max.tao
    --export_settings=off ←
```

You can use the following two commands to create the design, apply constraints, and compile the design:

```
quartus_sh -t setup_proj.tcl ←
quartus_sh --flow compile filtref ←
```

The `quartus_sh --flow compile` command performs a full compilation, and is equivalent to clicking the **Start Compilation** button in the toolbar.

Check Design File Syntax

The UNIX shell script example shown in [Example 2–9](#) assumes that the Quartus II software `fir_filter` tutorial project exists in the current directory. (You can find the `fir_filter` project in the `<Quartus II directory>/qdesigns/fir_filter` directory unless the Quartus II software tutorial files are not installed.)

The `--analyze_file` option performs a syntax check on each file. The script checks the exit code of the `quartus_map` executable to determine whether there is an error during the syntax check. Files with syntax errors are added to the `FILES_WITH_ERRORS` variable, and when all files are checked, the script prints a message indicating syntax errors. When options are not specified, the executable uses the project database values. If not specified in the project database, 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.

Example 2–9. Shell Script to Check Design File Syntax

```
#!/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, WYSIWYG primitive resynthesis, and other netlist optimization options, refer to 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 command shown in [Example 2-10](#) at a command prompt.

Example 2-10. Creating a Project & Synthesizing a Netlist Using Netlist Optimizations

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

Archive & Restore Projects

You can archive or restore a Quartus II project with a single command. This makes it easy to take snapshots of projects when you use batch files or shell scripts for compilation and project management. Use the `--archive` or `--restore` options for `quartus_sh` as appropriate. Type the command shown in [Example 2-11](#) at a system command prompt to archive your project.

Example 2-11. Archiving a Project

```
quartus_sh --archive <project name> ↵
```

The archive file is automatically named `<project name>.qar`. If you want to use a different name, rename the archive after it has been created. This command overwrites any existing archive with the same name.

To restore a project archive, type the command shown in [Example 2-12](#) at a system command prompt:

Example 2-12. Restoring a Project Archive

```
quartus_sh --restore <archive name> ↵
```

The command restores the project archive to the current directory and overwrites existing files.

Perform I/O Assignment Analysis

You can perform I/O assignment analysis with a single command. I/O assignment analysis checks pin assignments to ensure they do not violate board layout guidelines. I/O assignment analysis does not require a complete place and route, so it is a quick way to ensure your pin assignments are correct. The command shown in [Example 2–13](#) performs I/O assignment analysis for the specified project and revision.

Example 2–13. Performing I/O Assignment Analysis

```
quartus_fit --check_ios <project name> --rev=<revision name> ←
```

Update Memory Contents without Recompiling

You can use two simple commands to update the contents of memory blocks in your design without recompiling. Use the **quartus_cdb** executable with the `--update_mif` option to update memory contents from Memory Initialization Files or Hexadecimal (Intel-Format) Files. Then re-run the assembler with the **quartus_asm** executable to regenerate the SOF, POE, and any other programming files.

[Example 2–14](#) shows these two commands:

Example 2–14. Commands to Update Memory Contents without Recompiling

```
quartus_cdb --update_mif <project name> [--rev=<revision name>] ←  
quartus_asm <project name> [--rev=<revision name>] ←
```

[Example 2–15](#) shows the commands for a DOS batch file for this example. You can paste the following lines into a DOS batch file called **update_memory.bat**.

Example 2–15. Batch file to Update Memory Contents without Recompiling

```
quartus_cdb --update_mif %1 --rev=%2  
quartus_asm %1 --rev=%2
```

Type the following command at a system command prompt:

```
update_memory.bat <project name> <revision name> ←
```

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 causes 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 command shown in [Example 2-16](#) at a command prompt.

Example 2-16. Fitting a Project Quickly

```
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 the file **fir_filter.qpf**). If the tutorial files are installed on your system, this project exists in the `<Quartus II directory>/qdesigns<quartus_version_number>/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.

Example 2-17 is designed for use on UNIX systems using **sh** (the shell).

Example 2-17. Shell Script to Fit a Design Using Multiple Seeds

```
#!/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 about the DSE, type `quartus_sh --help=dse` at the command prompt, or refer to the *Design Space Explorer* chapter in volume 2 of the *Quartus II Handbook*, or see the Quartus II Help.

The QFlow Script

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

Start QFlow by typing the following command at a command prompt: `quartus_sh -g` ←. Figure 2-3 shows the QFlow interface.

Figure 2-3. QFlow Interface



The QFlow script is located in the
 <Quartus II directory>/common/tcl/apps/qflow/ directory.

Document Revision History

Table 2–4 shows the revision history for this document.

Table 2–4. Document Revision History

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history for this document.	
May 2006 v6.0.0	Added the TimeQuest timing analyzer feature.	
October 2005 v5.1.0	Updated for the Quartus II software version 5.1.0.	
May 2005 v5.0.0	Updated for the Quartus II software version 5.0.0.	
Dec. 2004 v2.1	<ul style="list-style-type: none"> ● Updates to tables and figures. ● New functionality in the Quartus II software version 4.2. 	
June 2004 v2.0	<ul style="list-style-type: none"> ● Updates to tables and figures. ● New functionality in the Quartus II software version 4.1. 	
Feb. 2004 v1.0	Initial release.	

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 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. 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 switching assignments to another project.

The Quartus II software Tcl commands follow the 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 automating the Quartus II software. You can modify these example scripts for use with your own designs. You can find more Tcl scripts in the Design Examples section of the Support area of Altera's website.

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, refer to [“References” on page 3–50](#).

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.

If you're unfamiliar with Tcl scripting, or are a Tcl beginner, refer to the [“Tcl Scripting Basics” on page 3–42](#) for an introduction to Tcl scripting.

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

Quartus II Tcl Packages

The Quartus II Tcl commands are grouped in packages by function. [Table 3–1](#) describes each Tcl package.

Package Name	Package Description
advanced_timing	Traverse the timing netlist and get information about timing nodes
backannotate	Back annotate assignments
chip_editor	Identify and modify resource usage and routing with the Chip Editor
database_manager	Manage version-compatible database files
device	Get device and family information from the device database
flow	Compile a project, run command-line executables and other common flows
insystem_memory_edit	Read and edit memory contents in Altera devices
jtag	Control the jtag chain
logic_analyzer_interface	Query and modify the logic analyzer interface output pin state
logiclock	Create and manage LogicLock regions
misc	Perform miscellaneous tasks
project	Create and manage projects and revisions, make any project assignments including timing assignments
report	Get information from report tables, create custom reports
sdic	Specifies constraints and exceptions to the TimeQuest Analyzer
simulator	Configure and perform simulations
sta	Contains the set of Tcl functions for obtaining advanced information from the TimeQuest Timing Analyzer
stp	Run the SignalTap® II logic analyzer
timing	Annotate timing netlist with delay information, compute and report timing paths
timing_assignment	Contains the set of Tcl functions for making project-wide timing assignments, including clock assignments; all Tcl commands designed to process Classic Timing Analyzer assignments have been moved to this package
timing_report	List timing paths

By default, only the minimum number of packages is 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 is automatically loaded, you must load other packages before you can run commands in 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 clear circle (○) means that the package is not available in that executable.

Table 3–2. Tcl Package Availability by Quartus II Executable (Part 1 of 2)

Packages	Quartus II Executable						
	Quartus_sh	Quartus_tan	Quartus_cdb	Quartus_sim	Quartus_stp	Quartus_sta	Tcl Console
advanced_timing	○	◐	○	○	○	○	○
backannotate	○	○	◐	○	○	○	◐
chip_editor	○	○	◐	○	○	○	○
device	●	◐	●	●	○	●	◐
flow	◐	◐	◐	◐	○	◐	◐
insystem_memory_edit	○	○	○	○	●	○	○
jtag	○	○	○	○	●	○	○
logic_analyzer_interface	○	○	○	○	●	○	○
logiclock	○	◐	◐	○	○	○	◐
misc	●	●	●	●	●	●	●
old_api	○	○	○	○	○	○	●
project	●	●	●	●	●	●	●
report	◐	◐	◐	●	○	●	◐
sdc	○	○	○	○	○	●	○
simulator	○	○	○	●	○	○	○
sta	○	○	○	○	○	●	○
stp	○	○	○	○	●	○	○

Table 3–2. Tcl Package Availability by Quartus II Executable (Part 2 of 2)

Packages	Quartus II Executable						
	Quartus_sh	Quartus_tan	Quartus_cdb	Quartus_sim	Quartus_stp	Quartus_sta	Tcl Console
timing	○	●	○	○	○	○	○
timing_assignment	●	●	●	●	●	○	○
timing_report	○	◐	○	○	●	○	●

Notes to Table 3–2:

- (1) A dark circle (●) indicates that the package is loaded automatically.
- (2) A half-circle (◐) means that the package is available but not loaded automatically.
- (3) A white circle (○) means that the package is not available for that executable.

Because different packages are available in different executables, you must run your scripts with executables that include the packages you use in the scripts. For example, if you use commands in the **timing** package, you must use the **quartus_tan** executable to run the script because the **quartus_tan** executable is the only one with support for the **timing** package.

Loading Packages

To load a Quartus II Tcl package, use the **load_package** command as follows:

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

This command is similar to the **package require** Tcl command (described in [Table 3–3 on page 3–7](#)), but you can easily alternate between different versions of a Quartus II Tcl package with the **load_package** command.



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

Quartus II Tcl API Help

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

```
quartus_sh --qhelp ←
```

This command runs the Quartus II Command-Line and 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.

In addition, the information in the Tcl API help is available in the *Quartus II Scripting Reference Manual*, which is available in PDF on the Quartus II Literature page on the Altera web site.

Quartus II Tcl help allows easy access to information about the Quartus II Tcl commands. To access the help information, type `help` at a Tcl prompt, as shown in [Example 3-1](#).

Example 3-1. Help Output

```
tcl> help
```

```
-----
Available Quartus II Tcl Packages:
-----
```

Loaded	Not Loaded
-----	-----
::quartus::misc	::quartus::device
::quartus::old_api	::quartus::backannotate
::quartus::project	::quartus::flow
::quartus::timing_assignment	::quartus::logiclock
::quartus::timing_report	::quartus::report

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

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.



The Tcl API help is also available in Quartus II online help. Search for the command or package name to find details about that command or package.

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

Table 3–3. 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 that includes the list of available Tcl commands. For convenience, you can omit the ::quartus:: package prefix, and type help -pkg <package name> ↵. 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. Examples: help -pkg ::quartus::p ↵ help -pkg ::quartus::project ↵ help -pkg project ↵ help -pkg project -version 1.0 ↵
<command_name> -h or <command_name> -help	To view short help for a Quartus II Tcl command for which the package is loaded. Examples: project_open -h ↵ 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. Example: package require ::quartus::project 1.0 ↵ 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. 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. Example: load_package ::quartus::project -version 1.0 ↵

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

Help Command	Description
<pre>help -cmd <command name> [-version <version number>] or <command name> -long_help</pre>	<p>To view long help for a Quartus II Tcl command. Only <code><command name> -long_help</code> requires that the associated Tcl package is loaded.</p> <p>If you do not specify the <code>-version</code> option, help for the currently loaded package is displayed by default.</p> <p>If the package for which you want help is not loaded, help for the latest version of the package is displayed by default.</p> <p>Examples:</p> <pre>project_open -long_help ← help -cmd project_open ← help -cmd project_open -version 1.0 ←</pre>
<pre>help -examples</pre>	<p>To view examples of Quartus II Tcl usage.</p>
<pre>help -quartus</pre>	<p>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.</p>
<pre>quartus_sh --qhelp</pre>	<p>To launch the Tk viewer for Quartus II command-line help and display help for the command-line executables and Tcl API packages.</p> <p>For more information about this utility, refer to the <i>Command-Line Scripting</i> chapter in volume 2 of the <i>Quartus II Handbook</i>.</p>

Executables Supporting Tcl

Some of the Quartus II command-line executables support Tcl scripting (refer to [Table 3-4](#)). Each executable supports different sets of Tcl packages. Refer to [Table 3-4](#) to determine the appropriate executable to run your script.

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 Classic 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	The Quartus II Simulator supports the automation of design simulation.
quartus_sta	The TimeQuest Timing Analyzer supports SDC terminology for constraint entry and reporting.
quartus_stp	The Quartus II SignalTap II executable supports in-system debugging tools.

The **quartus_tan** and **quartus_cdb** executables support supersets of the packages supported by the **quartus_sh** executable. Use the **quartus_sh** executable if you run Tcl scripts with only project management and assignment commands, or if you 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: -t, -s & --tcl_eval

Table 3–5 lists three command-line options you can use with executables that support Tcl.

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>/common/tcl/packages directory.

For example, to run a script called **myscript.tcl** with one argument, Stratix, type the following command at a system command prompt:

```
quartus_sh -t myscript.tcl Stratix ↵
```



Beginning with version 4.1, the Quartus II software supports the argv variable. In previous software versions, script arguments are accessed in the quartus(args) global variable.

Refer to “[Accessing Command-Line Arguments](#)” on page 3–36 for more information.

Interactive Shell Mode

Running an executable with the -s option starts an interactive Tcl shell that displays a tcl> prompt. For example, type quartus_tan -s ↵ at a system command prompt to open the Classic timing analyzer

executable in interactive shell mode. Commands you type in the Tcl shell are interpreted when you press **Enter**. You can run a Tcl script in the interactive shell with the following command:

```
source <script name> ↵
```

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.

For example, the following command runs the Tcl command that prints out the commands available in the **project** package.

```
quartus_sh --tcl_eval help -pkg project ↵
```

Using the Quartus II Tcl Console Window

You can run Tcl commands directly in the Quartus II Tcl Console window. On the View menu, click **Utility Windows**. By default, the Tcl Console window is docked in the bottom-right corner of the Quartus II 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 are shown in the Quartus II Tcl Console window.

Note that you can do limited timing analysis in the Tcl console in the Quartus II GUI. With the **timing_report** package, you can use the **list_path** command to get details on paths listed in the timing report. However, if you want to get information about timing paths that are not listed in the timing report, you must use the **quartus_tan** executable in shell mode or run a script that reports on the paths in which you are interested.

If your design uses the TimeQuest timing Analyzer, you should perform scripted timing analysis in the TimeQuest Tcl console.

As [Table 3–2](#) shows, the Tcl console in the Quartus II GUI does not include support for every package, so you cannot run scripts that use commands in packages that are not supported.

End-to-End Design Flows

You can use Tcl scripts to control all aspects of the design flow, including controlling other software if it includes a scripting interface.

Typically, EDA tools include their own script interpreters that extend core language functionality with tool-specific commands. For example, the Quartus II Tcl interpreter supports all core Tcl commands, and adds numerous commands specific to the Quartus II software. You can include commands in one Tcl script to run another script, which allows you to combine or chain together scripts to control different tools. Because scripts for different tools must be executed with different Tcl interpreters, it is difficult to pass information between the scripts unless one script writes information into a file and another script reads it.

Within the Quartus II software, you can perform many different operations in a design flow (such as synthesis, fitting, and timing analysis) from a single script, making it easy to maintain global state information and pass data between the operations. However, there are some limitations on the operations you can perform in a single script due to the various packages supported by each executable. For example, you cannot write a single script that performs simulation with commands in the **simulator** package while using commands in the **advanced_timing** package; those two packages are not available in the same executable. In a case where you wanted to include Tcl simulation and advanced timing analysis commands, you must write two scripts.

There are no limitations on running flows from any executable. Flows include operations found in the Start section of the Processing menu in the Quartus II GUI, and are also documented with the **execute_flow** Tcl command. If you can make settings in the Quartus II software and run a flow to get your desired result, you can make the same settings and run the same flow in any command-line executable.

To revisit the example with simulation and timing analysis, you could write one script that includes settings that configure a simulation, with settings that configure timing analysis. Then, run the simulation and timing analysis flows with the **execute_flow** command.

Configuring a simulation includes specifying settings such as name and location of the stimulus file, the duration of the simulation, whether to perform glitch detection or not, and more. Configuring timing analysis includes specifying settings such as the required clock frequency, the number of paths to report, and which timing model to use. You can make the settings, then run the flows with the **execute_flow** command, in any Quartus II command-line executable.

Creating 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. From the Project menu, click **Generate Tcl File for Project** to automatically generate a Tcl file with all of your assignments. 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.



Refer to “[Interactive Shell Mode](#)” on page 3–10 for information about sourcing a script. Scripting information for all Quartus II project settings and assignments is located in the *QSF Reference Manual*.

[Example 3–2](#) shows how to create a project, make assignments, and compile the project. It uses the **fir_filter** tutorial design files.

Example 3–2. Create & Compile a Project

```
load_package flow

# Create the project and overwrite any settings
# files that exist
project_new fir_filter -revision filtref -overwrite
# Set the device, the name of the top-level BDF,
# and the name of the top level entity
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C6F256C6
set_global_assignment -name BDF_FILE filtref.bdf
set_global_assignment -name TOP_LEVEL_ENTITY filtref
# Add other pin assignments here
set_location_assignment -to clk Pin_G1
# Create a base clock and a derived clock
create_base_clock -fmax "100 MHz" -target clk clocka
create_relative_clock -base_clock clocka -divide 2 \
    -offset "500 ps" -target clkx2 clockb
# Create a multicycle assignment of 2 between
# the two clock domains in the design.
set_multicycle_assignment -from clk -to clkx2 2
execute_flow -compile
project_close
```



The assignments created or modified while a project is open are 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.

HardCopy Device Design



For information about using a scripted design flow for HardCopy II designs, refer to the *Script-Based Design Flow for HardCopy Devices* chapter of the *HardCopy Handbook*. It contains sample scripts and recommendations to make your HardCopy II design flow easy.

A separate chapter in the *HardCopy Handbook* called *Timing Constraints for HardCopy II* also contains information about script-based design for HardCopy II devices, with an emphasis on timing constraints.

EDA Tool Assignments

You can target EDA tools for a project in the Quartus II software in Tcl with 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.

By default, the EDA interface options are set to <none>. Table 3–6 lists the EDA interface options available in the Quartus II software. Enclose interface assignment options that contain spaces in quotation marks.

Option	Acceptable Values
Design Entry (EDA_DESIGN_ENTRY_SYNTHESIS_TOOL)	<ul style="list-style-type: none"> ● Design Architect ● Design Compiler ● FPGA Compiler ● FPGA Compiler II ● FPGA Compiler II Altera Edition ● FPGA Express ● LeonardoSpectrum™ ● LeonardoSpectrum-Altera (Level 1) ● Synplify ● Synplify Pro ● ViewDraw ● Precision Synthesis ● Custom
Simulation (EDA_SIMULATION_TOOL)	<ul style="list-style-type: none"> ● ModelSim (VHDL output from the Quartus II software) ● ModelSim (Verilog HDL output from the Quartus II software) ● ModelSim-Altera (VHDL output from the Quartus II software) ● ModelSim-Altera (Verilog HDL output from the Quartus II software) ● SpeedWave ● VCS ● Verilog-XL ● VSS ● NC-Verilog (Verilog HDL output from the Quartus II software) ● NC-VHDL (VHDL output from the Quartus II software) ● Scirocco (VHDL output from the Quartus II software) ● Custom Verilog HDL ● Custom VHDL
Timing Analysis (EDA_TIMING_ANALYSIS_TOOL)	<ul style="list-style-type: none"> ● PrimeTime (VHDL output from the Quartus II software) ● PrimeTime (Verilog HDL output from the Quartus II software) ● Stamp (board model) ● Custom Verilog HDL ● Custom VHDL

Table 3–6. EDA Interface Options in the Quartus II Software (Part 2 of 2)

Option	Acceptable Values
Board level tools (EDA_BOARD_DESIGN_TOOL)	<ul style="list-style-type: none"> ● Signal Integrity (IBIS) ● Symbol Generation (ViewDraw)
Formal Verification (EDA_FORMAL_VERIFICATION_TOOL)	<ul style="list-style-type: none"> ● Conformal LEC
Resynthesis (EDA_RESYNTHESIS_TOOL)	<ul style="list-style-type: none"> ● PALACE ● Amplify

For example, to generate an NC-Sim Verilog simulation output file, EDA_SIMULATION_TOOL should be set to target NC-Sim Verilog as the desired output, as shown in [Example 3–3](#).

Example 3–3.

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



For information about using third-party simulation tools, refer to volume 3 of the *Quartus II Handbook*.

Example 3-4 shows compilation of the `fir_filter` design files, generating a VHDL Output (`.vho`) file output for NC-Sim Verilog simulation.

Example 3-4. Simple Design with .vho Output

```
# 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
```

Custom options are 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, refer to the Quartus II Help.

Using LogicLock Regions

You can use Tcl commands to work with LogicLock™ regions. The following examples show how to export and import LogicLock regions for use in other designs. The examples use the files in the LogicLock tutorial design.



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

To compile a design and export LogicLock regions, follow these steps:

1. Create a project and add assignments.
2. Assign virtual pins.
3. Create a LogicLock region.
4. Assign a design entity to the region.
5. Compile the project.
6. Back-annotate the region.
7. Export the region.

Example 3-5 shows a script that creates a project called **lockmult**, and makes all the required assignments to compile the project. Next, the script compiles the project, back-annotates the design, and exports the LogicLock region. The script uses a procedure called **assign_virtual_pins**, which is described after the example. Use the **quartus_cdb** executable to run this script.

Example 3-5. LogicLock Export Script

```
load_package flow
load_package logiclock
load_package backannotate

project_new lockmult -overwrite
set_global_assignment -name BDF_FILE pipemult.bdf
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C6F256C6
set_global_assignment -name TOP_LEVEL_ENTITY pipemult

# These two assignments cause the Quartus II software
# to generate a VQM file for the logic in the LogicLock
# region. The VQM file is imported into the top-level
# design.
set_global_assignment -name \
    LOGICLOCK_INCREMENTAL_COMPILE_FILE pipemult.vqm
set_global_assignment -name \
    LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT ON

create_base_clock -fmax 200MHz -target clk clk_200
assign_virtual_pins { clk }
#Prepare LogicLock data structures before
#LogicLock-related commands.
initialize_logiclock

# Create a region named lockmult and assign pipemult
# to it.
# The region is auto-sized and floating.
set_logiclock -region lockmult -auto_size true \
-floating true
set_logiclock_contents -region lockmult -to pipemult
execute_flow -compile

# Back annotate the LogicLock Region and export a QSF
logiclock_back_annotate -region lockmult -lock
logiclock_export -file_name pipemult.qsf

uninitialize_logiclock
project_close
```

The `assign_virtual_pins` command is a procedure that makes virtual pin assignments to all bottom-level design pins, except for signals specified as arguments to the procedure. The procedure is defined in [Example 3-6](#).

Example 3-6. assign_virtual_pins Procedure

```
proc assign_virtual_pins { skips } {  
  
    # Analysis and elaboration must be run first to get the pin names  
    execute_flow -analysis_and_elaboration  
  
    # Get all pin names as a collection.  
  
    set name_ids [get_names -filter * -node_type pin]  
    foreach_in_collection name_id $name_ids {  
        # Get the hierarchical path name of the pin.  
        set hname [get_name_info -info full_path $name_id]  
        #Skip the virtual pin assignment if the  
        #pin is in the list of signals to be skipped.  
        if {[lsearch -exact $skips $hname] == -1} {  
            post_message "Setting VIRTUAL_PIN on $hname"  
            set_instance_assignment -to $hname -name VIRTUAL_PIN ON  
        } else {  
            post_message "Skipping VIRTUAL_PIN for $hname"  
        }  
    }  
}
```

When the script runs, it generates a netlist file named `pipemult.vqm`, and a Quartus II Settings File named `pipemult.qsf`, which contains the back-annotated assignments. You can then import the LogicLock region in another design. This example uses the top-level design in the `topmult` directory.

To import it four times in the top-level LogicLock tutorial design, follow these steps:

1. Create the top-level project.
2. Add assignments.
3. Elaborate the design.
4. Import the LogicLock constraints.
5. Compile the project.

Example 3-7 shows a script that demonstrates the previous steps.

Example 3-7. LogicLock Import Script

```
load_package flow
load_package logiclock

project_new topmult -overwrite
set_global_assignment -name BDF_FILE topmult.bdf
set_global_assignment -name VQM_FILE pipemult.vqm
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C6F256C6
create_base_clock -fmax 200MHz -target clk clk_200

# The LogicLock region will be used four times
# in the top-level design. These assignments
# specify that the back-annotated assignments in
# the QSF will be applied to the four entities
# in the top-level design.
set_instance_assignment -name LL_IMPORT_FILE pipemult.qsf \
    -to pipemult:inst
set_instance_assignment -name LL_IMPORT_FILE pipemult.qsf \
    -to pipemult:inst1
set_instance_assignment -name LL_IMPORT_FILE pipemult.qsf \
    -to pipemult:inst2
set_instance_assignment -name LL_IMPORT_FILE pipemult.qsf \
    -to pipemult:inst3

execute_flow -analysis_and_elaboration
initialize_logiclock
logiclock_import
uninitialize_logiclock
execute_flow -compile
project_close
```



For additional information about the LogicLock design methodology, refer to the *LogicLock Design Methodology* chapter in volume 2 of the *Quartus II Handbook*.

Compiling Designs

You can run the Quartus II command-line executables from Tcl scripts. Use the included **flow** package to run various Quartus II compilation flows, or run each executable directly.

The flow Package

The **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 **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 the Tcl **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 **flow** package to ensure assignments are exported to the project database and settings file.



You should use the **qexec** command to make 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> ↵
```

Reporting

Once compilation finishes, it is sometimes necessary 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.

Use the commands that access report data one row at a time, or one 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. You may often 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.

Column headings in report panels are in row 0. If you use a loop that reads the report one row at a time, you can start with row 1 to skip the row with column headings. The `get_number_of_rows` command returns the number of rows in the report panel, including the column heading row. Because the number of rows includes the column heading row, your loop should continue as long as the loop index is less than the number of rows, as illustrated in the following example.

Report panels are hierarchically arranged, 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**.

[Example 3–8](#) shows code that prints a list of all report panel names in your project.

Example 3–8. Print All Report Panel Names

```
set panel_names [get_report_panel_names]
foreach panel_name $panel_names {
    post_message "$panel_name"
}
```

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**.

Example 3–9. Print Number of Failing Paths per Clock

```
load_package report
project_open my-project
load_report
set report_panel_name "Timing Analyzer||Timing Analyzer Summary"
set num_rows [get_number_of_rows -name $report_panel_name]

# Get the column indices for the Type and Failed Paths columns
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"]

# Go through each line in the report panel
for {set i 1} {$i < $num_rows} {incr i} {

    # Get the row of data, then the type of summary
    # information in the row, and the number of failed paths
    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
```

Creating CSV Files for Excel

The Microsoft Excel software is sometimes used to view or manipulate timing analysis results. You can create a CSV file to import into Excel with data from any Quartus II report. This example shows a simple way to create a CSV file with data from a timing analysis panel in the report. You could modify the script to use command-line arguments to pass in the name of the project, report panel, and output file to use.

Example 3–10. Create CSV Files from Reports

```

load_package report
project_open my-project

load_report

# This is the name of the report panel to save as a CSV file
set panel_name "Timing Analyzer|Clock Setup: 'clk'"
set csv_file "output.csv"

set fh [open $csv_file w]
set num_rows [get_number_of_rows -name $panel_name]

# Go through all the rows in the report file, including the
# row with headings, and write out the comma-separated data
for { set i 0 } { $i < $num_rows } { incr i } {
    set row_data [get_report_panel_row -name $panel_name \
        -row $i]
    puts $fh [join $row_data ","]
}

close $fh
unload_report

```

Short Option Names

Beginning with version 6.0 of the Quartus II software, you can use short versions of command options, as long as they distinguish between the command's options. For example, the **project_open** command supports two options: `-current_revision` and `-revision`. You can use any of the following shortened versions of the `-revision` option: `-r`, `-re`, `-rev`, `-revi`, `-revis`, and `-revisio`. You can use an option as short as `-r` because no other option starts with the same letters as `revision`. However, the **report_timing** command includes the options `-recovery` and `-removal`. You cannot use `-r` or `-re` to shorten either of those options, because they do not uniquely distinguish between either option. You could use `-rec` or `-rem`.

Timing Analysis

The Quartus II software includes comprehensive Tcl APIs for both the Classic and TimeQuest analyzers. This section includes simple and advanced script examples for the Classic analyzer and introductory scripting information about the TimeQuest Tcl API.

Classic 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 timing analysis discovers a negative slack for one of the clocks. The second part of the script adds a multiclock 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 with the `create_timing_netlist` and `report_timing` Tcl commands.



You must compile the project before running the script example shown in [Example 3-11](#).

Example 3-11. Classic Timing Analysis

```
# 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

```

Advanced Classic Timing Analysis

There may be times when the commands available for timing analysis reporting do not provide access to specific data you need. The **advanced_timing** package provides commands to access the data structures representing the timing netlist for your design. These commands provide low-level details about timing delays, node fan-in and fan-out, and timing data. Writing procedures to traverse the timing netlist and extract information gives you the most control to get exactly the data you need.

The timing netlist is represented with a graph, which is a collection of nodes and edges. Nodes represent elements in your design such as registers, combinational nodes, pins, and clocks. Edges connect the nodes and represent the connections between the logic in your design. Edges and nodes have unique positive integer IDs that identify them in the timing netlist. All the commands for getting information about the timing netlist use node and edge IDs instead of text-based names.

As an example of how to use the **advanced_timing** package, [Example 3-12](#) shows one way to show the register-to-pin delays from all registers that drive the pins of an output bus. Specify the name of an output bus (for example, `address`), and the script prints out the names of all registers driving the pins of the bus and the delays from registers to pins.

Example 3-12. Report Register to Pin Delays

```
load_package advanced_timing
package require cmdline

# This procedure returns a list of IDs for pins with names
# that match the bus name passed in
proc find { bus_name } {

    set to_return [list]

    foreach_in_collection node_id [get_timing_nodes -type pin] {
        set node_name [get_timing_node_info -info name $node_id]
        if { [string match $bus_name* $node_name] } {
            lappend to_return $node_id
        }
    }
    return $to_return
}

# Required arguments for the script are the name of the project and
# revision, as well as the name of the bus to analyze
set options { \
    { "project.arg" "" "Project name" } \
    { "revision.arg" "" "Revision name" } \
    { "bus_name.arg" "" "Name of the bus to get timing data for" } \
}
array set opts [::cmdline::getoptions quartus(args) $options]

project_open $opts(project) -revision $opts(revision)

# The timing netlist must be created before accessing it.
create_timing_netlist

# This creates a data structure with additional timing data
create_p2p_delays

# Walk through each pin in the specified bus
foreach pin_id [find $opts(bus_name)] {
    set pin_name [get_timing_node_info -info name $pin_id]
```

```

puts "$pin_name source registers and delays"
# The get_delays_from_keepers command returns a list of all the
# non-combinational nodes in the design that fan in to the
# specified timing node, with the associated delays.
foreach data [get_delays_from_keepers $pin_id] {
    set source_node [lindex $data 0]
    set max_delay [lindex $data 1]
    set source_node_name \
[get_timing_node_info -info name $source_node]
    puts " $source_node_name $max_delay"
}
}
project_close

```

Type the command shown in [Example 3-13](#) at a system command prompt to run this script.

Example 3-13.

```

quartus_tan -t script.tcl -project fir_filter
-revision filtref -bus_name yn_out ↵

```

TimeQuest Timing Analysis

The TimeQuest timing analyzer includes support for SDC commands in the `::quartus::sdc` package.



Refer to the *TimeQuest Timing Analysis* chapter of the *Quartus II Handbook* for detailed information about how to perform timing analysis with the TimeQuest timing analyzer.

TimeQuest Scripting

In versions of the Quartus II software before 6.0, the `::quartus::project` Tcl package contained the following SDC-like commands for making timing assignments:

- `create_base_clock`
- `create_relative_clock`
- `get_clocks`
- `set_clock_latency`
- `set_clock_uncertainty`
- `set_input_delay`
- `set_multicycle_assignment`
- `set_output_delay`
- `set_timing_cut_assignment`

These commands are not SDC-compliant. Beginning with version 6.0, these commands are in a new package named **::quartus::timing_assignment**. To ensure backwards compatibility with existing Tcl scripts, the **::quartus::timing_assignment** package is loaded by default in the following executables:

- **quartus**
- **quartus_sh**
- **quartus_cdb**
- **quartus_sim**
- **quartus_stp**
- **quartus_tan**

The **::quartus::timing_assignment** package is not loaded by default in the **quartus_sta** executable. The **::quartus::sdc** Tcl package includes SDC-compliant versions of the commands listed above. The package is available only in the **quartus_sta** executable, and it is loaded by default.

Automating Script Execution

Beginning with version 4.0 of the Quartus II software, you can configure scripts to run automatically at various points during compilation. Use this capability to automatically run scripts that perform custom reporting, make specific assignments, and perform many other tasks.

The following three global assignments control when a script is run automatically:

- **PRE_FLOW_SCRIPT_FILE** —before a flow starts
- **POST_MODULE_SCRIPT_FILE** —after a module finishes
- **POST_FLOW_SCRIPT_FILE** —after a flow finishes

The **POST_FLOW_SCRIPT_FILE** and **POST_MODULE_SCRIPT_FILE** assignments are supported beginning in version 4.0, and the **PRE_FLOW_SCRIPT_FILE** assignment is supported beginning in version 4.1.

A module is a Quartus II executable that performs one step in a flow. For example, two modules are Analysis & Synthesis (**quartus_map**) and timing analysis (**quartus_tan**).

A flow is a series of modules that the Quartus II software runs with predefined options. For example, compiling a design is a flow that typically consists of the following steps (performed by the indicated module):

1. Analysis and synthesis (**quartus_map**)
2. Fitter (**quartus_fit**)

3. Assembler (`quartus_asm`)
4. Timing Analyzer (`quartus_tan`)

Other flows are described in the help for the `execute_flow` Tcl command. In addition, many commands in the Processing menu of the Quartus II GUI correspond to this design flow.

Making the Assignment

To make an assignment to automatically run a script, add an assignment with the following form to your project's Quartus II Settings File:

```
set_global_assignment -name <assignment name> \
    <executable>:<script name>
```

The assignment name is one of the following:

- `PRE_FLOW_SCRIPT_FILE`
- `POST_MODULE_SCRIPT_FILE`
- `POST_FLOW_SCRIPT_FILE`

The executable is the name of a Quartus II command-line executable that includes a Tcl interpreter.

- `quartus_cdb`
- `quartus_sh`
- `quartus_sim`
- `quartus_sta`
- `quartus_stp`
- `quartus_tan`

The script name is the name of your Tcl script.

Script Execution

The Quartus II software runs the scripts as shown [Example 3–14](#).

Example 3–14.

```
<executable> -t <script name> <flow or module name> <project name> <revision name>
```

The first argument passed in the `argv` variable (or `quartus(args)` variable) is the name of the flow or module being executed, depending on the assignment you use. The second argument is the name of the project, and the third argument is the name of the revision.

When you use the `POST_MODULE_SCRIPT_FILE` assignment, the specified script is automatically run after every executable in a flow. You can use a string comparison with the module name (the first argument passed in to the script) to isolate script processing to certain modules.

Execution Example

Example 3–15 illustrates how automatic script execution works in a complete flow, assuming you have a project called **top** with a current revision called **rev_1**, and you have the following assignments in the Quartus II Settings File for your project.

Example 3–15.

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE quartus_sh:first.tcl
set_global_assignment -name POST_MODULE_SCRIPT_FILE quartus_sh:next.tcl
set_global_assignment -name POST_FLOW_SCRIPT_FILE quartus_sh:last.tcl
```

When you compile your project, the `PRE_FLOW_SCRIPT_FILE` assignment causes the following command to be run before compilation begins:

```
quartus_sh -t first.tcl compile top rev_1
```

Next, the Quartus II software starts compilation with analysis and synthesis, performed by the **quartus_map** executable. After the analysis and synthesis finishes, the `POST_MODULE_SCRIPT_FILE` assignment causes the following command to be run:

```
quartus_sh -t next.tcl quartus_map top rev_1
```

Then, the Quartus II software continues compilation with the Fitter, performed by the **quartus_fit** executable. After the Fitter finishes, the `POST_MODULE_SCRIPT_FILE` assignment runs the following command:

```
quartus_sh -t next.tcl quartus_fit top rev_1
```

Corresponding commands are run after the other stages of the compilation. Finally, after the compilation is over, the `POST_FLOW_SCRIPT_FILE` assignment runs the following command:

```
quartus_sh -t last.tcl compile top rev_1
```


Controlling Processing

The `POST_MODULE_SCRIPT_FILE` assignment causes a script to run after every module. Because the same script is run after every module, you may need to include some conditional statements that restrict processing in your script to certain modules.

For example, if you want a script to run only after timing analysis, you should include a conditional test like the one shown in [Example 3–16](#). It checks the flow or module name passed as the first argument to the script and executes code when the module is `quartus_tan`.

Example 3–16. Restrict Processing to a Single Module

```
set module [lindex $quartus(args) 0]

if [string match "quartus_tan" $module] {

    # Include commands here that are run
    # after timing analysis
    # Use the post-message command to display
    # messages
    post_message "Running after timing analysis"
}
```

Displaying Messages

Because of the way the Quartus II software runs the scripts automatically, you must use the `post_message` command to display messages, instead of the `puts` command. This requirement applies only to scripts that are run by the three assignments listed in [“Automating Script Execution”](#) on [page 3–30](#).



Refer to [“Using the post_message Command”](#) on [page 3–35](#) for more information about this command.

Other Scripting Features

The Quartus II Tcl API includes other general-purpose commands and features described in this section.

Natural Bus Naming

Beginning with version 4.2, the Quartus II software supports natural bus naming. Natural bus naming means that square brackets used to specify bus indexes in hardware description languages do not have to be escaped to prevent Tcl from interpreting them as commands. For example, one

signal in a bus named address can be identified as `address[0]` instead of `address\[0\]`. You can take advantage of natural bus naming when making assignments, as in [Example 3–17](#).

Example 3–17. Natural Bus Naming

```
set_location_assignment -to address[10] Pin_M20
```

The Quartus II software defaults to natural bus naming. You can turn off natural bus naming with the **disable_natural_bus_naming** command.

For more information about natural bus naming, type

```
enable_natural_bus_naming -h
```

 at a Quartus II Tcl prompt.

Using Collection Commands

Some Quartus II Tcl functions return very large sets of data that 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 collections, **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, see the Quartus II Help and search 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.

[Example 3–18](#) prints all instance assignments in an open project.

Example 3–18. Using Collection Commands

```
set all_instance_assignments [get_all_instance_assignments -name *]
foreach_in_collection asgn $all_instance_assignments {
    # Information about each assignment is
    # returned in a list. For information
    # about the list elements, refer to Help
    # for the get-all-instance-assignments command.
    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. [Example 3–19](#) prints the number of global assignments in an open project:

Example 3–19. get_collection_size Command

```
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"
```

Using the post_message Command

To print messages that are formatted like Quartus II software messages, use the **post_message** command. Messages printed by the **post_message** command appear in the **System** tab of the Messages window in the Quartus II GUI, and are written to standard at when scripts are run. Arguments for the **post_message** command include an optional message type and a required message string.

The message type can be one of the following:

- info (default)
- extra_info
- warning
- critical_warning
- error

If you do not specify a type, Quartus II software defaults to `info`.

When you are using the Quartus II software in Windows, you can color code messages displayed at the system command prompt with the **post_message** command. Add the following line to your **quartus2.ini** file:

```
DISPLAY_COMMAND_LINE_MESSAGES_IN_COLOR = on
```

[Example 3–20](#) shows how to use the **post_message** command.

Example 3–20. post_message command

```
post_message -type warning "Design has gated clocks"
```

Accessing Command-Line Arguments

Many Tcl scripts are designed to 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. [Example 3–21](#) shows code that prints all of the arguments in the `quartus (args)` variable.

Example 3–21. Simple Command-Line Argument Access

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

If you copy the script in the previous example to a file named `print_args.tcl`, it displays the following output when you type the command shown in [Example 3–22](#) at a command prompt.

Example 3–22. Passing Command-Line Arguments to Scripts

```
quartus_sh -t print_args.tcl my_project 100MHz ←
The value at index 0 is <my_project>
The value at index 1 is 100MHz
```

Beginning with version 4.1, the Quartus II software supports the Tcl variables `argv`, `argc`, and `argv0` for command-line argument access. [Table 3–7](#) shows equivalent information for earlier versions of the software.

<i>Table 3–7. Quartus II Support for Tcl Variables</i>	
Beginning with Version 4.1	Equivalent Support in Previous Software Versions
argc	<code>llength \$quartus(args)</code>
argv	<code>quartus(args)</code>
argv0	<code>info nameofexecutable</code>

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>`.

Example 3–23 uses the `cmdline` package:

Example 3–23. `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 see the following output when you type the command shown in Example 3–24 at a command prompt:

Example 3–24. Passing Command-Line Arguments for Scripts

```
quartus_sh -t print_cmd_args.tcl -project my_project -frequency 100MHz ↵
The project name is <my_project>
The frequency is 100MHz
```

Virtually all Quartus II Tcl scripts must open a project. [Example 3–25](#) opens a project, and you can optionally specify a revision name. The example checks whether the specified project exists. If it does, the example opens the current revision, or the revision you specify.

Example 3–25. Full-Featured Method to Open Projects

```
package require cmdline
variable ::argv0 $::quartus(args)
set options { \
{ "project.arg" "" "Project Name" } \
{ "revision.arg" "" "Revision Name" } \
}
array set optshash [::cmdline::getoptions ::argv0 $options]

# Ensure the project exists before trying to open it
if {[project_exists $optshash(project)]} {

    if {[string equal "" $optshash(revision)]} {

        # There is no revision name specified, so default
        # to the current revision
        project_open $optshash(project) -current_revision
    } else {

        # There is a revision name specified, so open the
        # project with that revision
        project_open $optshash(project) -revision \
            $optshash(revision)
    }
} else {
    puts "Project $optshash(project) does not exist"
    exit 1
}
# The rest of your script goes here
```

If you do not require this flexibility or error checking, you can use just the **project_open** command, as shown in [Example 3–26](#).

Example 3–26. Simple Method to Open Projects

```
set proj_name [lindex $argv 0]
project_open $proj_name
```



For more information about the **cmdline** package, refer to the documentation for the package at [<Quartus II installation directory>/common/tcl/packages](#).

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 finite impulse response (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 in [Example 3-27](#).

Example 3-27. Interactive Tcl Shell

```
tcl> quartus_sh -s
tcl> Info: *****
Info: Running Quartus II Shell
Info: Version 6.0 Internal Build 170 10/29/2006 SJ Full Version
Info: Copyright (C) 1991-2006 Altera Corporation. All rights reserved.
Info: Your use of Altera Corporation's design tools, logic functions
Info: and other software and tools, and its AMPP partner logic
Info: functions, and any output files any of the foregoing
Info: (including device programming or simulation files), and any
Info: associated documentation or information are expressly subject
Info: to the terms and conditions of the Altera Program License
Info: Subscription Agreement, Altera MegaCore Function License
Info: Agreement, or other applicable license agreement, including,
Info: without limitation, that your use is for the sole purpose of
Info: programming logic devices manufactured by Altera and sold by
Info: Altera or its authorized distributors. Please refer to the
Info: applicable agreement for further details.
Info: Processing started: Tue Apr 04 12:24:13 2006
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 <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>
```

Create a new project called **fir_filter**, with a revision called **filtref** by typing the following command at a Tcl prompt:

```
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.

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

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

```
set_global_assignment -name family Cyclone ←
```



To learn more about assignment names that you can use with the `-name` option, refer to the 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:

```
load_package flow ←
```

It returns the following:

```
1.0
```

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

```
help -pkg ::quartus::flow ←
```

Example 3-28 shows an alternative command and the resulting output:

Example 3-28. Help 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
-----
```

This help display gives information about the flow package and the commands that are available with the package. To learn about the options available for the **execute_flow** Tcl command, type the following command at a Tcl prompt:

```
execute_flow -h ↵
```

To view additional information and example usage type the following command at a Tcl prompt:

```
execute_flow -long_help ↵
```

or

```
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 [Example 3-29](#):

Example 3-29.

```
tcl> execute_flow -compile ↵
Info:*****
Info: Running Quartus II Analysis & Synthesis
Info: Version 6.0 SJ Full Version
Info: Processing started: Tues Apr 04 09:30:47 2006
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 selecting **Start Compilation** from the Processing menu in the Quartus II GUI.

When you are finished with a project, close it using the **project_close** command as shown in [Example 3-30](#).

Example 3-30.

```
project_close ↵
```

Then, to exit the interactive Tcl shell, type `exit` ↵ at a Tcl prompt.

Quartus II Legacy Tcl Support

Beginning with the Quartus II software version 3.0, 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 with the Quartus II Tcl console or by using the `quartus_cmd` executable at the system 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 EDA tools that do not generate Tcl scripts for the most recent version of the Quartus II software.



You should create all new projects and Tcl scripts with the latest version of the Quartus II Tcl API.

Tcl Scripting Basics

The core Tcl 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.

Tcl commands are executed immediately as they are typed in an interactive Tcl shell. You can also create scripts (including this chapter's examples) as files and run them with a Tcl interpreter. A Tcl script does not need any special headers.

To start an interactive Tcl interpreter, type `quartus_sh -s` at a command prompt. The commands you type are executed immediately at the interpreter prompt. If you save a series of Tcl commands in a file, you can run it with a Tcl interpreter. To run a script named `myscript.tcl`, type `quartus_sh -t myscript.tcl` at a command prompt.

Hello World Example

The following shows the basic "Hello world" example 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 in ["Substitutions" on page 3-43](#). 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.

[Example 3–31](#) assigns the value 1 to the variable named `a`.

Example 3–31. Assigning Variables

```
set a 1
```

To access the contents of a variable, use a dollar sign before the variable name. [Example 3–32](#) prints "Hello world" in a different way.

Example 3–32. Accessing Variables

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

Substitutions

Tcl performs three types of substitution:

- Variable value substitution
- Nested command substitution
- Backslash substitution

Variable Value Substitution

Variable value substitution, as shown in [Example 3–32](#), refers to accessing the value stored in a variable by using a dollar sign (“\$”) before the variable name.

Nested Command Substitution

Nested command substitution refers to how the Tcl interpreter evaluates Tcl code in square brackets. 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. [Example 3–33](#) sets `a` to the length of the string `foo`:

Example 3–33. Command Substitution

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

Backslash Substitution

Backslash substitution allows you to quote reserved characters in Tcl, such as dollar signs (“\$”) and braces (“[]”). You can also specify other special ASCII characters like tabs and new lines with backslash substitutions. The backslash character is the Tcl line continuation character, used when a Tcl command wraps to more than one line.

[Example 3–34](#) shows how to use the backslash character for line continuation:

Example 3–34. Backslash Substitution

```
set this_is_a_long_variable_name [string length "Hello \  
world."]
```

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.

[Example 3–35](#) sets **b** to the sum of the value in the variable **a** and the square root of 2:

Example 3–35. Arithmetic with the expr Command

```
set a 5  
set b [expr { $a + sqrt(2) }]
```

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

Lists

A Tcl list is a series of values. Supported list operations include creating lists, appending lists, extracting list elements, computing the length of a list, sorting a list, and more. [Example 3–36](#) sets **a** to a list with three numbers in it:

Example 3–36. Creating Simple Lists

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

You can use the **lindex** command to extract information at a specific index in a list. Indexes are zero-based. You can use the index `end` to specify the last element in the list, or the index `end-<n>` to count from the end of the list. [Example 3–37](#) prints the second element (at index 1) in the list stored in `a`.

Example 3–37. Accessing List Elements

```
puts [lindex $a 1]
```

The **llength** command returns the length of a list. [Example 3–38](#) prints the length of the list stored in `a`.

Example 3–38. List Length

```
puts [llength $a]
```

The **lappend** command appends elements to a list. If a list does not already exist, the list you specify is created. The list variable name is not specified with a dollar sign. [Example 3–39](#) appends some elements to the list stored in `a`.

Example 3–39. Appending to a List

```
lappend a 4 5 6
```

Arrays

Arrays are similar to lists except that they use a string-based index. Tcl arrays are implemented as hash tables. You can create arrays by setting each element individually or by using the **array set** command. To set an element with an index of `Mon` to a value of `Monday` in an array called `days`, use the following command:

```
set days(Mon) Monday
```

The `array set` command requires a list of index/value pairs. This example sets the array called `days`:

```
array set days { Sun Sunday Mon Monday Tue Tuesday \
                Wed Wednesday Thu Thursday Fri Friday Sat Saturday }
```

[Example 3–40](#) shows how to access the value for a particular index.

Example 3–40. Accessing Array Elements

```
set day_abbreviation Mon
puts $days($day_abbreviation)
```

Use the **array names** command to get a list of all the indexes in a particular array. The index values are not returned in any specified order. [Example 3–41](#) shows one way to iterate over all the values in an array.

Example 3–41. Iterating Over Arrays

```
foreach day [array names days] {
    puts "The abbreviation $day corresponds to the day \
name $days($day) "
}
```

Arrays are a very flexible way of storing information in a Tcl script and are a good way to build complex data structures.

Control Structures

Tcl supports common control structures, including **if-then-else** conditions and **for**, **foreach**, and **while** loops. The position of the curly braces as shown in the following examples ensures the control structure commands are executed efficiently and correctly. [Example 3–42](#) prints whether the value of variable `a` is positive, negative, or zero.

Example 3–42. If-Then_Else Structure

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

[Example 3–43](#) uses a **for** loop to print each element in a list.

Example 3–43. For Loop

```
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]"
}
```

Example 3-44 uses a **foreach** loop to print each element in a list.

Example 3-44. foreach Loop

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

Example 3-45 uses a **while** loop to print each element in a list.

Example 3-45. while Loop

```
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. Example 3-46 defines a procedure that multiplies two numbers and returns the result.

Example 3-46. Simple Procedure

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

[Example 3-47](#) shows how to use the **multiply** procedure in your code. You must define a procedure before your script calls it, as shown below.

Example 3-47. Using a Procedure

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

You should define 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. [Example 3-48](#) defines a procedure that prints an element in a global list of numbers, then calls the procedure.

Example 3-48. Accessing Global Variables

```
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
```

File I/O

Tcl includes commands to read from and write to files. You must open a file before you can read from or write to it, and close it when the read and write operations are done. To open a file, use the **open** command; to close a file, use the **close** command. When you open a file, specify its name and the mode in which to open it. If you do not specify a mode, Tcl defaults to read mode. To write to a file, specify **w** for write mode as shown in [Example 3-49](#).

Example 3-49. Open a File for Writing

```
set output [open myfile.txt w]
```

Tcl supports other modes, including appending to existing files and reading from and writing to the same file.

The **open** command returns a file handle to use for read or write access. You can use the **puts** command to write to a file by specifying a filehandle, as shown in [Example 3-50](#).

Example 3–50. Write to a File

```
set output [open myfile.txt w]
puts $output "This text is written to the file."
close $output
```

You can read a file one line at a time with the **gets** command.

Example 3–51 uses the **gets** command that prints out each line of the file, with its line number.

Example 3–51. Read from a File

```
set input [open myfile.txt]
set line_num 1
while { [gets $input line] >= 0 } {
    # Process the line of text here
    puts "$line_num: $line"
    incr line_num
}
close $input
```

Syntax & Comments

Arguments to Tcl commands are separated by white space, and Tcl commands are terminated by a newline character or a semicolon. As shown in “[Substitutions](#)” on page 3–43, you must use backslashes when a Tcl command extends more than one line.

Tcl uses the hash or pound character (#) to begin comments. The # character must begin a comment. If you prefer to include comments on the same line as a command, be sure to terminate the command with a semicolon before the # character. **Example 3–52** is a valid line of code that includes a **set** command and a comment:

Example 3–52. Comments

```
set a 1;# Initializes a
```

Without the semicolon, it will be an invalid command because the **set** command will not terminate until the new line after the comment.

The Tcl interpreter counts curly braces inside comments, which can lead to errors that are difficult to track down. [Example 3-53](#) causes an error because of unbalanced curly braces:

Example 3-53. Unbalanced Braces in Comments

```
# if { $x > 0 } {  
if { $y > 0 } {  
    # code here  
}
```

References



For more information about using Tcl, refer to 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
- Quartus II Tcl example scripts at www.altera.com/support/examples/tcl/tcl.html
- Tcl Developer Xchange at tcl.activestate.com

Document Revision History

Table 3–8 shows the revision history for this document.

Table 3–8. Document Revision History

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history to the document.	
May 2006 v6.0.0	Updated for the Quartus II software version 6.0.0. <ul style="list-style-type: none"> ● Reorganized content. ● Added the TimeQuest timing analyzer feature. 	
October 2005 v5.1.0	Updated for the Quartus II software version 5.1.0.	
August 2005 v5.0.1	Minor text changes.	
May 2005 v5.0.0	Updated for the Quartus II software version 5.0.0.	
Dec. 2004 v2.1	<ul style="list-style-type: none"> ● Updates to tables and figures. ● New functionality in the Quartus II software version 4.2. 	
Aug. 2004 v2.1	<ul style="list-style-type: none"> ● Minor typographical corrections ● Enhancements to example scripts. 	
June 2004 v2.0	<ul style="list-style-type: none"> ● Updates to tables, figures. ● New functionality in the Quartus II software version 4.1. 	
Feb. 2004 v1.0	Initial release.	

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 must track their project changes.

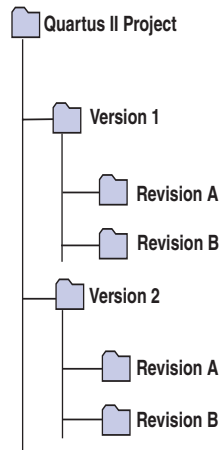
To help designers manage their FPGA designs, the Quartus® II software provides the following capabilities:

- Creating revisions
- Copying and archiving projects
- Making a version-compatible database
- Controlling message suppression

In the Quartus II software, a revision is one set of assignments and settings. A project typically has multiple revisions, with each revision having its own set of assignments and settings. Creating multiple revisions allows you to change assignments and settings while preserving the 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 Quartus II Version-Compatible Database feature allows Quartus II databases to be compatible across different versions of the Quartus II software, which saves valuable design time by avoiding unnecessary compilations (Figure 4-1). This chapter also discusses how to migrate your projects from one computing platform to another as well as message suppression.

Figure 4–1. Quartus II Version-Compatible Database Structure

Creating a New Project

A Quartus II project contains all of the design files, the settings files and other files necessary for the successful compilation of your design. These files include two Quartus II settings files:

- Quartus II Project File (.qpf) containing the name of your project and all revisions of your project, described in [“Using Revisions With Your Design” on page 4–3](#).
- Quartus II Settings File (.qsf) containing all assignments applied to your design including assignments to help fit your design and meet performance requirements. For more information on the Quartus II Settings File, refer to [“Quartus II Settings File” on page 4–25](#).

To start a new Quartus II project, use the **New Project Wizard**. From the File menu, click the **New Project Wizard**, and add the following project information:

- Project name and directory
- Name of the top-level design entity
- Project files and user libraries
- Target device family and device
- EDA tool settings



For more information on user libraries, refer to [“Specifying Libraries” on page 4–13](#) and [“Specifying Libraries Using Scripts” on page 4–29](#).

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 feature allows you to explore different assignments and settings for your design and then compare the results. You can use the Revisions feature in the following ways:

- Create a unique revision which is not based on a previous revision. Creating a unique revision lets you optimize a design for different fundamental reasons such as to optimize by area in one revision, then optimize for f_{MAX} in another revision. When you create a unique revision (a revision that is not based on an existing revision), all default settings are turned on.
- Create a revision based on an existing revision, but try new settings and assignments in the new revision. A new revision already includes all the assignments and settings applied in the previous revision. If you are not satisfied with the results in the new revision, you can revert to the original revision. You can compare revisions manually or with the Compare feature.

Creating & Deleting Revisions

All Quartus II assignments and settings are stored in the Quartus II Settings File. Each time you create a new revision of a project, the Quartus II software creates a new Quartus II Settings File and adds the name of the new revision to the list of revisions in the Quartus II Settings File.



The name of a new Quartus II Settings File matches the revision name.

You can manage revisions with the **Revisions** dialog box, which allows you to set the current revision, as well as create, delete, and compare revisions in a project.

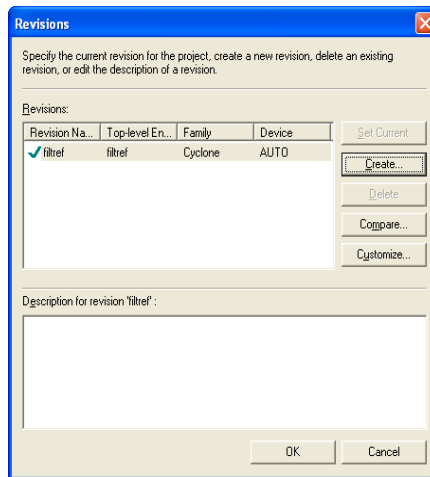
Creating a Revision

To create a revision, follow these steps:

1. If you have not already done so, create a new project or open an existing project. On the File menu, click **New Project Wizard** or **Open Project**.
2. On the Project menu, click **Revisions**.
3. To base the new revision on an existing revision for the current design, select the existing revision in the **Revisions** list.

4. Click **Create** (Figure 4-2).

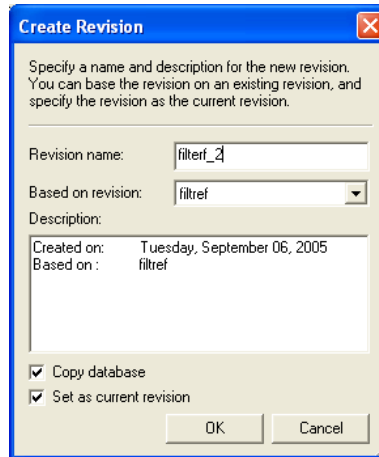
Figure 4-2. Revisions Dialog Box




5. In the **Create Revision** dialog box (Figure 4-3), type the name of the new revision in the **Revisions name** box.
6. To base the new revision on an existing revision for the current design, if you did not select the correct revision in Step 3, select the revision in the **Based on revision** list (Figure 4-3).

or

To create a unique revision that is not based on an existing revision of the current design, select the “blank entry” in the **Based on revision** list.

Figure 4–3. Create Revisions Dialog Box

7. Optionally, edit the description of the revision in the **Description** box (Figure 4–3).
8. Turn off the **Copy database** option if you do not want the new revision to contain the database information from the existing revision. The **Copy database** option is on by default.

 Copying the database allows you to view the results from the previous compilation while you are making changes to the settings of the new revision.
9. If you do not want to use the new revision immediately, turn off **Set as current revision**. The **Set as current revision** option is on by default.
10. In the **Create Revision** dialog box (Figure 4–3), click **OK**.

Delete a Revision

To delete a revision, follow these steps:

1. If you have not already done so, open an existing project by clicking **Open Project on the File menu and selecting a Quartus II Project File**.
2. On the Project menu, click **Revisions**.

- In the **Revisions** list, select the revision you want to delete and click **Delete**.



You cannot delete the current revision when it is active; you must first open another revision. For example, if revision A is currently active, you need to create (if the revision does not exist) and open revision B before you delete revision A.

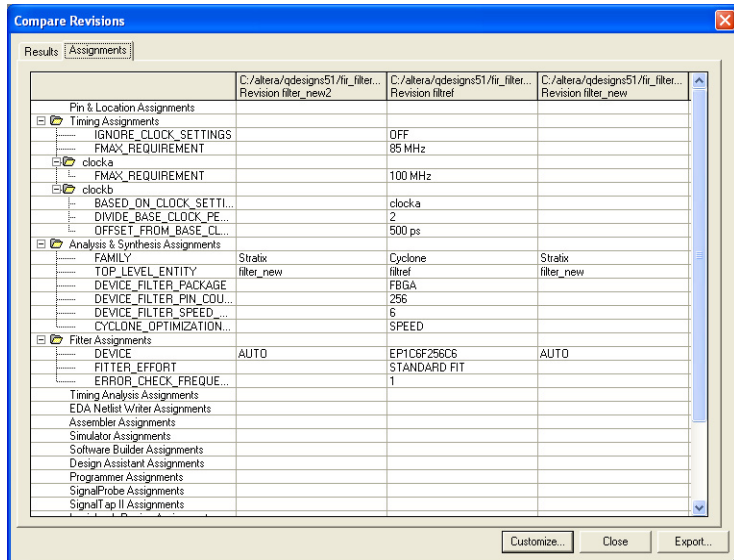
Comparing Revisions

You can compare the results of multiple revisions side by side with the **Compare Revisions** dialog box.

- On the **Project** menu, click **Revisions**.
- In the **Revisions** dialog box, click **Compare** to compare all revisions in a single window.

The **Compare Revisions** dialog box (Figure 4-4) compares the results of each revision in three assignment categories: Analysis & Synthesis, Fitter, and Timing Analyzer.

Figure 4-4. Compare Revisions Dialog Box



In addition to viewing the results of each revision, you also can show the assignments for each revision. Click the **Assignments** tab in the **Compare Revisions** dialog box to view all assignments applied to each revision (Figure 4-4). To export both **Results** and **Assignments** for your revisions, click on **Export**. When the dialog box appears, enter the name of the Comma-Separated Value (.csv) file into which the software will export the data when you click **OK**. Gain better understanding of how different optimization options affect your design by viewing the results of each revision and their assignments.

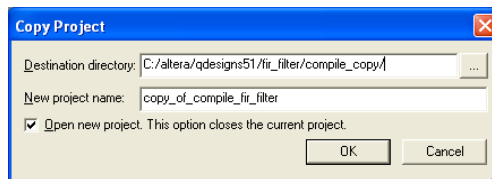
Creating Different Versions of Your Design

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 (all assignments and settings).

To create a new version of your project with the Quartus II software, create a copy of your project and edit your design files. An example is if you have a design that is compatible with a 32-bit data bus and you require 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 and edit the new version of the design files by performing the following steps:

1. On the Project menu, click **Copy Project**. The **Copy Project** dialog box appears (Figure 4-5).

Figure 4-5. Copy Project Dialog Box



2. Specify 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 **Open new project**. This option closes the current project option.
5. Click **OK**.

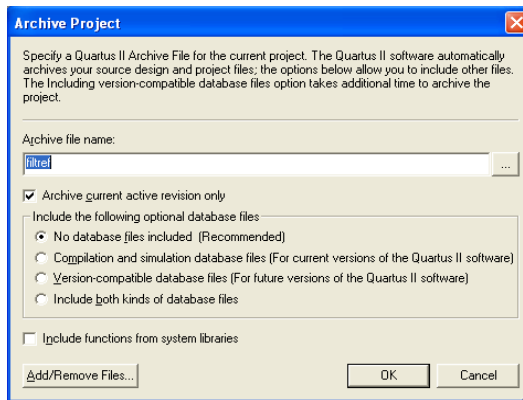
When creating a new version of your project with an Electronic Data Interchange Format (EDIF) or Verilog Quartus Mapping (.vqm) netlist from a third-party EDA synthesis tool, create a copy of your project and then replace the previous netlist file with the newly generated netlist file. On the Project menu, click **Copy Project** to create a copy of your design. On the Project menu, click the **Add/Remove Files** command to add and remove design files, if necessary.

Archiving Projects with the Quartus II Archive Project Feature

A single project can contain hundreds of files in many directories, which can make transferring a project between engineers difficult. 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. The Quartus II Archive File contains all the files, including the Quartus II Default Settings File (.qdf), required to perform a full compilation to restore the original results. The Quartus II Default Settings File contains all the project and assignment default settings from the current version of the Quartus II software. It is necessary to archive the Default Settings File to preserve your results when you restore the archive in a different version of the Quartus II software. For more information on the Quartus II Default Settings File, refer to “[Quartus II Default Settings File](#)” on page 4-26.

With the archive file generated by the Archive Project feature ([Figure 4-6](#)), you can easily share projects between engineers.

Figure 4-6. Archive Project Dialog Box



Archive a Project

To archive a project, perform these steps:

1. If you have not already done so, create a new project or open an existing project. On the File menu, click **New Project Wizard** or **Open Project**.
2. On the Processing menu, point to Start and click **Start Analysis & Elaboration**.



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

3. On the Project menu, click **Archive Project**.
4. In the **Archive file name** box, type the file name of the Quartus II Archive File you want to archive, or click **Browse** to select a Quartus II Archive File name.
5. Turn on **Archive current active revision only** to archive the currently active revision. If you do not turn on this option, all revisions within the project are included in the project archive.
6. Select one of the following items under **Include the following optional database files** in the **Archive Project** dialog box (Figure 4–6).
 - a. Select **No database files included** to exclude both compilation and simulation database files and version-compatible database files from the archive.
 - b. Select **Compilation and simulation database files** to include the compilation and simulation database files in the archive.
 - c. Select **Version-compatible database files** to include the version-compatible database files in the archive.
 - d. Select **Include both kinds of database files** to include both compilation and simulation database files and version-compatible database files in the archive.
7. Turn on **Include functions from system libraries** to include functions from system libraries in the archive.
8. Click **Add/Remove Files** to add or remove files from the archive.

9. Click **OK**.

Restore an Archived Project

To restore an archived project, perform the following steps:

1. On the Project menu, click **Restore Archived Project**.
2. In the **Archive file name** box, type the file name of the Quartus II Archive File you wish to restore, or click **Browse to** select a Quartus II Archive File.
3. In the **Destination folder** box, specify the directory path into which you will restore the contents of the Quartus II Archive File, or browse to a directory.
4. Click **Show log** to view the Quartus II Archive Log File (**.qarlog**) for the project you are restoring from the Quartus II Archive File.
5. Click **OK**.
6. If you did not include the compilation and simulation database files in the project archive ([Figure 4–6](#)), you must recompile the project.

Version- Compatible Databases

Prior to the Quartus II software version 4.1, 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. For example, using one 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 version 5.1 and later without recompiling your design. Using this feature eliminates unnecessary compilation time.

Migrate to a New Version

To migrate a design from one Quartus II software version to a newer version, perform the following steps:

1. On the File menu, open the older version of the Quartus II software project by clicking **Open Project**.
2. On the Project menu, click **Copy Project** to make a new copy of the project. The copied project will open in the older version.

3. On the Project menu, click **Export Database**. By default, the database is exported to the **export_db** directory of the copied project. If desired, a new directory can be created.
4. Open the copied project from the new version of the Quartus II software. The Quartus II software deletes the existing database but not the exported database.
5. On the Project menu, click **Import Database**. By default, the directory of the database you just exported is selected.

Save the Database in a Version-Compatible Format

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

1. On the Assignments menu, click **Settings**. The **Settings** dialog box displays.
2. In the **Category** list, select the **Compilation Process** page. The Compilation Process page displays.
3. Turn on the **Export version-compatible database** option.
4. Browse to the directory where you want to save the database.
5. Click **OK**.

Quartus II Project Platform Migration

When moving your project from one computing platform to another, you must think about the following cross-platform issues:

- Filenames and Hierarchy
- Specifying Libraries
- Quartus II Search Path Precedence Rules
- Quartus II-Generated Files for Third-Party EDA Tools
- Migrating Database Files

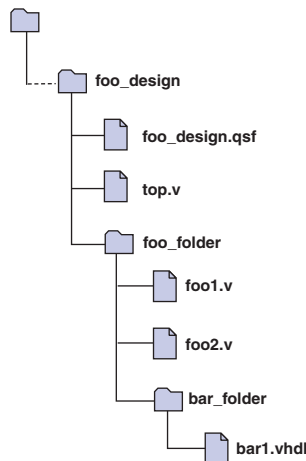
Filenames & Hierarchy

To ensure migration across platforms, you must consider a few basic differences between operating systems when naming source files, especially when interacting with these from the system-command prompt or a Tcl script:

- Some operating system file systems are case-sensitive. When writing scripts, ensure you specify paths exactly, even if the current operating system is not case-sensitive. For best results, use lowercase letters when naming files.
- Use a character set common to all the used platforms.
- You do not have to convert the forward-slash / and back-slash \ path separators in the Quartus Settings File because the Quartus II software changes all back-slash \ path separators to forward-slashes /.
- Observe the shortest file name length limit of the different operating systems you are using.

You can specify files and directories inside a Quartus II project as paths relative to the project directory. For instance, for a project titled **foo_design** with a directory structure shown in [Figure 4–7](#), specify the source files as: **top.v**, **foo_folder/foo1.v**, **foo_folder/foo2.v**, and **foo_folder/bar_folder/bar1.vhdl**.

Figure 4–7. All-Inclusive Project Directory Structure



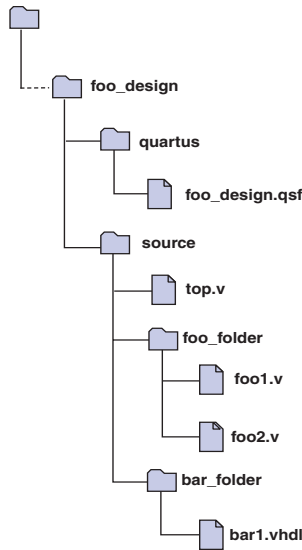
If the Quartus II Settings File is in a directory that is separate from the source files, you can specify paths using the following options:

- Relative paths
- Absolute paths
- Libraries

Relative Paths

If the source files are very near the Quartus II project directory, you can express relative paths using the `..` notation. For example, given the directory structure shown in [Figure 4–8](#), you can specify `top.v` as `../source/top.v` and `foo1.v` as `../source/foo_folder/foo1.v`.

Figure 4–8. Quartus II Project Directory Separate from Design Files



When you copy a directory structure to a different platform, ensure that any subdirectories are in the same hierarchical structure and relative path as in the original platform.

Specifying Libraries

You also can specify the directory (or directories) containing source files as a library that the Quartus II software searches when you compile your project. A Quartus II library is a directory containing design files used by your Quartus II project. You can specify the following two kinds of libraries in the Quartus II software:

- User libraries, which apply to a specific project
- Global libraries, which all projects use

Use the procedures in this section to specify user or global libraries.

All files in the directories specified as libraries are relative to the libraries. For example, if you want to include the file `/user_lib1/foo1.v` and the `user_lib1` directory is specified as a user library in the project, the `foo1.v` file can be specified in the Quartus II Settings File as `foo1.v`. The Quartus II software searches directories that are specified as libraries and finds the file.

Specifying User Libraries

To specify user libraries from the GUI: from the Assignments menu, click **Settings**, and select **User Libraries (Current Project)**. Type the name of the directory in the **Library name** box, or browse to it. User libraries are stored in the Quartus II Settings File of the current revision.

Specifying Global Libraries

Specify global libraries from the GUI: On the Tools menu, click **Options**, and select **Global User Libraries (All Project)**. Type the name of the directory in the **Library name** box, or browse to it. Global libraries are stored in the `quartus2.ini` file. The Quartus II software searches for the `quartus2.ini` file in the following order:

- `USERPROFILE`, for example,
`C:\Documents and Settings\<user name>`
- **Directory specified by the TMP** environmental variable
- **Directory specified by TEMP** environmental variable
- Root directory, for example, `C:`

For UNIX and Linux users, the file is created in the `altera.quartus` directory under the `<home>` directory, if the `altera.quartus` directory exists. If the `altera.quartus` directory does not exist, the file is created in the `<home>` directory.



Whenever you specify a directory name in the GUI or in Tcl, the name you use is maintained verbatim in the Quartus II Settings File rather than resolved to an absolute path.

If the directory is outside of the project directory, the path returned in the dialog box is an absolute path, and you can use the **Browse** button in either the **Settings** dialog box or the **Options** dialog box to select a directory. However, you can change this absolute path to a relative path by editing the absolute path displayed in the Library name field to create a relative path before you click **Add** to put the directory in the **Libraries** list.

When copying projects that specify user libraries, you must either copy your user library files along with the project directory or ensure that your user library files exist in the target platform.

Search Path Precedence Rules

If two files have the same file name, the file found is determined by the Quartus II software's search path precedence rules. The Quartus II software resolves relative paths by searching for the file in the following directories and order:

1. The project directory, which is the directory containing the Quartus II Settings File.
2. The project's database (**db**) directory.
3. User libraries are searched in the order specified by the **USER_LIBRARIES** setting of the Quartus II Settings File for the current revision.
4. Global user libraries are searched in the order specified by the **USER_LIBRARIES** setting on the **Global User Libraries** page in the **Options** dialog box.
5. The Quartus II software **libraries** directory.



For more information on libraries, refer to [“Specifying Libraries Using Scripts”](#) on page 4-29.

Quartus II-Generated Files for Third-Party EDA Tools

When you copy your project to another platform, regenerate any Quartus II software-generated files for use by other EDA tools, using the GUI or the **quartus_eda** executable.

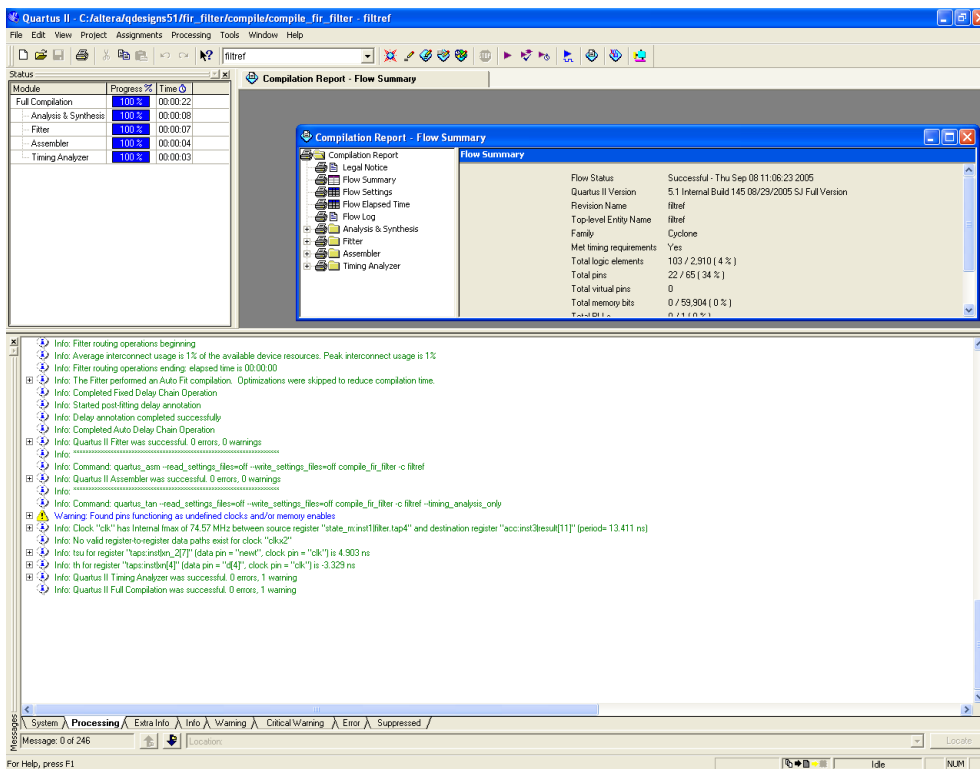
Migrating Database Files

There is nothing inherent in the file format and syntax of exported version-compatible database files that might cause problems for migrating the files to other platforms. However, the contents of the database can cause problems for platform migration. For example, use of absolute paths in version-compatible database files generated by the Quartus II software can cause problems for migration. Altera recommends that you change absolute paths to relative paths before migrating files whenever possible.

Working with Messages

The Quartus II software generates various types of messages, including Information, Warning, and Error messages. Some messages include information on software status during a compilation and alert you to possible problems with your design. Messages are displayed in the Messages window in the Quartus II GUI (Figure 4–9), and written to standard out and when you use command-line executables. In both cases, messages are written to Quartus II report files.

Figure 4–9. Viewing Quartus II Messages



You can right-click on a message in the Message window and get help on the message, locate the source of the message of your design, and manage messages.

Messages provide useful information if you take time to review them after each compilation. However, it can be tedious if there are thousands of them. Beginning with version 5.1 and later, the Quartus II software includes new features to help you manage messages.

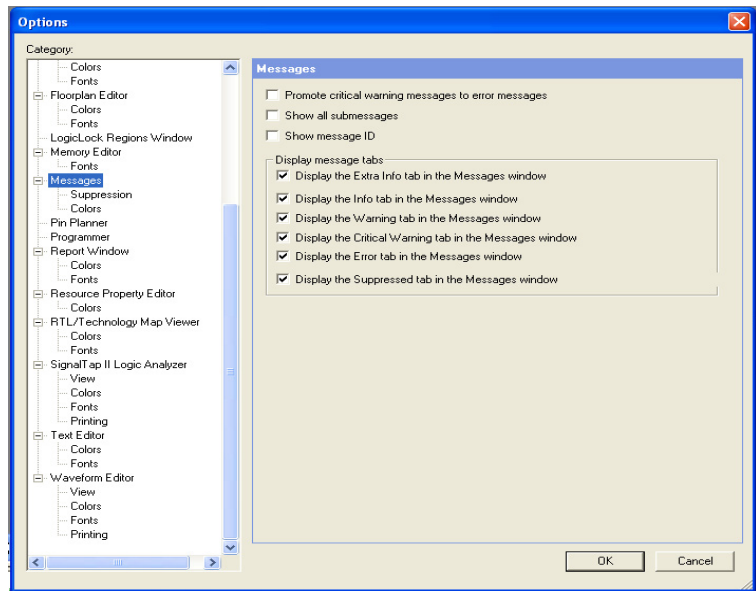
Messages Window

By default, the Messages window displays eight message tabs (Table 4–1), which makes it easy to review all messages of a certain type.

Message Tab	Description
System	Displays messages that are unrelated to processing your design. For example, messages generated during programming are displayed in the System tab.
Processing	Displays messages that are generated when the Quartus II software processes your most recent compilation, simulation, or software build; timing analysis messages appear as part of the compilation messages.
Info	Displays general informational messages during a compilation, simulation, or software build. For example: legal and compilation-success messages.
Extra Info	Displays detailed informational messages about the operations for designers. For example: extra fitting information messages.
Warning	Displays strong warning messages generated during a compilation, simulation, or software build. For example: detection of signal promotion to global and high fan-out nets.
Critical Warning	Displays critical warning messages generated during a compilation, simulation, or software build. For example: detection of combinational feedback loops, gated clocks, or register duplication.
Error	Displays processing and compilation error messages generated during a compilation, simulation, or software build. Error messages can sometimes stop processing and cannot be disabled.
Suppressed	Displays suppressed messages during the last processing operation.

The **Info**, **Extra Info**, **Warning**, **Critical Warning**, and **Error** tabs display messages grouped by type. Warning messages are shown with all other types of messages in the Processing message window; all warning messages also appear in the **Warning message** tab.

You can control which tabs are displayed by right-clicking in the Messages window and choosing options from the right button pop-up menu, and with the options in the **Display Message** tabs section of the Messages page in the **Options** dialog box of the Tools menu (Figure 4–10).

Figure 4–10. Message Tab Options

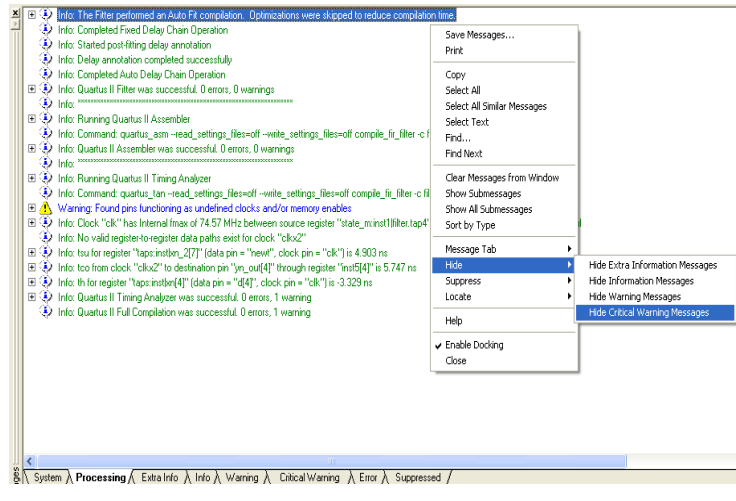
The **Suppressed** tab shows messages suppressed during the last processing operation. You can also prevent the **Suppressed** tab from being displayed with an option in the **Display Message** tabs section of the Messages pane in the **Options** dialog box of the Tools menu.

Hiding Messages

In the Messages window, you can hide all messages of a particular type. For example, to hide Info messages, follow these steps:

1. On the **Processing** tab, right-click in the **Processing** message window, and click the **Hide** option (Figure 4–11).
2. Select the Info message type.

Figure 4–11. Hiding Messages from the Processing Tab



All messages of the specified types are removed from the list of messages in the **Processing** tab, although they are still included in the separate tabs corresponding to the message type. For example, if you hide Info messages, no Info messages are shown in the Processing message window, but all the Info messages are shown in the Info messages window.

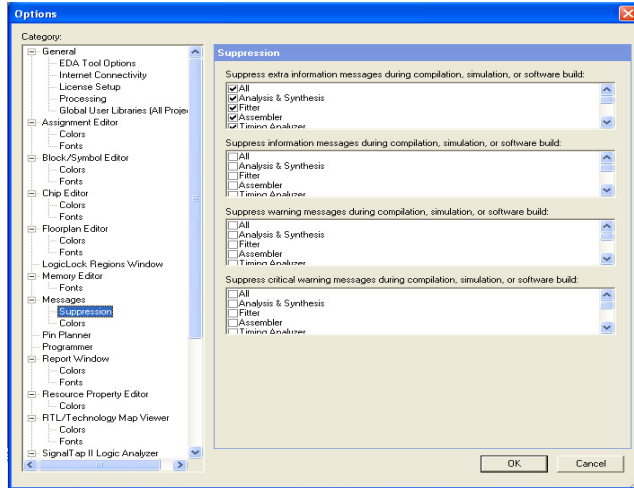
Message Suppression

Message suppression is a new feature in version 5.1 or later of the Quartus II software. You can use message suppression to reduce the number of messages to be reviewed after a compilation by preventing individual messages and entire categories of messages from being displayed. For example, if you review a particular message and determine that it is not caused by something in your design that should be changed or fixed, you can suppress the message so it is not displayed during subsequent compilations. This saves time because you see only new messages during subsequent compilations.

Every time you add a message to be suppressed, a suppression rule is created. Suppressing exact selected messages adds patterns that are exact strings to the suppression rules. Suppressing all similar messages adds patterns with wildcards to the suppression rules.

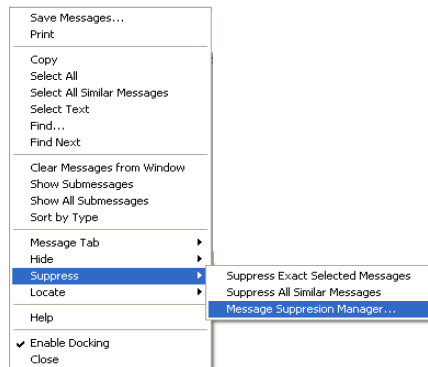
Furthermore, you can suppress all messages of a particular type in a particular stage of the compilation flow. On the Tools menu, click **Options**, and click **Suppression** from under the Messages section (Figure 4–12).

Figure 4–12. Controlling Suppression Messages



Suppressing individual messages is controlled in two locations in the Quartus II GUI. You can right-click on a message in the Messages window and choose commands in the Suppress sub-menu entry. To open the Message Suppression Manager, right-click in the Messages window. From the Suppress sub-menu item, click **Message Suppression Manager** (Figure 4–13).

Figure 4–13. Message Suppression Manager



Refer to “**Message Suppression Manager**” on page 4–22 for further information.

Message Suppression Methods

There are two methods you can use to create suppression rules: Suppress Exact Selected Messages and Suppress All Similar Messages. If you suppress a message with the exact selected messages option, only messages matching the exact text will be suppressed during subsequent compilations. The All Similar Messages option behaves like a wildcard pattern on variable fields in messages.

For an example of suppressing all similar messages, consider the following message:

```
Info: Found 1 design units, including 1 entities, in source file mult.v.
```

This type of message is common during synthesis and is displayed for each source file that is processed, with varying information about the number of design units, entities, and source file name.

Help for this message shows it is in the form Found *<number>* design units, including *<number>* entities, in source file *<name>*. Choosing to suppress all similar messages effectively replaces the variable parts of that message (*<number>*, *<number>*, and *<name>*) with wildcards, resulting in the following suppression rule:

```
Info: Found * design units, including * entities, in source file *.
```

As a result, all similar messages (ones that match the pattern) are suppressed.

Details & Limitations

The following limitations apply to which messages can be suppressed and how they can be suppressed:

- You cannot suppress error messages or messages with information about Altera legal agreements.
- Suppressing a message also suppresses all its submessages, if there are any.
- Suppressing a submessage causes matching submessages to be suppressed only if the parent messages are the same.
- You cannot create your own custom wildcards to suppress messages.
- You must use the Quartus II GUI to manage message suppression, including choosing messages to suppress. These messages are suppressed during compilation in the GUI and when using command-line executables.

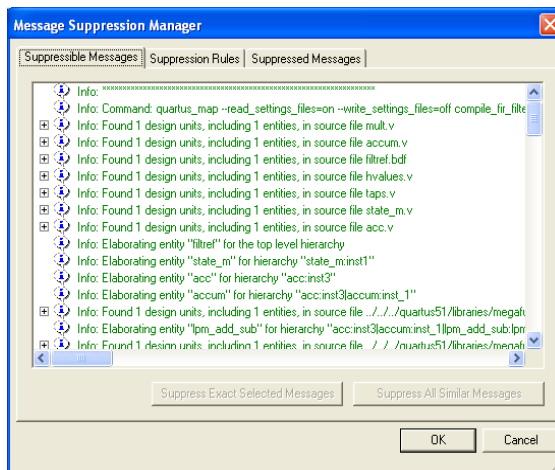
- Messages are suppressed on a per-revision basis, not for an entire project. Information about which messages to suppress is stored in a file called `<revision>.srf`. If you create a revision based on a revision for which messages are suppressed, the suppression rules file is copied to the new revision. You cannot make all revisions in one project use the same suppression rules file.
- You cannot remove messages or modify message suppression rules while a compilation is running.

Message Suppression Manager

You can use the Message Suppression Manager to view and suppress messages, view and delete suppression rules, and view suppressed messages.

Open the Message Suppression Manager by clicking the **Processing** tab. Right-click anywhere in the Messages window and click **Message Suppression Manager** from the Suppression sub-menu. The Message Suppression Manager has three tabs labeled **Suppressible Messages**, **Suppression Rules**, and **Suppressed Messages** (Figure 4-14).

Figure 4-14. Message Suppression Manager Window



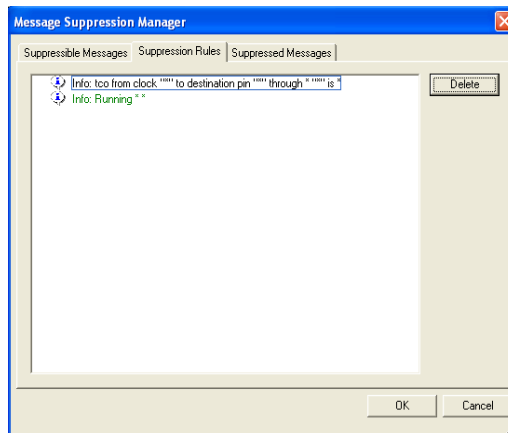
Suppressible Messages

Messages that are listed in the **Suppressible Messages** tab are messages that were not suppressed during the last compilation. These messages can be suppressed. The **Select All Similar Messages** option in the right click menu selects messages according to the example described in the [“Message Suppression Methods” on page 4–21](#). You can select all similar messages to see which messages are suppressed if you choose to suppress all similar messages.

Suppression Rules

Items listed in the **Suppression Rules** tab are the patterns that the Quartus II software uses to determine whether to suppress a message. Messages matching any of the items listed in the **Suppression Rules** tab are suppressed during compilations ([Figure 4–15](#)).

Figure 4–15. Message Suppression Manager



An entry in the **Suppression Rules** tab that includes a message with submessages indicates the submessage is suppressed only when all its parent messages match.

You can stop suppressing messages by deleting the suppression rules that match them (causing them to be suppressed). Merely deleting suppression rules does not cause the formerly suppressed messages to be added to the messages generated during the previous compilation; you must recompile the design for the changed suppression rules to take effect.

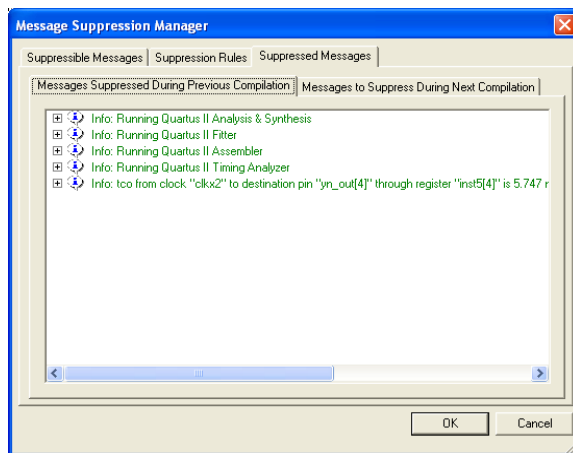
Suppressed Messages

Messages listed in the **Suppressed Messages** tab are divided in two sub-tabs:

- Messages Suppressed During Previous Compilation
- Messages to Suppress During Next Compilation

The messages listed in the Messages Suppressed During Previous Compilation sub-tab are all the suppressed messages from the previous compilation (Figure 4–16).

Figure 4–16. Messages Suppressed During Previous Compilation



These messages are also listed in the **Suppressed** tab in the Messages window. Messages listed in the Messages to Suppress During Next Compilation are messages that will be suppressed during the next compilation that match suppression rules created after the last compilation finished.

In addition to appearing in the **Suppressed** tab in the Messages window, suppressed messages are included in a Suppressed Messages entry in the Quartus II compilation report, viewable in the GUI. Suppressed messages are not included in the `<revision>.<module>.rpt` text files; they are written to a separate text report file called `<revision name>.<module>.smmsg`.

Quartus II Settings File

All assignments made in the Quartus II software are stored as Tcl commands in the Quartus II Settings File. The Quartus II Settings File is a text based file containing Tcl commands and comments. The Quartus II Settings File is not a Tcl script and does not support the full Tcl scripting language.

As you make assignments in the Quartus II software, the assignments are either stored temporarily in memory or written out to the Quartus II Settings File. This is determined by the **Update assignments to disk during design processing only** option, which is located in the Tools menu under Options on the Processing page. If the option is turned on, then all assignments are stored in memory and are written to the Quartus II Settings File when a compilation has started or when you save or close the project. By saving assignments to memory, the performance of the software is improved because it avoids unnecessary reading and writing to the Quartus II Settings File on the disk. This performance improvement is seen more dramatically when the project files are stored on a remote data disk.

Beginning with the Quartus II software version 5.1, you can add lines of comments into the Quartus II Settings File, such as are shown in the following example:

```
# Assignments for input pin clk
# Clk is being driven by FPGA 1
set_location_assignment PIN_6 -to clk
set_instance_assignment -name IO_STANDARD "2.5 V" -to clk
```

Sourcing other Quartus II Settings Files is supported using the following Tcl command:

```
source <filename>.qsf
```

Format Preservation

Beginning with the Quartus II software version 5.1, the Quartus II software maintains the order of assignments within the Quartus II Settings File. When you make new assignments, they are appended to the end of the Quartus II Settings File. If you modify an assignment, the corresponding line in the Quartus II Settings File is modified and the order of assignments in the Quartus II Settings File is maintained except when you add and remove project source files, or when you add, remove, and exclude members from an assignment group. In these cases, all assignments are moved to the end of the Quartus II Settings File. For example, if you add a new design file into the project, the list of all your design files is removed from its current location in the file and moved to the end of the Quartus II Settings File.



The header that is located at the beginning of the Quartus II Settings File is written only if the Quartus II Settings File is newly created.

The Quartus II software preserves all spaces and tabs for all unmodified assignments and comments. When you make a new assignment or modify an existing assignment, the assignment is written using the default formatting.

Quartus II Default Settings File

The Quartus II Default Settings File contains all the project and assignment default settings from the current version of the Quartus II software. The Quartus II Default Settings File, located in the win directory of the Quartus II installation path, is used to ensure consistent results when defaults are changed between versions of the Quartus II software.

The Quartus II software reads assignments from various files and stores the assignments in memory. The Quartus II software reads settings files in the following order shown below, so that assignments in subsequent files take precedence over earlier ones:

1. **assignment_defaults.qdf** from *<Quartus II Installation directory>/win*
2. **assignment_defaults.qdf** from project directory
3. *<revision name>_assignment_defaults.qdf* from project directory
4. *<revision name>.qsf* from project directory

As each new file is read, if an existing assignment from a previous file matches (following rules of case sensitivity, multi-value fields as well as other rules), then the old value is removed and replaced by the new. For example, if the first file has a non multi-valued assignment A=1, and the second file has A=2, then the assignment A=1, stored in memory, is replaced by A=2.

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
```

The *Scripting Reference Manual* includes the same information in PDF form.



For more information about Tcl scripting, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. Refer to the *Quartus II Settings File Reference Manual* for information about all settings and constraints in the Quartus II software. For more information about command-line scripting, refer to the *Command-Line Scripting* chapter in volume 2 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, refer to “[Creating & Deleting Revisions](#)” on page 4–3.

Creating Revisions

The following Tcl command creates a new revision called **speed_ch**, based on a revision called **chiptrip** and sets the new revision 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 specify 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 with a Tcl Command or at the Command Prompt

You can archive projects with a Tcl command or with a command run at the system command prompt. For more information about archiving projects, refer to [“Archiving Projects with a Tcl Command or at the Command Prompt”](#) on page 4–28.

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 called **top**:

```
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, refer to [“Restore an Archived Project”](#) on page 4–10.

The following Tcl command restores the project archive named **archive.qar** in the **restored** subdirectory 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 & 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, refer to [“Version-Compatible Databases”](#) on page 4–10.



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

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

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

Use the following Tcl commands from the `flow` package to import or export version-compatible databases. If you use the `flow` package, you must 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> ←
```

Specifying Libraries Using Scripts

In Tcl, use commands in the `::quartus::project` package to specify user libraries. To specify user libraries, use the `set_global_assignment` command. To specify global libraries use the `set_user_option` command.

The following examples show typical usage of the `set_global_assignment` and `set_user_option` commands:

```
set_global_assignment -name USER_LIBRARIES \
"../other_dir/library1"
set_user_option -name USER_LIBRARIES \
"../an_other_dir/library2"
```

To report any user libraries specified for a project and any global libraries specified for the current installation of the Quartus II software, use the **get_global_assignment** and **get_user_option** Tcl commands. The following Tcl script outputs the user paths and global libraries for an open Quartus II project:

```
get_global_assignment -name USER_LIBRARIES
get_user_option -name USER_LIBRARIES
```

Conclusion

Designers often try different settings and versions of their designs throughout the development process. Quartus II project revisions facilitate the creation and management of different assignments and settings.

In addition, understanding how to smoothly migrate your projects from one computing platform to another, controlling messages, and reducing compilation time is important as well. The Quartus II software facilitates efficient management of your design to accommodate today's more sophisticated FPGA designs.

Document Revision History

Table 4–2 shows the revision history for this document.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0		
May 2006 v6.0.0	Minor updates for the Quartus II software version 6.0.0.	
October 2005 v5.1.0	Updated for the Quartus II software version 5.1.0.	
May 2005 v5.0.0	Updated for the Quartus II software version 5.0.0.	
Dec. 2004 v1.1	Updated for Quartus II software version 4.2: <ul style="list-style-type: none"> ● General formatting and editing updates. ● Added new figures. ● Added new introduction to To Delete a Revision That is a Design's Current Revision. ● Added new section To Delete a Revision That is not a Design's Current Revision. ● Updated figures. ● Added new information about displaying assignments for multiple revisions. ● Updated Archive a Project. ● Updated Restore an Archived Project. ● Version-Compatible Databases describes migration to Quartus II software version 4.2. ● Corrected Tcl commands. 	
June 2004 v1.0	Initial release.	

This section provides an overview of the I/O planning process, Altera's FPGA pin terminology, as well as the various methods for importing, exporting, creating, and validating pin-related assignments using Quartus® II software, and describes the design flow that includes making and analyzing pin assignments using the **Start I/O Assignment Analysis** command in the Quartus II software, during and after the development of your HDL design.

This section includes the following chapters:

- [Chapter 5, I/O Management](#)
- [Chapter 6, Mentor Graphics PCB Design Tools Support](#)
- [Chapter 7, Cadence PCB Design Tools Support](#)



For information about the revision history for chapters in this section, refer to each individual chapter for that chapter's revision history.

Introduction

The process of managing I/Os for today's leading FPGA devices involves more than just fitting design pins into a package. The increasing complexity of today's I/O standards and pin placement guidelines are just some of the factors that influence pin-related assignments. The I/O capabilities of the FPGA device and board layout guidelines influence pin location and other types of assignments for each of your design pins. Therefore, it is necessary to begin I/O planning and printed circuit board (PCB) development even before starting the FPGA design.

This chapter provides an overview of the I/O planning process, FPGA pin terminology and the various methods for importing, exporting, creating, and validating pin-related assignments.

I/O Planning Overview

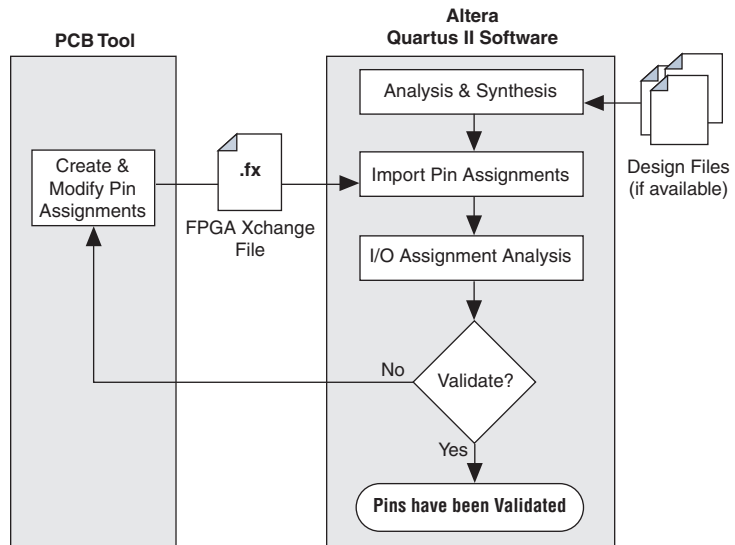
I/O planning includes creating pin-related assignments and validating them against pin placement guidelines. This process ensures a successful fit in your Altera® FPGA device. The Quartus® II software includes the Pin Planner and the I/O Assignment Analyzer to assist you in I/O planning.

The method you use to create your pin assignments depends on your requirements. If your PCB is partially designed, create your FPGA assignments in your PCB tool and import them into the Quartus II software for validation (Figure 5-1).



Currently, only the Mentor Graphics I/O Designer PCB tool supports in this reverse I/O planning flow.

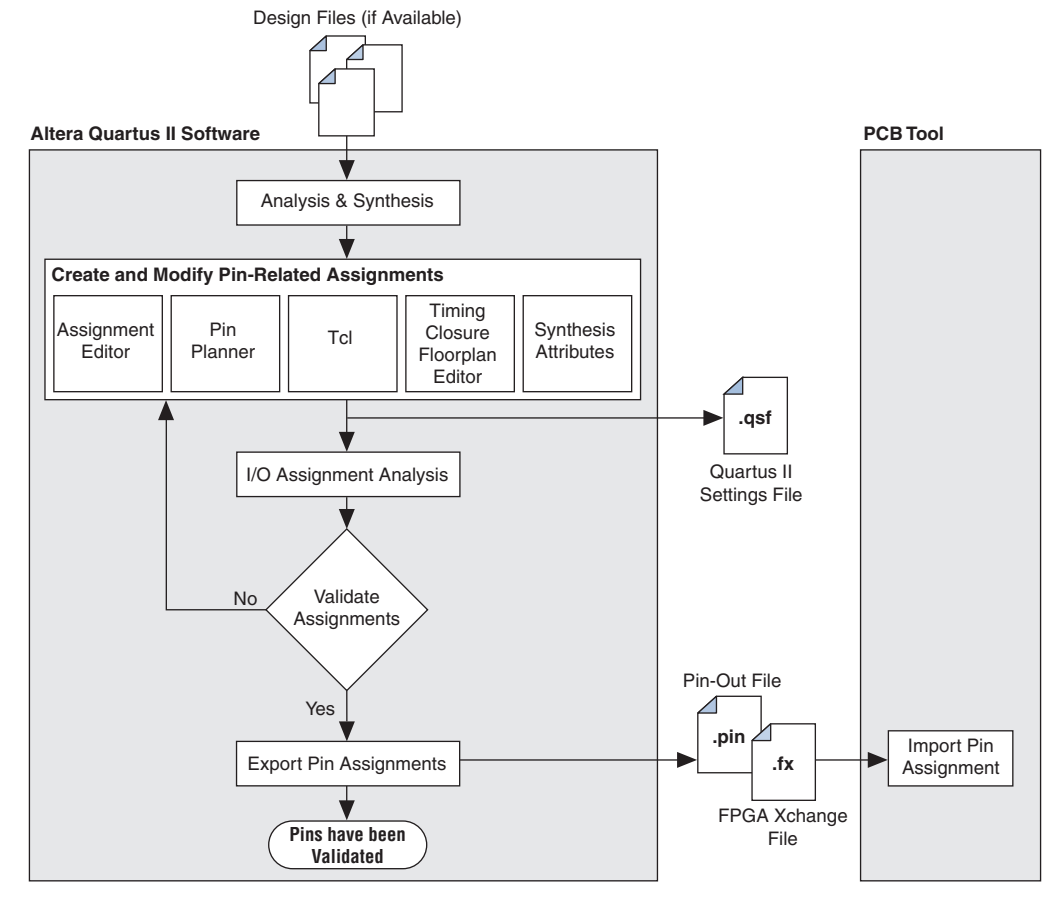
Figure 5–1. I/O Planning Flow Using an FPGA Exchange File from a PCB Tool



For more information about board layout and I/O pin assignment import and export, refer to the *Cadence PCB Design Tools Support* and the *Mentor Graphics PCB Design Tools Support* chapters in volume 2 of the *Quartus II Handbook*.

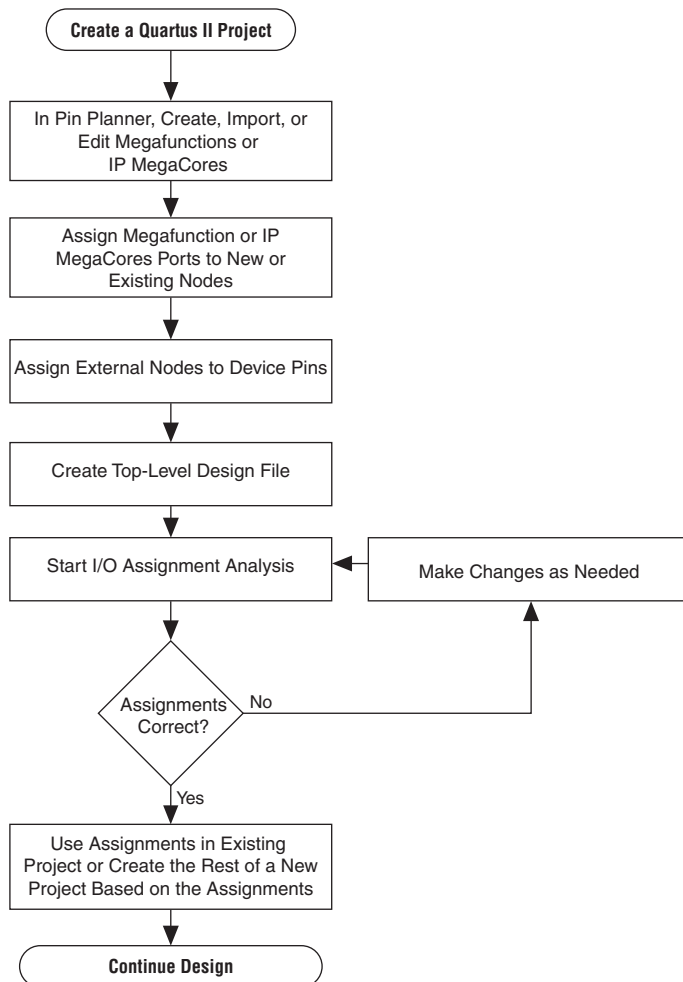
If you have not designed the PCB yet, create and validate your I/O assignments in the Quartus II software, then export them to the PCB tool (Figure 5–2). This is the typical design flow for creating I/O assignments for an FPGA design.

Figure 5–2. Quartus II Software I/O Planning Flow



When creating pin assignments, the preferred method for validating those assignments is to perform a full compilation first. If design files are not available, this may not be possible. You can create a top-level netlist wrapper file while making pin assignments and creating custom megafunctions without requiring any design files (Figure 5–3). With this wrapper file, you can use the I/O Assignment Analyzer to validate your I/O assignments early in the FPGA design process.

For more details about this early I/O planning design flow, refer to “[Early I/O Planning Using the Pin Planner](#)” on page 5–52.

Figure 5–3. Early I/O Planning Using the Pin Planner

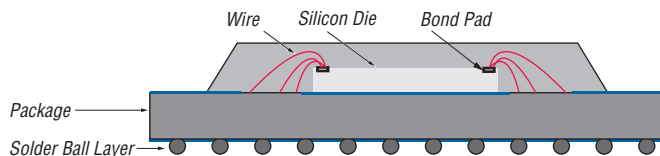
Understanding Altera FPGA Pin Terminology

Altera FPGA devices are available in a variety of packages to meet all of your complex design needs. To describe Altera FPGA pin terminology, this chapter uses a wirebond ball grid array (BGA) package in its examples. On the top surface of the silicon die, there is a ring of bond pads that connect to the I/O pins of the silicon. In a wirebond BGA package, the device is placed in the package and copper wires connect the bond pads to the solder balls of the package. [Figure 5–4](#) shows a cross section of a wirebond BGA package.



For a list of all BGA packages available for each Altera FPGA device, refer to the *Altera Device Package Information Datasheet*.

Figure 5-4. Wire Bond BGA

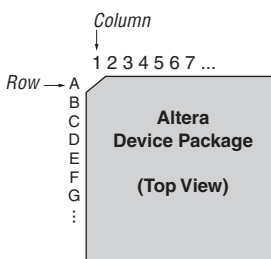


Package Pins

The pins of a BGA package are small solder balls arranged in a grid pattern on the bottom of the package. In the Quartus II software, the package pins are represented as pin numbers. The pin numbers are determined by their locations using a coordinate system with the letters and numbers identifying the row and column of the pins, respectively.

The upper-most row of pins is labeled “A” and continues alphabetically as you move downward (Figure 5-5). The left-most column of pins is labeled “1” and continues with increments of 1 as you move to the right. For example, pin number “A1” represents row “A” and column “1.”

Figure 5-5. Row & Column Labeling



The letters I, O, Q, S, X, and Z are never used in pin numbers. If there are more rows than letters of the alphabet, then the alphabet is repeated, prefixed with the letter “A.”

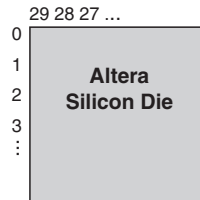


For more information about the pin numbers for your Altera device, refer to the device pin-outs available on the Altera web site, www.altera.com.

Pads

Package pins are connected to pads located on the perimeter of the top metal layer of the silicon die (Figure 5-4). Each pad is identified by a pad ID, which is numbered starting at 0, incrementing by 1 in a counter clockwise direction (Figure 5-6).

Figure 5-6. Pad Number Ordering



To prevent signal integrity issues, the Quartus II software uses pin placement rules to validate your pin placements and pin-related assignments. It is important that you understand to which pad locations your pins were assigned, because some pin placement rules describe pad placement restrictions. For example, in certain devices, there is a restriction on the number of I/O pins supported by a VREF pad to ensure signal integrity. There are also restrictions on the number of pads between single-ended input or output pins and a differential pin. The Quartus II software performs pin placement analysis, and if pins are not placed according to pin placement rules, the design compilation fails and the Quartus II software reports an error.



For more information about pin placement guidelines, refer to the *Design Consideration* section of the *Selectable I/O Standards* chapter in volume 1 of the appropriate device handbook.

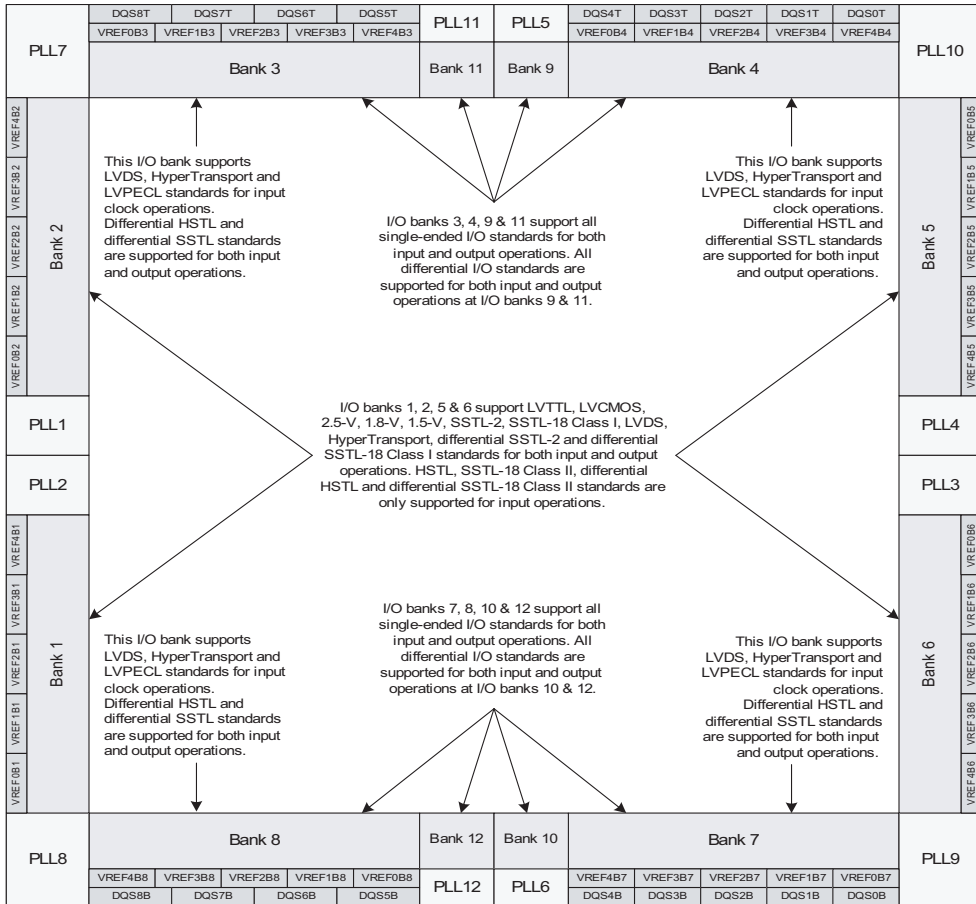
I/O Banks

I/O pins are organized into I/O banks designed to facilitate the various supported I/O standards. Each I/O bank is numbered and has its own voltage source pins, called VCCIO, to offer the highest I/O performance. Depending on the device and I/O standards for the pins within the I/O bank, the specified voltage of the VCCIO pin is between 1.5 V and 3.3 V. Each I/O bank can support multiple pins with different I/O standards that share the same VCCIO.

It is important to refer to the appropriate device handbook to determine the capabilities of each I/O bank. For example, the pins in the I/O banks on the left and right side of a Stratix® II device support high-speed I/O

standards such as LVDS, whereas the pins on the top and bottom I/O banks support all single-ended I/O standards, including DQS signaling (Figure 5–7). Pins belonging to the same I/O bank must use the same VCCIO signal.

Figure 5–7. Stratix II I/O Banks Notes (1), (2), (3), (4)



Notes to Figure 5–7:

- (1) Figure 5–7 is a top view of the silicon die which corresponds to a reverse view for flip chip packages. It is a graphical representation only.
- (2) Depending on size of the device, different device members have different number of V_{REF} groups. Refer to the pin list and the Quartus II software for exact locations.
- (3) Banks 9 through 12 are enhanced PLL external clock output banks.
- (4) Horizontal I/O banks feature SERDES and DPA circuitry for high speed differential I/O standards. For more information about differential I/O standards, refer to the *High-Speed Differential I/O Interfaces in Stratix II Devices* chapter in volume 2 of the *Stratix II Device Handbook*.

VREF Groups

A VREF group is a group of pins that includes one dedicated V_{REF} pin as required by voltage-referenced I/O standards. A VREF group is made up of a small number of pins, compared to the I/O bank, in order to maintain the signal integrity of the V_{REF} pin. One or more VREF groups exist in an I/O bank. Each pin in a VREF group shares the same V_{CCIO} and V_{REF} voltages.



For more information about I/O banks, VREF groups, and supported I/O standards, refer to the *Architecture and Selectable I/O Standards* chapters in volume 1 of the appropriate device handbook.

Importing & Exporting Pin Assignments

You can transfer pin-related assignments between the Quartus II software and other tools by importing and exporting these assignments in the following file formats: Comma Separated Value (.csv) file, Quartus II Settings File (.qsf), Tool command language (Tcl), FPGA Xchange (.fx) file, and Pin-Out (.pin) file (export only).

Comma Separated Value File

You can transfer pin-related assignments as a Comma Separated Value file. This file consists of a row of column headings followed by rows of comma-separated data. The row of column headings in the exported file is in the same order and format as the columns displayed in the Pin category in the Assignment Editor or in the All Pins list in the Pin Planner when the export is performed. Do not modify the row of column headings if you plan to import the Comma Separated Value file later.

To import a Comma Separated Value file into your project, on the Assignment menu, click **Import Assignments** and browse to the file.

You can export pin-related assignments from the Quartus II Pin Planner or the Assignment Editor. To export your pin-related assignments to a Comma Separated Value file, on the Assignment menu, click **Pin Planner** or **Assignment Editor**. For the Pin Planner, make sure the All Pins list is visible. If the list is not visible, on the View menu, click **All Pins List**. For the Assignment Editor, select the **Pin** category from the **Category** list. Then, to create the Comma Separated Value (.csv) file, on the File menu, click **Export**.



The All Pins list in the Pin Planner and the Pin category in the Assignment Editor display detailed properties about each pin of the device, similar to the device pin-out files (available on the Altera web site at www.altera.com) in addition to the pin name and pin number.



For more information about importing and exporting Comma Separated Value files and the Assignment Editor, refer to the *Assignment Editor* chapter in volume 2 of the *Quartus II Handbook*.

Quartus II Settings Files

You can transfer pin-related assignments as a Quartus II Settings File. The pin-related assignments are stored as Tcl commands in the Quartus II Settings File.

To import a Quartus II Settings File, on the Assignments menu, click **Import Assignments** and browse to the file. You can also import a Quartus II Settings File by sourcing the file in the Tcl console. To export a Quartus II Settings File, on the Assignments menu, click **Export Assignments**, type in a file name, and click **OK**.



For more information about Quartus II Settings Files, refer to the Quartus II *Project Management* chapter in volume 2 of the *Quartus II Handbook*.

Tcl Script

To import the pin-related assignments from a Tcl script, source the Tcl script in the Tcl console or run the Tcl script with the `quartus_sh` executable. For example, type the following command at a system command prompt:

```
quartus_sh -t my_pins.tcl ↵
```

You can export pin-related assignments from the Quartus II Pin Planner or the Assignment Editor. To export pin-related assignments as a Tcl script, on the Assignments menu, click **Pin Planner** or **Assignment Editor**. For the Pin Planner, make sure the All Pins list is visible. If the list is not visible, on the View menu, click **All pins List**. For the Assignment Editor, select the **Pin** category from the **Category** list. Then, to create the `.tcl` file, on the File menu, click **Export**. In the **Export** dialog box, type in a file name, select **Tcl Script File (*.tcl)**, and click **OK**. All pin-related assignments displayed in the All Pins list of the Pin Planner and the spreadsheet of the Assignment Editor are saved as Tcl commands in the Tcl script.



For more information about Quartus II scripting support including examples, refer to the *Tcl Scripting* and *Command-Line Scripting* chapters in volume 2 of the *Quartus II Handbook*.

FPGA Xchange File

An FPGA Xchange file contains device and pin-related information that allows you to transfer information between the Quartus II software and your PCB schematic or design tool. For example, you can use an FPGA Xchange file to transfer pin information from the Mentor Graphics I/O Designer software to the Quartus II software to validate pin assignment changes using the I/O Assignment Analyzer.

To import an FPGA Xchange file into the Quartus II software, perform the following steps:

1. On the Assignments menu, click **Import Assignments**.
2. In the **File name** box, click **Browse** and click FPGA Xchange Files (*.fx) from the **Files of type** list.
3. Browse to and select the **FPGA Xchange** file and click **Open**.
4. Click **OK**.

To generate an FPGA Xchange file in the Quartus II software, perform the following steps:

1. Perform an I/O Assignment Analysis or a full compilation.
2. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
3. In the **Category** list in the EDA Tool Settings, select **Board-Level**. In the **Board Level Symbol Format** list, select **FPGA Xchange**.
4. Set the Output directory to the location where you want to save the file. The default output file path is `<project directory>/symbols/fpgaxchange`.
5. Click **OK**.
6. On the Processing menu, point to Start and click **Start EDA Netlist Writer**. The output directory you selected is created when you generate the FPGA Xchange file.

Pin-Out File

A Pin-Out file is an ASCII text file containing pin location results and other pin information. To generate a Pin-Out file for your project, you must successfully perform an I/O Assignment Analysis or full compilation.

Use the Pin-Out file to understand which signals should be connected to which pins. You can also use the Pin-Out file to transfer the pin information of your project into third-party PCB tools for board development. Table 5–1 describes each header of the Pin-Out file, and Figure 5–8 is an example of a Pin-Out file.

Figure 5–8. Example of a Pin-Out File

Pin Name/Usage	Location	Dir.	I/O Standard	Voltage	I/O Bank	User Assignment
VCCA_PLL1 clk	9 10	power input	LVTTL	1.5V	1	N

Table 5–1. Pin-Out File Header Description

Column Name	Description
Pin Name/Usage	The name of a design pin, ground or power
Location	The pin number of the location on the device package
Dir	The direction of the pin
I/O Standard	The name of the I/O standard to which the pin is configured
Voltage	The voltage level that is required to be connected to this pin
I/O Bank	The I/O bank number that the pin belongs to
User Assignment	Y or N indicating if the location assignment for the design pin was user assigned (Y) or assigned by the Fitter (N)



For more information about the Pin Name/Usage, refer to the Device Pin-Out for the targeted device, available on the Altera web site at www.altera.com.



For more information about using Cadence PCB tools with the Quartus II software, refer to the *Cadence PCB Design Tools Support* chapter in volume 2 of the *Quartus II Handbook*. For more information about using the Mentor Graphics PCB tools with the Quartus II software, refer to the *Mentor Graphics PCB Design Tools Support* chapter in volume 2 of the *Quartus II Handbook*.

Creating Pin-Related Assignments

A pin-related assignment is any assignment applied to a pin. An example of a pin-related assignment is a pin location assignment that assigns a design pin to a pin number/location on the targeted device. Other pin-related assignments include assigning an I/O standard or current drive strength to a pin.

You can make pin-related assignments at any time during the design cycle, even before any design files have been developed. The accuracy and completeness of the pin-related assignments determines the accuracy of the I/O assignment analysis. If you do not have design files, create reserved pins to temporarily represent your top-level design I/O pins until the I/O pins are defined in your design files. If you do not have design files in your project, create an empty Verilog HDL or VHDL file with all the ports of the design defined.

Reserved pins are pins that you reserve for future use but that do not currently perform a function in your design. Reserved pins require a unique pin name and a pin location. Using reserved pins as place holders for future design pins increases the accuracy of the I/O assignment analysis.

The Quartus II software offers many tools and features for creating reserved pins and other pin-related assignments (Table 5–2). Each tool and feature is described in more detail in the following sections.

Table 5–2. Overview of Quartus II Tools & Features Used to Create Pin-Related Assignments (Part 1 of 2)

Feature	Overview
Pin Planner	<ul style="list-style-type: none"> ● Make pin location assignments to one or more node names by dragging and dropping unassigned pins into the package view ● Edit pin location assignments for one or more node names by dragging and dropping groups of pins within the package view ● Visually analyze pin resources in the package view ● Display I/O banks and VREF groups ● View the function of package pins using the pin legend ● Make correct pin location decisions by referring to the Pads view ● Create, import, and edit megafunctions and IP MegaCores for early I/O planning ● Generate a top-level wrapper file without design files based on early I/O assignments ● Configure board trace models of selected pins for use in “board-aware” signal integrity reports generated with the Enable Advanced I/O Timing option
Assignment Editor	<ul style="list-style-type: none"> ● Create and edit all types of pin-related assignments ● Create and edit multiple assignments simultaneously with the Edit bar ● Create pin assignments efficiently by viewing the different font styles used to display assigned and unassigned node names, as well as occupied and available pin locations ● Provides user-selectable information about each pin, including the pad number, the t_{CO} requirement, and the t_H requirement

Table 5–2. Overview of Quartus II Tools & Features Used to Create Pin-Related Assignments (Part 2 of 2)

Feature	Overview
Tcl	<ul style="list-style-type: none"> • Create any pin-related assignments for multiple pins • Store and reapply all pin-related assignments with Tcl scripts • Make assignments from the command line
Timing closure Floorplan	<ul style="list-style-type: none"> • Create and change pin locations by dragging and dropping pins into the floorplan • Make correct pin location decisions by referring to the pad ID number and spacing • Display I/O banks, VREF groups, and differential pin pairing information
Synthesis Attributes	<ul style="list-style-type: none"> • Embed pin-related assignments using attributes in the design files to pass assignments to the Quartus II software

The Pin Planner is the main interface for creating and editing pin-related assignments. Use the Pin Planner package view to make pin location and other assignments using a device package view instead of pin numbers. With the Pin Planner, you can identify I/O banks, VREF groups, and differential pin pairings to help you through the I/O planning process.



For more information about using the Pin Planner, refer to [“Using the Pin Planner”](#) on page 5–21.

While the Pin Planner is the recommended tool for creating and editing pin-related assignments, you may find some of the other tools for working with pin-related assignments useful.

Assignment Editor

The Assignment Editor provides a spreadsheet-like interface that allows you to create and change all pin-related assignments.

Assigning Pin Locations Using the Assignment Editor

You can use either of two methods for making pin assignments with the Assignment Editor. The first involves selecting from all assignable pin numbers of the device and assigning a pin name from your design to this location.

The second involves selecting from all pin names in your design and assigning a device pin number to the design pin name. In either method, you can take advantage of row background coloring (pin numbers within the same I/O bank have a common background color), auto fill node names, and pin numbers to assist in making your assignments.

Setting Pin Locations From the Device Pin Number List

It is important to understand the properties of a pin location before assigning that location to a pin in your design. For example, you must know which I/O bank or VREF group the pin belongs to when following pin placement guidelines.



For more information about pin placement guidelines, refer to the appropriate device handbook.

Before creating pin-related assignments, perform analysis and elaboration or analysis and synthesis on your design to create a database of your design pin names. Then perform the following steps:

1. **To open** the Assignment Editor, on the Assignments menu, click **Assignment Editor**.
2. From the **Category list**, select **Pin**.

Creating pin assignments can be difficult when you need to check which I/O bank the pin belongs to or which VREF pad the pin uses. By selecting the **Pin** category, more pin-related information is visible in the spreadsheet to help you create pin location assignments.



The Assignment Editor does not show assignments to individual nodes made with wildcards or assignment groups.

3. On the View menu, click **Show All Assignable Pin Numbers**.



You can also view all assignable pins in the All Pins list in the Pin Planner. Right-click anywhere in the Groups or All Pins lists, and click **Show Assignable Pins**. When the All Pins list filter is set to **Pins: unassigned** or **Pins: all**, all unassigned pin15

A list of all assignable pin numbers for the targeted device is shown in the **Location** column (Figure 5–9).

Figure 5–9. Assignment Editor with Show All Assignable Pin Numbers

	To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reserved	Enabled
40		PIN_H20	2	3.3-V LVTTL	Row I/O	DIFFIO_RX14p		Yes
41		PIN_H19	2	3.3-V LVTTL	Row I/O	DIFFIO_RX14n		Yes
42		PIN_G20	2	3.3-V LVTTL	Row I/O	DIFFIO_TX14p		Yes
43		PIN_G19	2	3.3-V LVTTL	Row I/O	DIFFIO_TX14n		Yes
44		PIN_G22	2	3.3-V LVTTL	Row I/O	DIFFIO_RX13p		Yes
45		PIN_G21	2	3.3-V LVTTL	Row I/O	DIFFIO_RX13n		Yes
46		PIN_J17	2	3.3-V LVTTL	Row I/O	DIFFIO_TX13p		Yes
47		PIN_J16	2	3.3-V LVTTL	Row I/O	DIFFIO_TX13n		Yes
48		PIN_H22	2	3.3-V LVTTL	Row I/O	DIFFIO_RX12p		Yes
49		PIN_H21	2	3.3-V LVTTL	Row I/O	DIFFIO_RX12n		Yes
50		PIN_J19	2	3.3-V LVTTL	Row I/O	DIFFIO_TX12p		Yes
51		PIN_J18	2	3.3-V LVTTL	Row I/O	DIFFIO_TX12n		Yes
52		PIN_J21	2	3.3-V LVTTL	Row I/O	DIFFIO_RX11p		Yes
53		PIN_J20	2	3.3-V LVTTL	Row I/O	DIFFIO_RX11n		Yes
54		PIN_K18	2	3.3-V LVTTL	Row I/O	DIFFIO_TX11p		Yes

- Find a pin number in the spreadsheet. In the same row, double-click the cell in the **To** column. Type the pin name or select a pin from the drop-down arrow. If analysis and elaboration has been performed, your design pins are listed in the drop-down arrow.



As you type in a pin name, the Assignment Editor automatically completes the field by looking up the pin names stored in the database created from the initial analysis and elaboration. Pin names already assigned to a pin location are shown in italics.

Setting Pin Locations from the Design Signal Name List

It is important to understand the properties of a pin location before assigning that location to a pin in your design. For example, you must know which I/O bank or VREF group the pin belongs to when following pin placement guidelines.




For more information about pin placement guidelines, refer to the appropriate device handbook.

To set the pin locations from the **design pin name** list, perform the following steps:

1. To open the Assignment Editor, on the Assignments menu of the Quartus II software, click **Assignment Editor**.
2. From the **Category** list, select **Pin**.

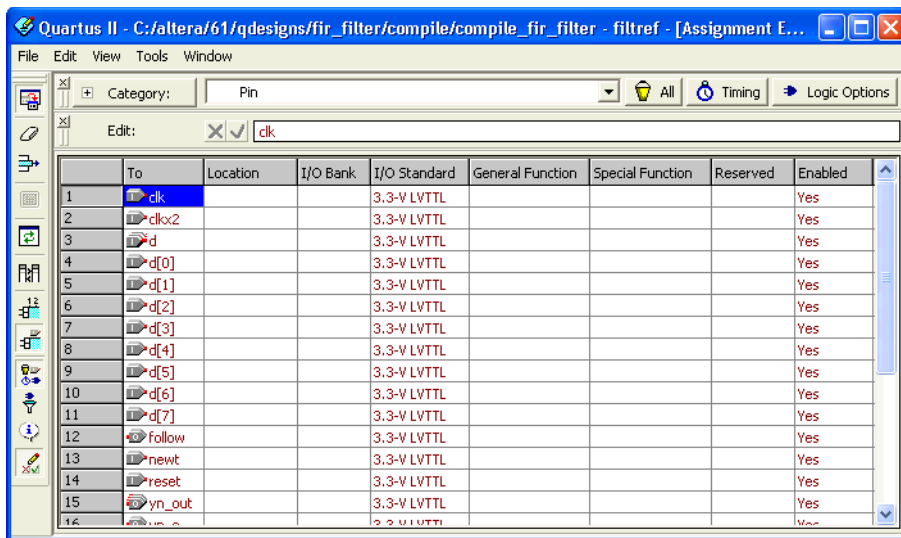
Creating pin assignments can be difficult when you have to check which I/O bank the pin belongs to, or which VREF pad the pin uses. By selecting the **Pin** category, more pin-related information is visible in the spreadsheet to help you create pin location assignments.

 The Assignment Editor does not show assignments to nodes made with wildcards or assignment groups.

3. On the View menu, click **Show All Known Pin Names**.

A list of all pin names in your design is shown in the **To** column (Figure 5–10).

Figure 5–10. Assignment Editor with Show All Known Pin Names





To list a selection of pin names from your design into the spreadsheet of the Assignment Editor, type the pin names with or without wild cards into the **Node Filter** bar. This is effective when you want to assign common pin-related assignments to a selection of pins in your design.



For more information about using the **Node Filter** bar, refer to the *Assignment Editor* chapter in volume 2 of the *Quartus II Handbook*.

4. Find a pin name in the spreadsheet, and double-click the **Location** cell in the same row. Select a pin number from the drop-down arrow which contains all assignable pin numbers in the selected device. You can also start typing the pin number and let the **Assignment Editor** automatically complete it for you. Instead of typing `PIN_AA3`, you can type `AA3` and let the Assignment Editor auto complete the pin number to `PIN_AA3`.



Pin locations that already have a pin name assignment appear in the Assignment Editor in italics.



For more information about using the Assignment Editor, refer to the *Assignment Editor* chapter in volume 2 of the *Quartus II Handbook*.

Tcl Scripts

Tcl scripting allows you to write scripts to create pin-related assignments. To run a Tcl script with your project, type the following command at a system prompt:

```
quartus_sh -t my_tcl_script.tcl ↵
```

You can also type individual Tcl commands into the Tcl console window. To use the Tcl console, on the View menu, point to Utility Windows and click **Tcl Console**. In the Tcl Console window, type your Tcl commands. The following example shows a list of Tcl commands that creates pin-related assignments to the input pin `address[10]`.

```
set_location_assignment PIN M20 -to address[10] -comment"Address pin to Second FPGA"
set_instance_assignment -name IO_STANDARD "2.5 V" -to address[10]
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to address[10]
```



For more information about using Tcl scripts to create pin-related assignments, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*.

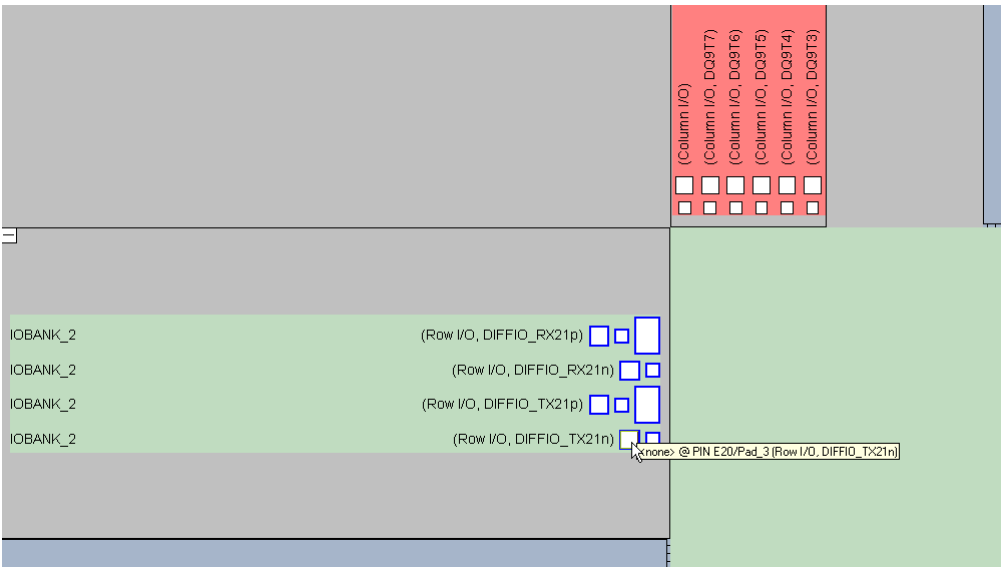
Timing Closure Floorplan

The Timing Closure Floorplan shows the pins in the same order as the pads of the device. Understanding the relative distance between a pad and related logic can help you meet your timing requirements. You can also use the Timing Closure Floorplan to find the distances between user I/O pads and V_{CC} , GND, and VREF pads to avoid signal integrity issues (Figure 5–11).



For more information about pin placement guidelines, refer to the *Selectable I/O Standards* chapter of the appropriate device handbook.

Figure 5–11. Timing Closure Floorplan of EP1C6F256I7



You can create a pin location assignment with the Timing Closure Floorplan by selecting a pin and selecting a desired location. To do this, perform the following steps:

1. On the Assignments menu, click **Timing Closure Floorplan**. The Timing Closure Floorplan opens, displaying the pin and pad layout of your selected device.
2. On the View menu, point to Utility windows, and click **Node Finder**. The **Node Finder** dialog box appears.

3. In the **Filter** list, select **Pins: all** and click **List** to see all the nodes in the design.
4. Select a node from the Nodes Found list and drag the selection into a pin location in the floorplan.



For more information about using the Timing Closure Floorplan, refer to the *Timing Closure Floorplan* chapter in volume 2 of the *Quartus II Handbook*.

Synthesis Attributes

Synthesis attributes allow you to embed assignments in your HDL code. The Quartus II software reads these synthesis attributes and translates them into assignments. The Quartus II integrated synthesis supports the `chip_pin`, `useioff`, and `altera_attribute` synthesis attributes.



For more information about integrated synthesis, refer to the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*.

For synthesis attribute support by third-party synthesis tools, contact your vendor.

chip_pin & useioff

You can use the `chip_pin` and `useioff` synthesis attributes to embed pin location and fast output/input register assignments, respectively. For all other assignments, including pin-related assignments, use the `altera_attribute` synthesis attribute as discussed in “[altera_attribute](#)” on page 5-20.

Synthesis attributes translated into assignments are stored in the database and take precedence over other assignments in the Quartus II Settings File. [Example 5-1](#) and [5-2](#) embed a location and fast input assignment into both a Verilog HDL and VHDL design file using the `chip_pin` and `useioff` synthesis attributes.

Example 5-1. Verilog HDL Example

```
input my_pin1 /* synthesis chip_pin = "C1" useioff = 1 */;
```

Example 5–2. VHDL Example

```
entity my_entity is
    port(
        my_pin1: in std_logic
    );
end my_entity;
attribute useioff : boolean;
attribute useioff of my_pin1 : signal is true;
attribute chip_pin : string;
attribute chip_pin of my_pin1 : signal is "C1";
```

altera_attribute

To create other pin-related assignments, use the `altera_attribute` attribute. The `altera_attribute` attribute is understood only by the Quartus II integrated synthesis and supports all types of instance assignments. [Example 5–3](#) and [5–4](#) use `altera_attribute` to embed the fast input register and I/O standard assignments into both a Verilog HDL and a VHDL design file.

Example 5–3. Verilog HDL Example

```
input my_pin1 /* synthesis altera_attribute = "-name FAST_INPUT_REGISTER
ON;
-name IO_STANDARD \"2.5 V\" " */ ;
```

Example 5–4. VHDL Example

```
entity my_entity is
    port(
        my_pin1: in std_logic
    );
end my_entity;

attribute altera_attribute : string;
attribute altera_attribute of my_pin1: signal is "-name FAST_INPUT_REGISTER
ON; -name IO_STANDARD \"2.5 V\"";
```



For detailed information about using synthesis attributes and their usage syntax, refer to the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*.

Using the Pin Planner

When planning your I/Os, it can be cumbersome to try to correlate pin numbers with their relative location on the package and their pin properties. The Pin Planner provides an intuitive graphical representation of the targeted device, also known as the package view, that makes it easy to plan your I/Os, create reserved pins, and make pin location assignments. When deciding on a pin location, you can use the Pin Planner to gather information about available resources, as well as the functionality of each individual pin, I/O bank, and VREF group. You can assign locations to design pins by dragging and dropping each pin into the Package view.

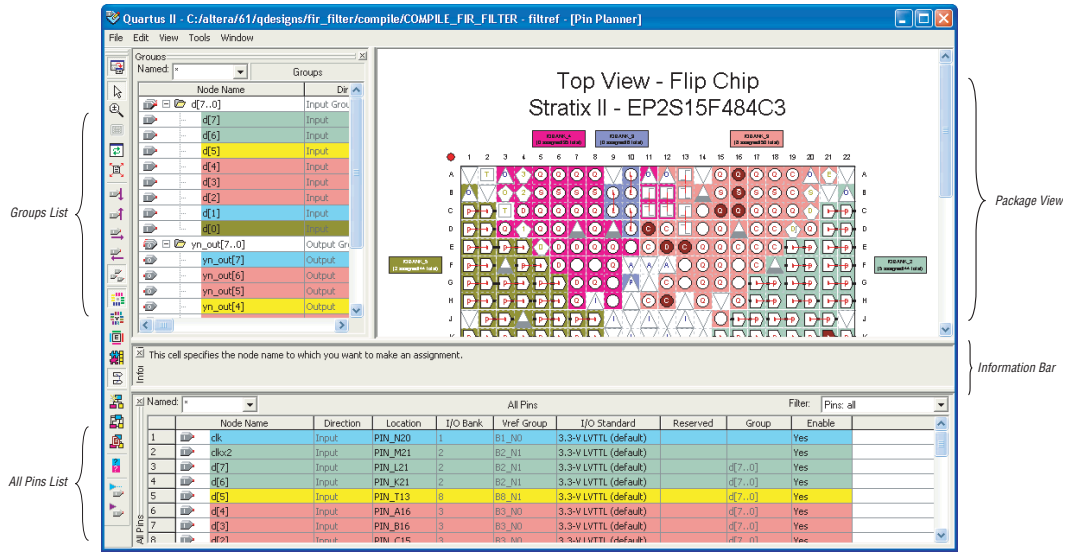


Maintaining good signal integrity (SI) requires that you follow pad distance and pin placement rules. Complementing the Pin Planner is the Pad View, which displays the pads in order around the silicon die.

The Pin Planner includes the following sections (refer to [Figures 5-12 through 5-17](#)):

- Package view
- All Pins list
- Groups list
- Info bar
- Floating Pad View window

Figure 5–12. Pin Planner



The Pin Planner feature supports cross-probing that allows you to select a pin in one view while simultaneously highlighting the pin in all of the different views. For example, if you select a pin in the package view of the Pin Planner, the corresponding pad in the Pad View window is highlighted. If the pin has an assigned node name, the node name in the **All Pins** list and the **Groups** list is highlighted.

Groups List

The **Groups** list displays all of the buses from the top-level ports of your design and all the assignment groups in your project (Figure 5–13). You can also filter the group names displayed by typing in a wild card filter into the **Named** list. The **Groups** list allows you to create your own custom groups of pins and make location assignments to groups by dragging them into the package view of the Pin Planner.



In the **Groups** list, all members of an assignment group are displayed, regardless of whether the member is a pin or an internal node.

The background color of pin locations in the Groups list easily identifies which pins belong to which I/O banks. The colors match the I/O bank colors in the Package View when **Show I/O Banks** is enabled. You can

disable the colors in both the Groups list and the All Pins list. From the Tools menu, click **Options**. In the **Category** list, select **Pin Planner**, and turn off **Show I/O bank color in node lists**.

You can create and organize custom groups and group members in the **Assignment Groups** dialog box or directly in the Groups list in the Pin Planner. To open the **Assignment Groups** dialog box, on the Assignments menu, click **Assignment (Time) Groups**.

Figure 5–13. Groups List

Node Name	Direction	Location	I/O Bank	Wref Group	I/O Standard	Reserved
d[7..0]	Input Group				3.3-V LVTTTL (default)	
d[7]	Input	PIN_L21	2	B2_N1	3.3-V LVTTTL (default)	
d[6]	Input	PIN_K21	2	B2_N1	3.3-V LVTTTL (default)	
d[5]	Input	PIN_T13	8	B8_N1	3.3-V LVTTTL (default)	
d[4]	Input	PIN_A16	3	B3_N0	3.3-V LVTTTL (default)	
d[3]	Input	PIN_B16	3	B3_N0	3.3-V LVTTTL (default)	
d[2]	Input	PIN_C15	3	B3_N0	3.3-V LVTTTL (default)	
d[1]	Input	PIN_N19	1	B1_N0	3.3-V LVTTTL (default)	
d[0]	Input	PIN_L3	2	B2_N1	3.3-V LVTTTL (default)	
yn_out[7..0]	Output Group				3.3-V LVTTTL (default)	
yn_out[7]	Output	PIN_N21	1	B1_N0	3.3-V LVTTTL (default)	
yn_out[6]	Output	PIN_D11	3	B3_N0	3.3-V LVTTTL (default)	
yn_out[5]	Output	PIN_E12	3	B3_N0	3.3-V LVTTTL (default)	
yn_out[4]	Output	PIN_U12	8	B8_N1	3.3-V LVTTTL (default)	
yn_out[3]	Output	PIN_H12	3	B3_N0	3.3-V LVTTTL (default)	

To add a new group to the **Groups** list without opening the **Assignment Groups** dialog box, perform the following steps:

1. In the **Node Name** column, double-click **new** in the **Groups** list.
2. Type the group name.
3. Press **Enter**. The **Add Members** dialog box appears.
4. Type node names, wild cards, and assignment groups in the **Members** box, or browse to and select the node names from the **Node Finder** dialog box.
5. Click **OK**.



For more information about using Assignment Groups, refer to the *Assignment Editor* chapter in volume 2 of the *Quartus II Handbook*.

You can also create a new group by selecting one or more node names within the **Groups** list or **All Pins** list. Right-click one of the selected node names, and click **Add to Group** on the shortcut (right-click) menu.

As you plan your I/O placement, you may decide to add and remove members from a group.


To add a member to a custom group in the **Groups** list without opening the **Assignment Groups** dialog box, perform the following steps:

1. Right-click a group name in the **Groups** list and click **Add Members**.
2. Type in the name of the member or click **browse** to select one or more nodes from the **Node Finder** dialog box.

To remove a member from a group in the **Groups** list, perform the following steps:

1. Expand the group from which you want to remove a member.
2. Select one or more members that you want to remove.
3. Right-click the selected members, point to Edit and click **Delete**.

The **Groups** list provides many columns, some for information purposes and others to make assignments. The only cells you can edit in addition to those in the **Node Name** column are Location, I/O Standard, Reserved cells, and Enable. You can make changes to any of the cells in these columns to adjust pin-related assignments. The other columns provide useful information during I/O planning, including the I/O Bank number, VREF group, and the direction. To show or hide a column, right-click the column and click **Customize Columns**. You can also reorder and sort the columns from this menu.

 If an assignment group contains pins with different directions, the direction of the assignment group is a `bidir` group.

You can edit the columns in the Groups list in the same manner as a spreadsheet. You can copy and paste the Location, I/O Standard, and Reserved assignments to other rows in the list within the same column. You can also use Auto Fill to copy these assignments to other rows quickly. To automatically fill a block of rows, set the desired assignment in one row and select the assignment's cell. Place the cursor over the lower right-hand corner of the cell until it changes to a cross with the word FILL (Figure 5-14). Click and drag up or down the column to select

which cells to fill. When all the desired cells are selected, release the mouse button. The selected assignment is copied to all of the selected cells.

Figure 5–14. Auto Fill the Groups List

Node Name	Direction	Location	I/O Bank	Vref Group	I/O Standard	Reserved
d[7..0]	Input Group				3.3-V LVTTTL (default)	
d[7]	Input	PIN_L21	2	B2_N1	2.5 V	
d[6]	Input	PIN_K21	2	B2_N1	3.3-V LVTTTL (default)	
d[5]	Input	PIN_T13	8	B8_N1	3.3-V LVTTTL (default)	
d[4]	Input	PIN_A16	3	B3_N0	3.3-V LVTTTL (default)	
d[3]	Input	PIN_B16	3	B3_N0	3.3-V LVTTTL (default)	
d[2]	Input	PIN_C15	3	B3_N0	3.3-V LVTTTL (default)	
d[1]	Input	PIN_N19	1	B1_N0	3.3-V LVTTTL (default)	FULL
d[0]	Input	PIN_L3	2	B2_N1	3.3-V LVTTTL (default)	
yn_out[7..0]	Output Group				3.3-V LVTTTL (default)	

All Pins List

The **All Pins** list displays all of the pins in your design, including user-created pins (Figure 5–15). The **All Pins** list does not display buses; instead, it displays each individual pin of the bus. The background color of pin locations in the **All Pins** list easily identifies which pins belong to which I/O banks. The colors match the I/O bank colors in the Package View when **Show I/O Banks** is turned on. You can turn off the colors in the **All Pins** list and the **Groups** list. On the **Tools** menu, click **Options**. In the **Category** list, select **Pin Planner**, and turn off **Show I/O bank color in node lists**.

You must perform **Analysis** and **Elaboration** successfully to display pins in your design in the **All Pins** list. Individual user-reserved pins and nodes with pin-related assignments are always shown in the **All Pins** list.

Figure 5–15. All Pins List

Node Name	Direction	Location	I/O Bank	Vref Group	I/O Standard	Reserved	Group	Enable
clk	Input	PIN_M20	1	B1_N0	3.3-V LVTTTL (default)			Yes
clkx2	Input	PIN_M21	2	B2_N1	3.3-V LVTTTL (default)			Yes
d[7]	Input	PIN_L21	2	B2_N1	2.5 V		d[7..0]	Yes
d[6]	Input	PIN_K21	2	B2_N1	3.3-V LVTTTL (default)		d[7..0]	Yes
d[5]	Input	PIN_T13	8	B8_N1	3.3-V LVTTTL (default)		d[7..0]	Yes
d[4]	Input	PIN_A16	3	B3_N0	3.3-V LVTTTL (default)		d[7..0]	Yes
d[3]	Input	PIN_B16	3	B3_N0	3.3-V LVTTTL (default)		d[7..0]	Yes
d[2]	Input	PIN_C15	3	B3_N0	3.3-V LVTTTL (default)		d[7..0]	Yes
d[1]	Input	PIN_M19	1	B1_N0	3.3-V LVTTTL (default)		d[7..0]	Yes
d[0]	Input	PIN_L3	2	B2_N1	3.3-V LVTTTL (default)		d[7..0]	Yes
follow	Output	PIN_L20	2	B2_N1	3.3-V LVTTTL (default)			Yes
newt	Input	PIN_M20	2	B2_N1	3.3-V LVTTTL (default)			Yes
reset	Input	PIN_M2			3.3-V LVTTTL (default)			Yes
yn_out[7]	Output	PIN_M21	1	B1_N0	3.3-V LVTTTL (default)		yn_out[7..0]	Yes
yn_out[6]	Output	PIN_D11	3	B3_N0	3.3-V LVTTTL (default)		yn_out[7..0]	Yes
yn_out[5]	Output	PIN_E12	3	B3_N0	3.3-V LVTTTL (default)		yn_out[7..0]	Yes
yn_out[4]	Output	PIN_U12	8	B8_N1	3.3-V LVTTTL (default)		yn_out[7..0]	Yes
yn_out[3]	Output	PIN_H12	3	B3_N0	3.3-V LVTTTL (default)		yn_out[7..0]	Yes
yn_out[2]	Output	PIN_W13	8	B8_N1	3.3-V LVTTTL (default)		yn_out[7..0]	Yes

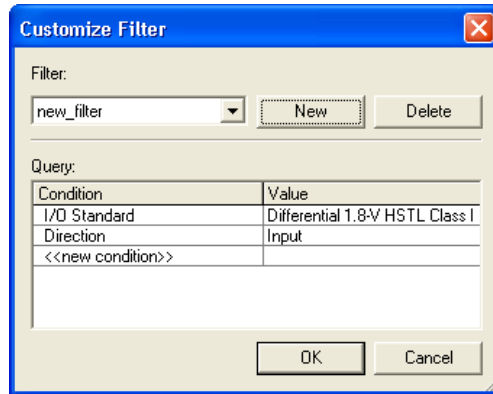
You can filter the list of pins in the **All Pins** list based on their node names by typing in a portion of the pin name in combination with wild card characters in the **Named** list. You can also filter the list of pins in the **All Pins** list based on the pins' attributes by selecting from the **Filter** list.

You can also create your own custom filter in the **Filter** list, where you can specify a set of conditions from the following list:

- Assigned or unassigned
- Current strength
- Direction
- Edge location
- I/O Bank location
- I/O Standard
- VREF Group

To create a new filter in the **All Pins** list, select <<new filter>> from the **Filter** list in the **All Pins** list. The **Customize Filter** dialog box appears (Figure 5–16).

Figure 5–16. Customized Filter Dialog Box



To create a custom filter for the **All Pins** list, perform the following steps:

1. In the **Customize Filter** dialog box, click **New**. The **New Filter** dialog box appears.
2. Enter the name of your custom filter in the **Filter name** text box.
3. You can base your new custom filter on existing filters by selecting from the **Based on Filter** list. If you do not want to base your custom filter on any other filter, select **Pins: all** from the **Based on Filter** list.
4. Click **OK**.
5. Add as many conditions as you require to the **Query** list. To add a condition, double-click **new condition** and select a condition from the **Condition** list. Select a value for the condition by double-clicking the cell next to your selected condition under the **Value** column.



To remove a condition from your filter, right-click the condition in the **Query** list and select **Delete**.

After specifying your conditions, the pins meeting the specified conditions are the only pins shown in the **All Pins** list. If the set of conditions contains a condition with more than one value, then the pins displayed must meet at least one of the values for that multiple-value condition.

To edit an existing custom filter, select <<*new filter*>> from the **Filter** list in the **All Pins** list. In the **Customize Filter** dialog box, select the custom filter you want to edit from the **Filter** list and add and remove conditions to the **Query** list.

Pins generated from a compilation or from a bus group are not editable. All other user-created pins are editable.

The **All Pins** list provides many columns, some for information purposes and others to make assignments. To show or hide a column, right-click the column heading and select **Customize Columns**. In addition, you can reorder and sort the columns from this menu.

You can edit the columns in the Groups list in the same manner as a spreadsheet. You can copy and paste assignments to other rows in the list within the same column. You can also use Auto Fill to quickly copy these assignments to other rows. To automatically fill a block of rows, set the desired assignment in one row and select the assignment's cell. Place the cursor over the lower right-hand corner of the cell until it changes to a cross with the word FILL as shown with the Groups list in [Figure 5-14 on page 5-25](#). Click and drag up or down the column to select which cells to fill. When all the desired cells are selected, release the mouse button. The selected assignment is copied to all the selected cells.

Pad View

To maintain good signal integrity in designs, use the Pad View to guide your pin placement decisions. Each device family is accompanied with pin placement rules, including pad spacing between various pin types.



For more information about pin placement rules, refer to the appropriate device handbook.

You can edit or make pin assignments in the Pad View by dragging and dropping a design pin into an available pad location.

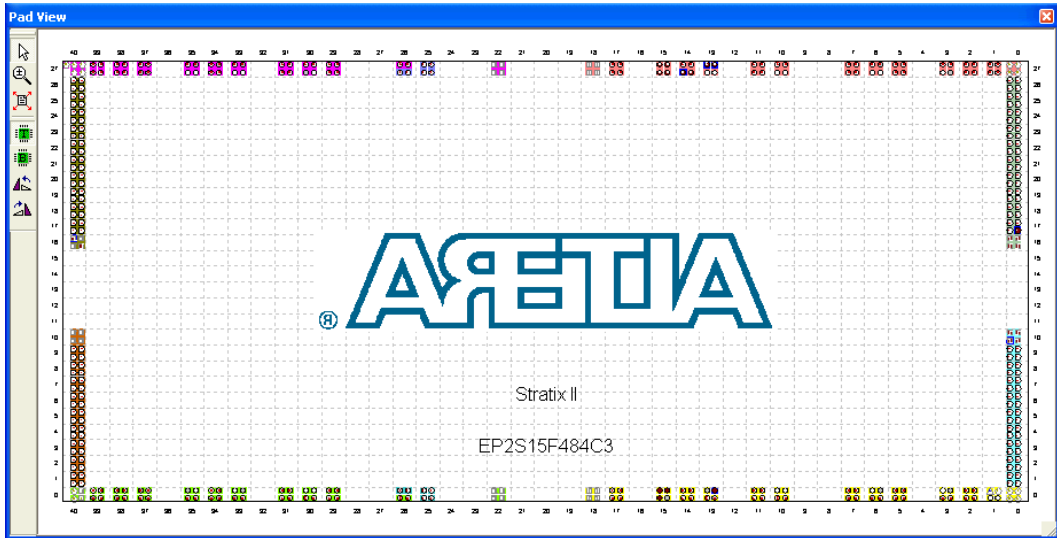
When you drag and drop a design pin into an available pad location, the corresponding pin number of the pad is assigned to the design pin. To assign a pad number to the design pin, perform the following steps:

1. On the Tools menu, click **Options**. The **Options** dialog box appears.
2. Click **Pin Planner** and turn on **Create pad assignment** in the Pad View window.

The column and row numbering around the Pad View helps identify which pad row or pad column each pad is located. This is useful when the pin placement guidelines for your targeted device refer to pad rows and columns.

Since the pad view is a view of the I/O ring of the silicon within the package, flip-chip packages appear inverted. Notice the reversed ALTERA logo in [Figure 5-17](#). To understand the correlation between the package pins and the pads on the silicon die, the Pad View window and Package View are closely integrated together. When a pad is selected, the corresponding pin in the package view is highlighted. This is also true when a pin is selected in the package view, the corresponding pad is highlighted in the Pad View window.

Figure 5-17. Pad View of a Stratix II Flip-Chip Device



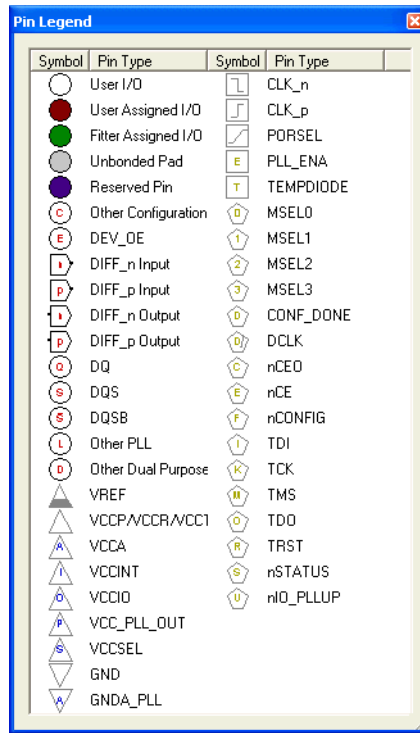
Package View

The package view in the Pin Planner uses annotated pin symbols in different shapes and colors as visual representations of pins of the actual package ([Figure 5-12 on page 5-22](#)). The package view eliminates the need to cross-reference each pin number with its physical location on the package described in the device package datasheet. When making pin location assignments in the Package view, switch between the different views to help you decide on a pin location. The different views in the Package view include I/O banks, VREF groups, Edges, DQ/DQS pins,

and differential pin pairs. For more information about the different views in the Package view, refer to the section in this chapter about the specific view you want to use.

The Pin Legend window provides a quick reference to the meanings of the pin symbol shapes, notations, and colors in the Package view. To view the Pin Legend window, on the View menu, click **Pin Legend** (Figure 5–18). You can also open the Pin Legend from the Pin Planner toolbar or from the right-click menu in the Package view.

Figure 5–18. Pin Legend Window

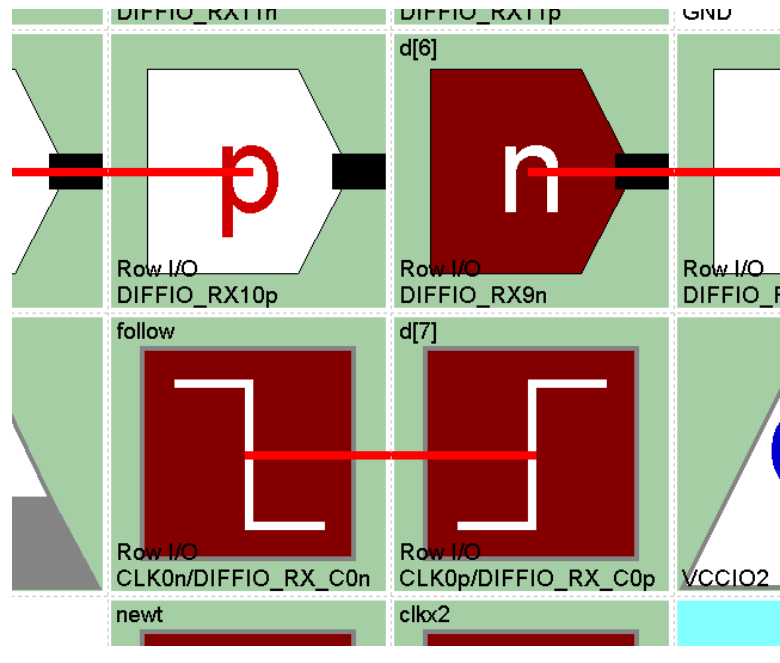


Planning your FPGA I/O assignments with your board design is necessary in today's market. If your FPGA device is oriented differently than how it appears in the package and pad view of the pin planner, rotate the package view. To rotate the package view, on the View menu, point to Show and click **Rotate Left 90°** or **Rotate Right 90°** until your FPGA is shown in the desired orientation in the Package view. The red

dot in the package view indicates the location of the first pin. For example, the red circle identifies where Pin A1 is located on a BGA package and where Pin 1 is located on a TQFP package.

You can also print the package view with the pin names and pin types visible (Figure 5–19). To show the pin name (if available) or pin type for each pin in the package view, on the View menu, click **Show Node Names** and **Show Pin Types**.

Figure 5–19. Package View with Show Node Names & Show Pin Types



To view pin resource usage, on the View menu, click **Resources Window**. The **Resources** dialog box appears (Figure 5–20).


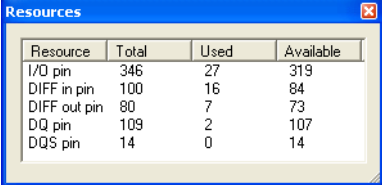
 For more detailed information about resources, view the Resource section of the Compilation Report.

Figure 5–20. Resources Window


Resource	Total	Used	Available
I/O pin	346	27	319
DIFF in pin	100	16	84
DIFF out pin	80	7	73
DQ pin	109	2	107
DQS pin	14	0	14

If a HardCopy® II companion device is selected, the Pin Planner shows the package view for the Stratix II device. To ensure correct pin migration between the Stratix II and HardCopy II devices, run the I/O Assignment Analysis command or the Fitter.

If a migration device is selected, the Pin Planner shows only pins that are available for migration. Selecting a migration device allows you to either vertically migrate to a different density while using the same package, or migrate between packages with different densities and ball counts.



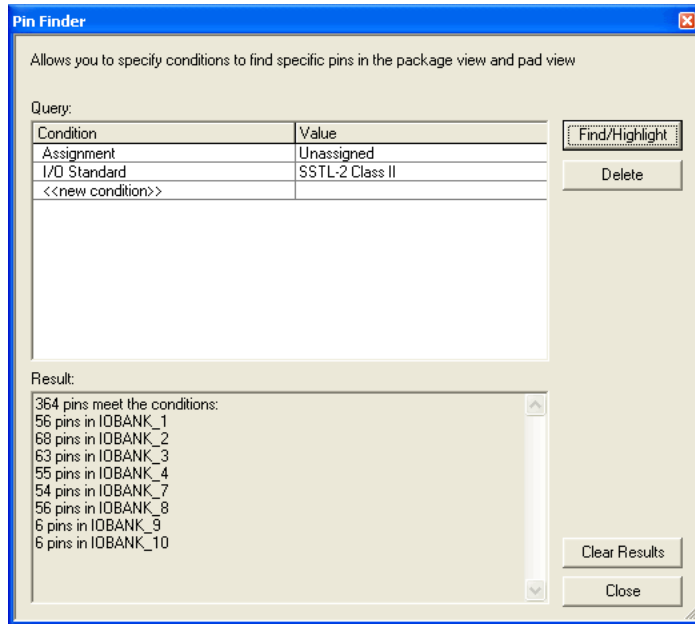
For more information about migration, refer to the Altera application note, *AN90: SameFrame Pin-Out Design for FineLine BGA Packages*. For more information about designing for HardCopy II devices, refer to the *Quartus II Support of HardCopy Series Devices* chapter in volume 1 of the *Quartus II Handbook*.

Using the Pin Finder to Find Compatible Pin Locations

As FPGA pin-counts and I/O capabilities continue to increase, it becomes more difficult to understand the capabilities of each I/O and to correctly assign your design I/Os. To help you address this problem, the Pin Planner highlights all pins that match the list of conditions that you enter. To enter your conditions, perform the following steps with the Pin Planner open:

1. On the View menu, click **Pin Finder**. The Pin Finder window appears (Figure 5–21).

Figure 5–21. Pin Finder Window



- In the Pin Finder window, create a list of conditions in the **Query** list.

To add a condition to the **Query** list, double-click <<new condition>>, and select a condition from the list. Double-click the cell next to the new condition and select a desired value. For example, if you want to highlight all available pins that support the SSTL-2 Class II I/O standard, create an assignment condition and an I/O standard condition as shown in [Figure 5–21](#).

If you add the same condition type more than once, the Pin Finder searches for results that match any of the specified values. If you add more than one condition type, the Pin Finder searches for results that match all of the specified conditions.

- In the Pin Finder window, click **Find/Highlight**. All of the pins that meet the specified conditions are highlighted in the Package view and in the Pad View window.

At the same time, the **Results** list in the Pin Finder window displays a summary of the number of pins in each I/O Bank that meet the specified conditions.

Creating Reserved Pin Assignments

You can make reserved pin assignments to act as place holders for future design pins in the Package view or in the **All Pins** list. To create a reserved pin in the Pin Planner Package view, perform the following steps:

1. In the Package View, right-click an available pin.
2. On the shortcut (right-click) menu, point to Reserve, and click one of the available configurations.

When you reserve a pin from the Package view, the name of the reserved pin is set to `user_reserve_<number>` by default, and the pin symbol is filled with a dark purple color. The number increments by 1 for each additional reserved pin.

Alternately, you can reserve a pin in the **All Pins** list by performing the following steps:

1. Type the pin name into an empty cell in the **Node Name** column. The pin name must not already exist in your design.
2. Select a pin configuration from the **Reserved** list (Figure 5–22).

The following configurations are available as:

- bidirectional
- input tri-stated
- output driving an unspecified signal
- output driving ground
- output driving VCC
- SignalProbe output

Figure 5–22. Reserving a Pin in the All Pins List

Node Name	Direction	Location	I/O Bank	Vref Group	I/O Standard	Reserved	Group	Enable
d[2]	Input	PIN_C15	3	B3_NO	3.3-V LVTTTL (default)		d[7..0]	Yes
d[1]	Input	PIN_N19	1	B1_NO	3.3-V LVTTTL (default)		d[7..0]	Yes
d[0]	Input	PIN_L3	7	B7_NO	3.3-V LVTTTL (default)			Yes
follow	Output	PIN_L20	2	B2_N1	3.3-V LVTTTL (default)			Yes
my_reserved_pin	Input	PIN_Y3	7	B7_N1	3.3-V LVTTTL (default)			Yes
newt	Input	PIN_M20	2	B2_N1	3.3-V LVTTTL (default)			Yes
reset	Input	PIN_M2	7	B7_NO	3.3-V LVTTTL (default)			Yes
yn_out[7]	Output	PIN_N21	1	B1_NO	3.3-V LVTTTL (default)			Yes
yn_out[6]	Output	PIN_D11	3	B3_NO	3.3-V LVTTTL (default)			Yes
yn_out[5]	Output	PIN_E12	3	B3_NO	3.3-V LVTTTL (default)			Yes
yn_out[4]	Output	PIN_U12	8	B8_N1	3.3-V LVTTTL (default)			Yes
yn_out[3]	Output	PIN_H12	3	B3_NO	3.3-V LVTTTL (default)		yn_out[7..0]	Yes

Release reserved pins by selecting the blank entry from the **Reserved** list.



The **Direction** column is a read-only column and changes direction depending on the reserved selection.

Creating Pin Location Assignments

You can create pin locations assignments for one or more pins with the following methods:

- Assigning a location for unassigned pins
- Assigning a location for differential pins
- Assigning an unassigned pin to a pin location

You can disable or prevent any of these assignments using the Enable column in either the Groups list or the All Pins list. The Enable column is a special column you can use to disable only the location assignment for a selected pin. You can change the value of the Enable cell for a selected pin from **Yes** to **No** by double-clicking the cell and selecting **No** from the list. A disabled pin only prevents location assignments when signals are assigned using drag and drop as described below. You can still make assignments directly in the Location columns in both the Groups and All Pins lists. To enable the location assignment again, change the Enable cell back to Yes.

Assigning Locations for Unassigned Pins

To assign locations for all of your design pins, perform the following steps:


1. On the Edit menu, select an assignment direction.

You can assign several pins simultaneously by choosing an assignment direction (Table 5-3). When assigning an entire bus, assignments are made in order from the most significant bit to the least significant bit.

<i>Table 5-3. Multiple Pins (Part 1 of 2)</i>	
Assignment	Pin Group
Assign Down	From the selected group of unassigned pins, assign each pin downwards starting from the selected pin.
Assign Up	From the selected group of unassigned pins, assign each pin upwards starting from the selected pin.
Assign Right	From the selected group of unassigned pins, assign each pin to the right of each other starting from the selected pin.

Table 5–3. Multiple Pins (Part 2 of 2)

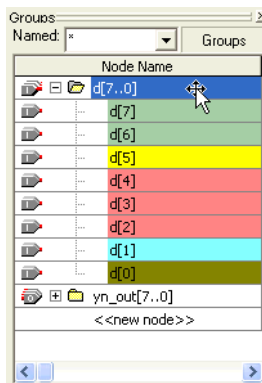
Assignment	Pin Group
Assign Left	From the selected group of unassigned pins, assign each pin to the left of each other starting from the selected pin.
Assign One by One	Select a pin location for each of the pins selected from the Unassigned Pins list.

 If there is an unassignable location in the path of the selected assignment direction, pins are assigned as far in the assignment direction as possible. Assign the rest of the pins in a separate location.

- In the **Filter** list, select **Pins: unassigned**.
- In the **All Pins** list, select one or more unassigned node names, or in the **Groups** list, select one or more buses.

You can click on multiple node names using the control and shift keys. When you click on a pin or bus in the **All Pins** or **Groups** list, the node name is highlighted, and a crossing arrow displays above the cursor. Drag the selected cells into the package view (Figure 5–23).

Figure 5–23. Drag Node Name in the Groups List



- Drag and drop the selected pins or buses from the **All Pins** or **Groups** list to a location in the package view.

Before you drag and drop your pins, you can optionally use the Pin Finder to locate pin locations that support your selected pins. When creating a query in the Pin Finder, add an Assignment condition and set it to **Unassigned**.

If you don't use the Pin Finder, you can drop pins directly into any of the following locations in the Pin Planner package view: an available user I/O pin, I/O Bank, VREF Group, or Edge. On the View menu, you can display either I/O banks, VREF groups, or edges by going to the Show submenu and toggling between **Show I/O Banks**, **Show VREF Groups**, and **Show Edges**. You can also toggle between these views from the Pin Planner toolbar or from the right-click menu in the Package View.

Available single-ended user I/O pins are represented by empty circles in the package view. The letter inside the circle provides information about the user I/O pin. Negative and positive differential pins are shaped like hexagons and contain the letters “n” and “p”, respectively. For a complete listing of I/O pin shapes, notations, and colors, open the Pin Legend window from the View menu, toolbar, or Package View right-click menu.

In the Pin Planner Package View, I/O banks are displayed as rectangles labeled `IOBANK_<number>` (Figure 5-27). In each I/O bank, there are one or more VREF groups. VREF groups are displayed as rectangles labeled `VREFGROUP_B<I/O Bank number>_N<index>` (Figure 5-29).

Edge locations are displayed as rectangles labeled `EDGE_<direction>`. To make an edge assignment, drag and drop pins into one of the four edges, `EDGE_TOP`, `EDGE_BOTTOM`, `EDGE_LEFT`, or `EDGE_RIGHT`.



You can also drag and drop pins from the **Node Finder** dialog box or from the Block Diagram/Schematic File into the package view.

Assigning a Location for Differential Pins

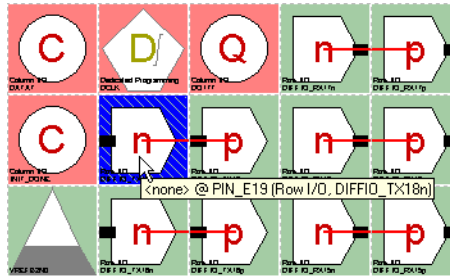
To identify and assign differential pins using the Pin Planner, perform the following steps:

1. On the View menu, click **Show Differential Pin Pair Connections**.

When you select **Show Differential Pin Pair Connections**, a red line connects the positive and negative pins of the differential pin pairing. The positive and negative pins are labeled in the package view with the letters “p” and “n”, respectively (Figure 5-24).

- Use the tool tips to identify LVDS-compatible pin locations by holding the mouse pointer over a differential pin in the package view (Figure 5–24).

Figure 5–24. Tool Tip of the Positive Differential Pin



The tool tip shows the design pin name and pin number, as well as its general and special functions.


The tool tip for differential receiver and transmitter channel pins that are also available as user I/O is shown in the following format:

<design pin name> @ PIN_*<Package Pin Number>* (*<Row | Column>* I/O, DIFFIO_*<RX/TX><differential pin pair number><p | n>*)


The tool tip for dual-purpose LVDS I/O channel pins is shown in the following format:

<design pin name> @ PIN_*<Package Pin Number>* (*<Row | Column>* I/O, LVDS*<differential pin pair number><p | n>*)

- Click on the differential pin from the **All Pins** or **Groups** list.
- Drag and drop the selected pin from the **All Pins** or **Groups** list to a differential positive pin location in the package view.

 Optionally, before you drag and drop your pins, you can use the Pin Finder to locate pin locations that support your selected pins. When creating a query in the Pin Finder, add an assignment condition set to **Unassigned** and an I/O standard condition set to your differential I/O standard.

The unassigned differential pin that you drag to the package view represents the positive pin of the differential pair. The Fitter automatically recognizes the negative pin of the differential pair and creates it in the Pin-Out file.

 If you assign a differential pin to a pin location, the negative pin becomes unassignable. The Quartus II software recognizes the negative pin as part of the differential pin pair assignment. However, the assignment is not entered in the QSF.

If you have a single-ended clock that feeds a PLL, assign the pin only to the positive clock pin in the targeted device. Single-ended pins, that feed a PLL and are assigned to the negative clock pin in the targeted device cause the design to fail to fit.



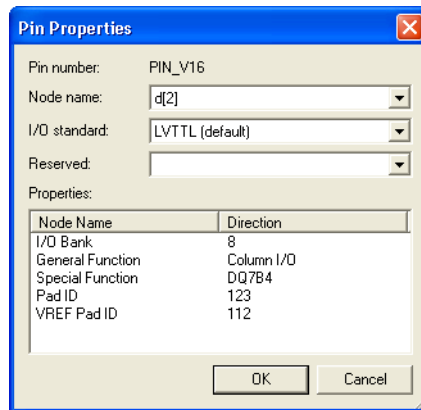
For more information about the general and special functions displayed by the tool tip, refer to the Device Pin-Outs available at www.altera.com.

Assigning an Unassigned Pin to a Pin Location

Use the following steps to select a pin location and assign a design pin to that location:

1. In the package view, select an available pin location.
2. On the View menu, click **Pin Properties**. The **Pin Properties** dialog box appears (Figure 5–25).

Figure 5–25. Pin Properties Dialog Box



You can use the **Pin Properties** dialog box to create pin location and I/O standard assignments. The **Pin Properties** dialog box also displays the properties of the pin location, including the pad ID (Table 5–4). The pad ID is important information when following pin spacing guidelines. Adjacent pin numbers do not always represent adjacent pads on the die. Use the Pads view to help correlate pad location and distance between your user I/O pins and VREF pins.

3. Select a pin from the **Node Name** list.
4. To assign or change the I/O standard, select an I/O standard from the **I/O standard** list.
5. Click **OK**.



For more information about pin placement, refer to the appropriate device handbook.

Table 5–4 provides a description of each field in the **Pin Properties** dialog box.

Table 5–4. Pin Properties	
Pin Property	Description
Pin Number	Pin number used in package (1)
Node Name	Node name assigned to the pin location
I/O Standard	I/O standard assigned to the pin name and location
Reserved	If reserved, determines how to reserve this pin
I/O Bank	I/O bank number of the pin
General Function	General function of the pin (row/column I/O, dedicated clock pin VCC, GND)
Special Function	Special function of the pin (LVDS, PLL)
Pad ID	Pad number connected to pin
VREF Pad ID	The pad ID for the VREF pin used for voltage referenced I/O standards

Note to Table 5–4:

- (1) For more information about how pin numbers are derived, refer to the device pin-out on the Altera web site, www.altera.com.



You can also open the **Pin Properties** dialog box by double-clicking on a pin in the package view of the Pin Planner, or by right-clicking the pin in the package view of the Pin Planner, and clicking **Pin Properties**.

Error Checking Capability

The Pin Planner has basic pin placement checking capability, preventing pin placements that violate the fitting rules. The following checks are performed by the Pin Planner as you make pin-related assignments:

- An I/O Bank or VREF Group is an unassignable location if there are no available pins in the I/O bank or VREF group.
- The negative pin of a differential pair is unassignable if the positive pin of the differential pair has been assigned with a node name with a differential I/O standard.
- Dedicated input pins (for example, dedicated clock pins) are an unassignable location if you attempt to assign an output or bidirectional node name.
- Pin locations that do not support the I/O standard assigned to the selected node name become unassignable.
- All nodes in the same VREF group must have the same VREF voltage. Apply this only to HSTL- and SSTL-type I/O standards.



To perform a more comprehensive check on your pin placements, perform I/O assignment analysis.



For more information about assignment analysis, refer to [“Using I/O Assignment Analysis to Validate Pin Assignments”](#) on page 5–58.

After creating a pin location, the **Location**, **I/O Bank**, and **VREF Group** fields are populated in both the **All Pins** list and the **Groups** list. In the package view, the occupied pins are filled with a dark brown color.

Changing Pin Locations

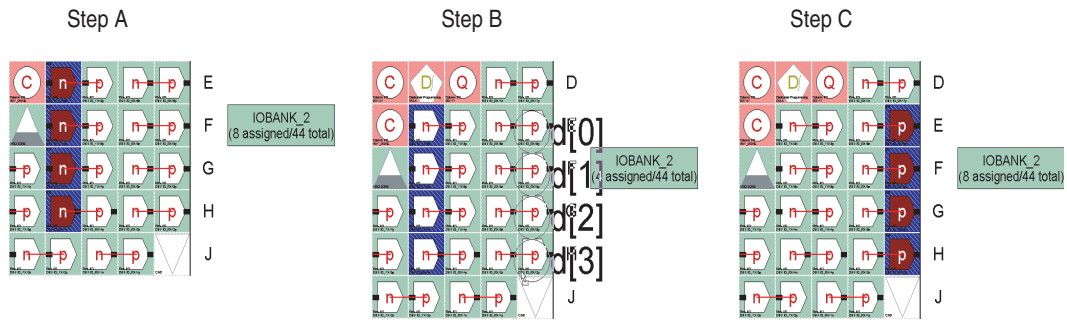
The Pin Planner allows you to change the location of multiple pins simultaneously. To change pin locations, select one or more pins in the package view or pad view, and drag the pins to a new location.

It is quick and easy to change pin locations if you understand which user I/O pins are available and where they are, physically, on the device. For example, in the package view, you can move a column of pins closer to the edge of the device for easier PCB routing ([Figure 5–26](#)). In this example, you are moving multiple I/O pins to the area closest to the edge of the I/O bank. To change pin locations, perform the following steps:

1. In the Package view, select multiple pins by holding down the left mouse button and dragging over the pins you want to move ([Figure 5–26](#), step A).
2. Drag the group of pins to the area of placement ([Figure 5–26](#), step B).

- Drop the pins into the area closest to the edge of the I/O bank (Figure 5–26, step C).

Figure 5–26. Changing the Locations for a Group of Pins



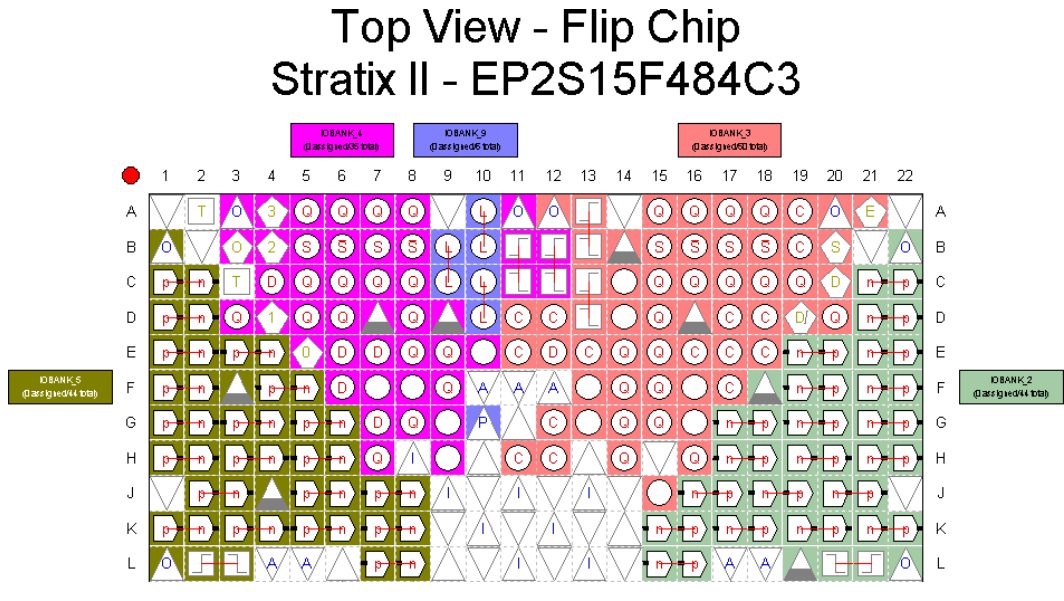
Show I/O Banks

When **Show I/O Banks** is turned on in the View menu, in the Show submenu, or in the shortcut (right-click) menu in the Package view, the Package view groups I/O pins that share the same `VCCIO` pin using different colors (Figure 5–27). When planning your I/O pins, it is important to guide your pin placement decisions by placing pins with compatible I/O standards into the same I/O bank. For example, you cannot place an LVTTTL pin with an I/O standard of LVTTTL in the same bank as another pin with an I/O standard of 1.5 V HSTL Class I.

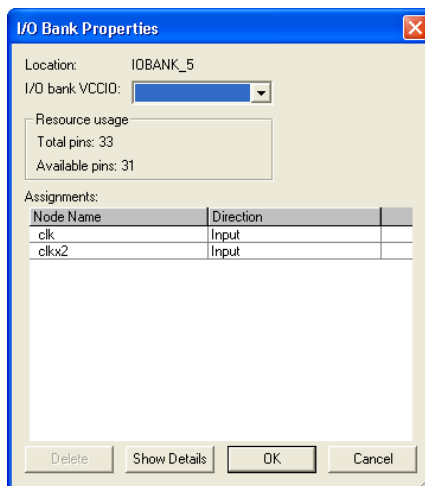


For more information about compatible I/O standards, refer to the appropriate device handbook.

Figure 5–27. Package View with I/O Banks



When **Show I/O Banks** is turned on, the package view allows you to view the properties of each I/O bank. Select an I/O Bank in the package view. On the View menu, click **I/O Bank Properties**. The **I/O Bank Properties** dialog box appears (Figure 5–28). The **I/O Bank Properties** dialog box lists all node names assigned to that I/O Bank. To view all node names that are assigned within the I/O Bank, click **Show Details** in the **I/O Bank Properties** dialog box. You can also assign the VCCIO for the I/O Bank by selecting a voltage from the **I/O bank VCCIO** list.

Figure 5–28. I/O Bank Properties

The Resource Usage section of the dialog box shows the total number of pins in the I/O Banks including assignable and unassignable pins, as well as the total number of available assignable pins.

Show VREF Groups

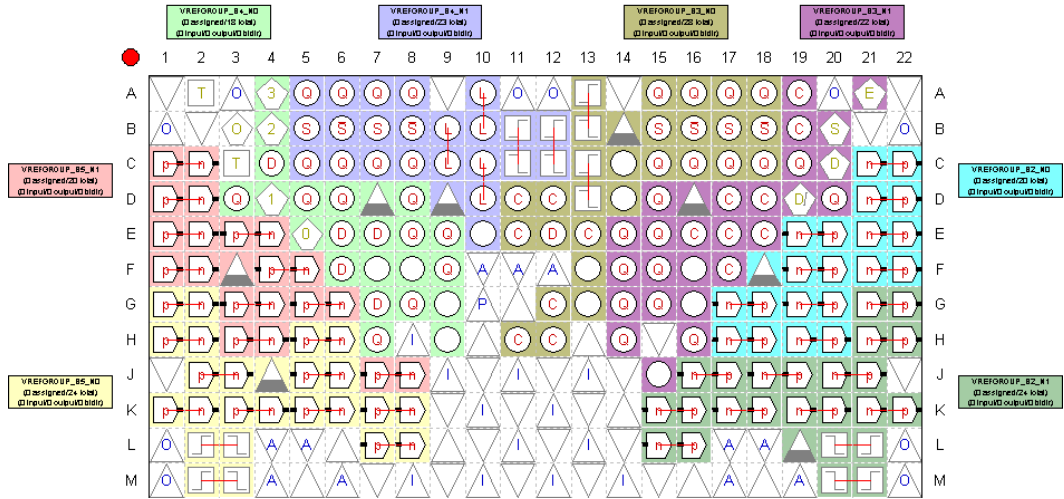
When **Show VREF Groups** is turned on in the View menu, in the Show submenu, or in the Package view shortcut (right-click) menu, the Package view uses different colors to indicate different groups of I/O pins sharing the same VCCIO and VREF pins (Figure 5–29). When planning your I/O pins, it is important to place pins with compatible voltage-referenced I/O standards in the same I/O bank. To guide your pin placement decisions by placing compatible I/O standards requiring VREF pins into the same VREF group, on the View menu, in the Show submenu, click **Show VREF Groups**. For example, pins with I/O standards SSTL-18 Class II and 1.8V-HSTL Class II are compatible and can be placed into the same VREF group. It is also important to be aware of the number and direction of pins within a VREF group for simultaneous switching noise (SSN) analysis.



For more information about compatible I/O standards, refer to the appropriate device handbook.

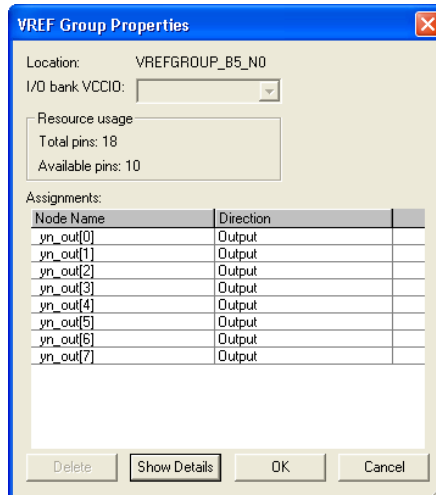
Figure 5–29. Package View with VREF Groups

Top View - Flip Chip Stratix II - EP2S15F484C3



When **Show VREF Groups** is turned on, the package view allows you to show the properties of each VREF group. Select a VREF group in the package view, and on the View menu, click **VREF Group Properties**. The **VREF Group Properties** dialog box appears (Figure 5–30). In the **VREF Group Properties** dialog box, all node names assigned to the VREF group are listed. Click **Show Details** to view node names that are assigned to pin numbers within the VREF group. Any design pins that are assigned to the VREF group and not to a pin number are listed in the **Assignments** list. The Resource usage section describes the total number of pins in the VREF group and the total number of available assignable pins. It also keeps a running tally on the input, output, and bidirectional pins.

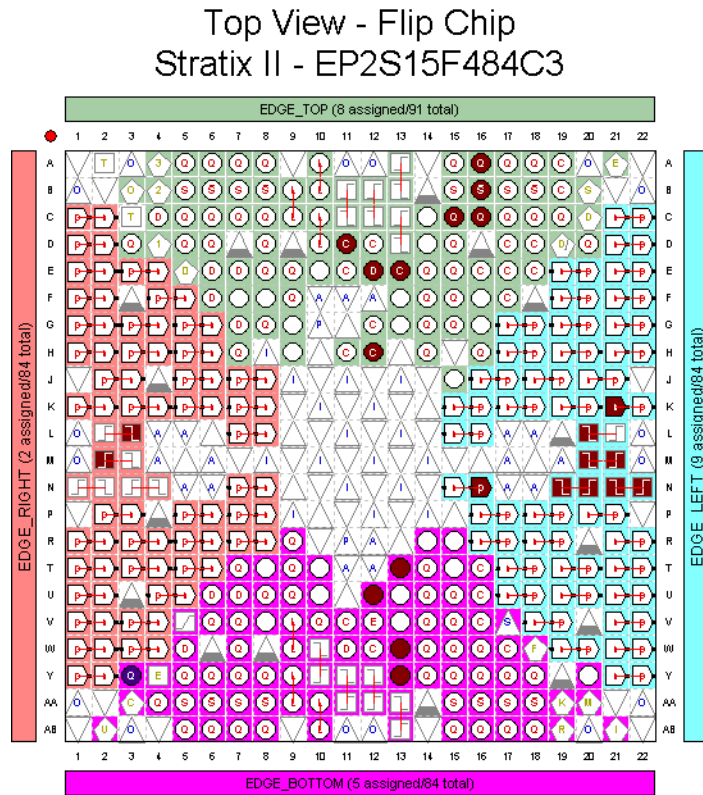
Figure 5–30. VREF Group Properties



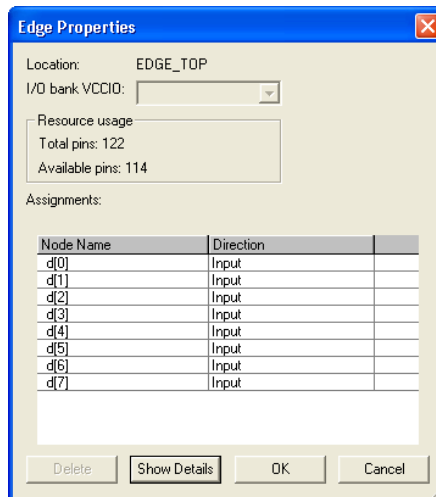
Show Edges

When **Show Edges** is turned on in the View menu, in the Show submenu, or in the shortcut (right-click) menu of the Package view, the Package view uses different colors to indicate the four edges of the package (Figure 5–31). If the exact location of a pin is not a priority when planning your I/O pins, use an Edge assignment.

Figure 5–31. Package View with Edges



When Show Edges is turned on, the package view allows you to show the properties of each Edge. Select an Edge in the package view and on the View menu, click **Edge Properties**. The **Edge Properties** dialog box appears. In the **Edge Properties** dialog box, all node names assigned to the Edge are listed (Figure 5–32). To view all node names assigned to a pin number within an Edge, in the **Edge Properties** dialog box, click **Show Details**.

Figure 5–32. Edge Properties

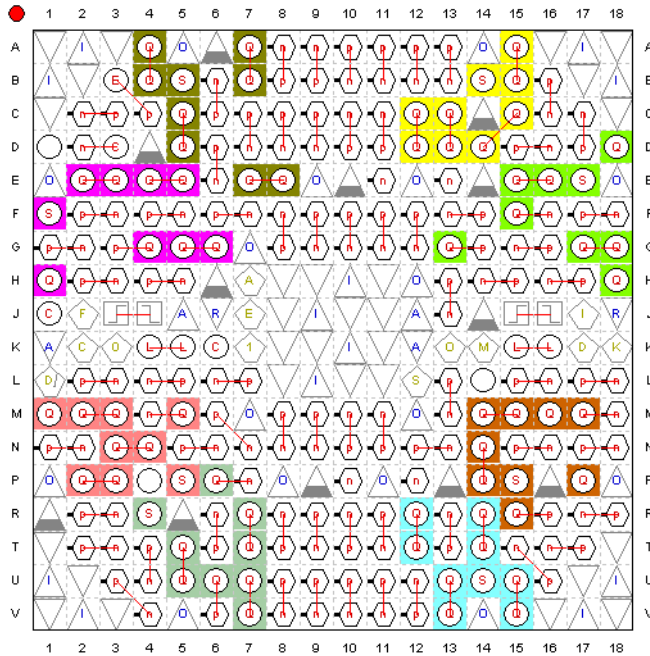
Show DQ/DQS Pins

When **Show DQ/DQS Pins** is turned on in the View menu in the Show submenu or in the shortcut (right-click) menu in the Package view, the package view uses different colors to highlight groups of DQ and DQS pins (Figure 5–33). Highlighting these DQ/DQS groups easily identifies which DQ pins are associated with a specific DQS strobe pin. You can select between the following DQ/DQS modes:

- In ×4 Mode
- In ×8/×9 Mode
- In ×16/×18 Mode
- In ×32/×36 Mode

Figure 5–33. DQ/DQS Pins (1)

Top View - Wire Bond Cyclone - EP1C4F324C6



Note to Figure 5–33:

(1) This DQ/DQS view shows an x8 mode.

For example, when implementing DDR II in a Stratix II device, there are dedicated pins designed specifically to be used as DQ and DQS pins.



For more information about using the `altdq` and `altdqs` megafunction, refer to the *altdq & altdqs Megafunction User Guide*.

Displaying & Accepting Fitter Placements

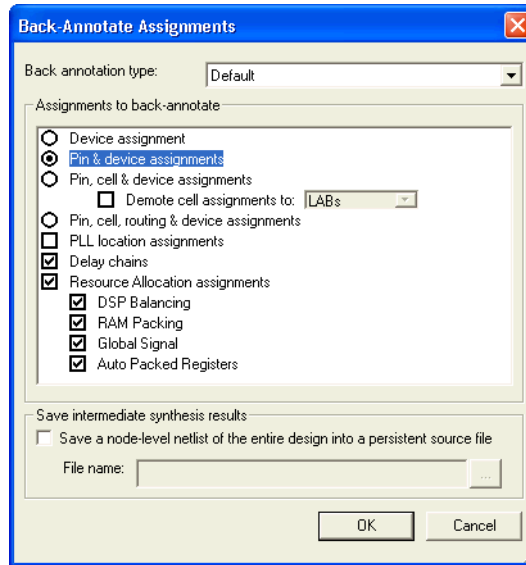
In addition to the **Show I/O Banks**, the **Show VREF Groups**, and the **Show Edge** views, you can also show pins placed by the Fitter by selecting **Show Fitter Placements** on the View menu in the Show submenu, in the Pin Planner toolbar, or in the shortcut (right-click) menu in the Package view.

The Fitter provides optimal placement to unassigned pins based on design constraints when you perform a compilation or an I/O Assignment Analysis. When you select **Show Fitter Placements** on the View menu, in the Show submenu, the Fitter-placed pins are shown as green-filled pins in the package view of the Pin Planner. You can create a copy of the fitter placements in your project Quartus II Settings File using the Back-Annotate Assignments command.

To create assignments for all Fitter-placed pins into your project Quartus II Settings File, perform the following steps in the Quartus II software:

1. On the Processing menu, click **Start Compilation**, or on the Processing menu, point to Start and click **I/O Assignment Analysis**.
2. On the Assignments menu, click **Pin Planner**. The Pin Planner appears.
3. On the View menu, point to Show and click **Show Fitter Placements**. You can also access this command from the Pin Planner toolbar or the shortcut (right-click) menu in the Package view. Review the Fitter placements.
4. To create location assignments for these fitter placements, perform the following steps:
 - a. On the Assignments menu, click **Back-Annotate Assignments**. The **Back-Annotate Assignments** dialog box appears.
 - b. Select **Pin & device assignments** (Figure 5-34).
 - c. Click **OK**.

Figure 5–34. Back-Annotate Assignments Dialog Box



To create assignments for a selection of the Fitter-placed pins, perform the following steps:

1. On the Processing menu, click **Start Compilation**, or on the Processing menu, point to Start, and click **I/O Assignment Analysis**.
2. On the Assignments menu, click **Pin Planner**.
3. On the View menu, point to Show and click **Show Fitter Placements**, and review the placements.
4. In the Pin Planner, select one or more fitter-placed pins for which you want to create assignments.
5. Right-click one of the selected pins, and click **Back Annotate**.
6. On the File menu, click **Save Project**. The Assignments are written to the Quartus II Settings File.



For more information about how the Quartus II software writes and updates the Quartus II Settings File, refer to the *Quartus II Project Management* chapter in volume 2 of the *Quartus II Handbook*.

Early I/O Planning Using the Pin Planner

It may be difficult to plan your I/Os early in the design cycle because design files, including the top-level design, may not be available yet. However, the interfaces between your FPGA and other devices are typically determined and documented in the design specifications. By adding the bus or memory interfaces needed to connect your FPGA with these other devices into the Pin Planner, you can efficiently plan your FPGA I/Os without design files.

The Pin Planner can interface with the MegaWizard® Plug-in Manager and gives you the ability to create or import custom megafunctions and IP cores. You can add many types of interfaces, including Megafunctions such as `altpll` and `altddio` as well as IP MegaCores such as PCI Compiler, QDR II, and Rapid IO. The advantage of adding the interface information while planning your I/Os is that it eliminates the possibility of not assigning a required pin while removing the need to manually create each pin individually in the Pin Planner.

After you add the interfaces used in your design, the Groups list is automatically populated with new groups named after the megafunctions or IP MegaCores you created. The members of the new groups include all the external pins of your megafunctions and IP MegaCores.

You can then make I/O pin assignments for all the external pins of your interfaces, create the required top-level wrapper file, and validate the assignments. [Figure 5-3 on page 5-4](#) shows a flow diagram for this type of early I/O planning flow with megafunctions and IP cores created in the Pin Planner.

Once you complete and validate the I/O assignments, you can proceed with your design in a number of ways:

- You can transfer the assignments to an existing project that includes design files, making sure the pin names match the design.
- You can continue working with this early I/O project, adding design files to work with the planned I/O assignments.
- You can make a revision of your existing design that uses the wrapper file and verified I/O assignments and decide later whether to integrate them with your project.



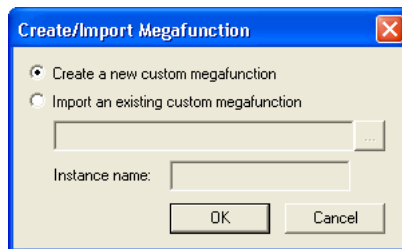
For information about revisions in the Quartus II software, refer to Quartus II Help.

Create a Megafunction or IP MegaCore Variation from the Pin Planner

To create a megafunction or IP MegaCore variation from the Pin Planner, perform the following steps:

1. In the Pin Planner, right-click anywhere in the Groups or All Pins lists.
2. On the shortcut (right-click) menu, click **Create/Import Megafunction**. The **Create/Import Megafunction** dialog box is displayed as shown in [Figure 5–35](#). You can also open this dialog box from the Edit menu.

Figure 5–35. Create/Import Megafunction Dialog Box



3. To create a new megafunction, select **Create a new megafunction** and click **OK**. The **MegaWizard Plug-In Manager** dialog box appears.
4. Under the Installed Plug-Ins, a list of all of the supported Megafunctions and IP MegaCores are shown. Select the Megafunction or IP MegaCore you want to create, and complete the wizard.
5. After you complete the wizard, a new group, based on the file name you provided, is created and all the I/O names, directions, and I/O standards are listed as members of the group in the **Groups** list. Make pin location assignments for the group or to each individual pin.



More more information about a particular megafunction, refer to the appropriate megafunction user guide.

Import a Megafunction or IP MegaCore Variation from the Pin Planner

To import a Variation from the Pin Planner, perform the following steps:

1. In the Pin Planner, right-click anywhere in the Groups or All Pins lists.
2. On the shortcut (right-click) menu, click **Create/Import Megafunction**. The **Create/Import Megafunction** dialog box is displayed, as shown in [Figure 5-35](#). You can also open this dialog box from the Edit menu.
3. Select **Import an existing custom megafunction**, and click **browse**. Select the Pin Planner File (.ppf) that was generated along with your megafunction variation or your IP MegaCore files.
4. In the Instance name box, type in an instance name and click **OK**.



To avoid pin name conflicts when there are more than one instances of a megafunction or IP MegaCore, the instance name is appended to the beginning of each pin name.

After you finish running the wizard, a new group based on the file name you provided is created and all of the I/Os that are used externally are listed as members of the group. Make pin location assignments for the group or to each individual pin.

Create a Top-Level Netlist for I/O Analysis

You can create a top-level design file after you add megafunctions or IP MegaCores to your project with the Pin Planner and make I/O assignments for their external nodes. Though no internal logic may exist yet, the top-level design file lets you validate your I/O assignments and provides a base on which to build the rest of your design. The creation of a top-level design file is a two-step process. First, you must configure the megafunctions and IP MegaCores created in the Pin Planner for integration with each other and the rest of the design. Then you create the actual top-level design file.

Configure Megafunctions for Creating a Top-Level Design File

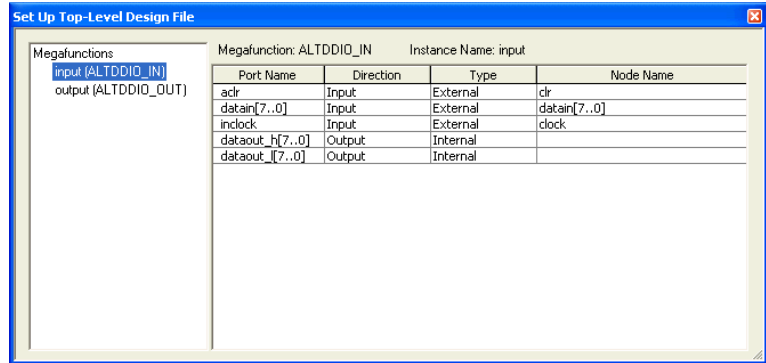
After creating or importing custom Megafunctions or IP MegaCores in the Pin Planner, you must configure how they will be connected to each other. You do this by specifying matching node names for selected ports of the megafunctions or IP MegaCores.



In this section, ports and port names refer to the generic port names of megafunctions and IP MegaCores in the MegaWizard® Plug-In Manager. Node names refer to the unique names assigned to ports when the megafunction or IP MegaCore is created based on the instance name given when the MegaWizard Plug-In Manager is started. By default, node names are the original port names appended with *<instance name>_.*

To configure your custom megafunctions and IP MegaCores for creating a top-level design file, on the Edit menu or in the shortcut (right-click) menu of the Package view, click **Set Up Top-Level Design File**. The **Set Up Top-Level Design File** dialog box appears (Figure 5-36).

Figure 5-36. Set Up Top-Level Design File Dialog Box



Click the name of a megafunction or IP MegaCore in the list on the left. Only megafunctions created in the Pin Planner appear in the list. The list on the right contains all of the ports for the selected megafunction.

The columns in the **Set Up Top-Level Design File** dialog box provide information about megafunctions created in the Pin Planner and allow you to make adjustments to connect megafunctions together. The **Direction** column indicates the direction of the port or port group as defined by the megafunction. The direction of a port cannot be changed.

The **Type** column indicates whether a port is available externally to the device. By default, all ports on all megafunctions created through the Pin Planner are of the **External** type, meaning they appear in the Pin Planner **Groups** and **All Pins** lists and can be assigned to I/O pins. You can change the port type by double-clicking the Type cell for a port and selecting **Internal** or **External** from the list. Any ports on any

megafunction connected to a port that has its type changed have their type changed to match automatically. This prevents internal and external megafunction ports from being connected to each other accidentally. Internal ports do not appear in the **Groups** or **All Pins** lists. If all the ports of a megafunction are Internal, the megafunction does not appear in the Groups list.

The **Node Name** column is used to assign node names or device pins to ports. You can double-click a cell in the **Node Name** column and select an existing node or device pin to connect the port to an existing location. Enter a new node name in the **Node Name** column to rename the selected port. This only changes the name as it appears as a group member in the Pin Planner **Groups** list. Enter a node name that matches the node name of the ports of other megafunctions or between a megafunction and an existing node to connect the ports to each other.

Figure 5–37 shows an example of the port names of the megafunction named “output” as shown in Figure 5–36. When the port types and node names for both the input and output megafunctions are configured as in the two figures, they create a circuit similar to the one shown in Figure 5–38. In this way, you can connect megafunctions to each other and to other nodes in the design, improving the thoroughness of an I/O assignment analysis. This is especially useful for clock networks that are typically attached to multiple megafunctions or IP MegaCores.

Figure 5–37. Port Names of the Megafunction Named “Output”

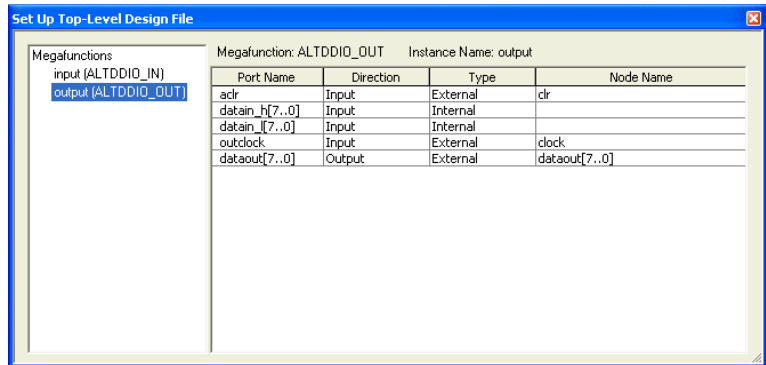
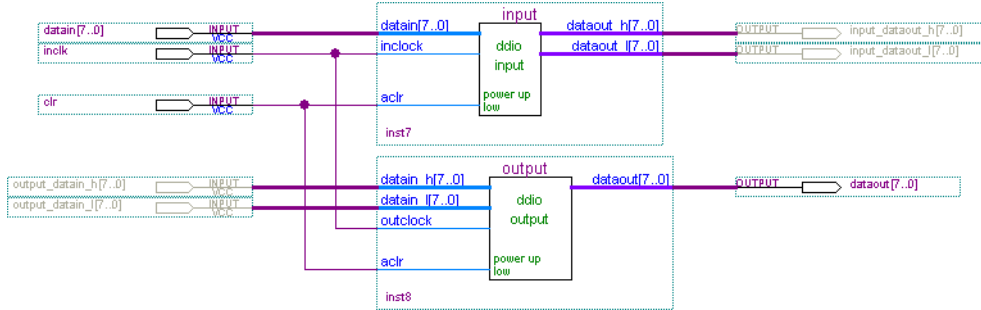


Figure 5–38. Schematic Representing Connections between Input & Output Megafunctions



Note to Figure 5–38:

- (1) Gray pins indicate internal nodes.

You can edit a megafunction or IP MegaCore you have created in the Pin Planner. Select it in the Groups list. On the selected megafunction's shortcut (right-click) menu or on the Edit menu, click **Edit Megafunction** to reopen the MegaWizard Plug-In Manager and make changes as necessary. If you make changes to a megafunction, you must import it again and reconfigure its node connections in the **Set Up Top-Level Design File** dialog box.

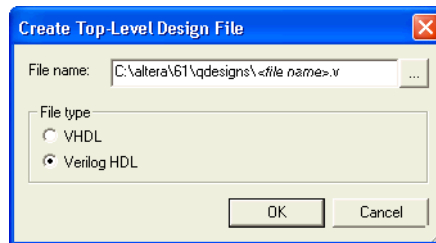


Always use the **Edit Megafunction** command to make changes to megafunctions and IP MegaCores created with the Pin Planner. If the generated megafunction code files are edited manually or the MegaWizard Plug-In Manager is used outside of the Pin Planner, you will not be able to configure the megafunction ports as described, or create a top-level design file based on the edited megafunctions.

Create a Top-Level Design File

Once you have configured the megafunctions or IP MegaCores created in the Pin Planner, you can create a top-level design file. You can use this as the basis for the rest of your project, or just to validate the I/O assignments you have made.

To generate a top-level design file, right-click in the Package view and click **Create Top-Level Design File**. You can also generate a top-level file on the File menu by pointing to Create/Update and clicking **Create Top-Level Design File From Pin Planner**. The **Create Top-Level Design File** dialog box appears (Figure 5–39).

Figure 5–39. The Create Top-Level Design File Dialog Box

Enter a name and select a file type. If the file already exists, you can select to create a backup of the original file.

The newly created file is set as the top-level design file in the Project Navigator. You can now use this file as the basis for the rest of your project and perform I/O Assignment Analysis, described in the next section, to validate these early I/O assignments. The Pin Planner gives virtual pin assignments to internal nodes, so internal nodes are not assigned to device pins during compilation. If you use this top-level file as the basis for your project, internal megafunction ports must be connected to internal logic.



The top-level design file must be updated whenever changes are made to the design. This includes any node changes made in the **Set Up Top-Level Design File** dialog box. You do not have to update the top-level design file if you make pin assignment changes to the megafunctions because these changes are stored in the Quartus II Default Settings File (.qdf).

Using I/O Assignment Analysis to Validate Pin Assignments

This section describes a design flow that includes making and analyzing pin assignments with the **Start I/O Assignment Analysis** command in the Quartus II software during and after the development of your HDL design.

The **Start I/O Assignment Analysis** command allows you to check your I/O assignments early in the design process. Use this command to check the legality of pin assignments before, during, or after you compile 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 can be used for designs that target Stratix series, Cyclone® series, and MAX® II device families.

I/O Assignment Analysis Design Flows

The I/O assignment analysis design flows depend on whether your project contains design files. The following examples show two different circumstances in which I/O assignment analysis can be used:

- When the board layout must be complete before starting the FPGA design, use the flow shown in [Figure 5-40 on page 5-60](#). This flow does not require design files and checks the legality of your pin assignments.
- With a complete design, use the flow shown in [Figure 5-42 on page 5-63](#). This flow thoroughly checks the legality of your pin assignments against any design files provided. For more information about creating assignments, refer to 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.

You should run the analysis each time you add or modify a pin-related assignment. You can use the **Start I/O Assignment Analysis** command frequently because 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 resources such as a phase-locked loops (PLLs), low-voltage differential signals (LVDS), or gigabit transceiver blocks.

Design Flow without Design Files

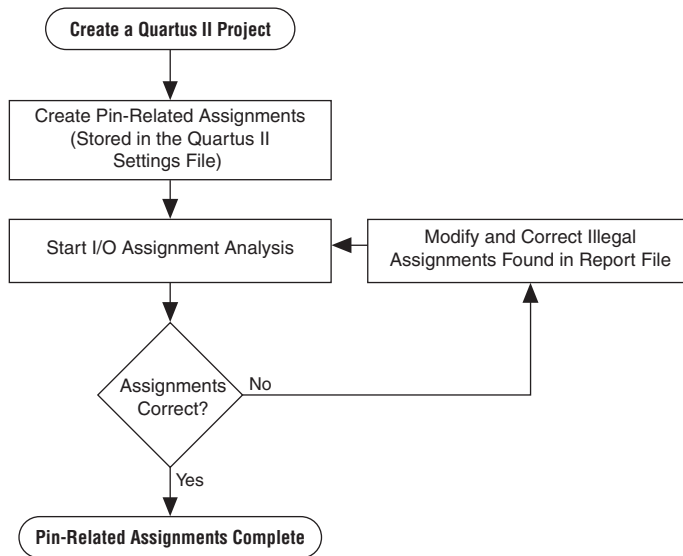
During the early stages of developing an FPGA device, board layout engineers may request preliminary or final pin-outs. It is time consuming to manually check whether the pin-outs violate any design rules. Instead, you can use the **Start I/O Assignment Analysis** command to quickly perform basic checks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



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

The I/O Assignment Analysis command can perform limited checks on pin assignments made in a Quartus II project that has a device specified, but may not yet include any HDL design files. For example, you can create a Quartus II project with only a target device specified and create pin-related assignments based on circuit board layout considerations that are already determined. Even though the Quartus II project does not yet contain any design files, you can reserve input and output pins and make pin-related assignments for each pin using the Pin Planner or Assignment Editor. After you assign an I/O standard to each reserved pin, run the I/O Assignment Analysis to ensure that there are no I/O standard conflicts in each I/O bank.

Figure 5–40. Assigning & Analyzing Pin-Outs without Design Files



To assign and analyze pin-outs using the **Start I/O Assignment Analysis** command without design files, perform the following steps:

1. In the Quartus II software, create a project.
2. Use the Pin Planner, Assignment Editor, or a Tcl script to create pin locations and related assignments. For the I/O assignment analysis to determine the type of pin, you must reserve your I/O pins. For information about reserving pins in the Pin Planner, refer to [“Creating Reserved Pin Assignments” on page 5–34](#). For information about reserving pins in the Assignment Editor, refer to [“Reserving Pins” on page 5–66](#).



If you make pin-related assignments in the Mentor Graphics I/O Designer software, you can import an FPGA Xchange file into the Quartus II software.

3. To start the analysis, on the Processing menu, point to Start, and click **Start I/O Assignment Analysis**.



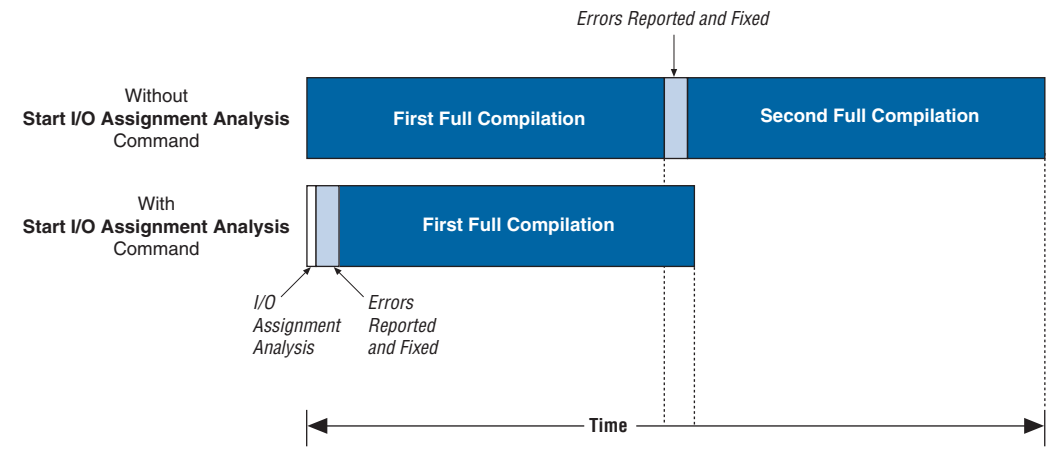
For information about using a Tcl script or command prompt to start the analysis, refer to [“Scripting Support” on page 5–76](#).

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.

Repeat the above steps 1 through 5 until all of the errors are corrected.

Design Flow with Design Files

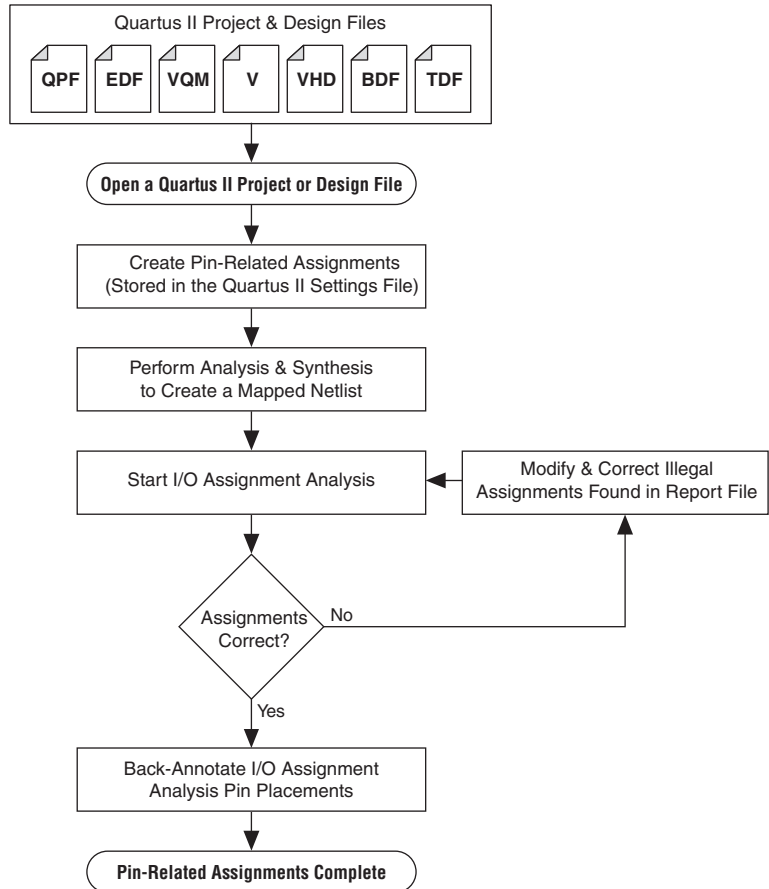
During a full compilation, the Quartus II software does not report illegal pin assignments until the fitter stage. To validate pin assignments sooner, you can run the **Start I/O Assignment Analysis** command after performing analysis and synthesis and before performing a full compilation. Typically, the analysis takes a short time. [Figure 5–41](#) shows the benefits of using the **Start I/O Assignment Analysis** command.

Figure 5–41. Saving Compilation Time with the Start I/O Assignment Analysis Command

The rules that are checked by the I/O assignment analysis depend 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, which can be just the top-level wrapper file, 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, or 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 does not account for 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.

Analyze 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 Plug-In Manager-generated files in your project for analysis (Figure 5–42).

Figure 5–42. Assigning & Analyzing Pin-Outs with Design Files

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

1. In the Quartus II software, create a project including your design files.
2. Create pin-related assignments with the Pin Planner or Assignment Editor.



You can also create pin-related assignments by importing them from a CSV or FPGA Xchange file, executing Tcl commands, or editing the Quartus II Settings File directly. On the Processing menu, point to Start and click **Start Analysis & Synthesis** to generate an internal mapped netlist.

For information about using a Tcl script or the command prompt to start the analysis, refer to [“Scripting Support” on page 5–76](#).

3. On the Processing menu, point to Start, and click **Start I/O Assignment Analysis** to start the analysis.
4. View the messages in the Compilation Report or in the Messages window.
5. Use the Pin Planner or Assignment Editor to correct any errors and violations reported.
6. Use the **Start I/O Assignment Analysis** command until all errors are corrected.

Using Output Enable Group Logic Option Assignments with I/O Assignment Analysis

Each device has a certain number of VREF pins, and each VREF pin supports a certain number of I/O pins. Check the device pin-outs to locate the VREF pins and its associated I/O pins. The VREF pin, including its supported I/O pins, is called a VREF bank. The VREF pins are only used for VREF I/O standards; for example, SSTL and HSTL input pins. VREF outputs do not require the VREF pin. When a voltage-referenced input is present in a VREF bank, there can be only a certain number of outputs that are allowed to be present in that VREF bank. For the Stratix II flip chip package, only 20 outputs can be present in a VREF bank when a VREF I/O standard input is present in that bank.

For interfaces that use bidirectional VREF I/O pins, the VREF restriction must be met when the pins are driving in either direction. If a set of bidirectional signals are controlled by different output enables, the **I/O Assignment Analysis** command treats these as independent output enables. Use the output enable group logic option assignment to treat the set of bidirectional signals as a single output enable. This is important in the case of external memory interfaces.

For example, in the case of a DDR2 interface in a Stratix II device, a Stratix II device can have 30 pins in a VREF group. Each byte lane for a $\times 8$ DDR2 interfaces has 1 DQS pin and 8 DQ pins, for a total of 9 pins per byte lane. DDR2 uses SSTL18 as its I/O standard, which is a VREF I/O standard. In typical interfaces, each byte lane has its own output enable. In this example, the DDR2 interface has 4 byte lanes. Using 30 I/O pins in a VREF group, there are 3 byte lanes, and an extra byte lane that supports the 3 remaining pins. If you do not use the output enable group logic option assignment, the I/O Assignment Analysis command analyzes each byte lane as an independent group driven by a unique output enable. With this arrangement, the worst-case scenario is when the 3 pins are inputs, and the other 27 pins are outputs. In this case, the 27 output pins violate the 20-output pin limit.

In a DDR2 interface, all DQS and DQ pins are always driven in the same direction. Therefore, the I/O Assignment Analysis reports an error that is not applicable to your design. Assigning an output enable group logic option assignment to the DQS and DQ pins forces the I/O Assignment Analyzer to check these pins as a group driven by a common output enable. When using the output enable group logic option assignment, the DQS and DQ pins are checked as all input pins or all output pins. This does not violate the rules described in [Tables 5-5 and 5-6](#).

The value for the output enable group logic option assignment should be an integer value. All sets of signals that are driving in the same direction should be given the same integer value. The output enable group logic option assignment can also be used with pins that are driven only at certain times. For example, the data mask signal in DDR2 interfaces are only outputs, but are driven only when the DDR2 is writing (bidirectional signals are outputs). Therefore, an output enable group logic option assignment should be assigned to the data mask with the same value of the DQ and DQS signals.

Output enable groups can also be used on VREF input pins. If the VREF input pins are not active during the time the outputs are driving, you can add the VREF input pins to the output enable group. This removes the VREF input pins from the VREF analysis. For example, the QVLD signal for RLD RAM II is only active during a read. During a write, the QVLD is

not active and so it does not count as an active VREF input pin within the VREF group. The QVLD pins can be placed in the same output enable group as the RLDRAM II data pins.

Inputs for I/O Assignment Analysis

The **Start I/O Assignment Analysis** command reads the following inputs:

- Internal mapped netlist
- Quartus II Settings File

The internal mapped netlist is used when you have a partial or complete design. The Quartus II Settings File is always used to read all pin-related assignments for analysis.

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

To generate a mapped netlist, on the Processing menu, point to Start, and click **Start Analysis & Synthesis**.

To use the `quartus_map` executable to run analysis and synthesis, type the following command at a system command prompt:

```
quartus_map <project name> ←
```

Creating Pin-Related Assignments

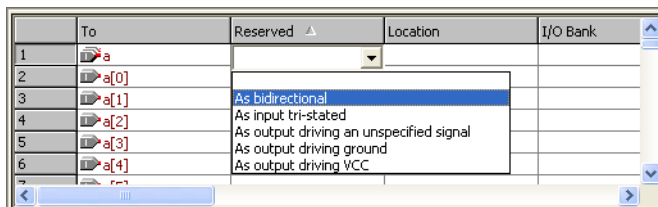
The **I/O Assignment Analysis** command reads a Quartus II Settings File containing all of your pin-related assignments. These pin-related assignments include pin settings such as I/O standards, drive strength, and location assignments. The following sections highlight some of the location assignments you can make.

Reserving Pins

If you do not have any design files, you can still reserve pin locations and create pin-related assignments. Reserving pins is necessary so that the **Start I/O Assignment Analysis** command has information about the pin and the pin type (input, output, or bidirectional) to correctly analyze the pins. To reserve a pin, on the Assignments menu, click **Assignment Editor**. In the **Category** list, click **Pin** to open the Pin assignment category.

Double-click the cell in the **Reserved** column that corresponds to the pin that you want to reserve. Use the drop-down arrow to select from the reserve pin options (Figure 5-43).

Figure 5-43. Reserving an Input Pin with the Assignment Editor



For more information about using the Assignment Editor, refer to the *Assignment Editor* chapter in volume 2 of the *Quartus II Handbook*.

You can also reserve pins using the Pin Planner. For more information about the Pin Planner, refer to “[Creating Reserved Pin Assignments](#)” on page 5-34.

Location Assignments

You can create the following types of location assignments for your design and its reserved pins:

- Pin number
- I/O bank
- VREF group
- Edge



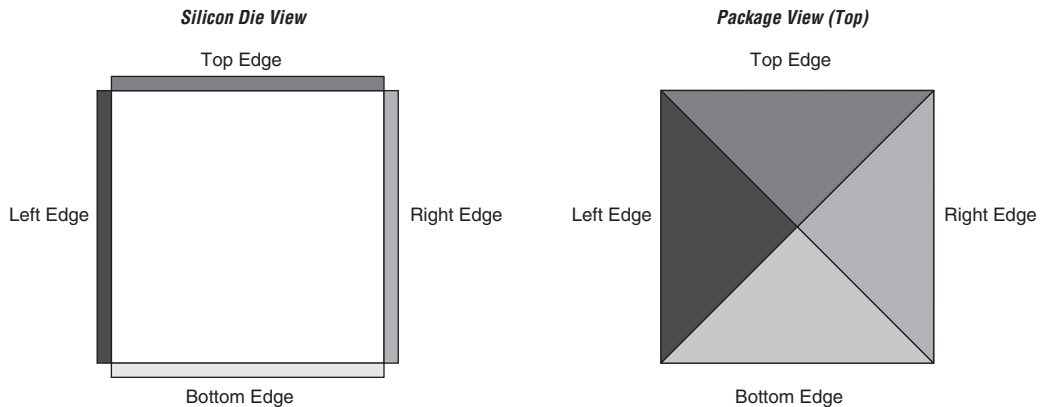
I/O bank, VREF group, and Edge location assignments are supported only for Stratix and Cyclone series device families.

You can assign a location to your pins using the Pin Planner or the Assignment Editor. To make a pin location assignment using the Assignment Editor, on the Assignments menu, click **Assignment Editor** and select the **Pin** category from the **Category** list. Type the pin name and select a location from the **Location** list.

It is common to place a group of pins (or bus) with compatible I/O standards in the same I/O bank or VREF group. For example, two buses with two compatible I/O standards, such as 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 the circuit board routing ability of large buses, because they are close together near an edge. [Figure 5-44](#) shows the Altera device package edges.

Figure 5-44. Die View & Package View of the Four Edges on an Altera Device



Suggested & Partial Placement

The **Start I/O Assignment Analysis** command automatically assigns suggested pin locations to unassigned pins in your design so it can perform pin legality checks. For example, if you assign an edge location to a group of LVDS pins, the **I/O Assignment Analysis** command assigns pin locations for each LVDS pin in the specified edge location and then performs legality checks.

To accept these suggested pin locations, on the Assignments menu, click **Back-Annotate Assignments**, select **Pin & device** assignments, and click **OK**. Back-annotation saves your pin and device assignments in the Quartus II Settings File.

Understanding the I/O Assignment Analysis Report & Messages

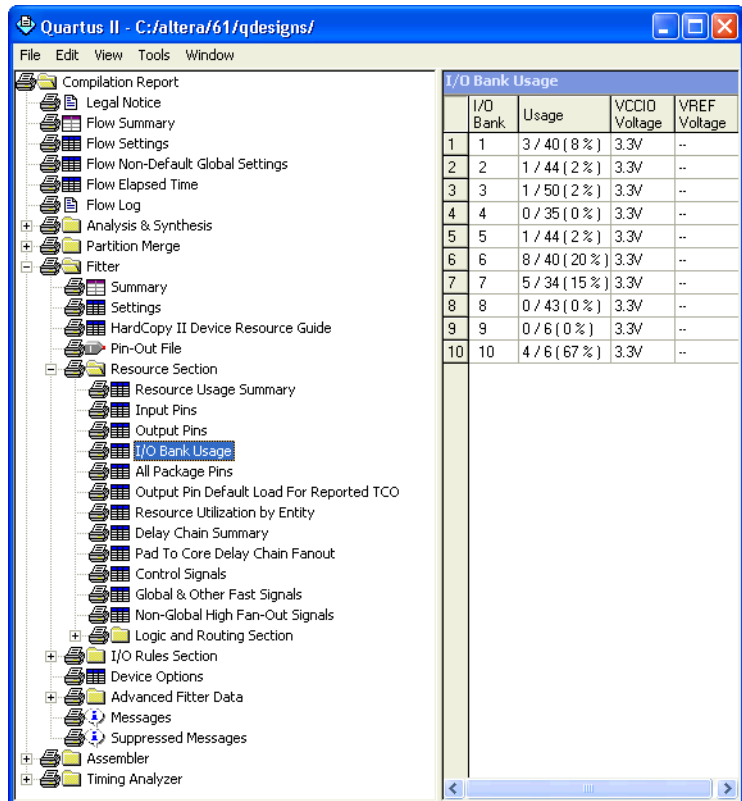
The **Start I/O Assignment Analysis** command generates detailed analysis reports and a Pin-Out file. The detailed messages in the reports help you quickly understand and resolve pin assignment errors. Each message includes a related node name and a description of the problem.

To view the report file, on the Project menu, click **Compilation Report**. The **Fitter** section of the Compilation Report contains the following sections:

- Summary
- Settings
- Resource Section
- I/O Rules Section
- Device Options
- Advanced Fitter Data
- Pin-Out File
- Fitter Messages

The Resource Section categorizes the pins as **Input Pins**, **Output Pins**, and **Bidir Pins**. You can view the utilization of each I/O bank in your device in the **I/O Bank Usage** section ([Figure 5-45](#)).

Figure 5–45. Summary of the I/O Bank Usage in the I/O Assignment Analysis Report



The I/O Rules Section includes detailed information about the I/O rules tested during I/O Assignment Analysis. Three sub-reports are generated. The I/O Rules Summary report provides a quick summary of the number of I/O rules tested and how many applicable rules passed, how many failed, and how many were unchecked because of other failing rules (Figure 5–46).

Figure 5–46. I/O Rules Summary Report

I/O Rules Statistic	Total	
1 Total I/O Rules	31	
2 Number of I/O Rules Passed	7	
3 Number of I/O Rules Failed	0	
4 Number of I/O Rules Unchecked	0	
5 Number of I/O Rules Inapplicable	24	

The I/O Rules Details report provides detailed information on all I/O rules. Applicable rules indicate whether they passed, failed, or could not be checked (Figure 5–47). All rules are given a level of severity from Low to Critical to indicate their level of importance for an effective analysis.

Figure 5–47. I/O Rules Details Report

Status	ID	Category	Rule Description	Severity
Pass	IO_000001	Capacity Checks	Number of pins in an I/O bank should not exceed the number of locations available.	Critical
Inapplicable	IO_000002	Capacity Checks	Number of clocks in an I/O bank should not exceed the number of clocks available.	Critical
Pass	IO_000003	Capacity Checks	Number of pins in a Vrefgroup should not exceed the number of locations available.	Critical
Inapplicable	IO_000004	Voltage Compatibility Checks	The I/O bank should support the requested VCCIO.	Critical
Inapplicable	IO_000005	Voltage Compatibility Checks	The I/O bank should not have competing VREF values.	Critical
Pass	IO_000006	Voltage Compatibility Checks	The I/O bank should not have competing VCCIO values.	Critical
Pass	IO_000007	Valid Location Checks	Checks for unavailable locations.	Critical
Inapplicable	IO_000008	Valid Location Checks	Checks for reserved locations.	Critical
Pass	IO_000009	I/O Properties Checks for One I/O	The location should support the requested I/O standard.	Critical
Pass	IO_000010	I/O Properties Checks for One I/O	The location should support the requested I/O direction.	Critical
Inapplicable	IO_000011	I/O Properties Checks for One I/O	The location should support the requested Current Strength.	Critical
Inapplicable	IO_000012	I/O Properties Checks for One I/O	The location should support the requested On Chip Termination value.	Critical
Inapplicable	IO_000013	I/O Properties Checks for One I/O	The location should support the requested Bus Hold value.	Critical
Inapplicable	IO_000014	I/O Properties Checks for One I/O	The location should support the requested Weak Pull Up value.	Critical
Inapplicable	IO_000015	I/O Properties Checks for One I/O	The location should support the requested PCI Clamp Diode.	Critical
Inapplicable	IO_000018	I/O Properties Checks for One I/O	The I/O standard should support the requested Current Strength.	Critical
Inapplicable	IO_000019	I/O Properties Checks for One I/O	The I/O standard should support the requested On Chip Termination value.	Critical
Inapplicable	IO_000020	I/O Properties Checks for One I/O	The I/O standard should support the requested PCI Clamp Diode.	Critical
Inapplicable	IO_000021	I/O Properties Checks for One I/O	The I/O standard should support the requested Weak Pull Up value.	Critical
Inapplicable	IO_000022	I/O Properties Checks for One I/O	The I/O standard should support the requested Bus Hold value.	Critical

The I/O Rules Matrix shows how each I/O rule was tested on each pin in the design (Figure 5–48). Applicable rules that could be checked either pass or fail for each pin. You can quickly find and make pin assignment adjustments on any pin that fails. Right-click a pin name that failed an I/O rule. Point to Locate, and select a location where the pin exists, such as the Pin Planner. Make appropriate changes to fix the pin assignments and rerun I/O Assignment Analysis. Check the resulting I/O Rules Matrix to see that your changes fixed the problem and allowed the failing pin assignment to pass.

Figure 5–48. I/O Rules Matrix

	Pin/Rules	IO_000001	IO_000002	IO_000003	IO_000004	IO_000005	IO_000006	IO_000007
1	Total Pass	22	0	22	0	0	22	22
2	Total Unchecked	0	0	0	0	0	0	0
3	Total Inapplicable	0	22	0	22	22	0	0
4	Total Fail	0	0	0	0	0	0	0
5	yvalid	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
6	follow	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
7	yn_out[7]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
8	yn_out[6]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
9	yn_out[5]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
10	yn_out[4]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
11	yn_out[3]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
12	yn_out[2]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
13	yn_out[1]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
14	yn_out[0]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
15	clk	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
16	reset	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
17	clkx2	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
18	newt	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
19	d[7]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
20	d[6]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
21	d[5]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
22	d[4]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
23	d[3]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
24	d[2]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
25	d[1]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass
26	d[0]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass

The **Fitter Messages** page stores all messages including errors, warnings, and information messages.

You can view the detailed messages in the **Fitter Messages** page in the compilation report and in the **Processing** tab in the **Messages** window. To open the **Messages** window, on the View menu, point to **Utility windows**, and click **Messages**.

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

Figure 5–49 shows an example of error messages reported by I/O assignment analysis.

Figure 5–49. 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 LVTTTL
 - ⊕ 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 Filter was unsuccessful. 2 errors, 1 warning

The effectiveness of the I/O Assignment Analysis is relative to the completeness of your pin-related assignments and design. To ensure your design functions correctly, include all pin-related assignments and as many design files as possible in your Quartus II project.

Tables 5–5 and 5–6 list a subset of the I/O rule checks performed when you run an I/O Assignment Analysis with and without design files.



For more detailed information about each I/O rule, refer to the appropriate device handbook.

Table 5–5. Examples of I/O Rules Checks (Part 1 of 2) *Note (1)*

Rule	Description	Device Families	HDL Required?
I/O bank capacity	Checks the number of pins assigned to an I/O bank against the number of pins allowed in the I/O bank.	All	No
I/O bank V _{CCIO} voltage compatibility	Checks that no more than one V _{CCIO} is required from the pins assigned to the I/O bank.	All	No
I/O bank V _{REF} voltage compatibility	Checks that no more than one V _{REF} is required from the pins assigned to the I/O bank.	All	No
I/O standard and location conflicts	Checks whether the pin location supports the assigned I/O standard.	All	No
I/O standard and signal direction conflicts	Checks if the pin location supports the assigned I/O standard and direction. For example, certain I/O standards on a particular pin location can only support output pins.	All	No

Table 5–5. Examples of I/O Rules Checks (Part 2 of 2) *Note (1)*

Rule	Description	Device Families	HDL Required?
Differential I/O standards cannot have open drain turned on	Checks that open drain is turned off for all pins with a differential I/O standard.	All	No
I/O standard and drive strength conflicts	Checks whether the drive strength assignments are within the specifications of the I/O standard.	All	No
Drive strength and location conflicts	Checks whether the pin location supports the assigned drive strength.	All	No
BUSHOLD and location conflicts	Checks whether the pin location supports BUSHOLD. For example, dedicated clock pins do not support BUSHOLD.	All	No
WEAK_PULLUP and location conflicts	Checks whether the pin location supports WEAK_PULLUP (for example, dedicated clock pins do not support WEAK_PULLUP)	All	No
Electromigration check	Checks whether combined drive strength of consecutive pads exceeds a certain limit. For example, the total current drive for 10 consecutive pads on a Stratix II device cannot exceed 200 mA.	All	No
PCI_IO clamp diode, location, and I/O standard conflicts	Checks whether the pin location along with the I/O standard assigned supports PCI_IO clamp diode.	All	No
SERDES and I/O pin location compatibility check	Checks that all pins connected to a SERDES in your design are assigned to dedicated SERDES pin locations.	All	Yes
PLL and I/O pin location compatibility check	Checks whether pins connected to a PLL are assigned to the dedicated PLL pin locations.	All	Yes

Note to Table 5–40:

- (1) “All” includes the following device families: Stratix III, Stratix II, Stratix II GX, Stratix, Stratix GX, Cyclone II, Cyclone, MAX II, and HardCopy devices.

Table 5–6. SSN-Related Rules (Part 1 of 2)

Rule	Description	Device Families ⁽¹⁾	HDL Required?
I/O bank can not have single-ended I/O when DPA exists	Checks that no single-ended I/O pin exists in the same I/O bank as a DPA.	Stratix II Stratix GX	No
A PLL I/O bank does not support both a single-ended I/O and a differential signal simultaneously	Checks that there are no single-ended I/O pins present in the PLL I/O Bank when a differential signal exists.	Stratix II	No
Single-ended output is required to be a certain distance away from a differential I/O pin	Checks if single-ended output pins are a certain distance away from a differential I/O pin.	All	No

Table 5–6. SSN-Related Rules (Part 2 of 2)

Rule	Description	Device ⁽¹⁾ Families	HDL Required?
Single-ended output has to be a certain distance away from a VREF pad	Checks if single-ended output pins are a certain distance away from a VREF pad.	Cyclone II Cyclone	No
Single-ended input is required to be a certain distance away from a differential I/O pin	Checks if single-ended input pins are a certain distance away from a differential I/O pin.	Cyclone II Cyclone	No
Too many outputs or bidirectional pins in a VREFGROUP when a VREF is used	Checks that there are no more than a certain number of outputs or bidirectional pins in a VREFGROUP when a VREF is used.	All	No
Too many outputs in a VREFGROUP	Checks if too many outputs are in a VREFGROUP.	All	No

Note to Table 5–6:

- (1) “All” includes the following device families: Stratix II, Stratix II GX, Stratix, Stratix GX, Cyclone II, Cyclone, MAX II, and HardCopy devices.

Scripting Support

A Tcl script allows you to run procedures and make settings described in this chapter. 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 the following command at a system command prompt to run the Quartus II command-Line and Tcl API Help browser:

```
quartus_sh --qhelp ←
```



For more information about Quartus II scripting support, including examples, refer to the *Tcl Scripting* and *Command-Line Scripting* chapters in volume 2 of the *Quartus II Handbook*.

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, refer to “[Understanding the I/O Assignment Analysis Report & Messages](#)” on page 5–68.

Tcl Command

Enter the following in a Tcl console or script:

```
execute_flow -check_ios
```

Command Prompt

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

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

Generating a Mapped Netlist

You can generate a mapped netlist with a Tcl command or with a command-line command. For more information about generating a mapped netlist, refer to [“Generating a Mapped Netlist” on page 5–66](#).

Tcl Command

Enter the following in the Tcl console or in a script:

```
execute_module -tool map
```

The `execute_module` command is in the flow package.

Command Prompt

Type the following at a system command prompt:

```
quartus_map <project name>␣
```

Reserving Pins

Use the following Tcl command to reserve a pin. For more information about reserving pins, refer to [“Reserving Pins” on page 5–66](#).

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

Valid values are: "AS BIDIRECTIONAL", "AS INPUT TRI-STATE", "AS OUTPUT DRIVING AN UNSPECIFIED SIGNAL", "AS OUTPUT DRIVING GROUND" and "AS SIGNALPROBE 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, refer to [“Location Assignments” on page 5–67](#).

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

Valid locations are pin location names, such as `PIN_A3`. The Stratix series 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`, in which n is the number of I/O banks in a particular device.



With Stratix III devices only, I/O banks are in the form of `IOBANK_nx` where n is a number and x is the letter *A*, *B*, or *C*. Though I/O banks may share the same number with different letters, such as 1A and 1C, they are separate banks and not related to each other. For more information, refer to the *Stratix III Device Handbook*.

Incorporating PCB Design Tools

Signal and pin assignments are initially made by the FPGA or ASIC designer, and it is up to the board designer to correctly transfer these assignments to the symbols used in their system circuit schematics and board layout. As the board design progresses, pin reassignments may be requested or required to optimize the layout. These reassignments must in turn be relayed to the FPGA designer, so that the new assignments can be validated with the I/O Assignment Analyzer and processed through an updated place-and-route of the FPGA.

The Quartus II software interacts with board layout tools by importing and exporting pin information files, including the Quartus II Settings File, Pin-Out file, and the FPGA Xchange file.



For more information about incorporating PCB design tools, refer to the *Cadence PCB Design Tools Support* and the *Mentor Graphics PCB Design Tools Support* chapters in volume 2 of the *Quartus II Handbook*.

Advanced I/O Timing

As part of I/O planning, especially with high-speed designs, you should take board-level signal integrity and timing into account. When adding an FPGA device with high-speed interfaces to a board design, the quality of the signal at the far end of the board route, as well as the propagation delay in getting there, is vital for proper system operation.

The Quartus II software provides features to take these factors into consideration, making the software “board-aware.” The Quartus II software can take into account board routing and external devices to generate advanced timing reports and board simulation modeling files. Three different methods of analysis are possible:

- I/O timing using a default or user-specified capacitive load with no signal integrity analysis (default)

- The Quartus II **Enable Advanced I/O Timing** option utilizing a user-defined board trace model to produce enhanced timing reports from accurate, “board-aware” simulation models
- Full board routing simulation in third-party tools using Altera-provided or generated IBIS or HSPICE I/O models

If no changes are made to device settings, the first method of timing analysis is used. Timing reports created by the TimeQuest Timing Analyzer and the Classic Timing Analyzer measure t_{co} to an I/O pin using a default or user-specified value for a capacitive load.

The second method, the Quartus II **Enable Advanced I/O Timing** option, lets you configure a complete board trace model for each I/O standard or pin used in your design. With **Enable Advanced I/O Timing** turned on, the TimeQuest Timing Analyzer uses the results of simulations of the I/O buffer, package, and board trace model to generate more accurate I/O delays and extra reports to give insight into signal behavior at the system level. You can use these advanced timing reports as a guide to make changes to your I/O assignments and board design to improve timing and signal integrity.

This section details the first and second methods. The third method of analysis, the creation of simulation model files for use by third-party board simulation tools, is achieved with the IBIS and HSPICE Writers. The IBIS and HSPICE Writers in the Quartus II software can export accurate simulation models for use in applications such as Mentor Graphics HyperLynx and Synopsys HSPICE.

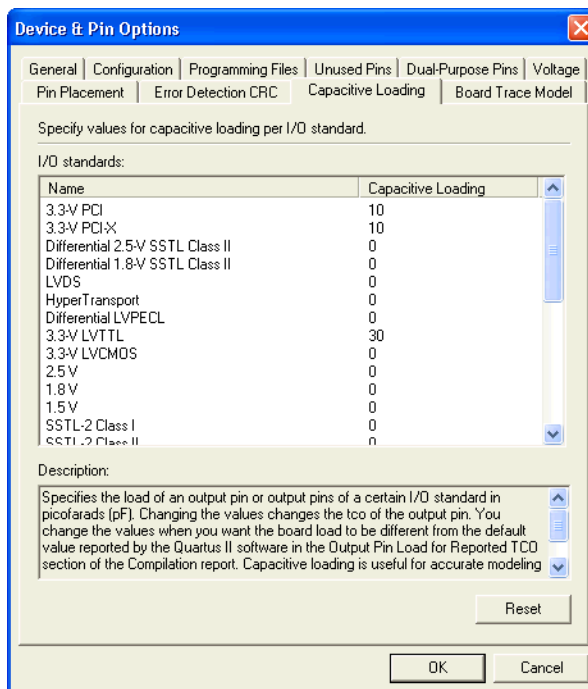


For information about creating IBIS and HSPICE models with the Quartus II software and integrating those models into HyperLynx and HSPICE simulations, refer to the *Signal Integrity Analysis with Third-Party Tools* chapter in volume 3 of the *Quartus II Handbook*.

Default I/O Timing & Power with Capacitive Loading

When calculating t_{co} and power for output and bidirectional pins, the TimeQuest Timing Analyzer, the Classic Timing Analyzer, and power analysis use a bulk capacitive load. This is the default method for these pins. You can adjust the value of the capacitive load per I/O standard to get t_{co} and power measurements that more accurately reflect the behavior of the output or bidirectional net on your PCB. Input pins ignore this setting. To do this, on the Assignments menu, click **Device**. Click **Device & Pin Options**, and click the **Capacitive Loading** tab (Figure 5-50).

Figure 5–50. Capacitive Tab of the Device & Pin Options Dialog Box



All of the available I/O standards for your selected device are listed with their default loading values in picofarads (pF). Adjust the loading values as desired for the I/O standards used in your design. Power and t_{co} measurements in the Compilation Report are adjusted based on the settings.



You can also adjust the load on any individual pin in the **Groups** or **All Pins** lists in the Pin Planner by adding the Output Pin Load column. Right-click anywhere in either list and select **Customize Columns**. Select **Output Pin Load** from the list of available custom columns, and add it to the list of visible columns. You can customize the load for individual pins or multiple pins with different I/O standards.



For more information about capacitive loading, the devices that support it, and how t_{co} and power are adjusted based on the setting, see Quartus II Help.

Enabling & Configuring Advanced I/O Timing

With the Quartus II **Enable Advanced I/O Timing** turned on, you can expand upon the basic timing and power measurements made with the **Capacitive Loading** settings. Advanced I/O Timing gives you the ability to fully define not only the capacitive load, but also any termination components and trace impedances in the board routing for any output pin or bidirectional pin in output mode. You can configure an overall board trace model for each I/O standard as well as customize the model for specific pins using a graphical interface.

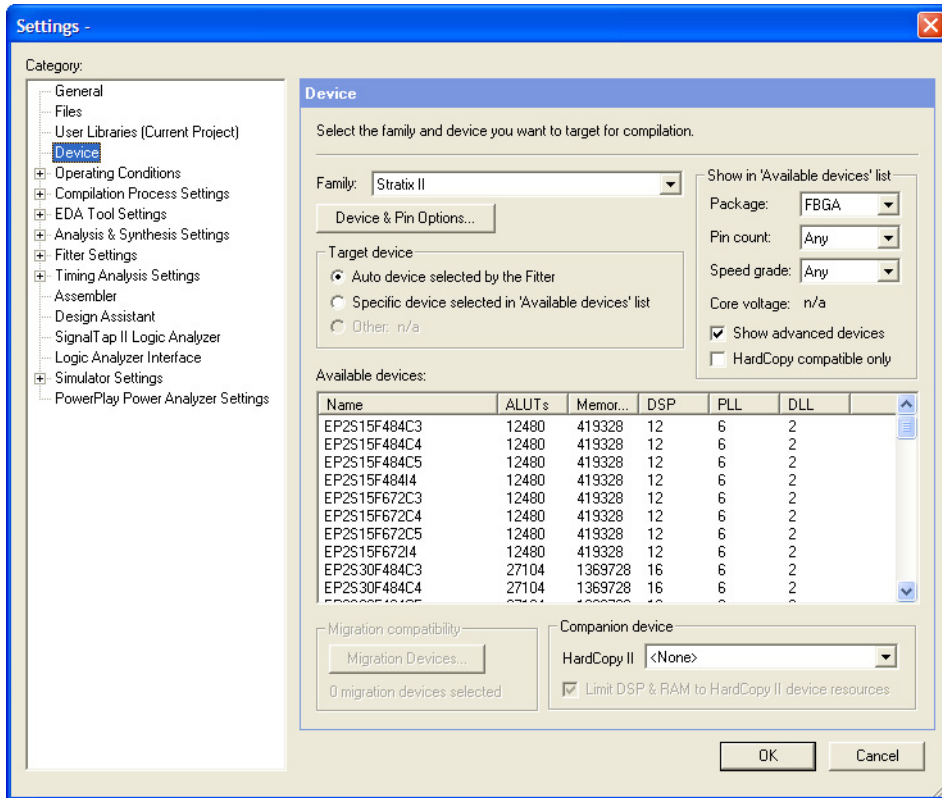
When the **Enable Advanced I/O Timing** option is turned on, the board trace model replaces the **Capacitive Loading** tab settings because the load is included in the model. For timing measurements, the entire board trace model is taken into account when calculating I/O delays. For power measurements, an effective capacitive load is used based on the sum of the capacitive elements in the model. This includes the **Near capacitance**, **Far capacitance**, and **Transmission line distributed capacitance** elements of the model.



Advanced I/O Timing is currently supported only for Stratix II devices. All other devices use Capacitive Loading only for I/O t_{co} and power measurements. Check the Altera web site at www.altera.com to determine which devices are supported in newer versions of the Quartus II software.

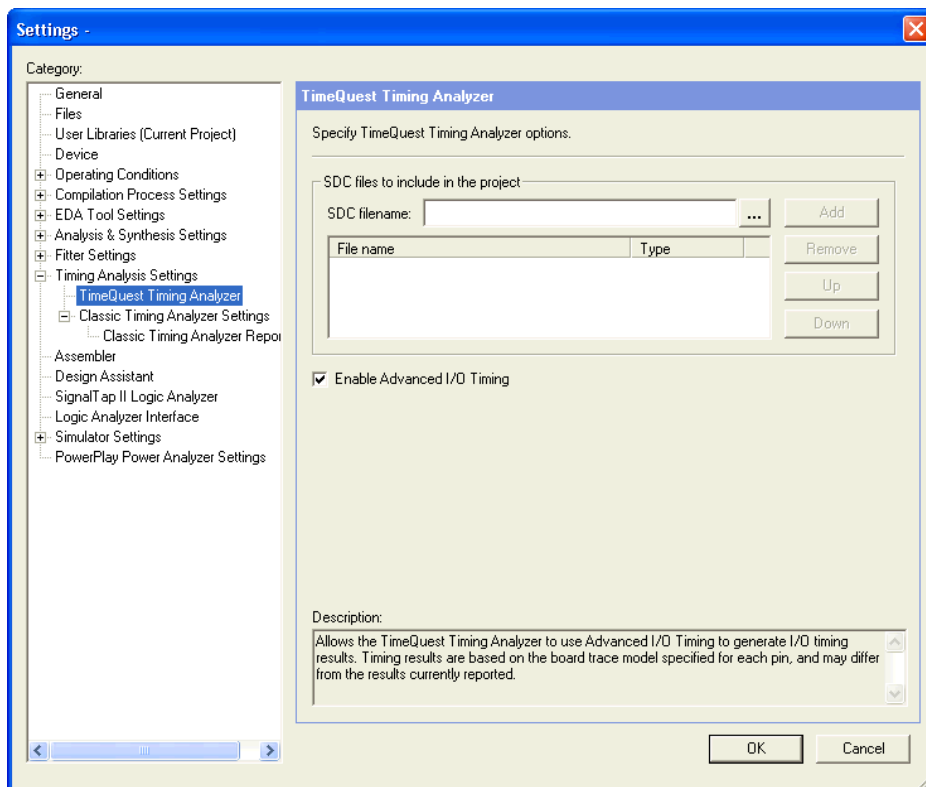
Before you can configure a board trace model for Advanced I/O Timing, you must select a device from a supported device family for your design and you must turn on the Advanced I/O Timing option. To select a device that supports Advanced I/O Timing, on the Assignments menu, click **Device** to open the **Settings** dialog box with the **Device** page selected (Figure 5-51). From the **Family** list, select **Stratix II**. You can set the other controls under **Show in 'Available devices list'** to filter the **Available devices** list and to select any migration devices. Under the **Available devices** list, select a device. All devices in a supported family work with Advanced I/O Timing.

Figure 5–51. Device Page of the Settings Dialog Box



Now you must turn on **Enable Advanced I/O Timing**. If the **Settings** dialog box is not currently open, on the **Assignments** menu, click **Settings**. In the **Category** list, click the **+** icon to expand **Timing Analysis Settings**. Select **TimeQuest Timing Analyzer**. The **TimeQuest Timing Analysis** page appears (Figure 5–52). Turn on **Enable Advanced I/O Timing**.

Figure 5–52. TimeQuest Timing Analyzer Settings

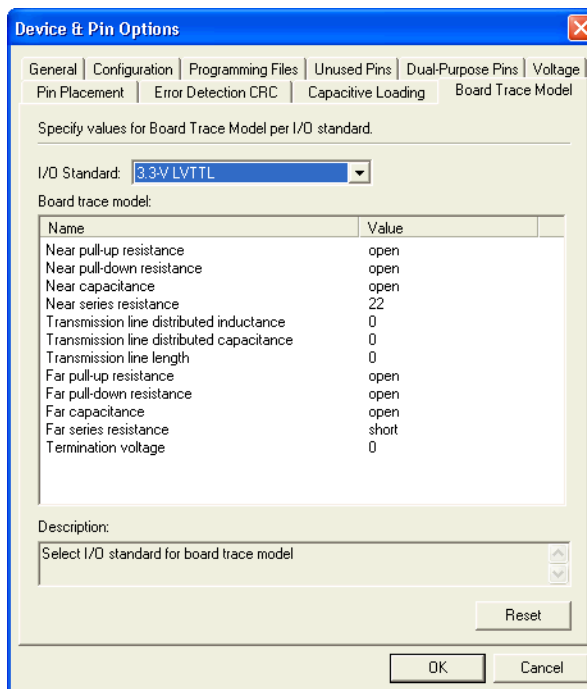


Define Overall Board Trace Models

You can now define an overall board trace model for each I/O standard in your design. This is the default model for all pins that use a particular I/O standard. After configuring the overall board trace model, you can customize the model for specific pins using the Board Trace Model view in the Pin Planner.

With the **Settings** dialog box open, in the **Category** list, click **Device**. Click **Device & Pin Options** and click the **Board Trace Model** tab.

Figure 5–53. Board Trace Model Tab of the Device & Pin Options Dialog Box



You can still click the **Capacitive Loading** tab. However, because you can configure all capacitive loading settings as part of the board trace model, the tab indicates that you must use the settings in the **Board Trace Model** tab.

All of the I/O standards available to the device are listed. Select any I/O standard from the list. The **Board trace model** list displays the names and values of all configurable components of the board trace for the selected I/O standard. Components of the model are initially set to **short**, **open**, or a numeric value depending on the component. The default settings for components in the model for each I/O standard are device-specific and match the default test model used for calculating delay when the **Enable Advanced I/O Timing** option is turned off. In this way, default delay measurements are the same whether or not the **Enable Advanced I/O Timing** option is used.



For information about the default models used for measuring I/O delay, refer to the *DC & Switching Characteristics* chapter of the *Stratix II Device Handbook*.

All of the component values listed in [Figure 5–53](#) are adjustable. For differential I/O standards, the component values you set are used for both the positive and negative signals of a differential pair. An additional component, **Far differential resistance**, is also included. To reset individual settings to their defaults, leave the setting blank. If you want all the settings for an I/O standard to revert to their original settings, click **Reset**. Click **OK** to close the **Device & Pin Options** dialog box. Click **OK** again to close the **Settings** dialog box.

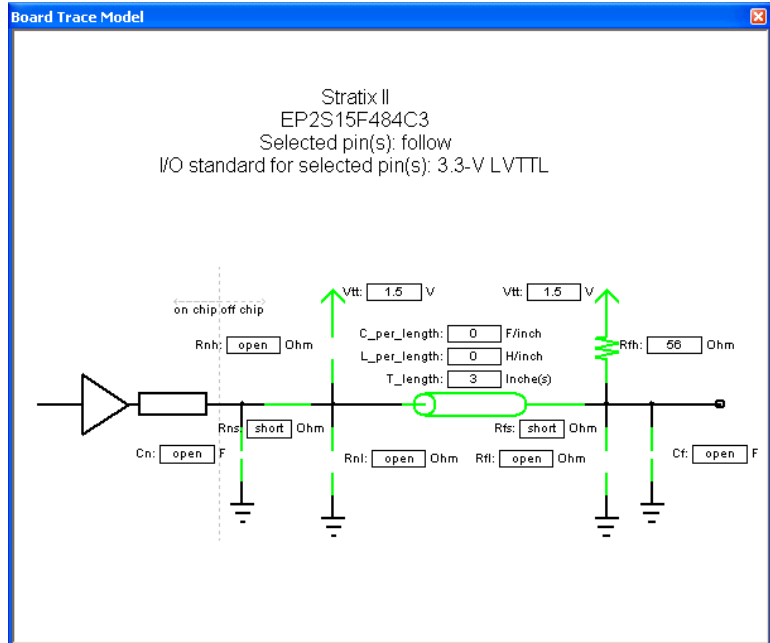


Any component value changes made in the **Board Trace Model** tab for a particular I/O standard are reflected in the Board Trace Model view in the Pin Planner of all pins assigned with the same I/O standard (described in “[Customize the Board Trace Model in the Pin Planner](#)”). However, custom component value changes made to selected pins in the Board Trace Model view in the Pin Planner take priority and are not affected by changes made to an I/O standard in the **Board Trace Model** tab.

Customize the Board Trace Model in the Pin Planner

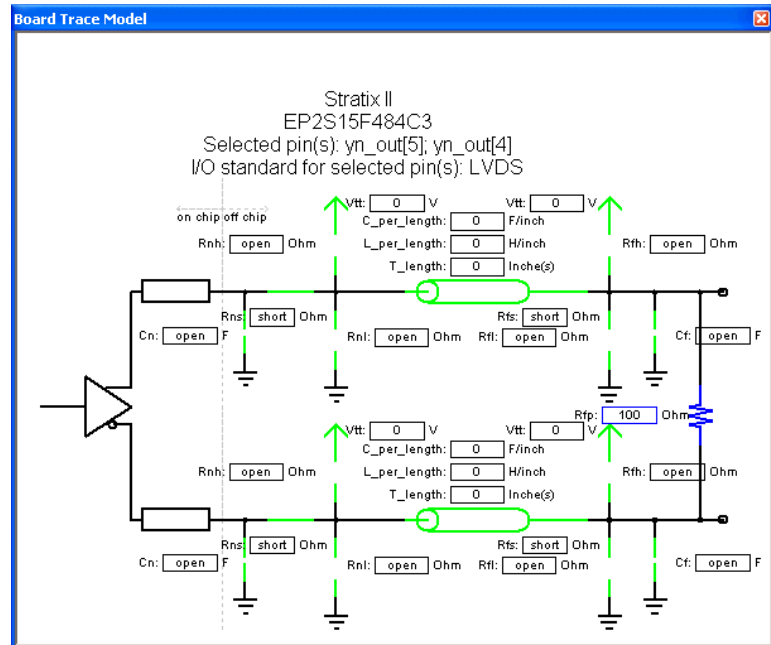
Along with the other views available in the Package view in the Pin Planner, you can also view a graphical representation of the board trace model you have configured using the Board Trace Model view. To open the Board Trace Model view, right-click an output or bidirectional pin in the **Groups** list, the All Pins list, or the Package view and click **Board Trace Model**. The Board Trace Model view opens in a floating window ([Figure 5–54](#)).

Figure 5–54. Board Trace Model View



For differential signals, the Board Trace Model view displays the routing and components for both the positive and negative signals of the differential pair (Figure 5–55).

Figure 5–55. Differential Board Trace Model View



Any changes made to the Board Trace Model view for a differential signal pair must be performed on the positive signal of the pair. The settings must match between the positive and negative signals of a differential pair, so the changes are automatically reflected in the settings for the negative signal.

Double-click a component value to edit it. For numerical values, use standard unit prefixes such as *p*, *n*, and *k* to represent pico, nano, and kilo, respectively. To short across a series component or have an open circuit for a parallel component, double-click the component value and select **short** or **open** from the list.



For more details about configuring component values for a board trace model, including a complete list of the supported unit prefixes, refer to Quartus II Help.

To view a display of a model for a particular pin, click on the pin in the Package view, **Groups** list, or **All Pins** list. This changes the Board Trace Model view to display the model of the pin. You can select multiple pins that share the same I/O standard, open the Board Trace Model view, and edit the model for all of the selected pins. If an input pin or multiple pins

with different I/O standards are selected, the Board Trace Model view window indicates that it cannot display the model for the selected pin or pins.

The components in the Board Trace Model view correspond to the components listed in the **Board Trace Model** tab directly, and the settings match initially. You can click and edit any value in the Board Trace Model view to customize the model for the selected pin or pins. Changes made in the Board Trace Model view do not affect the settings in the **Board Trace Model** tab.

To configure board trace models for the pins in your design efficiently with these two methods of entry, define the model for each I/O standard in the **Board Trace Model** tab. With the overall model defined, use the Board Trace Model view in the Pin Planner to customize individual pins as needed. These customizations take priority over the settings in the **Board Trace Model** tab on a per pin and per model component basis, so they will not affect the settings on any other pin.

Create Signal Integrity Result Reports

Once you have turned on **Enable Advanced I/O Timing** and configured board trace models for the pins you want to analyze, compile your project or run the TimeQuest Timing Analyzer after a full compilation. The **Enable Advanced I/O Timing** option creates signal integrity sub-reports under TimeQuest Timing Analyzer in the Compilation Report window.

The Board Trace Model Assignments report (Figure 5–56) summarizes the board trace model component settings for each output and bidirectional signal.

Figure 5–56. Board Trace Model Assignments Report

Pin	I/O Standard	Near Series R	Near Pull-up R	Near Pull-down R	Near C	Time Length	Time L per Length	Time C per Length	Far Series R	Far Pull-up R	Far Pull-down R	Far C	Termination Voltage
1	yvalid	3.3V LVTTTL	short	open	open	0	0	0	short	open	open	open	0
2	follow	3.3V LVTTTL	short	open	open	0	0	0	short	open	open	open	0
3	yn_out[7]	3.3V LVTTTL	22	open	open	0	0	0	short	open	open	open	30P
4	yn_out[5]	3.3V LVTTTL	22	open	open	0	0	0	short	open	open	open	30P
5	yn_out[5]	3.3V LVTTTL	22	open	open	0	0	0	short	open	open	open	30P
6	yn_out[4]	3.3V LVTTTL	22	open	open	0	0	0	short	open	open	open	30P
7	yn_out[3]	3.3V LVTTTL	22	open	open	0	0	0	short	open	open	open	30P
8	yn_out[2]	3.3V LVTTTL	22	open	open	0	0	0	short	open	open	open	30P
9	yn_out[1]	3.3V LVTTTL	22	open	open	0	0	0	short	open	open	open	30P
10	yn_out[0]	3.3V LVTTTL	22	open	open	0	0	0	short	open	open	open	30P

The Signal Integrity Metrics subfolder contains detailed reports listing all of the metrics calculated by the Enable Advanced I/O Timing option (Figure 5–57).

Figure 5–57. Example of Slow-Corner Signal Integrity Metrics Report

Pin	I/O Standard	Board Delay on Rise	Board Delay on Fall	Steady State Voh at FPGA Pin	Steady State Vol at FPGA Pin	Voh Max at FPGA Pin	Vol Min at FPGA Pin	Ringback Margin on Rise at FPGA Pin	Ringback Margin on Fall at FPGA Pin	10-90 Pin at FPGA
1	yvalid	3.3V LVTTTL 0.000e+000	0.000e+000	3.134e+000	6.459e-004	3.136e+000	-3.629e-002	1.454e-002	1.035e-001	5.517e-0
2	follow	3.3V LVTTTL 0.000e+000	0.000e+000	3.134e+000	6.459e-004	3.136e+000	-4.066e-002	1.004e-003	2.089e-001	4.697e-0
3	yn_out[7]	3.3V LVTTTL 7.015e-010	6.882e-010	3.134e+000	6.488e-004	3.134e+000	6.488e-004	3.386e-002	1.309e-002	4.545e-0
4	yn_out[5]	3.3V LVTTTL 7.015e-010	6.882e-010	3.134e+000	6.488e-004	3.134e+000	6.488e-004	3.386e-002	1.309e-002	4.545e-0
5	yn_out[4]	3.3V LVTTTL 7.001e-010	6.869e-010	3.134e+000	6.488e-004	3.134e+000	6.488e-004	3.567e-002	1.388e-002	4.600e-0
6	yn_out[3]	3.3V LVTTTL 7.001e-010	6.869e-010	3.134e+000	6.488e-004	3.134e+000	6.488e-004	3.567e-002	1.388e-002	4.600e-0
7	yn_out[2]	3.3V LVTTTL 7.015e-010	6.882e-010	3.134e+000	6.488e-004	3.134e+000	6.488e-004	3.386e-002	1.309e-002	4.545e-0
8	yn_out[1]	3.3V LVTTTL 7.001e-010	6.869e-010	3.134e+000	6.488e-004	3.134e+000	6.488e-004	3.567e-002	1.388e-002	4.600e-0
10	yn_out[0]	3.3V LVTTTL 7.015e-010	6.882e-010	3.134e+000	6.488e-004	3.134e+000	6.488e-004	3.386e-002	1.309e-002	4.545e-0

The Slow- and Fast-Corner Signal Integrity Metrics reports are generated by the Enable Advanced I/O Timing option. They list, in tabular format, all of the signal integrity metrics calculated by the **Enable Advanced I/O Timing** option, based on the board trace model settings for each output or bidirectional pin. The reports contain many metrics, including measurements at both the FPGA and at the far-end load of board delay, steady state voltages, and rise and fall times.

The slow- or fast-corner reports are generated depending on the Timing Netlist option in the TimeQuest Timing Analyzer. To select whether to create a slow- or a fast-corner report, in the TimeQuest Timing Analyzer on the Netlist menu, click **Create Timing Netlist**. Under Delay model, select **Slow corner** or **Fast corner** to create reports of that type.



For complete descriptions about all of the metrics calculated when the Enable Advanced I/O Timing option is turned on and diagrams illustrating the metrics on output waveforms, refer to Quartus II Help. For more information about board level signal integrity and tips on how to improve signal integrity in your high-speed designs, refer to the Altera Signal Integrity Center. For information about the configuration and use of the TimeQuest Timing Analyzer, refer to Quartus II Help or *Section II. Timing Analysis* in volume 3 of the *Quartus II Handbook*.

Conclusion

The Quartus II software provides many tools and features to help you with the I/O planning process. The I/O assignment analysis process offers the ability to validate pin assignments in all design stages, even before the development of the design. The ability to import and export assignments between the Quartus II software and other PCB tools also enables you to make iterative changes efficiently. Finally, the ability to enter a board trace model and create advanced timing reports based on how I/O signals are routed on a board truly makes the Quartus II software “board-aware.”

Document Revision History

Table 5–7 shows the revision history for this document.

Date / Version	Changes Made	Summary of Changes
November 2006 v6.1.0	<ul style="list-style-type: none"> ● Updated text and graphics to reflect GUI changes. ● Added information about setting up and creating a top-level design file from megafunctions and IP MegaCores created in the Pin Planner. ● Added spreadsheet functionality information to lists in the Pin Planner. ● Added descriptions of new reports generated by I/O Assignment Analysis. ● Added information the Advanced I/O Timing option, including the configuration of board trace models. 	
May 2006 v6.0.0	Updated for the Quartus II software version 6.0.0. <ul style="list-style-type: none"> ● Updated text and graphics to reflect the GUI changes. ● Added pin filtering information. ● Added pin assignments and Pad View information. ● Added Package view information. 	
October 2005 v5.1.0	<ul style="list-style-type: none"> ● Updated for the Quartus II software version 5.1.0. ● I/O Assignment Analysis material incorporated into chapter. 	
May 2005 v5.0.0	Initial release.	

Introduction

With today's large, high-pin-count and high-speed FPGA devices, good and correct printed circuit board (PCB) design practices are more essential than ever for ensuring correct system operation. Typically, the PCB design takes place concurrently with the design and programming of the FPGA. Signal and pin assignments are initially made by the FPGA or ASIC designer, and the board designer must correctly transfer these assignments to the symbols used in their system circuit schematics and board layout. As the board design progresses, pin reassignments may be needed to optimize the PCB layout. These reassignments must in turn be relayed back to the FPGA designer so that the new assignments can be processed through an updated placement and routing of the FPGA design.

Mentor Graphics® provides tools to support this type of design flow. This chapter discusses how the Quartus® II software interacts with the Mentor Graphics I/O Designer software and the DxDesigner software to provide a completely cyclical FPGA-to-board integration design workflow. This chapter covers the following topics:

- General design flow between the Quartus II software, the Mentor Graphics I/O Designer software, and the DxDesigner software
- Setting up the Quartus II software to create the design flow files
- Creating an I/O Designer database project to incorporate the Quartus II software signal and pin assignment data
- Updating signal and pin assignment changes between the I/O Designer software and the Quartus II software
- Generating symbols in the I/O Designer software
- Creating symbols in the DxDesigner software from the Quartus II software output files without the use of the I/O Designer software

This chapter is intended primarily for board design and layout engineers who want to start the FPGA board integration while the FPGA is still in the design phase. Optionally, the board designer can plan the FPGA pinout and routing requirements in the Mentor Graphics tools and pass the information back to the Quartus II software for place-and-route. In addition, part librarians benefit from learning how to take output from the Quartus II software and use it to create new library parts and symbols.

The procedures in this chapter require the following software:

- The Quartus II software version 5.1 or higher
- DxDesigner software version 2004 or higher

Mentor Graphics I/O Designer software is optional.

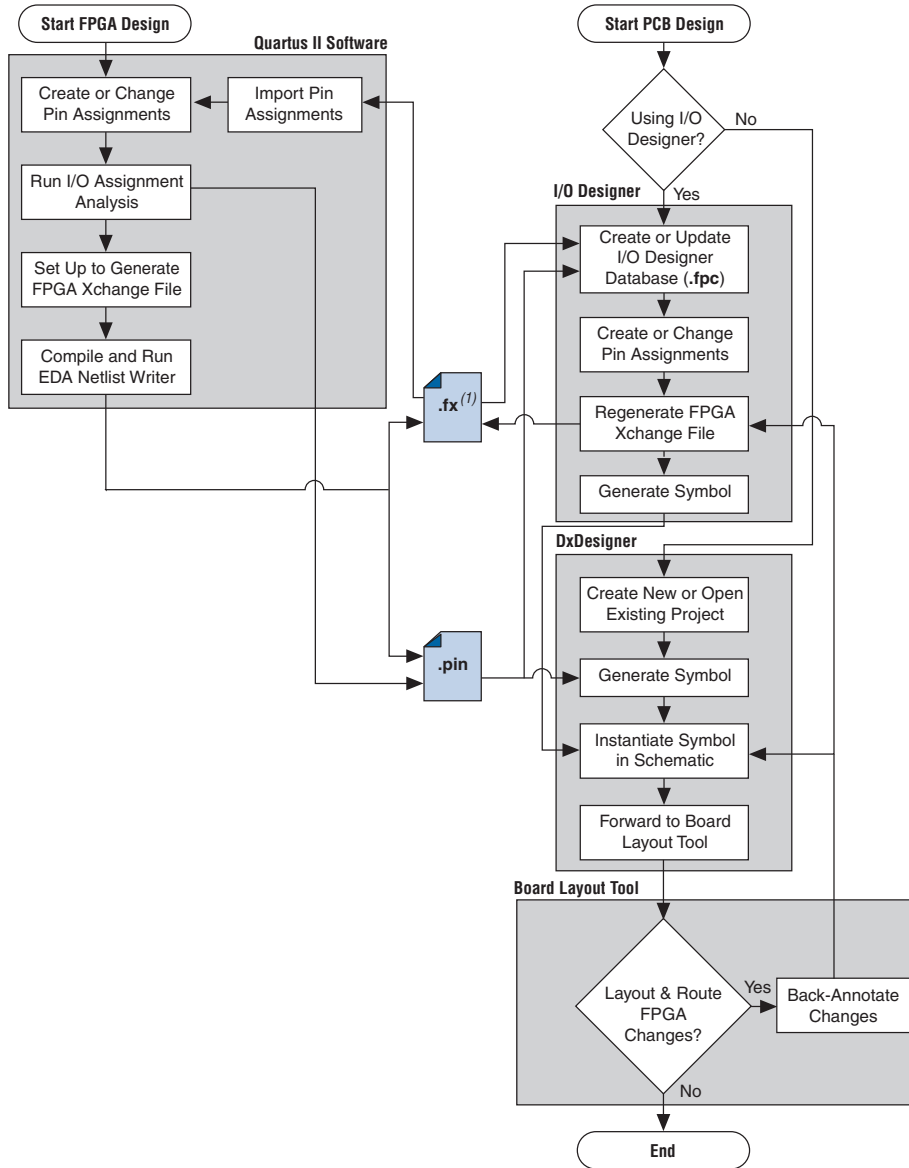


To obtain and license the Mentor Graphics tools and obtain product information, support, and training, go to the Mentor Graphics website at www.mentor.com.

FPGA-to-PCB Design Flow

In the examples in this section, you create a design flow integrating an Altera® FPGA design from the Quartus II software, and a circuit schematic in the DxDesigner software. [Figure 6–1](#) shows the design flow with and without the I/O Designer software.

Figure 6–1. Design Flow with & without the I/O Designer Software



Note to Figure 6–1:

- (1) The Quartus II software generates the FPGA Xchange file in the output directory you specify in the Board-Level Assignment Settings. However, the Quartus II software and the I/O Designer software can import pin assignments from an FPGA Xchange file located in any directory. Altera recommends that you work with a backup of the FPGA Xchange file to prevent overwriting existing assignments or importing invalid assignments.

The following tasks, which are described in this chapter, describe how to proceed through the design flow shown in [Figure 6-1](#):

- Set up the board-level assignment settings to generate an FPGA Xchange file (.fx) for symbol generation in the Quartus II software
- Compile the design and generate the FPGA Xchange file and the Pin-Out file (.pin), which are located in the Quartus II project directory
- Create a board design using the DxDesigner software together with the I/O Designer software, which involves the following steps:
 - Create a new I/O Designer database based on the FPGA Xchange file and the Pin-Out file
 - Make adjustments to signal and pin assignments in the I/O Designer software
 - Regenerate the FPGA Xchange file in the I/O Designer software to reflect the I/O Designer software changes in the Quartus II software
 - Generate a single or fractured symbol for use in the DxDesigner software
 - Add the symbol to the **sym** directory of a DxDesigner project, or specify a new DxDesigner project with the new symbol
 - Instantiate the symbol in your DxDesigner schematic and export the design to the board layout tool
 - Back-annotate pin changes created in the board layout tool to the DxDesigner software and back to the I/O Designer software and the Quartus II software
- Create a board design using the DxDesigner software without the I/O Designer software, which involves the following steps:
 - Create a new DxBoardLink symbol using the Symbol Wizard and reference the Pin-Out file output from the Quartus II software in an existing DxDesigner project
 - Instantiate the symbol in your DxDesigner schematic and pass the design to a board layout tool

The I/O Designer software allows you to take advantage of the full FPGA symbol design, creation, editing, and back-annotation flow supported by Mentor Graphics tools.

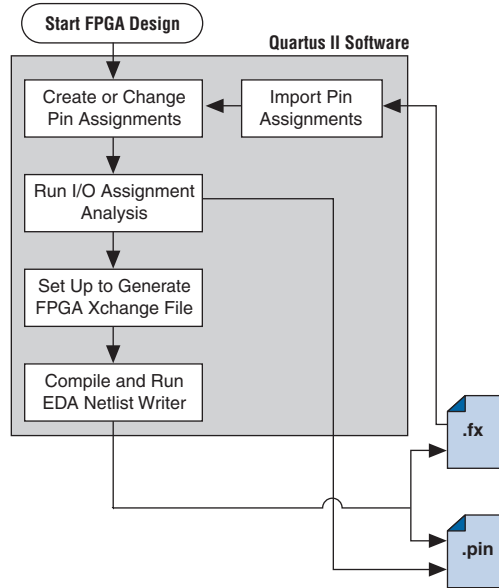


Symbols can be updated with design changes at any point with or without the I/O Designer software. However, if symbols are changed in the DxDesigner software, the I/O Designer software does not see the changes. If you change symbols using the DxDesigner software, you must reimport the symbols into I/O Designer to avoid overwriting your symbol changes.

Setting Up the Quartus II Software

You can transfer pin and signal assignments from the Quartus II software to the Mentor Graphics tools by generating two output files, a Pin-Out file (.pin) and an FPGA Xchange file (.fx) (Figure 6-2).

Figure 6-2. Pin-Out Files & FPGA Xchange Files *Note (1)*



Note to Figure 6-2:

- (1) Refer to Figure 6-1 for the full design flow, which includes the I/O Designer software, the DxDesigner software, and the board layout tool flowchart details.

The two output files, the Pin-Out file and the FPGA Xchange file, are described in [Table 6-1](#).

Table 6-1. Pin Assignment Output File Format Comparison

File Format	Description
Pin-Out file (.pin) (1)	An output file generated by the Quartus II Fitter. The file cannot be imported into the Quartus II software to change pin assignments. The file contains a complete list of the device pins including any unused I/O pins, and provides the following basic information fields for each assigned pin on a device: <ul style="list-style-type: none"> ● Pin signal name/usage ● Pin number ● Signal direction ● I/O standard ● Voltage ● I/O Bank ● User or fitter assigned
FPGA Xchange file (.fx) (1),(2)	An input/output file generated by the Quartus II software and the I/O Designer software that can be imported and exported from both programs. Industry standard with room for future changes and additions. The FPGA Xchange file generated by the Quartus II software lists only assigned pins. The file provides the following advanced information fields for each pin on a device: <ul style="list-style-type: none"> ● Pin number ● I/O Bank ● Signal name ● Signal direction ● I/O standard ● Drive strength (mA) ● Termination enabling ● Slew rate ● IOB Delay ● Swap group ● Differential pair type When generated by the I/O Designer software, all pins, including unused pins, are listed and the following fields are added: <ul style="list-style-type: none"> ● Swap group ● Differential pair type ● Device pin name ● Pin set ● Pin set position ● Pin set group ● Super pin set group ● Super pin set position

Notes to Table 6-1:

- (1) For additional information about these file formats, refer to the Quartus II Help.
- (2) For additional information about the information fields added by the Mentor Graphics software, refer to the Mentor Graphics website at www.mentor.com.

The I/O Designer software can also read from or update a Quartus II Settings File (.qsf). The Quartus II Settings File is used in the design flow in a similar manner to the FPGA Xchange file, but does not transfer pin swap group information between the I/O Designer software and the Quartus II software.



The **Quartus II Settings File** also contains additional important information about your project that is not used by the I/O Designer software. Because of this, Altera recommends that you use the FPGA Xchange file instead of the Quartus II Settings File for this design flow



For more information about the Quartus II Settings File, refer to the *Quartus II Settings File Reference Manual*.

Generating Pin-Out Files

The Quartus II Fitter generates the Pin-Out file whenever you perform a full compilation or I/O Assignment Analysis on your design. The file is generated and placed in your design directory and your file is named *<project name>.pin*. The Mentor Graphics tools do not alter this file. The Quartus II software cannot import assignments from an existing Pin-Out file.

Generating FPGA Xchange Files

The FPGA Xchange file is not created automatically. To set up the Quartus II software to create the FPGA Xchange file, follow these steps:

1. Start the Quartus II software. On the Assignments menu, click **Settings**. The Settings dialog box appears.
2. Under EDA Tool Settings, click **Board-Level**. In the Board-Level Symbol Format list, choose **FPGA Xchange**.
3. Set the Output directory to the location where you want to save the file. The default output file path is *<project directory>/symbols/fpgaxchange*. Click **OK**.
4. On the Processing menu, point to Start and click **Start EDA Netlist Writer**.

The output directory you selected is created when you generate the FPGA Xchange file.



Both the Quartus II software and the I/O Designer software can export and import an FPGA Xchange file. It is therefore possible to overwrite the FPGA Xchange file and import incorrect assignments into one or both programs. To prevent this occurrence from happening, make a backup copy of the file before importing, and import the copy instead of the file generated by the Quartus II software. In addition, assignments in the Quartus II software can be protected by following the steps in “[Protecting Assignments in the Quartus II Software](#)” on page 6–23.

Creating a Backup Quartus II Settings File

To create a backup Quartus II Settings File, perform the following steps:

1. On the Assignments menu, click **Import Assignments**. The Import Assignments dialog box appears.
2. In the Import Assignments dialog box, browse to your project and turn on **Copy existing assignments into** *<project name>.qsf.bak*.
3. Click **OK**.

Following these steps automatically creates a backup Quartus II Settings File of your current pin assignments.



For more information about pin and signal assignment transfer, and files the Quartus II software can import and export, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

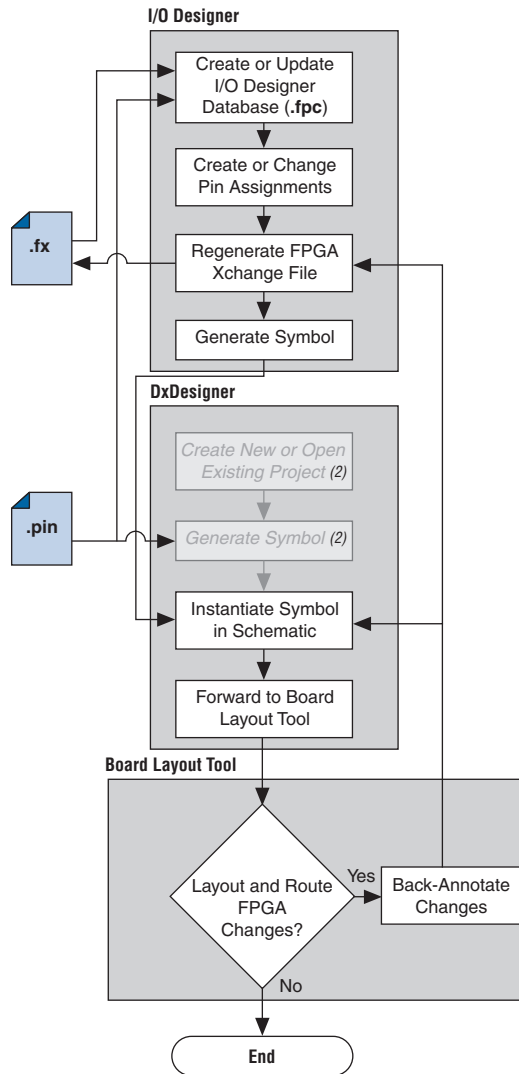
FPGA-to-Board Integration with the I/O Designer Software

The Mentor Graphics I/O Designer software allows you to integrate your FPGA and PCB designs. Pin and signal assignment changes can be made anywhere in the design flow, typically using either the Quartus II Pin Planner or the I/O Designer software. The I/O Designer software facilitates moving these changes, as well as synthesis, placement, and routing changes, between the Quartus II software, an external synthesis tool (if used), and a schematic capture tool such as the DxDesigner software.

This section describes how to use the I/O Designer software to transfer pin and signal assignment information to and from the Quartus II software with the FPGA Xchange file, and how to create symbols for the DxDesigner software.

[Figure 6–3](#) shows the design flow using the I/O Designer software.

Figure 6–3. Design Flow Using the I/O Designer Software *Note (1)*



Notes to Figure 6–3:

- (1) Refer to Figure 6–1 for the full design flow including the Quartus II software flowchart details.
- (2) These are DxDesigner software-specific steps in the design flow and are not part of the I/O Designer flow.



For more information about the I/O Designer software, and to obtain usage, support, and product updates, use the Help menu in the I/O Designer software or refer to the Mentor Graphics website at www.mentor.com.

I/O Designer Database Wizard

All I/O Designer project information is stored in an I/O Designer Database (.fpc) file. You can create a new database that incorporates the FPGA Xchange file and Pin-Out file information generated by the Quartus II software by using the I/O Designer Database Wizard. You can also create a new, empty database and manually add the assignment information. If there is no signal or pin assignment information currently available, you can create an empty database that contains only a selection of the target device. This is useful if you know the signals in your design and the pins you want to assign. You can transfer this information at a later time to the Quartus II software for place-and-route.

It is possible to create an I/O Designer database with only one type of file or the other. However, if only a Pin-Out file is used, any I/O assignment changes made in the I/O Designer software cannot be imported back into the Quartus II software without first generating an FPGA Xchange file. If only an FPGA Xchange file is used to create the I/O Designer database, the database may not contain a complete picture of all of the I/O assignment information available. The FPGA Xchange file generated by the Quartus II software only lists pins with assigned signals. Since the Pin-Out file lists all device pins—whether signals are assigned to them or not—its use, along with the FPGA Xchange file, produces the most complete set of information for creating the I/O Designer Database.

To create a new I/O Designer database using the Database Wizard, perform the following steps:



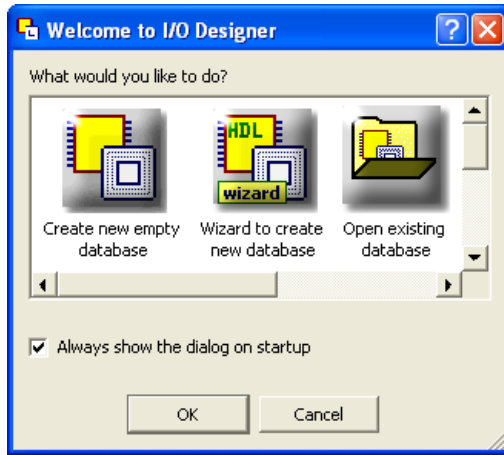
If you skip a step in this process, you can complete the skipped step later, filling in the appropriate information. To return to a skipped step, on the Properties menu, click **File**.

1. Start the I/O Designer software. The **Welcome to I/O Designer** dialog box appears (Figure 6-4). Select **Wizard to create new database** and click **OK**.



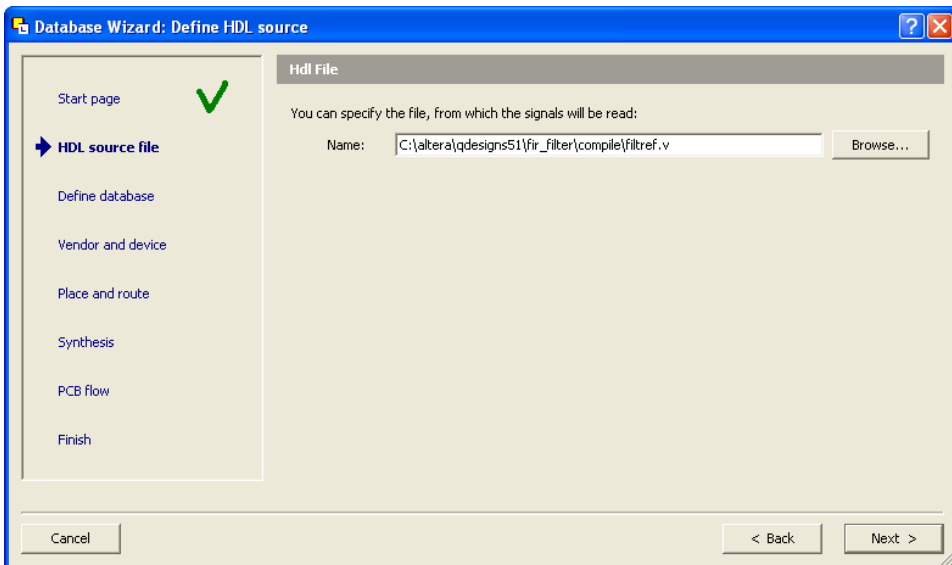
If the **Welcome to I/O Designer** dialog box is not shown because it was disabled, you can access the Wizard through the menus. To access the Wizard on the File menu, click **Database Wizard**.

Figure 6–4. I/O Designer Welcome Dialog



2. Click Next. The Define HDL source file page opens (Figure 6–5).

Figure 6–5. Database Wizard HDL File Page





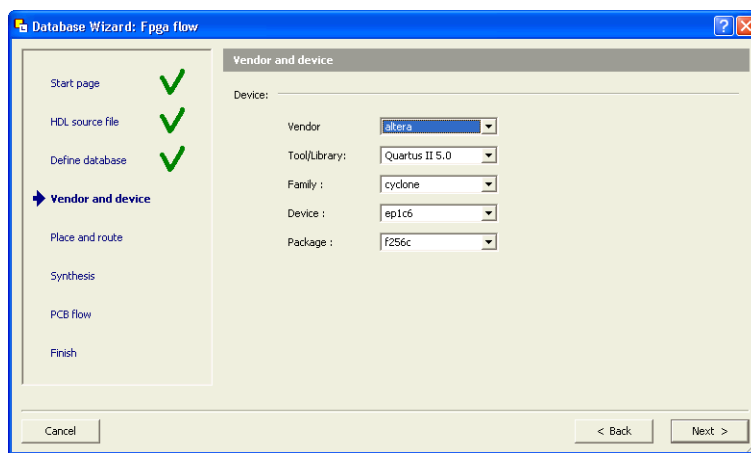
For more information about creating and using HDL files in the Quartus II software, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*, or refer to the I/O Designer Help.



If no HDL files are available, or if your signal and pin assignments are already contained in the FPGA Xchange file, you do not have to complete step 3 and can proceed to step 4.

3. If you have created a Verilog HDL or VHDL file in your Quartus II software design, you can enter a top-level Verilog HDL or VHDL file. Adding a file allows you to create functional blocks or get signal names from your design. All physical pin assignments must be created in I/O Designer if no FPGA Xchange file or Pin-Out file is used. Click **Next**. The Database Name page is shown.
4. In the Database Name window, enter your database file name. Click **Next**. The Database Location window is shown.
5. Enter a path to the new database or an existing one in the **Location** field, or browse to a database location. Click **Next**. The **FPGA flow** page is shown (Figure 6–6).

Figure 6–6. Database Wizard Vendor & Device Page



6. In the Vendor menu, click **Altera**.
7. In the Tool/Library menu, click **Quartus II 5.0**, or a later version of the Quartus II software.

8. Select the appropriate device family, device, package, and speed (if applicable), from the corresponding menus. Click **Next**. The **Place and route** page is shown (Figure 6-7).

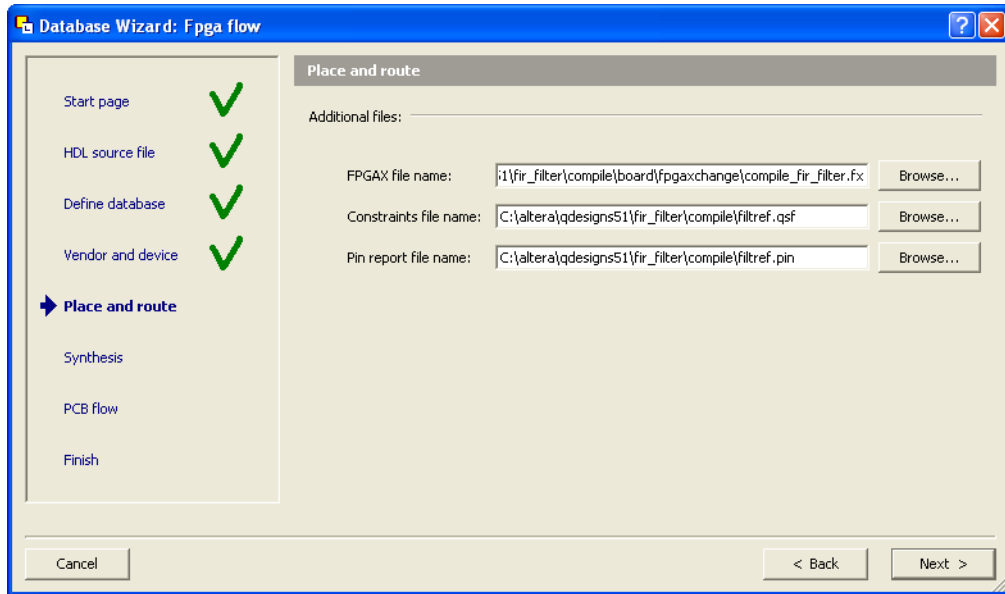


The Quartus II software version selections in the Tool/Library menu may not reflect the version of the Quartus II software currently installed on your system even if you are using the most current version of the I/O Designer software. The version number selection in this window is used in the I/O Designer software to identify the devices that were available or obsolete in that particular version of the Quartus II software. If you are unsure of the version to select, use the most recent version listed in the menu. If the device you are targeting does not appear in the device menu after making this selection, the device may be new and not yet added to the I/O Designer software. A list of unsupported devices for recent versions of the I/O Designer software can be found in Table 6-2. For I/O Designer software updates, contact Mentor Graphics or refer to their website at www.mentor.com.

Table 6-2. I/O Designer Unsupported Devices

Version of I/O Designer	Unsupported Devices
2005 (current release)	Stratix II GX
2004.2	Hardcopy II Stratix II GX

Figure 6–7. Database Wizard Place and Route Page



9. In the **FPGAX file name** field, type or browse to the backup copy of the FPGA Xchange file generated by the Quartus II software.
10. In the **Pin report file name** field, type or browse to the Pin-Out file generated by the Quartus II software. Click **Next**.

In addition, you can select a Quartus II Settings File for update. The I/O Designer software can update the pin assignment information in the Quartus II Settings File without affecting any other information contained in the file.



You can select a Pin-Out file without selecting an FPGA Xchange file for import. The I/O Designer software does not generate a Pin-Out file. To transfer assignment information to the Quartus II software, select an additional file and file type. Altera recommends selecting an FPGA Xchange file in addition to a Pin-Out file for transferring all of the assignment information contained within both types of files.



In some versions of the I/O Designer software, the standard file picker may incorrectly look for a Pin-Out file instead of an FPGA Xchange file. In this case, select **All Files (*.*)** from the **Save as type** list and select the file from the list.

11. The **Synthesis** page displays. On the **Synthesis** page, you can specify an external synthesis tool and a synthesis constraints file for use with the tool. If you do not use an external synthesis tool, click **Next**.



For more information about third-party synthesis tools, refer to volume 3 of the *Quartus II Handbook*.

12. The **PCB Flow** page is shown (Figure 6-8). On the **PCB Flow** page, you can select an existing schematic project or create a new project as a destination for symbol information.
 - To select an existing project, select **Choose existing project** and click **Browse** after the Project Path field. The **Select project** dialog box appears. Select the project.
 - To create a new project, in the **Select project** dialog box, select **Create new empty project**. Enter the project file name in the **Name** field and browse to the location where you want to save the file (Figure 6-9). Click **OK**.

Figure 6–8. PCB Flow Page

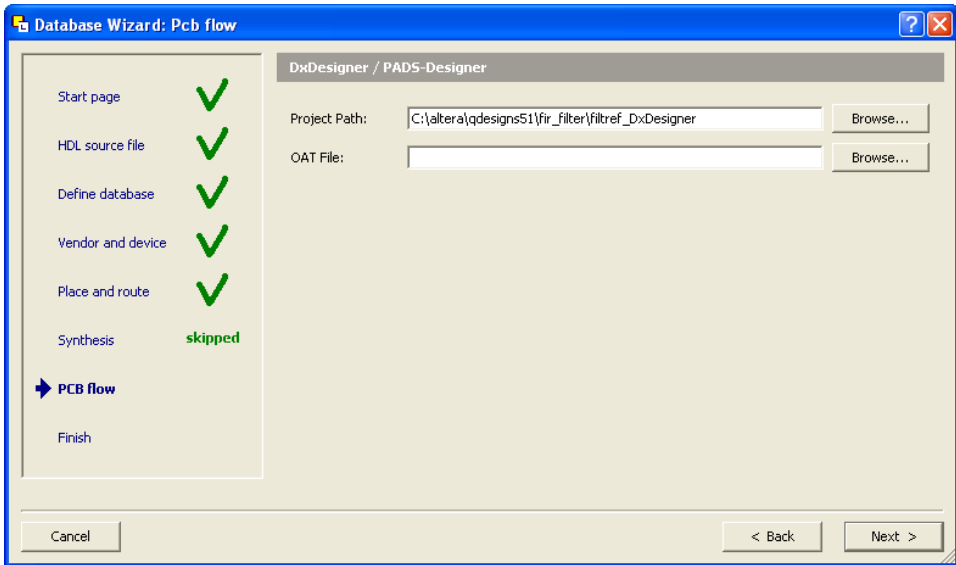
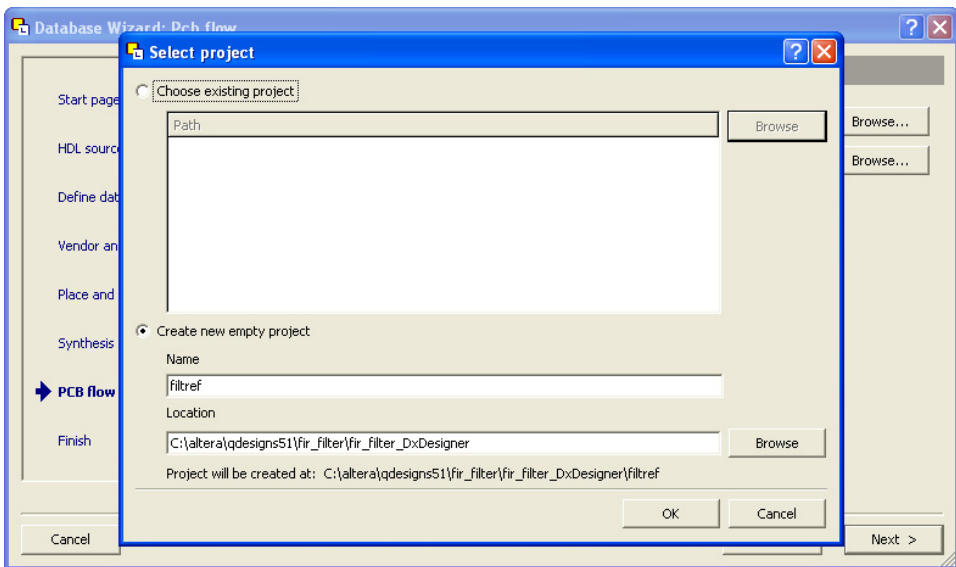


Figure 6–9. Select Project Dialog Box



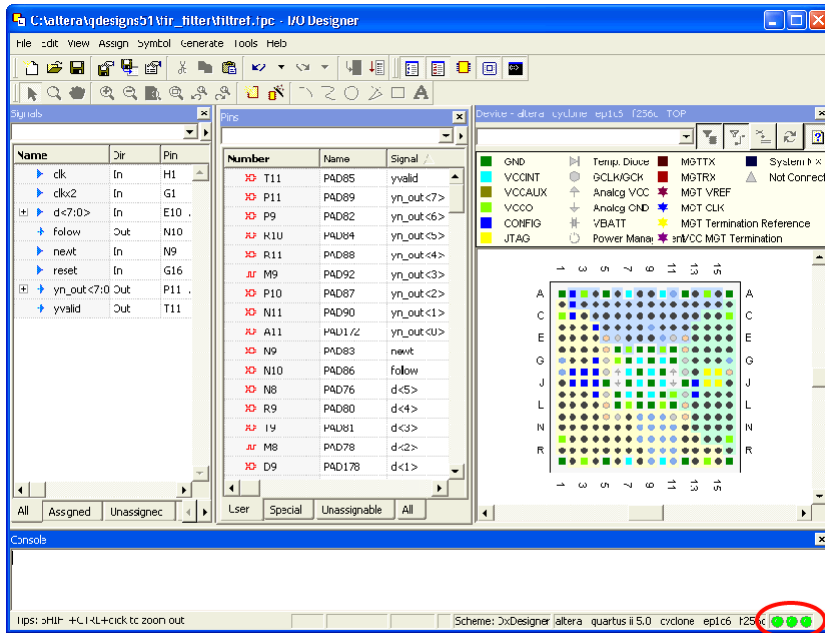
If you have not specified a design tool for sending symbol information to in I/O Designer, click **Advanced** in the **PCB Flow** page and select your design tool. If the DxDesigner software is selected, you have the option of specifying a Hierarchical Occurrence Attributes (.oat) file to import into the I/O Designer software (Figure 6–8). Click **Next**, then click **Finish** to create the database.



In I/O Designer version 2005 or later, the Update Wizard (refer to Figure 6–13 on page 6–21) is shown when you finish creating the database using the database wizard. Use the Update Wizard to confirm creation of the I/O Designer database using the selected FPGA Xchange and Pin-Out files.

Use the I/O Designer software and your newly created database to make pin assignment changes, create pin swap groups, or adjust signal and pin properties in the I/O Designer GUI (Figure 6–10).

Figure 6–10. I/O Designer Main Window



Database Update Indicator

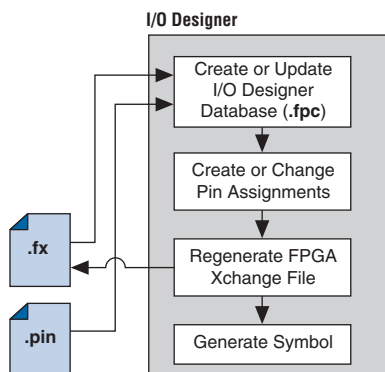


For more information about using the I/O Designer software and the DxDesigner software, refer to the Mentor Graphics website at www.mentor.com or refer to the I/O Designer software or the DxDesigner Help.

Updating Pin Assignments from the Quartus II Software

As the design process continues, the FPGA designer may need to make changes to the logic design in the Quartus II software that place signals on different pins after the design is recompiled, or manually by using the Quartus II Pin Planner. These types of changes must be carried forward to the circuit schematic and board layout tools to ensure that signals are connected to the correct pins on the FPGA. Updating the FPGA Xchange file and the Pin-Out file in the Quartus II software facilitates this flow (Figure 6–11).

Figure 6–11. Updating the I/O Designer Pin Assignments in the Design Flow
Note (1)



Note to Figure 6–11:

- (1) Refer to Figure 6–1 for the full design flow, which includes the Quartus II software, the DxDesigner software, and the board layout tool flowchart details.

To update the FPGA Xchange file and the Pin-Out file in the Quartus II software after making changes to the design, run a full compilation, or on the Start menu, point to Processing and click **Start EDA Netlist Writer**. The FPGA Xchange file in your selected output directory and the Pin-Out file in your project directory are updated. You must rerun the I/O Assignment Analyzer whenever you make I/O changes in the Quartus II software. To rerun the I/O Assignment Analyzer, on the Processing menu, click **Start Compilation**, or to run a full compilation, on the Processing menu, point to Start and click **Start I/O Assignment Analysis**.



Refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook* for more information about setting up the FPGA Xchange file and running the I/O Assignment Analyzer.

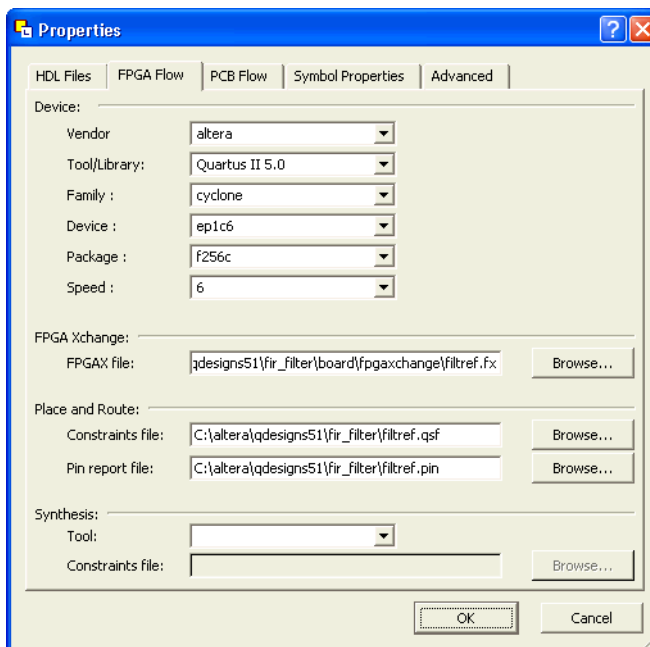


If your I/O Designer database points to the FPGA Xchange file generated by the Quartus II software instead of a backup copy of the file, updating the file in the Quartus II software overwrites any changes made to the file by the I/O Designer software. If there are I/O Designer assignments in the FPGA Xchange file that you want to preserve, create a backup copy of the file before updating it in the Quartus II software, and verify that your I/O Designer database points to the backup copy. To point to the backup copy, perform the steps in the following section.

Whenever the FPGA Xchange file or the Pin-Out file is updated in the Quartus II software, the changes can be imported into the I/O Designer database. You must set up the locations for the files in the I/O Designer software.

1. To set up the file locations if they are not already set, on the File menu, click **Properties**. The Project Properties dialog box appears (Figure 6–12).

Figure 6–12. Project Properties Dialog Box



2. Under **FPGA Xchange**, click **Browse** to select the FPGA Xchange file name and file location.
3. To specify a Pin report file, under **Place and Route**, click **Browse** to select the Pin-Out file name and file location.

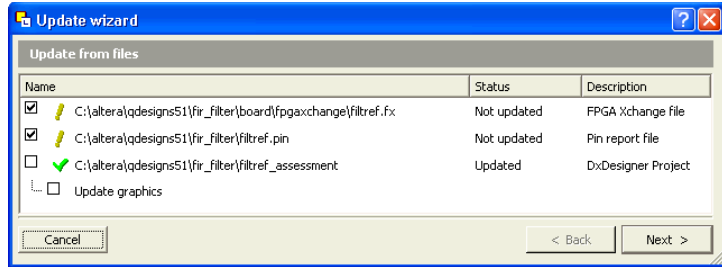
Once you have set up these file locations, the I/O Designer software monitors these files for changes. If the FPGA Xchange file or Pin-Out file changes during the design flow, three indicators flash red in the lower right-hand corner of the I/O Designer main window (see Figure 6–10 on page 6–17). You can continue working or click on the indicators to open the I/O Designer Update Wizard. If you have made changes to your design in the Quartus II software that result in an updated FPGA Xchange file or Pin-Out file and the update indicators do not flash or you have previously canceled an indicated update, manually open the Update Wizard. To open the Wizard, on the File menu, click **Update**.



In versions of the I/O Designer software before version 2005, instead of using flashing indicators, the I/O Designer software displays a dialog box asking if you want to open the Update Wizard.

The I/O Designer Update Wizard lists the updated files associated with the database (Figure 6–13).

Figure 6–13. Update Wizard Dialog Box



The paths to the updated files have yellow exclamation points and the **Status** column shows **Not updated**, indicating that the database has not yet been updated with the newer information contained in the files. A checkmark to the left of any updated file indicates that the file will update the database. Turn on any files you want to use to update the I/O Designer database, and click **Next**. If you are not satisfied with the database update, on the Edit menu, click **Undo**.

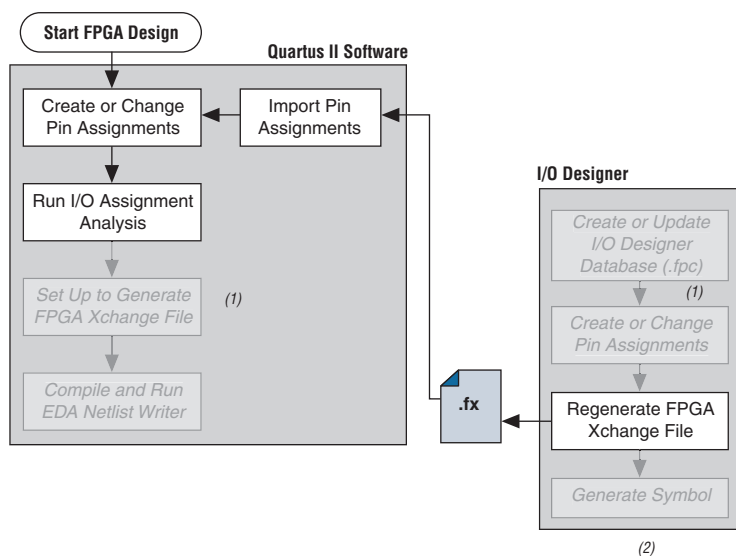


You can update the I/O Designer database using both the FPGA Xchange file and the Pin-Out file at the same time. Turning on both the FPGA Xchange file and the Pin-Out file for update causes the Update Wizard to provide options for using assignments from one file or the other exclusively or merging the assignments contained in both files into the I/O Designer database. Versions of the I/O Designer software older than version 2005 simply merge assignments contained in multiple files.

Sending Pin Assignment Changes to the Quartus II Software

In the same way that the FPGA designer can make adjustments that affect the PCB design, the board designer can make changes to optimize signal routing and layout that must be applied to the FPGA. The FPGA designer can take these required changes back into the Quartus II software to refit the logic to match the adjustments to the pinout. The I/O Designer software can accommodate this reverse flow as shown in Figure 6–14.

Figure 6–14. Updating the Quartus II Pin Assignments in the Reverse Design Flow



Notes to Figure 6–14:

- (1) These are software-specific steps in the design flow and are not necessary for the reverse flow steps of the design.
- (2) Refer to Figure 6–1 for the full design flow, which includes the complete I/O Designer software, the DxDesigner software, and the board layout tool flowchart details.

Pin assignment changes are made directly in the I/O Designer software, or the software automatically updates changes made in a board layout tool that are back-annotated to a schematic entry program such as the DxDesigner software. You must update the FPGA Xchange file to reflect these updates in the Quartus II software. To perform this update in the I/O Designer software, on the Generate menu, click **FPGA Xchange File**.



If your I/O Designer database points to the FPGA Xchange file generated by the Quartus II software instead of a backup copy, updating the file from the I/O Designer software overwrites any changes that may have been made to the file by the Quartus II software. If there are assignments from the Quartus II software in the file that you want to preserve, make a backup copy of the file before updating it in the I/O Designer software, and verify that your I/O Designer database points to the backup copy. To point to the backup copy, perform the steps in [“Updating Pin Assignments from the Quartus II Software”](#) on page 6–18.

After the FPGA Xchange file is updated, you must import it into the Quartus II software. To import the file, perform the following steps:

1. Start the Quartus II software and open your project.
2. On the Assignments menu, click **Import Assignments**.
3. In the File name box, click **Browse** and from the **Files of type** list, select **FPGA Xchange Files (*.fx)**.
3. Select the FPGA Xchange file and click **Open**.
4. Click **OK**.



Both the Quartus II software and the I/O Designer software can export and import an FPGA Xchange file. It is therefore possible to overwrite the FPGA Xchange file and import incorrect assignments into one or both programs. To prevent this occurrence from happening, make a backup copy of the file before importing, and import the copy instead of the file generated by the Quartus II software. In addition, assignments in the Quartus II software can be protected by following the steps in [“Protecting Assignments in the Quartus II Software”](#) on page 6–23.

Protecting Assignments in the Quartus II Software

To protect assignments in the Quartus II software, use the following steps:

1. Start the Quartus II software.
2. On the Assignments menu, click **Import Assignments**. The Import Assignments dialog box appears.
3. Turn on **Copy existing assignments into <project name>.qsf.bak before importing** before importing the FPGA Xchange file. This action automatically creates a backup Quartus II constraints file that contain all of your current pin assignments.

Generating Symbols for the DxDesigner Software

Along with circuit simulation, circuit board schematic creation is one of the first tasks required in the design of a new PCB. Schematics are required to understand how the PCB will work, and to generate a netlist

that is passed to a board layout tool for board design and routing. The I/O Designer software provides the ability to create schematic symbols based on the FPGA design exported from the Quartus II software.

Most FPGA devices contain hundreds of pins, requiring large schematic symbols that may not fit on a single schematic page. Symbol designs in the I/O Designer software can be split or fractured into a number of functional blocks, allowing multiple part fractures on the same schematic page or across multiple pages. In the DxDesigner software, these part fractures are joined together with the use of the HETERO attribute.

The I/O Designer software can generate symbols for use in a number of Mentor Graphics schematic entry tools, and can import changes back-annotated by board layout tools to update the database and feed updates back to the Quartus II software using the FPGA Xchange file. This section discusses symbol creation specifically for the DxDesigner software.

Schematic symbols are created in the I/O Designer software in the following ways:

- Manually
- Using the I/O Symbol Wizard
- Importing previously created symbols from the DxDesigner software

The I/O Designer Symbol Wizard can be used as a design base that allows you to quickly create a symbol for manual editing at a later time. If you have already created symbols in a DxDesigner project and want to apply a different FPGA design to them, you can manually import these symbols from the DxDesigner project. To import the symbols, open the I/O Designer software, and on the File menu, click **Import Symbol**.



For more information about importing symbols from the DxDesigner software into an I/O Designer database, refer to the I/O Designer Help.

Symbols created in the I/O Designer software are either functional, physical (PCB), or a combination of functional and physical. A functional symbol is based on signals imported into the database, usually from Verilog HDL or VHDL files. No physical device pins must be associated with the signals to generate a functional symbol. This section focuses on board-level PCB symbols with signals directly mapped to physical device pins through assignments in either the Quartus II Pin Planner or in the I/O Designer database.



For information about manually creating symbols, importing symbols, and editing symbols in the I/O Designer software, as well as the different types of symbols the software can generate, refer to the I/O Designer Help.

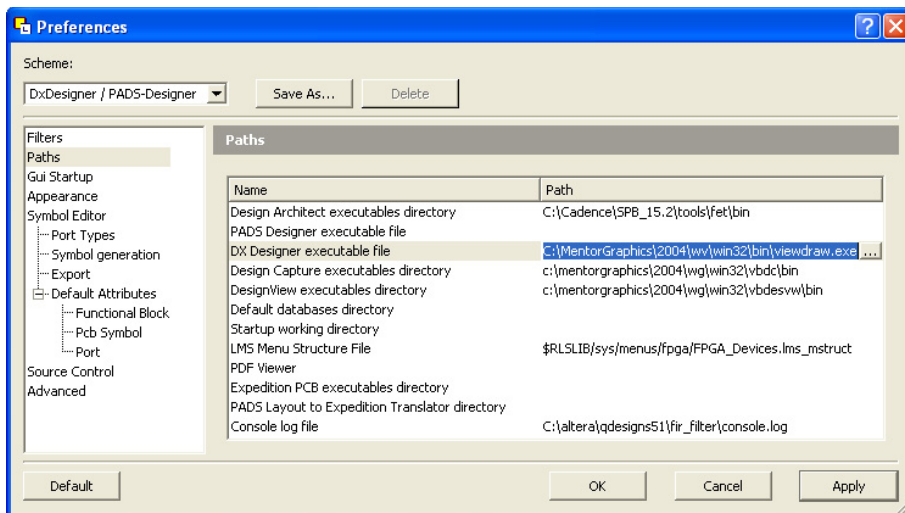
Setting Up the I/O Designer Software to Work with the DxDesigner Software

If you created your I/O Designer database using the Database Wizard, you may already be set up to export symbols to a DxDesigner project. To verify this, or to manually set up the I/O Designer software to work with the DxDesigner software, you must set the path to the DxDesigner executable, set the export type to DxDesigner, and set the path to a DxDesigner project directory.

To set these options, perform the following steps:

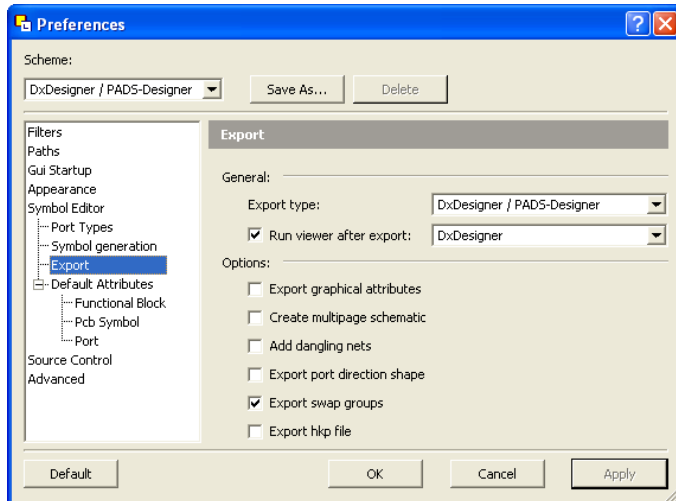
1. Start the I/O Designer software.
2. On the Tools menu, click **Preferences**. The Preferences dialog box appears.
3. Click **Paths**, double-click on the DxDesigner executable file path field, and click **Browse** to select the location of the DxDesigner application (Figure 6–15). Click **Apply**.

Figure 6–15. Path Preferences Dialog Box



4. Click **Symbol Editor** and click **Export**. In the Export type menu, under General, select **DxDesigner/PADS-Designer** (Figure 6–16).
5. Click **Apply** and click **OK**.

Figure 6–16. Symbol Editor Export Preferences



6. On the File menu, click **Properties**. The project Properties dialog box appears.
7. Click the **PCB Flow** tab and click **Path to a DxDesigner project directory**.
8. Click **OK**.

If you did not create a new DxDesigner project in the Database Wizard and you do not already have a DxDesigner project, you must create a new database using the DxDesigner software, and point the I/O Designer software to this new project.



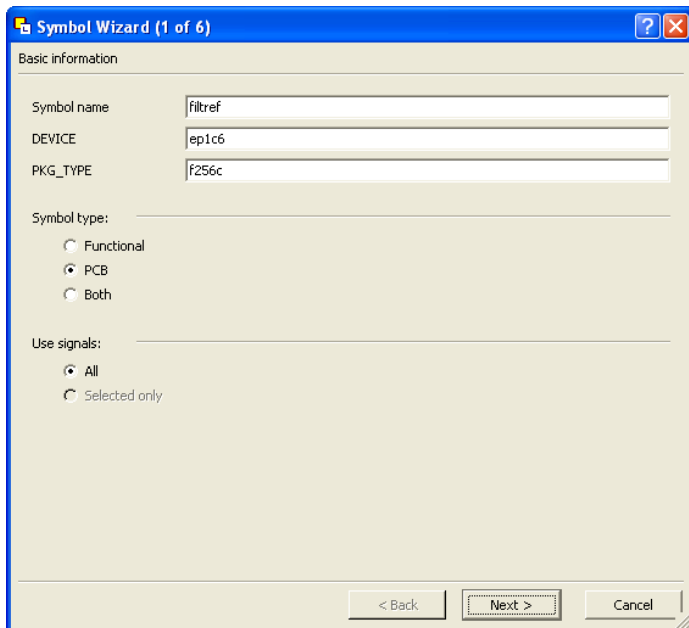
For information about creating and working with DxDesigner projects, refer to the DxDesigner Help.

Create Symbols with the Symbol Wizard

FPGA symbols based on Altera devices can be created, fractured, and edited using the I/O Designer Symbol Wizard. To create a symbol based on a selected Altera FPGA device:

1. Start the I/O Designer software.
2. Click **Symbol Wizard** in the toolbar, or on the Symbol menu, click **Symbol Wizard**. The **Symbol Wizard (1 of 6)** page is shown (Figure 6–17).

Figure 6–17. Symbol Wizard



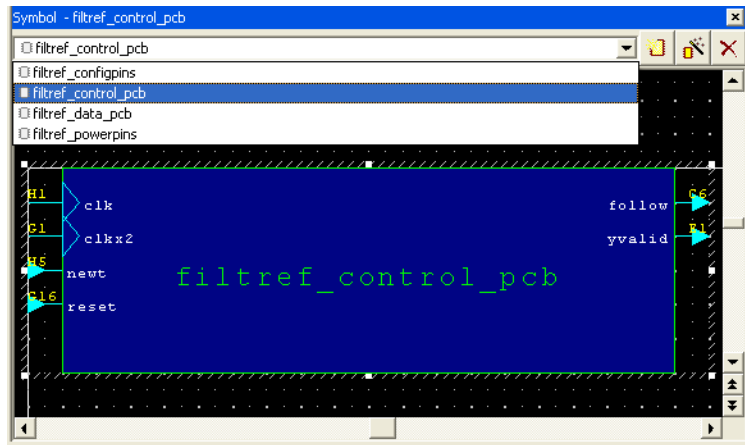
3. On the first Symbol Wizard page, in the **Symbol name** field, enter the symbol name. The **DEVICE** and **PKG_TYPE** fields are populated with the device and package information automatically. Under Symbol type, click **PCB**. Under Use signals, click **All**.
4. Click **Next**. The **Symbol Wizard (2 of 6)** page is shown.



If the **DEVICE** and **PKG_TYPE** fields are blank or incorrect, cancel the Symbol Wizard and select the correct device information. On the File menu, click **Properties**. In the Properties window, click the **FPGA Flow** tab and enter the correct device information.

5. On page 2 of the Symbol Wizard, select fracturing options for your symbol. If you are using the Symbol Wizard to edit a previously created fractured symbol, you must turn on **Reuse existing fractures** so that your current fractures are not altered. Select other options on this page as appropriate for your symbol.
6. Click **Next**. The **Symbol Wizard (3 of 6)** page is shown.
7. Additional fracturing options are available on page 3 of the Symbol Wizard. After selecting the desired options, click **Next**. The **Symbol Wizard (4 of 6)** page is shown.
8. On page 4 of the Symbol Wizard, select the options for how the symbols will look. Select the desired options and click **Next**. The **Symbol Wizard (5 of 6)** page is shown.
9. On page 5 of the Symbol Wizard, define what information will be labeled for the entire symbol and for individual pins. Select the desired options and click **Next**. The **Symbol Wizard (6 of 6)** page is shown.
10. On the final page of the Symbol Wizard, add additional signals and pins that have not already been placed in the symbol. Click **Finish** when you complete your selections.

Your symbol is complete. You can view your symbol and any fractures you created using the Symbol Editor ([Figure 6-18](#)). You can edit parts of the symbol, delete fractures, or rerun the Symbol Wizard.

Figure 6–18. The I/O Designer Symbol Editor

If assignments in the I/O Designer database are updated, the symbols created in the I/O Designer software automatically reflect these changes. Assignment changes can be made within the I/O Designer software, with an updated FPGA Xchange file from the Quartus II software, or from a back-annotated change in your board layout tool.

Export Symbols to the DxDesigner Software

After you have completed your symbols, export the symbols to your DxDesigner project. To generate all the fractures of a symbol, on the Generate menu, click **All Symbols**. To generate a symbol for the currently displayed symbol in Symbol Editor, click **Current Symbol Only**. Each symbol in the database is saved as a separate file in the `/sym` directory in your DxDesigner project. The symbols can be instantiated in your DxDesigner schematics.



For more information about working with DxDesigner projects, refer to the DxDesigner Help.

Scripting Support

The I/O Designer software features a command line Tcl interpreter. All commands issued through the GUI in the I/O Designer software are translated into Tcl commands that are run by the tool. You can view the generated Tcl commands and run scripts, or enter individual commands in the I/O Designer Console window.

The following section includes commands that perform some of the operations described in this chapter.

If you want to change the FPGA Xchange file from which the I/O Designer software updates assignments, type the following command at an I/O Designer Tcl prompt:

```
set_fpga_xchange_file <file name>
```

After the FPGA Xchange file is specified, use the following command to update the I/O Designer database with assignment updates made in the Quartus II software:

```
update_from_fpga_xchange_file
```

Use the following command to update the FPGA Xchange file with changes made to the assignments in the I/O Designer software for transfer back into the Quartus II software:

```
generate_fpga_xchange_file
```

If you want to import assignment data from a Pin-Out file created by the Quartus II software, use the following command:

```
set_pin_report_file -quartus_pin <file name>
```

Run the I/O Designer Symbol Wizard with the following command:

```
symbolwizard
```

Set the DxDesigner project directory path where symbols are saved with the following command:

```
set_dx_designer_project -path <path>
```



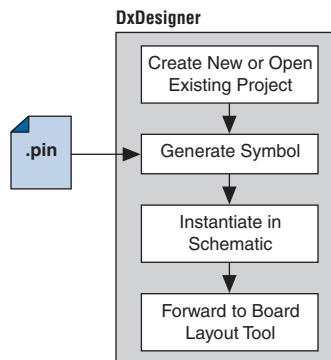
For more information about Tcl scripting and Tcl scripting with the Quartus II software, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. For more information about the Tcl scripting capabilities of the I/O Designer software as well as a list of all the commands available, refer to the I/O Designer Help.

FPGA-to-Board Integration with the DxDesigner Software

The Mentor Graphics DxDesigner software is a design entry tool for schematic capture. You can use it to create flat circuit schematics for all types of PCB design. You can also use the DxDesigner software to create hierarchical schematics that facilitate design reuse and a team-based design. You can use the DxDesigner software in the design flow alone or in conjunction with the I/O Designer software. However, if you use the DxDesigner software without the I/O Designer software, the design flow is one-way, using only the Pin-Out file generated by the Quartus II software.

Signal and pin assignment changes can be made only in the Quartus II software and are reflected in updated symbols in a DxDesigner schematic. You cannot back-annotate changes made in a board layout tool or in a DxDesigner symbol to the Quartus II software. [Figure 6–19](#) shows the design flow when the I/O Designer software is not used.

Figure 6–19. Design Flow Without the I/O Designer Software *Note (1)*



Note to [Figure 6–19](#):

- (1) Refer to [Figure 6–1](#) for the full design flow, which includes the Quartus II software, the I/O Designer software, and the board layout tool flowchart details.



For more information about the DxDesigner software, including usage, support, training, and product updates, refer to the Mentor Graphics web page at www.mentor.com, or choose Schematic Design Help Topics in the DxDesigner Help.

DxDesigner Project Settings

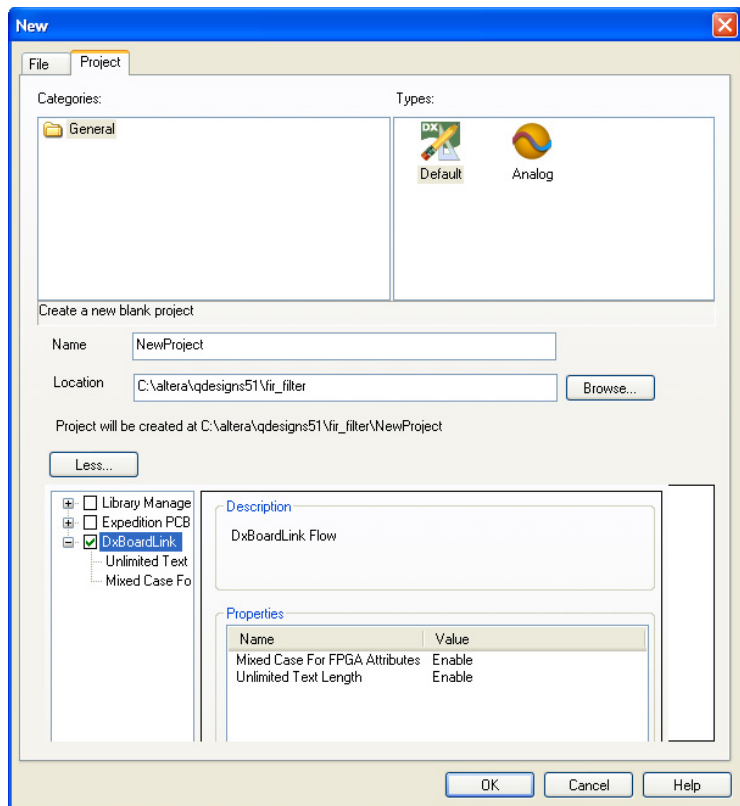
New projects in the DxDesigner software are already set up to create FPGA symbols by default. However, for complete support and compatibility with the I/O Designer software, if it is used with the DxDesigner software, you should enable the DxBoardLink Flow options.

You can enable the DxBoardLink flow design configuration while creating a new DxDesigner project or after a project is created.

To enable the DxBoardLink flow design configuration when creating a new DxDesigner project, perform the following steps:

1. Start the DxDesigner software.
2. On the File menu, click **New** and click the **Project** tab. The New Project dialog box appears.

Figure 6–20. New Project Dialog Box

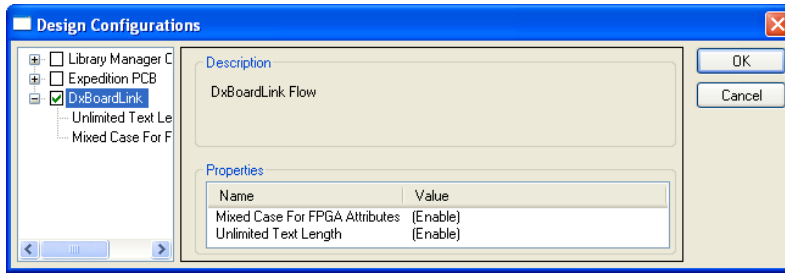


3. Click **More**. Turn on **DxBoardLink**. (Figure 6–20)



To enable the DxBoardLink Flow design configuration in an existing project, click **Design Configurations** in the Design Configuration toolbar and turn on **DxBoardLink** (Figure 6–21).

Figure 6–21. DxBoardLink Design Configuration



DxDesigner Symbol Wizard

In addition to circuit simulation, circuit board schematic creation is one of the first tasks required in the design of a new PCB. Schematics are required to understand how the PCB will work, and to generate a netlist that is passed on to a board layout tool for board stackup design and routing.

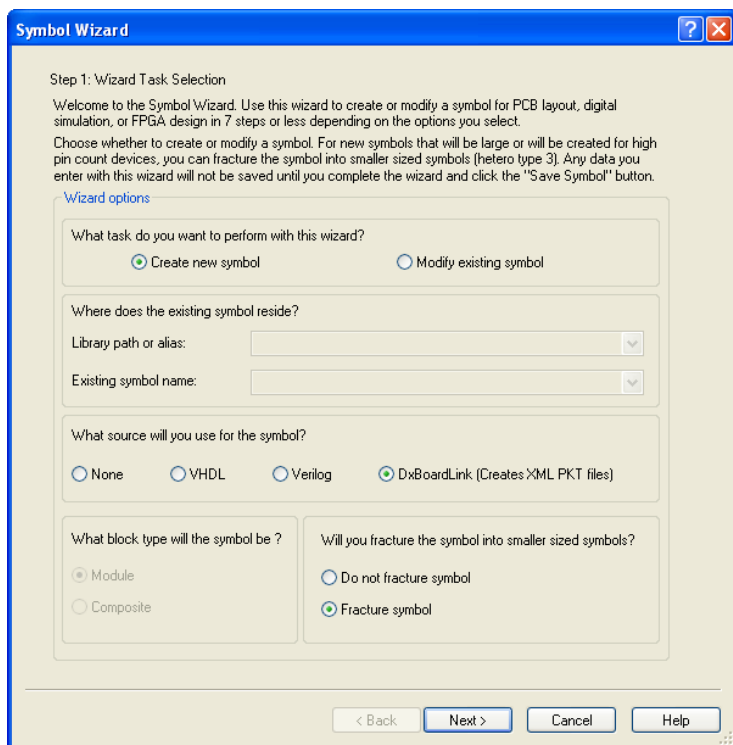
You can create schematic symbols using the DxDesigner software based on FPGA designs exported from the Quartus II software through the Pin-Out file for instantiation in DxDesigner schematic design files. Most FPGA devices are physically large with hundreds of pins, requiring large schematic symbols that may not fit on a single schematic page. You can split or fracture symbols created in the DxDesigner software into a number of functional blocks, allowing multiple part fractures on the same schematic page or across multiple pages. In the DxDesigner software, these part fractures are joined together with the use of the HETERO attribute.

You can create schematic symbols in the DxDesigner software manually or with the Symbol Wizard. The DxDesigner Symbol Wizard is similar to the I/O Designer Symbol Wizard, but with fewer fracturing options.

FPGA symbols based on Altera devices can be created, fractured, and edited using the DxDesigner Symbol Wizard. To start the Symbol Wizard, perform the following steps:

1. Start the DxDesigner software.
2. Click **Symbol Wizard** in the toolbar, or on the File menu, click **New**. The New window is shown. Click the **File** tab and create a new file of type **Symbol Wizard**.
3. Enter the new symbol name in the name field and click **OK**. The **Symbol Wizard** page is shown (Figure 6–22).

Figure 6–22. Wizard Task Selection

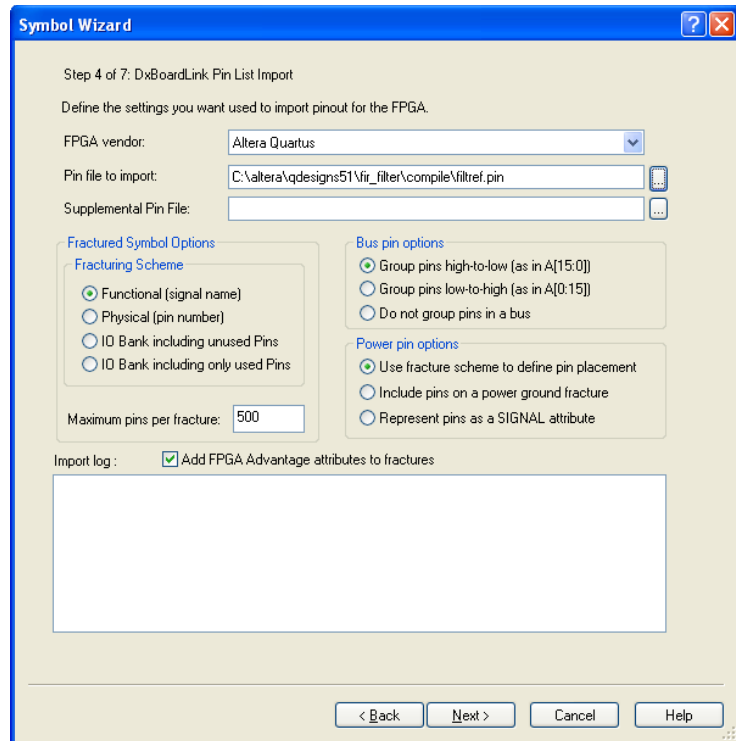


4. On the **Wizard Task Selection** page, choose to create a new symbol or modify an existing symbol. If you are modifying an existing symbol, specify the library path or alias, and select the existing symbol. If you are creating a new symbol, select **DxBoardLink** for the symbol source. The DxDesigner block type defaults to **Module**

because the FPGA design does not have an underlying DxDesigner schematic. Define whether or not to fracture the symbol. After making your selections, click **Next**. The **New Symbol and Library Name** page is shown.

5. On the **New Symbol and Library Name** page, enter a name for the symbol, an overall part name for all of the symbol fractures, and a library name for the new library created for this symbol. By default, the part and library names are the same as the symbol name. Click **Next**. The **Symbol Parameters** page is shown.
6. On the **Symbol Parameters** page, decide how the generated symbol will look and how it will match up with the grid you have set in your DxDesigner project schematic. After making your selections, click **Next**. The **DxBoardLink Pin List Import** page is shown (Figure 6–23).

Figure 6–23. DxBoardLink Pin List Import



7. On the **DxBoardLink Pin List Import** page, in the **FPGA vendor** list, select **Altera Quartus**. In the Pin-Out file to import field, browse to and select the Pin-Out file from your Quartus II design project directory. Additionally, select choices from the Fracturing Scheme options, Bus pin options, and Power pin options. After you make your selections, click **Next**. The **Symbol Attributes** page is shown.
8. On the **Symbol Attributes** page, select to create or modify symbol attributes for use in the DxDesigner software. After you make your selections, click **Next**. The **Pin Settings** page is shown.
9. On the **Pin Settings** page, make any final adjustments to pin and label location and information. Each tabbed spreadsheet represents a fracture of your symbol. After you make your selections, click **Save Symbol**.

After you save the symbol, you can examine and place any fracture of the symbol in your schematic. When you are finished with the Symbol Wizard, all the fractures you created are saved as separate files in the library you specified or created in the **/sym** directory in your DxDesigner project. You can add the symbols to your schematics or you can edit the symbols manually or with the Symbol Wizard.



Symbols created in the DxDesigner software can be edited and updated with newer versions of the Pin-Out file generated by the Quartus II software. However, symbol fracturing is fixed, and the symbol cannot be fractured again. To create new fractures for your design, create a new symbol in the Symbol Wizard, and follow the steps in “[DxDesigner Symbol Wizard](#)” on page 6–33.



For more information about creating, editing, and instantiating component symbols in DxDesigner, choose Schematic Design Help Topics from the Help menu in the DxDesigner software.

Conclusion

Transferring a complex, high-pin-count FPGA design to a PCB for prototyping or manufacturing is a daunting process that can lead to errors in the PCB netlist or design, especially when multiple engineers are working on different parts of the project. The design workflow available when the Quartus II software is used in conjunction with the Mentor Graphics toolset assists the FPGA designer and the board designer in preventing errors and focusing their attention on the design.

Document Revision History

Table 6–3 shows the revision history for this document.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history to the document.	
May 2006, v6.0.0	Was chapter 7 in v5.1. Minor updates for the Quartus II software version 6.0.0.	
November 2005, v5.1.1	Corrected text in steps 3 and 4 on page 11.	
October 2005, v5.1.0	Initial release.	

Introduction

With today's large, high-pin-count and high-speed FPGA devices, good printed circuit board (PCB) design practices are more essential than ever to ensure the correct operation of your system. Typically, the PCB design takes place concurrently with the design and programming of the FPGA. Signal and pin assignments are initially made by the FPGA or ASIC designer, and it is up to the board designer to correctly transfer these assignments to the symbols used in their system circuit schematics and board layout. As the board design progresses, pin reassignments may be requested or required to optimize the layout. These reassignments must in turn be relayed to the FPGA designer so that the new assignments can be processed through the FPGA using updated place-and-route.

Cadence provides tools to support this type of design flow. This chapter addresses how the Quartus II software interacts with the Cadence Allegro Design Entry HDL software and the Allegro Design Entry CIS (Component Information System) software (also known as OrCAD Capture CIS) to provide a complete FPGA-to-board integration design workflow. This chapter provides information about the following topics:

- Cadence tool description, history, and comparison
- The general design flow between the Quartus II software and the Cadence Allegro Design Entry HDL software and the Cadence Allegro Design Entry CIS software
- Generating schematic symbols from your FPGA design for use in the Cadence Allegro Design Entry HDL software
- Updating Design Entry HDL symbols when signal and pin assignment changes are made in the Quartus II software
- Creating schematic symbols in the Cadence Allegro Design Entry CIS software from your FPGA design
- Updating symbols in the Cadence Allegro Design Entry CIS software when signal and pin assignment changes are made in the Quartus II software
- Using Altera®-provided device libraries in the Cadence Allegro Design Entry CIS software

This chapter is intended primarily for board design and layout engineers who want to begin the FPGA board integration process while the FPGA is still in the design phase. In addition, part librarians benefit from learning how to take output from the Quartus II software and use it to create new library parts and symbols.

The instructions in this chapter require the following software:

- The Quartus II software version 5.1 or later
- The Cadence Allegro Design Entry HDL or the Cadence Allegro Design Entry CIS software version 15.2 or later
- If you are using the OrCAD Capture software, you must have version 10.3 or later (CIS is optional)



Because the Cadence Allegro Design Entry CIS software is based on OrCAD Capture, these programs are very similar. For this reason, this chapter refers to the Allegro Design Entry CIS software in directions; however, these directions also apply to OrCAD Capture unless otherwise noted.



To obtain and license the Cadence tools described in this chapter, and for product information, support, and training, refer to the Cadence website, www.cadence.com. For information about OrCAD Capture and the CIS option, refer to the OrCAD website, www.orcad.com. For Cadence and OrCAD support and training, refer to the EMA Design Automation website, www.ema-eda.com.

Product Comparison

The design tools described in this chapter have similar functionality, but there are differences in their use as well as where to access product information. [Table 7-1](#) lists the products described in this chapter and provides information about changes, product information, and support.

	Cadence Allegro Design Entry HDL	Cadence Allegro Design Entry CIS	OrCAD Capture CIS
Former Name	Concept HDL Expert	Capture CIS Studio	N/A
History	More commonly known by its former name, Cadence renamed all board design tools in 2004 under the Allegro name.	Based directly on OrCAD Capture CIS, this tool is still developed by OrCAD but sold and marketed by Cadence. EMA provides support and training.	The basis for Design Entry CIS is still developed by OrCAD for continued use by existing OrCAD customers. EMA now provides support and training for all OrCAD products.
Vendor Design Flow	Cadence Allegro 600 series, formerly known as Expert Series, for high-end, high-speed design.	Cadence Allegro 200 series, formerly known as Studio Series, for small- to medium-level design.	N/A
Information & Support	www.cadence.com www.ema-eda.com	www.cadence.com www.ema-eda.com www.orcad.com	www.ema-eda.com www.orcad.com

FPGA-to-PCB Design Flow

In the examples in this section, you create a design flow integrating an Altera FPGA design from the Quartus II software through a circuit schematic in the Allegro Design Entry HDL software (Figure 7-1) or the Allegro Design Entry CIS software (Figure 7-2).

Figure 7-1. Design Flow with the Allegro Design Entry HDL Software

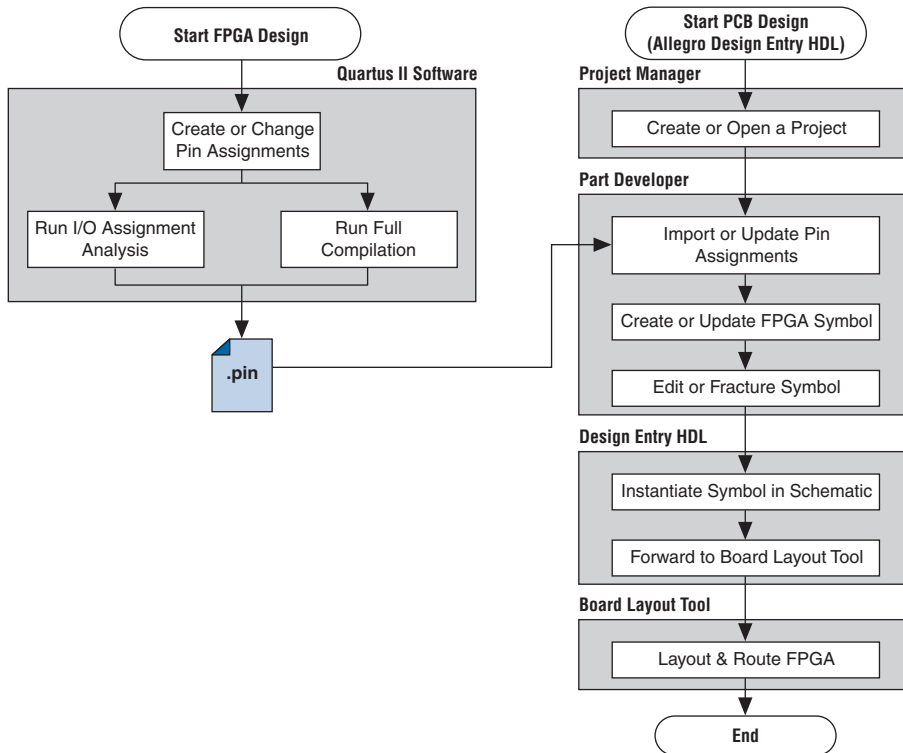
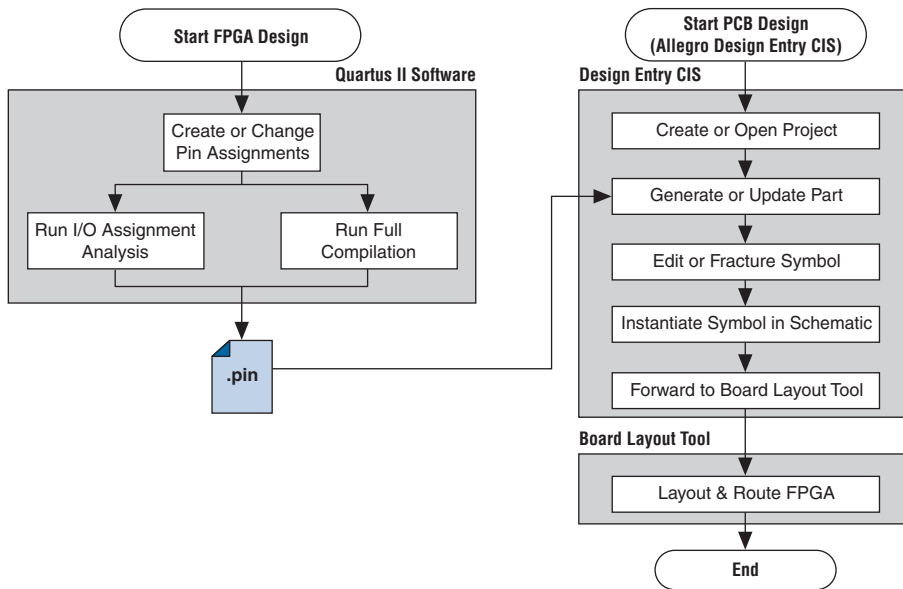


Figure 7-2. Design Flow with the Allegro Design Entry CIS Software



The basic steps in a complete design flow to integrate an Altera FPGA design starting in the Quartus II software through to a circuit schematic in Design Entry HDL or Design Entry CIS are as follows:

- Start the Quartus II software.
- In the Quartus II software, compile your design to generate a Pin-Out (.pin) file to transfer assignments to the Cadence tool.
- If you are using the Cadence Allegro Design Entry HDL software for your schematic design:
 - Open an existing project or create a new project in the Allegro Project Manager.
 - Construct a new symbol or update an existing symbol using the Allegro PCB Librarian Part Developer.
 - With the Part Developer, edit your symbol or fracture it into smaller parts, if desired.
 - Instantiate the symbol in your Design Entry HDL software schematic and transfer the design to your board layout tool.

- If you are using the Cadence Allegro Design Entry CIS software for your schematic design, perform the following steps:
 - Generate a new part within an existing or new Allegro Design Entry CIS project, referencing the Pin-Out file output from the Quartus II software. You can update an existing symbol with a new Pin-Out file.
 - Split the symbol into smaller parts as desired.
 - Instantiate the symbol in your Design Entry CIS schematic and transfer the design to your board layout tool.

Figures 7-1 and 7-2 show the possible design flows, depending on your tool choice. The Cadence PCB Librarian Expert license is required to use the PCB Librarian Part Developer to create FPGA symbols. You can update symbols with changes made to the FPGA design at any point using any of these tools.

Setting Up the Quartus II Software

You can transfer pin and signal assignments from the Quartus II software to the Cadence design tools by generating the Quartus II project Pin-Out file. The Pin-Out file is an output file generated by the Quartus II Fitter that contains pin assignment information. Use the Quartus II Pin Planner or Assignment Editor to set and change the assignments contained in the Pin-Out file. This file cannot be used to import pin assignment changes into the Quartus II software. Use it only to transfer assignments for use with the Cadence design tools.

The Pin-Out file lists all used and unused pins on your selected Altera device. It also provides the following basic information fields for each assigned pin on a device:

- Pin signal name and usage
- Pin number
- Signal direction
- I/O standard
- Voltage
- I/O bank
- User or Fitter-assigned



For information about using the Quartus II Pin Planner to create or change pin assignment details, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Generating Pin-Out Files

The Quartus II software automatically generates the Pin-Out file when your FPGA design is fully compiled or when you start I/O Assignment Analysis. To start I/O Assignment Analysis, on the Processing menu, point to Start and click **Start I/O Assignment Analysis**. The file is output by the Quartus II Fitter. The file is generated and placed in your Quartus II design directory with the name *<project name>.pin*. The Cadence design tools do not generate or change this file.



For more information about pin and signal assignment transfer and the files that the Quartus II software can import and export, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

FPGA-to-Board Integration with the Cadence Allegro Design Entry HDL Software

The Cadence Allegro Design Entry HDL software is Cadence's high-end schematic capture tool (part of the Cadence 600 series design flow). Use this software to create flat circuit schematics for all types of PCB design. The Cadence Allegro Design Entry HDL software can also create hierarchical schematics to facilitate design reuse and team-based design. With the Cadence Allegro Design Entry HDL software, the design flow from FPGA-to-board is one-way, using only the Pin-Out file generated by the Quartus II software. Signal and pin assignment changes can only be made in the Quartus II software and are reflected in updated symbols in a Design Entry HDL project.



Routing or pin assignment changes made in a board layout tool or a Design Entry HDL symbol cannot be back-annotated to the Quartus II software.

Figure 7-1 shows the design flow with the Cadence Allegro Design Entry HDL software.



For more information about the Cadence Allegro Design Entry HDL software and the Part Developer, including licensing, support, usage, training, and product updates, refer to the Help in the software or refer to the Cadence web page at www.cadence.com.

Symbol Creation

In addition to circuit simulation, circuit board schematic creation is one of the first tasks required in the design of a new PCB. Schematics are required to understand how the PCB works, and to generate a netlist that is passed on to a board layout tool for board design and routing. The Allegro PCB Librarian Part Developer provides the ability to create schematic symbols based on FPGA designs exported from the Quartus II software.

Create symbols for Design Entry HDL with the Allegro PCB Librarian Part Developer available in the Allegro Project Manager. The Part Developer is the recommended method for importing FPGA designs into the Cadence Allegro Design Entry HDL software.

You must have a PCB Librarian Expert license from Cadence to run the Part Developer. The Part Developer provides a graphical interface with many options for creating, editing, fracturing, and updating symbols. If you do not use the Part Developer, you must create and edit symbols manually in the Symbol Schematic View in the Cadence Allegro Design Entry HDL software.



If you do not have a PCB Librarian Expert license, you can still automatically create FPGA symbols using the programmable IC (PIC) design flow found in the Allegro Project Manager. For more information about using the PIC design flow, refer to the Help in the Cadence design tools, or go to the Cadence website at www.cadence.com.

Before you create a symbol from an FPGA design, you must open or create a Design Entry HDL design project. You can do this with the Allegro Project Manager, the main interface to all of the Cadence tools.

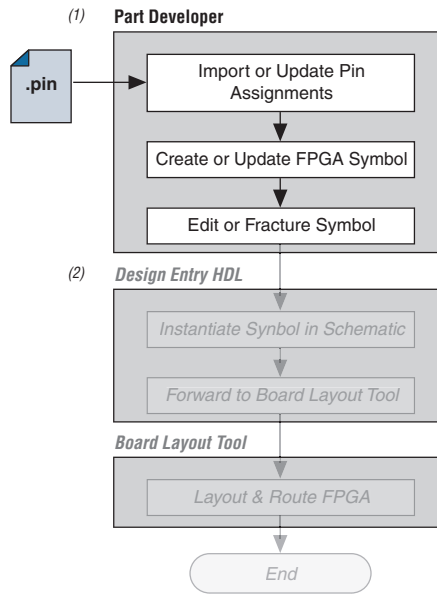
To open an existing design in the Allegro Project Manager, on the File menu, click **Open** and select the main design file for your project (found in your Allegro Design Entry HDL project directory and called *<project directory>.cpm*).

To create a new project, on the File menu, point to New and click **New Design**. The New Project Wizard appears. Use the wizard to name your new project, set the file location, and define associated part libraries.

Allegro PCB Librarian Part Developer

Create, fracture, and edit schematic symbols for your FPGA designs in Altera devices using the Part Developer. Most FPGA devices are physically large with hundreds of pins, requiring large schematic symbols that may not fit on a single schematic page. Symbols designed in the Part Developer can be split or fractured into a number of functional blocks called slots, allowing multiple smaller part fractures to exist on the same schematic page or across multiple pages. [Figure 7-3](#) highlights how the Part Developer fits into the design flow.

Figure 7-3. Part Developer in the Design Flow



Notes to Figure 7-3:

- (1) Refer to Figure 7-1 for the full design flow flowchart details.
- (2) Grayed out steps are not part of the FPGA Symbol creation or update process.

Run the Part Developer from the Project Manager (Figure 7-4). To start the Part Developer in the Project Manager, on the Flows menu, click **Library Management**. Click **Part Developer** to start the tool.

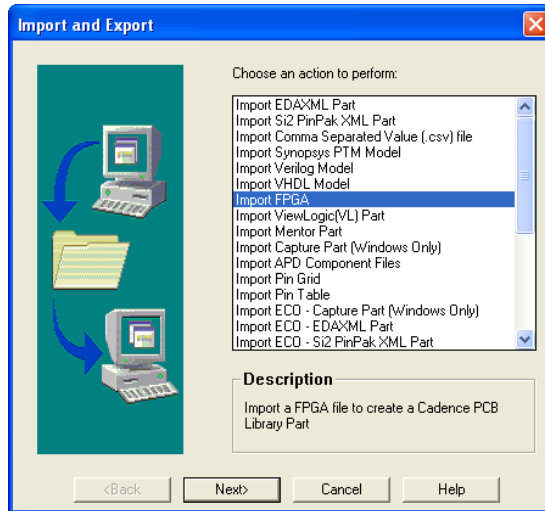
Figure 7-4. Invoking the Part Developer from the Project Manager

Import and Export Wizard

Once you are in the Part Developer, you can use the Import and Export Wizard to import your pin assignments from the Quartus II software. To access the Wizard, perform the following steps:

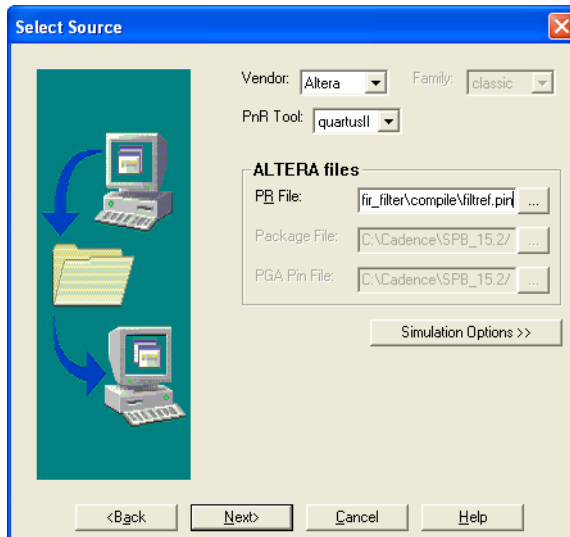
1. On the File menu, click **Import and Export**. The Import and Export Wizard appears (Figure 7-5).

Figure 7-5. Import and Export Wizard



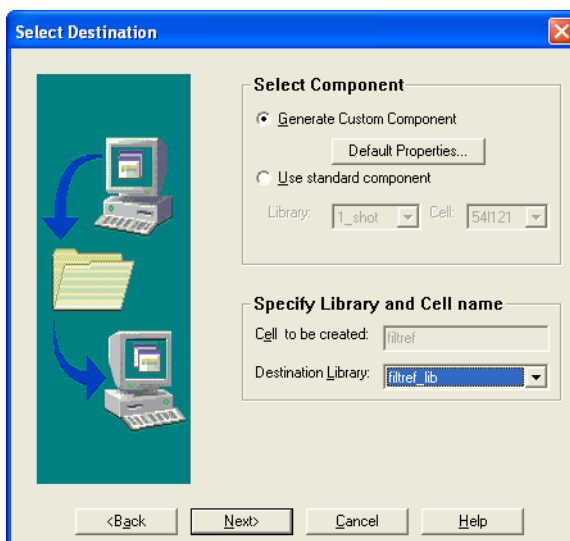
2. Select **Import FPGA**. Click **Next**. The **Select Source** page appears (Figure 7-6).

Figure 7-6. Select Source Page



- In the **Vendor** list, select **Altera**. In the **PnR Tool** list, select **quartusII**. To specify the Pin-Out file in the **PR File** field, select the Pin-Out file in your Quartus II project directory. Click **Simulation Options** if you want to select simulation input files. Click **Next**. The **Select Destination** page is shown (Figure 7-7).

Figure 7-7. Select Destination Page



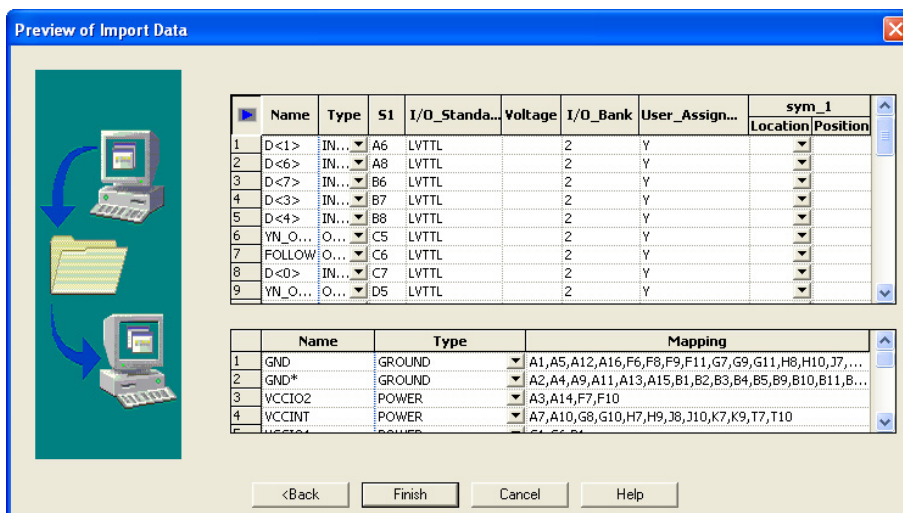
- To create a new component in a library, click **Generate Custom Component**. To base your symbol on an existing component, click **Use standard component**.




You may want to do this if you previously created generic symbols for an FPGA device. You can place your pin and signal assignments from the Quartus II software on this symbol and reuse the symbol as a base any time you have a new FPGA design.


In the **Library** list, select an existing library. You can now select from the cells contained in the selected library. Each cell represents all of the symbol versions and part fractures for that particular part. In the **Cell** list, select the existing cell to use as a base for your part. In the **Destination Library** list, select a destination library for the component. Click **Next**. A preview of your import data is shown (Figure 7-8).

Figure 7–8. Preview of Import Data Window



- Review the assignments you are importing into the Part Developer based on the data in the Pin-Out file. The location of each pin is not included in the information in this window, but inputs are placed on the left side of the created symbol, outputs on the right, power pins on the top, and ground pins on the bottom. Make any desired changes. When you have completed your changes, click **Finish** to create the symbol. The Part Developer main screen is shown.

 If the Part Developer is not set up to point to your PCB Librarian Expert license file, an error message displays in red at the bottom of the message text window of the **Part Developer** when you select the Import and Export command. To point to your PCB Librarian Expert license, on the **File** menu, click **Change Product** and select the correct product license.

 For more information about licensing and obtaining licensing support, contact Cadence or refer to their website at www.cadence.com.

Edit & Fracture Symbol

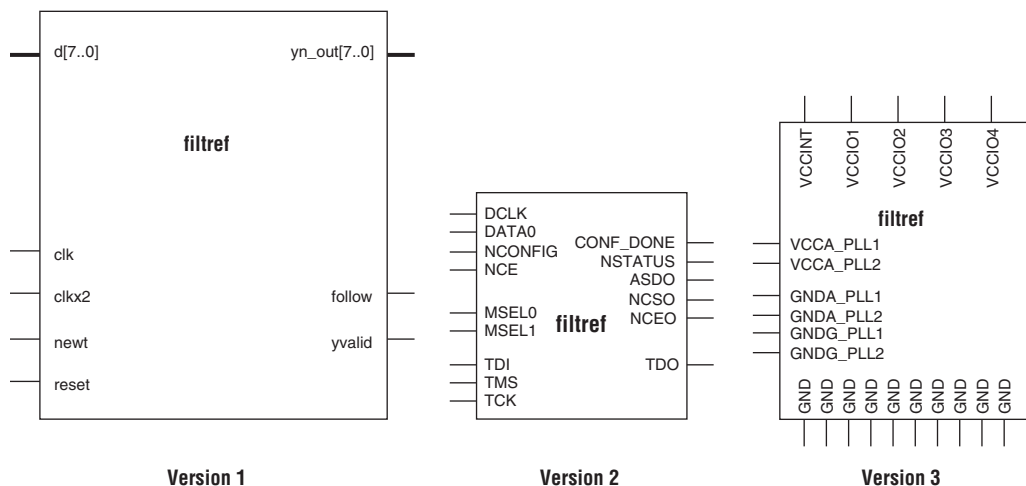
After you save your new symbol in the Part Developer software, you can edit the symbol graphics, fracture the symbol into multiple slots, and add or change package or symbol properties. These actions are available from the **Part Developer** main window.

The Part Developer Symbol Editor contains many graphical tools to edit the graphics of a particular symbol. Select the symbol in the cell hierarchy to edit the symbol graphics. The **Symbol Pins** tab is shown. Edit the preview graphic of the symbol in the **Symbol Pins** tab.

Fracturing a Part Developer package into separate symbol slots is especially useful for FPGA designs. A single symbol for most FPGA packages may be too large for a single schematic page. Splitting the part into separate slots allows you to organize parts of the symbol by function, creating cleaner circuit schematics. For example, you could create one slot for an I/O symbol, a second slot for a JTAG symbol, and a third slot for a power/ground symbol.

Figure 7–9 shows a part fractured into separate slots.

Figure 7–9. Splitting a Symbol into Multiple Slots Notes (1), (2)



Notes to Figure 7–9:

- (1) Figure 7–9 represents a Cyclone device with JTAG or passive serial (PS) mode configuration option settings. Symbols created for other devices or other configuration modes may have different sets of configuration pins, but can be fractured in a similar manner.
- (2) Symbol fractures are referred to in different ways in each of the tools described in this chapter. Refer to Table 7–2 for the specific tool naming conventions.
- (3) The power/ground slot shows only a representation of power and ground pins. In actuality, the device contains a high number of power and ground pins.



While the Part Developer software refers to symbol fractures as slots, the other tools described in this chapter use different names to refer to symbol fractures. Table 7–2 lists the symbol fracture naming conventions for each of the tools addressed in this chapter.

Table 7–2. Symbol Fracture Naming

	Allegro PCB Librarian Part Developer Software	Allegro Design Entry HDL Software	Allegro Design Entry CIS Software
During symbol generation	Slots	N/A	Sections
During symbol schematic instantiation	N/A	Versions	Parts

To fracture a part into separate slots, or modify the slot locations of pins on parts that are already fractured in the Part Developer, perform the following steps:

1. Start the Cadence Allegro Design Project Manager.
2. On the Flows menu, click **Library Management**. The Library Management design flow is shown. Click **Part Developer**. The Part Developer launches.
3. Click on the name of the package you want to change in the cell hierarchy. The **Package Pin** tab appears.
4. Click **Functions/Slots**. If you are not creating new slots but want to change the slot location of some pins, proceed to step 5. If you are creating new slots, click **Add**. A dialog box appears, allowing you to add extra symbol slots. Set the number of extra slots you want to add to the existing symbol, not the total number of desired slots for the part. Click **OK**.
5. Click **Distribute Pins**. Set the slot where each pin should reside. Use the checkboxes in each column to move pins from one slot to another. You can use the standard cut, copy, and paste keyboard commands on selected groups of checkboxes to move multiple pins from one slot to another. Click **OK**.
6. After distributing the pins, click the **Package Pin** tab and click **Generate Symbol(s)**. the **Generate Symbols** dialog box appears.
7. Select whether to create a new symbol or modify an existing symbol in each slot. Click **OK**.

The newly generated or modified slot symbols display as separate symbols in the cell hierarchy. Each of these symbols can be edited individually.



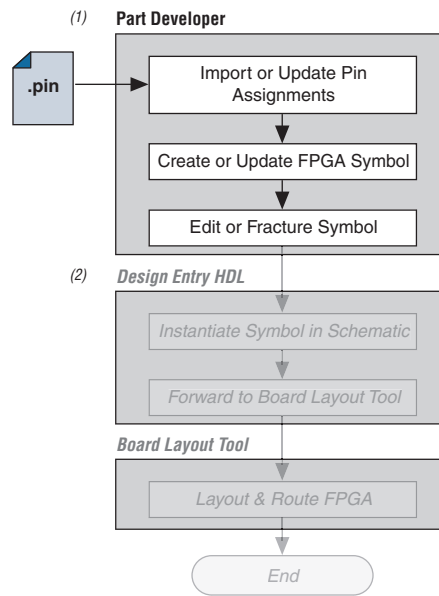
The Part Developer lets you remap pin assignments in the Package Pin tab of the main **Part Developer** window. If signals are remapped to different pins in the Part Developer, the changes are reflected only in regenerated symbols for use in your schematics. You cannot transfer pin assignment changes to the Quartus II software from the Part Developer, which creates a potential mismatch of the schematic symbols and assignments in the FPGA design. If pin assignment changes are necessary, make the changes in the Quartus II Pin Planner instead of the Part Developer, and update the symbol as described in the following sections.



For more information about creating, editing, and organizing component symbols with the Allegro PCB Librarian Part Developer, refer to the Part Developer Help.

Update FPGA Symbol

As the design process continues, you may need to make changes to the logic design in the Quartus II software, placing signals on different pins after the design is recompiled, or use the Quartus II Pin Planner to make changes manually. The board designer may request such changes to improve the board routing and layout. These types of changes must be carried forward to the circuit schematic and board layout tools to ensure signals are connected to the correct pins on the FPGA. Updating the Pin-Out file in the Quartus II software facilitates this flow. [Figure 7–10](#) shows this part of the design flow.

Figure 7-10. Updating the FPGA Symbol in the Design Flow**Notes to Figure 7-10:**

- (1) Refer to [Figure 7-1](#) for the full design flow flowchart details.
- (2) Grayed out steps are not part of the FPGA Symbol update process.

Once the Pin-Out file has been updated, perform the following steps to update the symbol using the Allegro PCB Librarian Part Developer:

1. On the File menu, click **Import and Export**. The Import and Export Wizard appears.
2. In the list of actions to perform, select **Import ECO - FPGA**. Click **Next**. The **Select Source Page** is shown.
3. Select the updated source of the FPGA assignment information. In the **Vendor** list, select **Altera**. In the **PnR Tool** list, select **quartusII**. In the **PR File** field, click browse to specify the updated Pin-Out file in your Quartus II project directory. Click **Next**. The **Select Destination** window is shown.
4. Select the source component and a destination cell for the updated symbol. To create a new component based on the updated pin assignment data, select **Generate Custom Component**. This replaces the cell listed under the Specify Library and Cell name header with a new, non-fractured cell. Any symbol edits or fractures

are lost. You can preserve these edits by selecting **Use standard component and select the existing library and cell**. Select the destination library for the component and click **Next**. The Preview of Import Data page is shown.

5. Make any additional changes to your symbol. Click **Next**. A list of ECO messages displays summarizing what changes will be made to the cell. To accept the changes and update the cell, click **Finish**.
6. The main **Part Developer** window is shown. You can edit, fracture, and generate the updated symbols as usual from this window.



If the Part Developer is not set up to point to your PCB Librarian Expert license file, an error message displays in red at the bottom of the message text window of the **Part Developer** when you select the **Import and Export** command. To point to your PCB Librarian Expert license, on the File menu, click **Change Product**, and select the correct product license. For more information about licensing and obtaining licensing support, contact Cadence or refer to their website at www.cadence.com.

Instantiating the Symbol in the Cadence Allegro Design Entry HDL Software

Once the new symbol is saved in the Part Developer, instantiate the symbol in your Design Entry HDL schematic.

1. In the Allegro Project Manager, switch to the board design flow.
2. On the Flows menu, click **Board Design**.
3. Click **Design Entry** to start the Design Entry HDL software.
4. To add the newly created symbol to your schematic, right-click in the main schematic window and choose **Add Component**, or on the Component menu, click **Add**. The **Add Component** dialog box appears.
5. Select the new symbol library location, and select the name of the cell you created from the list of cells.

The symbol is now “attached” to your cursor for placement in the schematic. If you fractured the symbol into slots, right-click the symbol and choose **Version** to select one of the slots for placement in the schematic.



For more information about the Cadence Allegro Design Entry HDL software, including licensing, support, usage, training, and product updates, refer to the Help in the software or go to the Cadence website at www.cadence.com.

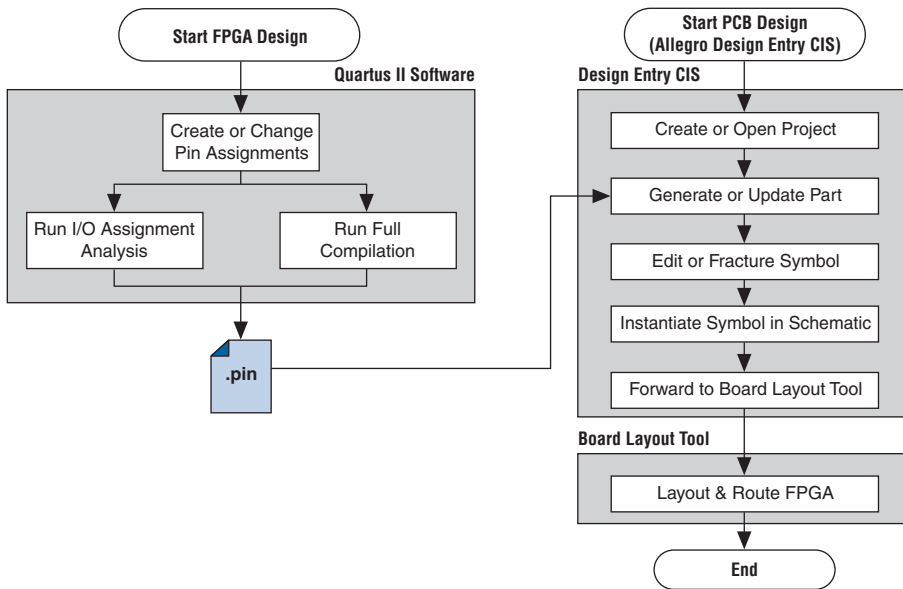
FPGA-to-Board Integration with Allegro Design Entry CIS

The Cadence Allegro Design Entry CIS software is Cadence’s mid-level schematic capture tool (part of the Cadence 200 series design flow based on OrCAD Capture CIS). Use this software to create flat circuit schematics for all types of PCB design. You can also create hierarchical schematics to facilitate design reuse and team-based design using this software. With the Cadence Allegro Design Entry CIS software, the design flow from FPGA-to-board is unidirectional using only the Pin-Out file generated by the Quartus II software. Signal and pin assignment changes can only be made in the Quartus II software and are reflected in updated symbols in a Design Entry CIS schematic project.



Routing or pin assignment changes made in a board layout tool or a Design Entry CIS symbol cannot be back-annotated to the Quartus II software. [Figure 7–11](#) shows the design flow with the Cadence Allegro Design Entry CIS software.

Figure 7–11. Design Flow with the Cadence Allegro Design Entry CIS Software





For more information about the Cadence Allegro Design Entry CIS software, including licensing, support, usage, training, and product updates, refer to the Help in the software, go to the Cadence website at www.cadence.com, or go to the EMA Design Automation website at www.ema-eda.com.

Allegro Design Entry CIS Project Creation

The Cadence Allegro Design Entry CIS software has built-in support for creating schematic symbols using pin assignment information imported from the Quartus II software.

If you have not already created a new project in the Cadence Allegro Design Entry CIS software, perform the following steps to create a new project:

1. On the File menu, point to **New** and click **Project**. The New Project Wizard starts.

When you create a new project, you can select the PC Board Wizard, the Programmable Logic Wizard, or a blank schematic.

2. Select the **PC Board Wizard** to create a project where you can select which part libraries to use, or select a blank schematic.

The Programmable Logic Wizard is used only to build an FPGA logic design in the Cadence Allegro Design Entry CIS software, which is unnecessary when using the Quartus II software.

No other special configuration for your project is required. Your new project is created in the specified location and initially consists of two files: the OrCAD Capture Project (**.opj**) file and the Schematic Design (**.dsn**) file.

Generate Part

After you create a new project or open an existing project in the Allegro Design Entry CIS software, you can generate a new schematic symbol based on your Quartus II FPGA design. You can also update an existing symbol if your Pin-Out file has been updated in the Quartus II software. The Cadence Allegro Design Entry CIS software stores component symbols in OrCAD Library (**.olb**) files. When a symbol is placed in a library attached to a project, it is immediately available for instantiation in the project schematic.

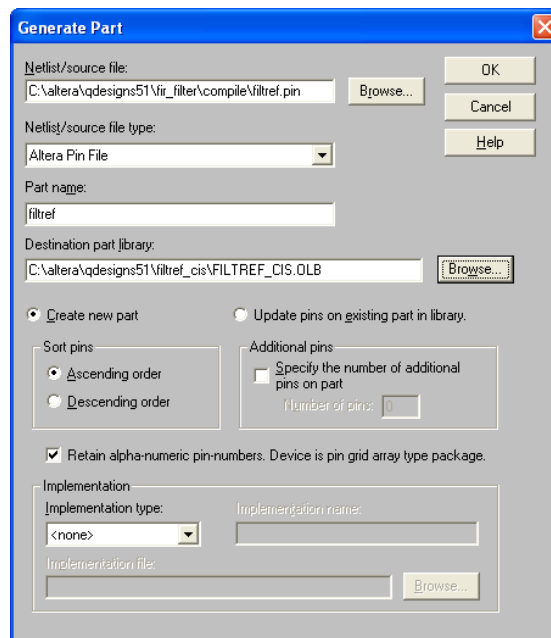
You can add symbols to an existing library or you can create a new library specifically for the symbols generated from your FPGA designs. To create a new library, perform the following steps:

1. On the File menu, point to New and click **Library** in the Cadence Allegro Design Entry CIS software to create a default library named library1.olb. This library appears in the Library folder in the **Project Manager** window of the Cadence Allegro Design Entry CIS software.
2. Right-click the new library and choose **Save As** to specify a desired name and location for the library. The library file is not created until you save the new library.

You can now create a new symbol to represent your FPGA design in your schematic. To generate a schematic symbol, perform the following steps:

1. Start the Cadence Allegro Design Entry CIS software.
2. On the Tools menu, click **Generate Part**. The **Generate Part** dialog box appears (Figure 7–12).

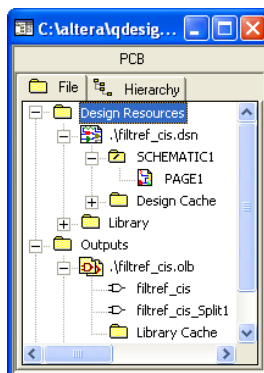
Figure 7–12. Generate Part Dialog Box



3. In the **Netlist/source file type** field, click **Browse** to specify the Pin-Out file from your Quartus II design.
4. In the **Netlist/source file type** list, select **Altera Pin File**.
5. Enter the new part name.
6. Specify the Destination part library for the symbol. If you do not select an existing library for the part, a new library is created with a default name that matches the name of your Design Entry CIS project.
7. Select **Create new part** if you are creating a brand new symbol for this design. Select **Update pins on existing part in library** if you updated your Pin-Out file in the Quartus II software and want to transfer any assignment changes to an existing symbol.
8. Select any other desired options and set Implementation type to **<none>**. The symbol is for a primitive library part based only on the Pin-Out file and does not need a special implementation. Click **OK**.
9. Review the Undo warning and click **Yes** to complete the symbol generation.

The symbol is generated and placed in the selected library or in a new library found in the Outputs folder of the design in the **Project Manager** window. Double-click the name of the new symbol to see its graphical representation and edit it manually using the tools available in the Cadence Allegro Design Entry CIS software.

Figure 7–13. Project Manager Window





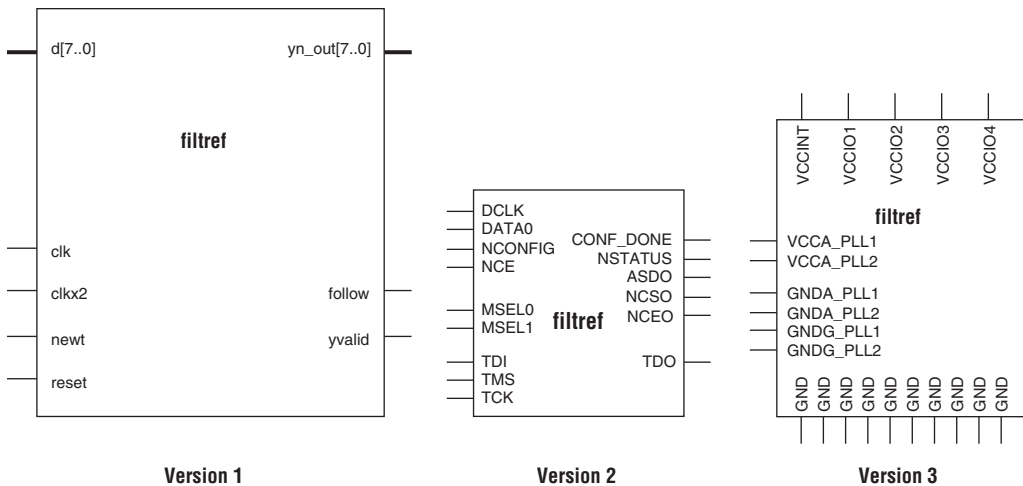
For more information about creating and editing symbols in the Allegro Design Entry CIS software, refer to the Help in the software.

Split Part

Once a new symbol is saved in a project’s library, you can fracture the symbol into multiple parts called sections. Fracturing a part into separate sections is especially useful for FPGA designs. A single symbol for most FPGA packages may be too large for a single schematic page. Splitting the part into separate sections allows you to organize parts of the symbol by function, creating cleaner circuit schematics. For example, you could create one slot for an I/O symbol, a second slot for a JTAG symbol, and a third slot for a power/ground symbol.

Figure 7–14 shows a part fractured into separate sections.

Figure 7–14. Splitting a Symbol into Multiple Sections *Notes (1), (2)*



Notes to Figure 7–14:

- (1) Figure 7–14 represents a Cyclone device with JTAG or passive serial (PS) mode configuration option settings. Symbols created for other devices or other configuration modes may have different sets of configuration pins, but can be fractured in a similar manner.
- (2) Symbol fractures are referred to in different ways in each of the tools described in this chapter. Refer to Table 7–2 for the specific tool naming conventions.
- (3) The power/ground section shows only a representation of power and ground pins. In actuality, the device contains a high number of power and ground pins.



While symbol generation in the Design Entry CIS software refers to symbol fractures as sections, the other tools described in this chapter use different names to refer to symbol fractures. Refer to [Table 7-2 on page 7-14](#) for the symbol fracture naming conventions for each of the tools addressed in this chapter.

To split a part into sections, select the part in its library in the **Project Manager** window of Design Entry CIS. On the Tools menu, click **Split Part** or right-click the part and choose **Split Part**. The **Split Part Section Input Spreadsheet** is shown ([Figure 7-15](#)).

Figure 7-15. Split Part Section Input Spreadsheet

Section Column

Number	Name	Type	Order	Length	User Assig	I/O Bank	Voltage	I/O Standard	Location	Section
1	H1	clk	Input	0	Line				Left	1
2	G1	clkx2	Input	1	Line				Left	1
3	K13	CONF_DONE	Passive	2	Line				Left	2
4	C7	d[0]	Input	3	Line				Left	1
5	A6	d[1]	Input	4	Line				Left	1
6	D7	d[2]	Input	5	Line				Left	1
7	B7	d[3]	Input	6	Line				Left	1
8	B8	d[4]	Input	7	Line				Left	1
9	M7	d[5]	Input	8	Line				Left	1
10	A8	d[6]	Input	9	Line				Left	1
11	B6	d[7]	Input	10	Line				Left	1
12	H2	DATA0	Input	11	Line				Left	2
13	K4	DCLK	Bidirectional	12	Line				Left	2
14	C6	follow	Output	13	Line				Right	1
15	J3	MSEL0	Passive	14	Line				Left	2
16	J2	MSEL1	Passive	15	Line				Left	2
17	J4	nCE	Passive	16	Line				Left	2
18	H4	nCEO	Passive	17	Line				Left	2
19	H3	nCONFIG	Passive	18	Line				Left	2
20	H5	newt	Input	19	Line				Left	1
21	J13	nSTATUS	Passive	20	Line				Left	2
22	G16	reset	Input	21	Line				Left	1

Each row in the spreadsheet represents a pin in the symbol. The spreadsheet column labeled Section indicates the section of the symbol to which each pin is assigned. By default, all pins in a new symbol are located in section 1. Change the values in this column to assign pins to different, new sections of the symbol. You can also specify the side of a section on which the pin will reside by changing the values in the Location column. When you are finished, click **Split**. A new symbol appears in the same library as the original with the name *<original part name>_Split1*.

View and edit each section individually. To view the new sections of the part, double-click the part. The **Part Symbol Editor** window is shown. The first section of the part is displayed for editing. On the View menu, click **Package** to view thumbnails of all the part sections. Double-click a thumbnail to edit that section of the symbol.

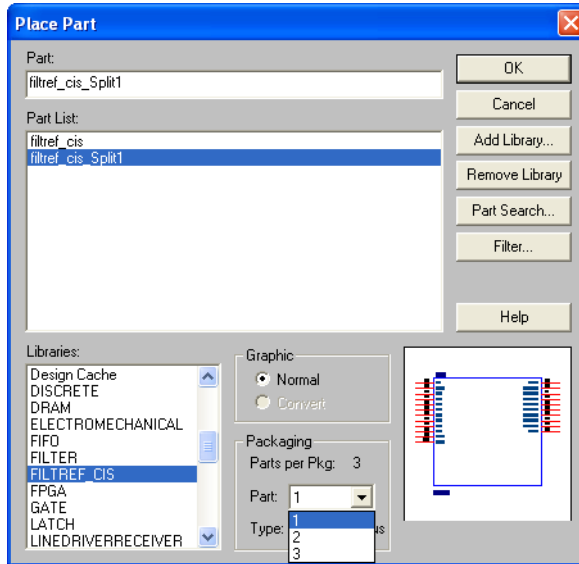


For more information about splitting parts into sections and editing symbol sections in the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

Instantiate Symbol in Design Entry CIS Schematic

After a new symbol is saved in a library in your Design Entry CIS project, you can instantiate it on a page in your schematic. Open a schematic page in the **Project Manager** window of the Cadence Allegro Design Entry CIS software. On the schematic page, to add the newly created symbol to your schematic, on the Place menu, click **Part**. The **Place Part** dialog box appears (Figure 7–16).

Figure 7–16. Place Part Dialog Box



Select the new symbol library location and the newly created part name. If you select a part that is split into sections, you can select the section to place from the Part pop-up menu. Click **OK**. The symbol is now attached to your cursor for placement in the schematic. Click on the schematic page to place the symbol.



For more information about using the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

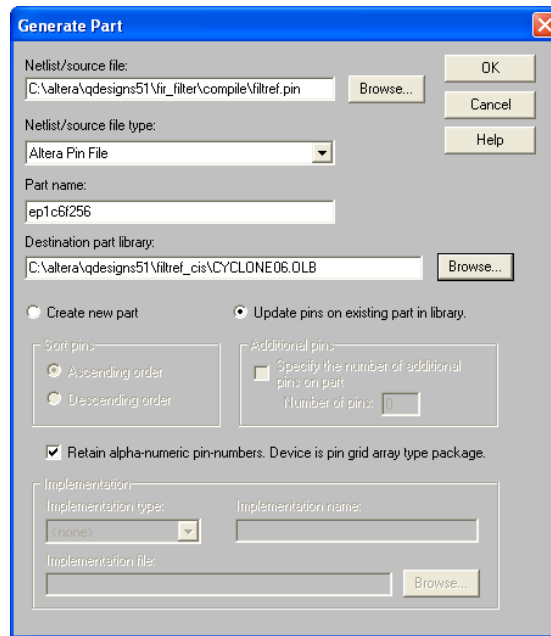
Altera Libraries for Design Entry CIS

Altera provides downloadable OrCAD Library Files for many of its device packages. You can add these libraries to your Design Entry CIS project and update the symbols with the pin assignments contained in the Pin-Out file generated by the Quartus II software. This allows you to use the downloaded library symbols as a base for creating custom schematic symbols with your pin assignments that you can edit or fracture as desired. This can increase productivity by reducing the amount of time it takes to create and edit a new symbol.

To use the Altera-provided libraries with your Design Entry CIS project, perform the following steps:

1. Download the library of your target device from the Download Center page found through the Support page on the Altera website at www.altera.com.
2. Make a copy of the appropriate OrCAD Library file so that the original symbols are not altered. Place the copy in a convenient location such as your Design Entry CIS project directory.
3. In the **Project Manager** window of the Cadence Allegro Design Entry CIS software, click once on the Library folder to select it. On the Edit menu, click **Project** or right-click the Library folder and choose **Add File** to select the copy of the downloaded OrCAD Library file and add it to your project. The new library is added to the list of part libraries for your project.
4. On the Tools menu, click **Generate Part**. The **Generate Part** dialog box appears (Figure 7-17).

Figure 7-17. Generate Part Dialog Box



5. In the **Netlist/source file type** field, click **Browse** to specify the Pin-Out file in your Quartus II design.
6. From the **Netlist/source file type** list, select **Altera Pin File**.
7. For the part name, enter the name of the target device the same as it appears in the downloaded library file. For example, if you are using a device from the CYCLONE06.OLB library, set the part name to match one of the devices in this library such as ep1c6f256. You can rename the symbol later in the **Project Manager** window after the part is updated.
8. Set the Destination part library to the copy of the downloaded library you added to the project.
9. Select **Update pins on existing part in library**. Click **OK**, then click **Yes**.

The symbol is updated with your pin assignments. Double-click the symbol in the **Project Manager** window to view and edit the symbol. On the View menu, click **Package** if you want to view and edit other sections

of the symbol. If the symbol in the downloaded library is already fractured into sections, as some of the larger packages are, you can edit each section but you cannot further fracture the part. Generate a new part without using the downloaded part library if you require additional sections.



For more information about creating, editing, and fracturing symbols in the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

Conclusion

Transferring a complex, high-pin-count FPGA design to a PCB for prototyping or manufacturing is a daunting process that can lead to errors in the PCB netlist or design, especially when different engineers are working on different parts of the project. The design workflow available when the Quartus II software is used with tools from Cadence assists the FPGA designer and the board designer in preventing such errors and focusing all attention on the design.

Document Revision History

Table 7-3 shows the revision history for this document.

Table 7-3. Document Revision History

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history to the document.	
May 2006, v6.0.0	Was chapter 6 in v5.1. Minor updates for the Quartus II software version 6.0.0.	
November 2005, v5.1.1	Realigned figures 6-9 and 6-14.	
October 2005, v5.1.0	Initial release.	

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:

- [Chapter 8, Area & Timing Optimization](#)
- [Chapter 9, Power Optimization](#)
- [Chapter 10, Timing Closure Floorplan](#)
- [Chapter 11, Netlist Optimizations & Physical Synthesis](#)
- [Chapter 12, Design Space Explorer](#)
- [Chapter 13, LogicLock Design Methodology](#)
- [Chapter 14, Synplicity Amplify Physical Synthesis Support](#)



For information about the revision history for chapters in this section, refer to each individual chapter for that chapter's revision history.

Introduction

Techniques for achieving the highest quality of results 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 design requirements by facilitating optimization at multiple points in the design process.

This chapter explains techniques to reduce resource usage, improve timing performance, and reduce compilation times when designing for Altera® devices. It also explains how and when to use some of the features described in other chapters of the *Quartus II Handbook*. The first section, “[Optimization Process Stages](#)”, describes the various stages in a design optimization process, and points you to the appropriate sections in the chapter for area, timing, or compilation time optimization.



For more information about power optimization, refer to the *Power Optimization* chapter in volume 2 of the *Quartus II Handbook*.

The effects of these techniques vary from design to design. Applying each technique does not always improve design results. Settings and options in the Quartus II software have default values that generally provide the best trade-off between compilation time, resource utilization, and timing performance. You can adjust these settings to determine whether other settings provide better results for your design. When using advanced optimization settings and tools, it is important to benchmark their effect on your quality of results and to use them only if they improve results for your design.

Use the optimization flow described in this chapter to explore various compiler settings and determine the techniques that provide the best results.

Optimization Process Stages

The first stage in the optimization process is to perform an initial compilation to view the quality of results for your design. “[Initial Compilation](#)” on page 8–5 provides guidelines on some of the settings and assignments that are recommended for your initial compilation. “[Design Analysis](#)” on page 8–14 explains how to analyze the compilation results.

After you have analyzed the compilation results, perform the optimization stages in the recommended order, as described in this chapter.

For LUT-based devices (FPGAs and MAX[®] II CPLDs), perform optimizations in the following order:

1. If your design does not fit, refer to “[Resource Utilization Optimization Techniques \(LUT-Based Devices\)](#)” on page 8–23 before trying to optimize I/O timing or f_{MAX} timing.
2. If the I/O timing performance requirements are not met, refer to “[I/O Timing Optimization Techniques \(LUT-Based Devices\)](#)” on page 8–40 before trying to optimize f_{MAX} timing.
3. If f_{MAX} performance requirements are not met, refer to “ [\$f_{MAX}\$ Timing Optimization Techniques \(LUT-Based Devices\)](#)” on page 8–47.

For macrocell-based devices (MAX 7000 and MAX 3000 CPLDs), perform optimizations in the following order:

1. If your design does not fit, refer to “[Resource Utilization Optimization Techniques \(Macrocell-Based CPLDs\)](#)” on page 8–65 before trying to optimize I/O timing or f_{MAX} timing.
2. If the timing performance requirements are not met, refer to “[Timing Optimization Techniques \(Macrocell-Based CPLDs\)](#)” on page 8–73.

For techniques to reduce compilation time, which are device-independent, refer to “[Compilation-Time Optimization Techniques](#)” on page 8–80.

You can use all these techniques in the GUI or with Tcl commands. For more information about scripting techniques, refer to “[Scripting Support](#)” on page 8–86.

Design Space Explorer

The Design Space Explorer (DSE) automates the process of running multiple compilations with different settings. You can use DSE to try the techniques described in this chapter. The DSE utility automates the process of finding the best set of options for your design. DSE explores the design space by applying various optimization techniques and analyzing the results.



For more information, refer to the *Design Space Explorer* chapter in volume 2 of the *Quartus II Handbook*.

Optimization Advisors

The optimization advisors provide guidance in making settings that optimize your design. On the Tools menu, point to **Advisors**, and click **Resource Optimization Advisor** or **Timing Optimization Advisor**. The advisors describe many of the suggestions listed in this chapter. The Power Optimization Advisor is also available, to provide guidance for reducing power consumption. In addition, the Incremental Compilation Advisor provides suggestions to improve your quality of results when partitioning your design for a hierarchical or team-based design flow using the Quartus II incremental compilation feature.



For more information about using the Power Optimization Advisor, refer to the *Power Optimization* chapter in volume 2 of the *Quartus II Handbook*. For more information about using the Incremental Compilation Advisor, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

If you open the advisors after compilation, the Resource and Timing Optimization Advisors display icons indicating which resources or timing constraints were not met. The example in [Figure 8–1](#) shows the Timing Optimization Advisor after compiling a design that meets its frequency requirements, but recommends changes be made to the settings to improve the timing.

Figure 8–1. Timing Optimization Advisor

Recommendation	Description
Direct Quartus II Integrated Synthesis to optimize the design for speed.	Most synthesis tools will optimize a design to meet your speed requirements. Some synthesis tools offer an easy way to optimize for speed instead of area. The Quartus II software can optimize a design for speed, choosing a design implementation that has the fastest maximum frequency (fmax).
Summary	The following areas will be affected by the recommended changes: + Delay may decrease (fmax may increase) - Logic element usage may increase = Compilation time is unaffected
Action	For Quartus II Integrated Synthesis, choose Speed under Optimization Technique in the Analysis & Synthesis Settings page of the Settings dialog box (Assignments). It is also recommended to set the optimization technique to Balanced if it is currently set to Area. Balanced technique gives better fmax than Area, worse than Speed. But resource usage is better than with Speed (worse than with Area). You can also specify the Optimization Technique logic option for specific entities in your design using the Assignment Editor (Assignments menu), while leaving the project 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).
Current Global Settings:	Optimization Technique -- Cyclone = AREA (Recommended: SPEED)

This button makes the recommended changes automatically.

These options open the **Settings** dialog box or **Assignment Editor** so you can manually change the settings.

When you expand one of the categories in the Advisor, such as **Maximum Frequency (fmax)** or **I/O Timing (tsu, tco, tpd)**, the recommendations are divided 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. In [Figure 8-1](#), the check mark icons in the list of recommendations for **Stage 1** indicate recommendations that are already implemented. The warning icons indicate recommendations that are not followed for this compilation. The information icon indicates general suggestions. For these entries, the advisor does not report whether these recommendations were followed, but instead explains how you can achieve better performance. Refer to the “How to use” page in the Advisor for a legend that provides more information for each icon.

There is a link from each recommendation to the appropriate location in the Quartus II user interface where you can change the settings. For example, the **Synthesis Netlist Optimizations** page of the **Settings** dialog box or the **Global Signals** category in the Assignment Editor. This approach provides the most control over which settings are made, and helps you learn about the settings in the software. In some cases, you can also use the **Correct the Settings** button, shown in the advisor in [Figure 8-1](#), to automatically make the suggested change to global settings.

For some entries in the advisor, a button appears that allows you to further analyze your design and gives you more information. For example, [Figure 8-2](#) shows the guidelines for the **Use Global Clocks** entry, after the user has clicked **List all clocks**. The advisor provides a table with the clocks in the design, and indicates whether they have been assigned a timing constraint.

Figure 8–2. Timing Optimization Advisor

The screenshot shows the Timing Optimization Advisor window. On the left, a tree view shows the following structure:

- Timing Summary
 - How to use the Timing Optimization Advisor
 - General Recommendations
 - Maximum Frequency (fmax)
 - Specify Fitter effort
 - Optimize for speed
 - Optimize specific clock domains for speed
 - Turn off restructure multiplexers
 - Use netlist optimizations
 - Turn on Auto Global Clock
 - Use global clocks** (highlighted)
 - Stage 1
 - Stage 2
 - Stage 3
 - I/O Timing (tsu, tco, tpd)
 - Hold Time & Minimum Delay Timing

The main panel displays the details for the 'Use global clocks' recommendation:

- Recommendation:** Use global clocks to improve timing
- Description:** Besides minimizing the delay from the register to output pin, minimizing the delay from the clock pin to the register can also improve the tco timing. Altera recommends that you always use the global clock for low-skew and speed-critical signals.
- Summary:** The following areas will be affected by the recommended changes:
 - + Delay may decrease (fmax may increase)
 - = Logic element usage is unaffected
 - = Compilation time is unaffected
- Action:** Use the Assignment Editor (Assignments menu) and the Global Signal option to assign the clocks in your design to global clocks.
 - List all clocks
 - Table with columns: Node Name, Global Resource Used

	Node Name	Global Resource Used
1	clk	Global Clock
2	clkx2	Global Clock

Below the table is a link: [Open Assignment Editor - Global Signals category](#)

Initial Compilation

This section describes the basic assignments and settings to make for your 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.

Device Setting

Assigning a specific device determines the timing model that the Quartus II software uses during compilation. Choose the correct speed grade to obtain accurate results and the best optimization. The device size and the package determine the device pin-out and how many resources are available in the device.

To choose the target device, on the Assignments menu, click **Device**.

Smart Compilation Setting

Smart compilation can reduce compilation time by skipping compiler stages that are not needed to recompile the design. This is especially useful when you perform multiple compilation iterations during the optimization phase of the design process. However, smart compilation uses more disk space. To turn on smart compilation, on the Assignments menu, click **Settings**. In the Category list, select **Compilation Process Settings** and turn on **Use Smart compilation**.



This feature skips entire compiler stages (such as Analysis & Synthesis) when they are not needed. This feature is different from incremental compilation, which you can use to compile parts of your design while preserving results for unchanged parts. For information about using the incremental compilation feature to reduce your compilation time, refer to [“Incremental Compilation” on page 8–80](#).

Partitions & Floorplan Assignments for Incremental Compilation

The Quartus II incremental compilation feature enables hierarchical and team-based design flows where you can compile parts of your design while other parts of the design remain unchanged, or import parts of your design from separate Quartus II projects. For information about using the incremental compilation feature to reduce your compilation time, refer to [“Incremental Compilation” on page 8–80](#).

If you want to take advantage of this feature for a team-based design flow, to reduce your compilation times, or to improve the timing performance of your design during iterative compilation runs, it becomes important that you make meaningful design partitions as well as create a floorplan for your design partitions. These assignments can negatively affect a design's quality of results if you do not follow Altera's recommendations. Good assignments can improve your quality of results.



For guidelines about how to create partition and floorplan assignments for your design, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

Timing Requirement Settings

An important step in the optimal quality of results, especially for high-performance FPGA designs, is to make comprehensive timing requirement settings. It is important to apply these settings for the following reasons:

- 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 8–11](#).
- The Quartus II software performs physical synthesis optimizations based on timing requirements (refer to [“Synthesis Netlist Optimizations & Physical Synthesis Optimizations” on page 8–48](#) for more information).

- Correct timing assignments allow the software to work hardest to optimize the performance of the timing-critical parts of the design, and make trade-offs for performance. This optimization can also save area or power utilization in non-critical parts of the design.



As a general rule, do not over-constrain the software by applying timing requirements that are higher than your design requirements. Use your real design requirements to get the best results. Power utilization may also be larger in an over-constrained design, when the software balances power and performance during compilation.

In some designs with multiple clocks, it may be possible to improve the timing performance on one clock domain while reducing the performance on other clock domains by over-constraining the most important clock. If you use this technique, ensure that any performance improvements that you see are real gains by performing a sweep over multiple seeds. For more information, refer to [“Fitter Seed” on page 8–56](#).

- The Timing Analyzer (Classic or TimeQuest) checks your design against the timing assignments. The Compilation Report and timing analysis reporting commands show whether timing requirements are met, and provide detailed timing information about paths that violate timing requirements.

To make clock assignments for the classic timing analyzer, on the Assignments menu, click **Timing Analysis Settings**. Select the **Classic Timing Analyzer Settings** page. Use the **Delay requirements**, **Minimum delay requirements**, and **Clock Settings** boxes to make global settings, or to apply settings to individual clocks, click **Individual Clocks** (recommended for multiple-clock designs). Create the clock setting, and apply it to the appropriate clock node in the design. The Timing Wizard can also step you through the process of making individual clock constraints for the classic timing analyzer. To run the Timing Wizard, on the Assignments menu, click **Timing Wizard**.

To make clock and timing assignments for the TimeQuest timing analyzer, create a Synopsys Design Constraint (.sdc) file that contains all of your constraints. You can also create constraints in the TimeQuest GUI. Use the `write_sdc` command, or, in the TimeQuest analyzer, on the Constraints menu, click **Write SDC File** to write your constraints to an SDC file. You can add an SDC file to your project on the **TimeQuest Timing Analyzer** page under **Timing Analysis Settings**.

Ensure that every clock signal has an accurate clock setting assignment. If clocks come from a common oscillator, they may be considered related. Ensure that all related or derived clocks are set up correctly in the assignments. All I/O pins that require I/O timing optimization must have settings. You should also specify minimum timing constraints as applicable. If there is more than one clock or there are different I/O requirements for different pins, make multiple clock settings and individual I/O assignments instead of using the global settings.

Make any complex timing assignments required in the design, including any cut-timing and multicycle path assignments. Common situations for these types of assignments include reset or static control signals, cases where it is not important how long it takes a signal to reach a destination, and paths that can operate in more than one clock cycle. These assignments allow the Quartus II software to make appropriate trade-offs between timing paths, and can enable the Compiler to improve timing performance in other parts of the design. Specify these settings in the Assignment Editor.



For more information about timing assignments and timing analysis, refer to the *Classic Timing Analyzer* and the *TimeQuest Timing Analyzer* chapters in volume 3 of the *Quartus II Handbook*.

Timing Constraint Check—Report Unconstrained Paths

To ensure that all constraints or assignments have been applied to design nodes, you can report all unconstrained paths in your design.

Use the following steps to report unconstrained paths in your design using the classic timing analyzer:

1. On the Assignments menu, click **Timing Analysis Settings**. The Settings dialog box appears.
2. Under the Category list, click **Timing Analysis Settings**.
3. Click **Classic Timing Analyzer Settings**, and click **More Settings**.
4. In the **More Timing Settings** dialog box, from the **Existing option settings** list, select **Report Unconstrained Paths**.
5. From the **Setting** drop-down list, select **On**.

When using the TimeQuest timing analyzer, create the report with the **Report Unconstrained Paths** command in the **Task** pane, or the `report_ucp` command.

Optimize Hold Timing

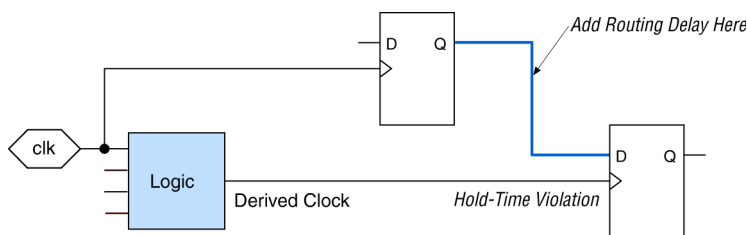
The **Optimize hold timing** option directs the Quartus II software to optimize minimum delay timing constraints. This option is available only for the Stratix® series of devices, Cyclone® series of devices, and MAX II devices. When you turn on this option, the Quartus II software adds delay to connections to guarantee that the minimum delay requirements are satisfied.

When using the Classic Timing Analyzer, if you choose **I/O Paths and Minimum TPD Paths** (the default choice if you turn on **Optimize hold timing**), the Fitter works to meet the following criteria:

- Hold times (t_H) from device input pins to registers
- Minimum delays from I/O pins to I/O registers or from I/O registers to I/O pins
- Minimum clock-to-out time (t_{CO}) from registers to output pins

If you select **All paths** (or if you are using the TimeQuest Timing Analyzer), the Fitter also works to meet hold requirements from registers to registers, as in [Figure 8–3](#), 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, Altera recommends that you correct the problems by making changes to your design, such as using a clock enable signal instead of a derived or gated clock.

Figure 8–3. Optimize Hold Timing Option Fixing an Internal Hold Time Violation



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

Optimize Fast Corner Timing

By default, the Fitter optimizes constraints using the worst-case timing model, which uses the worst-case or slowest timing delay numbers. You can instruct the Fitter to also optimize and analyze the fast delay corner, which uses the best-case or fastest timing numbers.

On the Assignments menu, click **Settings**. In the Category list, select **Fitter Settings** and turn on **Optimize Fast Corner Timing**. Using the two different timing models can be important to account for process, voltage, and temperature variations for each device. Turning this option on increases compilation time by approximately 10%.

Asynchronous Control Signal Recovery/Removal Analysis

Determine whether you require the software to analyze the results of recovery and removal checks for paths that end at an asynchronous clear, preset, or load signal of a register. Recovery time is the minimum length of time an asynchronous control signal, for example, clear and preset, must be stable before the active clock edge. Removal time is the minimum length of time an asynchronous control signal must be stable after the active clock edge.

When using the Quartus II Classic Timing Analyzer for timing analysis, Recovery/Removal analysis is turned off by default. Turning on the option adds additional constraints during placement and routing which can increase compilation time and reduce performance. If this analysis is required, on the Assignments menu, click **Settings**. In the Category list, select **Timing Requirements & Options**, then click **More Settings**. Turn on **Enable Recovery/Removal analysis**.

When using TimeQuest for timing analysis, Recovery/Removal analysis and optimization are always performed during placement and routing. You can use the `create_timing_summary` Tcl command to report the recovery and removal analysis. The slack for Removal/Recovery is determined in a very similar way to Setup and Hold checks.



For more details about Recovery/Removal analysis with TimeQuest, refer to the *TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

For designs containing FIFOs, Altera recommends turning Recovery/Removal analysis on. Recovery/Removal analysis helps to analyze corner-case conditions to achieve better functional coverage.

Fitter Effort Setting

On the Assignments menu, click **Settings**. In the Category list, select **Fitter Settings**. The default setting depends on the device family specified.

Use the **Standard Fit** option to exceed specified timing requirements and achieve the best possible timing results for your design. However, this setting usually increases 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%, resulting in a fit that has, on average, 10% lower f_{MAX} than that achieved using the **Standard Fit** setting. For a small fraction of hard-to-fit circuits, the reduced optimization that results from using the **Fast Fit** option can cause the first fitting attempt to fail due to routing problems, resulting in multiple fitting attempts and increased compilation time.

The **Auto Fit** option (available only for the Stratix and Cyclone series of devices, and MAX II devices) decreases compilation time by directing the Fitter to reduce Fitter effort after meeting the design's timing requirements and 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, specify a minimum slack before reducing Fitter effort in the **Desired worst case slack** box.

The **Auto Fit** option also causes the Quartus II Fitter to optimize for shorter compilation times instead of maximum performance when there are no timing assignments. For designs with no timing assignments, the resulting f_{MAX} is, on average, 10% 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 compilation time is the same as using the **Standard Fit** option. For designs with no timing requirements, or easily achieved timing requirements, you can achieve an average compilation time reduction of 40% by using the **Auto Fit** option.



Selecting this option does not guarantee that the Fitter meets the design's timing requirements, and specifying a minimum slack does not guarantee that the Fitter achieves the slack requirement.

I/O Assignments

The I/O standards and drive strengths specified for a design affect I/O timing. Specify I/O 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, leave pin locations unconstrained to achieve the best results. If your pin locations are already fixed, make pin assignments to constrain the compilation appropriately. “[Resource Utilization Optimization Techniques \(Macrocell-Based CPLDs\)](#)” on page 8–65 includes recommendations for making pin assignments that can have a larger affect on your quality of results in smaller macrocell-based architectures.

Use the **Assignment Editor** and **Pin Planner** to assign I/O standards and pin locations.



For more information about I/O standards and pin constraints, refer to the appropriate handbook. For information about planning and checking I/O assignments, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*. For information about using the Assignment Editor, refer to the *Assignment Editor* chapter in volume 2 of the *Quartus II Handbook*.

Early Timing Estimation

The Quartus II software provides an Early Timing Estimation feature that estimates your design’s timing results before the software performs full placement and routing. On the Processing menu, point to Start, and click **Start Early Timing Estimate** to generate initial compilation results after you have run analysis and synthesis. When you want a quick estimate of a design’s performance before proceeding with further design or synthesis tasks, this command can save significant compilation time. Using this feature provides a timing estimate up to 45× faster than running a full compilation, and the fit is not fully optimized or routed. Therefore, the timing report is only an estimate. On average, the estimated delays are within 11% of those achieved by a full compilation compared to the final timing results.

You can specify what type of delay estimates to use with this feature. On the Assignments menu, click **Settings**. In the Category list, select **Compilation Process Settings**, and select **Early Timing Estimate**. On the **Early Timing Estimate** page, the following options are available:

- The **Realistic** option, which is the default, generates delay estimates that will likely be close to the results of a full compilation.
- The **Optimistic** option uses delay estimates that are lower than those likely to be achieved by a full compilation, which results in an optimistic performance estimate.
- The **Pessimistic** option uses delay estimates that are higher than those likely to be achieved by a full compilation, which results in a pessimistic performance estimate.

All three options offer the same reduction in compilation time.

You can use the Timing Closure Floorplan or Chip Planner (for supported devices) to view the placement estimate created by this feature to identify critical paths in the design. Then, if necessary, you can add or modify floorplan constraints such as LogicLock™ regions, or make other changes to the design. You can then rerun the Early Timing Estimator to quickly assess the impact of any floorplan assignments or logic changes, enabling you to try different design variations and find the best solution.

Design Assistant

You can run the Design Assistant to analyze the post-fitting results of your design during a full compilation. The Design Assistant checks rules related to areas such as gated clocks, reset signals, asynchronous design practices, and signal race conditions. This is especially useful during the early stages of your design, so that you can work on any areas of concern in your design before proceeding with design optimization.

On the Assignments menu, click **Settings**. In the Category list, select **Design Assistant** and turn on **Run Design Assistant during compilation**.

You can also specify which rules you want the Design Assistant to apply when analyzing and generating messages for a design.



For more information about the rules in the design assistant, refer to the *Design Recommendations for Altera Devices* chapter in volume 1 of the *Quartus II Handbook*.

Design Analysis

The initial compilation establishes whether the design achieves a successful fit and meets the specified performance. This section describes how to analyze your design results in the Quartus II software. After design analysis, proceed to optimization as described in “[Optimization Process Stages](#)” on page 8–1.

Error & Warning Messages

After your initial compilation, it is important to evaluate all error and warning messages to see if any design or setting changes are required. If needed, make these changes and recompile the design before proceeding with design optimization.

To suppress messages that you have evaluated and that can be ignored, right-click on the message in the Messages window and click **Suppress**.



For more information about message suppression, refer to the *Message Suppression* section in the *Quartus II Project Management* chapter in volume 2 of the *Quartus II Handbook*.

Ignored Timing Assignments

You can use the Ignored Timings Assignments page in the Compilation Report to view any assignments that were ignored by the Classic Timing Analyzer during the previous compilation. The Classic Timing Analyzer ignores assignments that are invalid, conflict with other assignments, or that become obsolete through the use of other assignments. If any assignments have been ignored, analyze why they have been ignored. If needed, correct the assignments and recompile the design before proceeding with design optimization.

If you are using TimeQuest for timing analysis, you can use the following command to get a listing of ignored timing constraints:

```
report_sdc -ignored -panel_name "Ignored Constraints"
```



For more information about the `report_sdc` command and its options, refer the *TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Resource Utilization

Determining device utilization is important regardless of whether a successful fit is achieved. If your compilation results in a no-fit error, resource utilization information is important so you can 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 other design changes will introduce fitting difficulties.

To determine resource usage, refer to 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 (for Stratix and Stratix II devices) or 18-bit elements (for Stratix III devices), 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**.



For the Stratix II and Stratix III devices, a device with low utilization does not have the lowest adaptive logic module (ALM) utilization possible. For these devices, the Fitter uses adaptive look-up tables (ALUTs) in different ALMs, even when the logic can be placed within one ALM, so that it can achieve the best timing and routability results. In achieving these results, logic might be spread throughout the device. 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. Therefore, a design that is reported as close to 100% full might still have space for extra logic if logic and registers can be packed together more aggressively.

If resource usage is reported as less than 100% and a successful fit cannot be achieved, either there are not enough routing resources or some assignments are illegal. In either case, a message appears in the **Processing** tab of the Messages window describing the problem.

If the Fitter finishes very quickly compared to fitter runs on similar designs, then a resource might be over-utilized or there might be an illegal assignment. If the Quartus II software seems to run for an excessively long time compared to runs on similar designs, then a legal placement or route probably cannot be found. Look for errors and warnings that indicate these types of problems.

You can use the Timing Closure Floorplan or Chip Planner (for supported devices) to find areas of the device that have routing congestion.



For details about using the Timing Closure Floorplan, refer to the *Timing Closure Floorplan* chapter in volume 2 of the *Quartus II Handbook*. For details about using the Chip Planner, refer to the *Design Analysis & Engineering Change Management with Chip Planner* chapter in volume 3 of the *Quartus II Handbook*.

I/O Timing (Including t_{PD})

The TimeQuest Timing Analyzer supports Synopsys Design Constraints (SDC) format for constraining your design. While using TimeQuest for timing analysis, use the `set_input_delay` constraint to specify the data arrival time at an input port with respect to a given clock. For output ports, use the `set_output_delay` command with respect to a given clock. You can use the `report_timing` Tcl command to generate the required I/O timing reports.




The rest of this section refers to timing settings and analysis in the Quartus II classic timing analyzer. For more details about equivalent settings and analysis in the TimeQuest Timing Analyzer, refer to the *TimeQuest Timing Analyzer* and *Switching to the TimeQuest Timing Analyzer* chapters in volume 3 of the *Quartus II Handbook*.

When using the classic timing analyzer, from the Compilation Report, use the **Timing Analyzer** to determine whether or not I/O timing has been met. The t_{SU} , t_H , and t_{CO} reports list the I/O paths, together with the required timing number if you have made a timing requirement, the actual timing number for the timing as reported by the Quartus II software, and the slack, or difference between your requirement and the actual number. If you have any point-to-point propagation delay (t_{PD}) assignments, the t_{PD} report lists the corresponding paths.

The I/O paths that do not meet the required timing performance are reported as having negative slack and are displayed in red (Figure 8-4). In cases when you do not make an explicit I/O timing assignment to an I/O pin, the Quartus II timing analysis software still reports the **Actual** number, which is the timing number that must be met for that timing parameter when the device runs in your system.

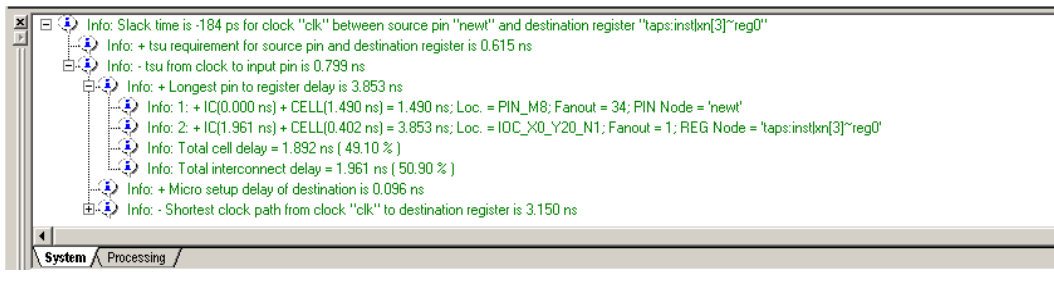
Figure 8-4. I/O Timing Analyzer Report

	Slack	Required tsu	Actual tsu	From	To	To Clock
1	-0.049 ns	0.300 ns	0.349 ns	newt	taps:instlkn_2[6]	clk
2	-0.049 ns	0.300 ns	0.349 ns	newt	taps:instlkn[6]	clk
3	-0.049					clk
4	-0.049					clk
5	0.061 r					ap1 clk
6	0.065 r					dle clk
7	0.072 r					clk
8	0.173 r					clk
9	0.173 r					clk
10	0.173 r					clk
11	0.173 r					clk
12	0.173 r					clk
13	0.173 ns	0.300 ns	0.127 ns	newt	taps:instlkn_2[0]	clk
14	0.173 ns	0.300 ns	0.127 ns	newt	taps:instlkn[0]	clk
15	0.173 ns	0.300 ns	0.127 ns	newt	taps:instlkn_3[5]	clk
16	0.173 ns	0.300 ns	0.127 ns	newt	taps:instlkn_3[4]	clk

To analyze the reasons why your timing requirements are not met, right-click an entry in the report and click **List Paths** (Figure 8-4). A message listing the paths appears in the **System** tab of the Messages window. To expand a selection, click the  icon at the beginning of the line (Figure 8-5). This is a good method to determine where the greatest delay is located along the path.

The List Paths report lists the slack time and how that slack time was calculated. By expanding the various 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 8–5. I/O Slack Report



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



For more information about how timing numbers are calculated, refer to the *Classic Timing Analyzer* or *TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

f_{MAX} Timing

When you are using TimeQuest for timing analysis, you can use constraints such as `set_input_delay` and `set_output_delay` to analyze the path between any two registers. Use the `report_timing` command to generate the required timing reports for any register-to-register path.




The remainder of this section refers to timing settings and analysis in the Quartus II classic timing analyzer. For more information about equivalent settings and analysis in the TimeQuest Timing Analyzer, refer to the *TimeQuest Timing Analyzer* or the *Switching to the TimeQuest Timing Analyzer* chapters in volume 3 of the *Quartus II Handbook*.

When using the classic timing analyzer, in the Compilation Report window, use the **Timing Analyzer** section to determine whether f_{MAX} timing requirements are met. The **Clock Setup** folder gives you figures for the actual register-to-register f_{MAX} for each clock as reported by the Quartus II software, and the slack, or difference between the timing

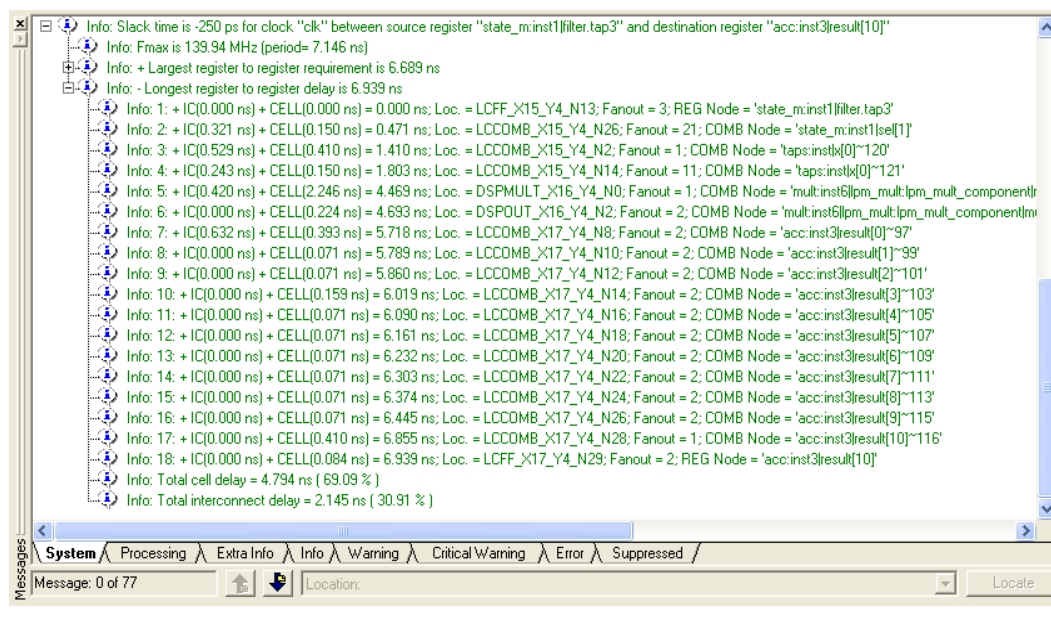
requirement you specified and the actual compilation results. The paths that do not meet timing requirements are shown with a negative slack and appear in red (Figure 8–6).

Figure 8–6. f_{MAX} Timing Analysis Report

	Slack	Actual fmax (period)	From	To	From Clock
1	-0.321 ns	138.56 MHz (period = 7.217 ns)	state_m_inst1 filter.tap3	acc_inst3 result[11]	clk
2	-0.250 ns	139.94 MHz (period = 7.146 ns)	state_m_inst1 filter.tap3	acc_inst3 result[10]	clk
3	-0.179 ns		filter.tap3	acc_inst3 result[9]	clk
4	-0.173 ns		filter.tap4~_Duplicate_1	acc_inst3 result[11]	clk
5	-0.108 ns		filter.tap3	acc_inst3 result[8]	clk
6	-0.102 ns		filter.tap4~_Duplicate_1	acc_inst3 result[10]	clk
7	-0.037 ns		filter.tap3	acc_inst3 result[7]	clk
8	-0.031 ns		filter.tap4~_Duplicate_1	acc_inst3 result[9]	clk
9	0.026 ns		filter.tap2	acc_inst3 result[11]	clk
10	0.034 ns		filter.tap3	acc_inst3 result[6]	clk
11	0.040 ns		filter.tap4~_Duplicate_1	acc_inst3 result[8]	clk
12	0.097 ns		filter.tap2	acc_inst3 result[10]	clk
13	0.105 ns	147.25 MHz (period = 6.791 ns)	state_m_inst1 filter.tap3	acc_inst3 result[5]	clk
14	0.111 ns	147.38 MHz (period = 6.785 ns)	state_m_inst1 filter.tap4~_Duplicate_1	acc_inst3 result[7]	clk
15	0.176 ns	148.81 MHz (period = 6.720 ns)	state_m_inst1 filter.tap3	acc_inst3 result[4]	clk

To analyze why your timing requirements were not met, right-click on an entry in the report and click **List Paths** (Figure 8–6). A message listing the paths appears in the **System** tab of the Messages window. To expand a selection, as shown in Figure 8–7, click the  icon at the beginning of the line. This is a good way to determine where the greatest delay is located along the path.

The List Paths report shows the slack time and how that slack time was calculated. By expanding the various 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 8-7. f_{MAX} Slack Report

To visually analyze f_{MAX} paths, right-click on a path, point to **Locate**, and click **Locate in Timing Closure Floorplan**. For Stratix III devices, click on **Locate in Chip Planner** for performing this analysis. The Timing Closure Floorplan or the Chip Planner is shown and the path is highlighted. Use the **Critical Path Settings** to select which failing paths to show. To turn critical paths on or off, use the **Show Critical Paths** command.



For more information about how timing analysis results are calculated, refer to the *Classic Timing Analyzer* or the *TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

You also can see the logic in a particular path by cross-probing to the RTL Viewer or Technology Map Viewer. These viewers allow you to see a gate-level or technology-mapped representation of your design netlist. To locate a timing path in one of the viewers, right-click on a path in the report, point to **Locate**, and click **Locate in RTL Viewer** or **Locate in Technology Map Viewer**. When you locate a timing path in the Technology Map Viewer, the annotated schematic displays the same delay information that is shown when you use the **List Paths** command.



For more information about the netlist viewers, refer to the *Analyzing Designs with Quartus II Netlist Viewers* chapter in volume 1 of the *Quartus II Handbook*.

Tips for Analyzing Failing Paths

When you are analyzing clock path failures, focus on improving the paths that show the worst slack. The Fitter works hardest on paths with the least slack. If you fix these paths, the Fitter may be able to improve the other failing timing paths in the design.

Check for particular nodes that appear in many failing paths. Look for paths that have common source registers, destination registers, or common intermediate combinational nodes. In some cases, the registers may not be identical, but are part of the same bus. In the timing analyzer report panels, clicking on the **From** or **To** column headers can be helpful to sort the paths by the source or destination registers. Clicking first on **From**, then on **To**, uses the **To** register as the primary sort and **From** as the secondary sort. If you see common nodes, these nodes indicate areas of your design that could possibly be improved through source code changes or Quartus II optimization settings. Constraining the placement for just one of the paths could decrease the timing performance for other paths by moving the common node further away in the device.

Tips for Analyzing Failing Clock Paths that Cross Clock Domains

When analyzing clock path failures, check whether these paths cross between two clock domains. This is the case if the **From Clock** and **To Clock** in the timing analysis report are different. There could also be paths that involve a different clock in the middle of the path, even if the source and destination register clock are the same. To analyze these paths in more detail, right-click on the entry in the report and click **List Paths**.

Expand the **List Paths** entry in the Messages window and analyze the largest register-to-register requirement. Evaluate the setup relationship between the source and destination (launch edge and latch edge) to determine if that is reducing the available setup time. For example, the path may go from a rising edge to a falling edge, which reduces the setup relationship by one half clock cycle.

Check if the PLL phase shift is reducing the setup requirement. You may be able to adjust this using PLL parameters and settings.

Check if the PLL compensation delay is reducing the setup relationship. If you are using the classic timing analyzer, you can direct the software to analyze this delay as clock skew by enabling Clock Latency. On the **Assignments** menu, click **Settings** and choose **Timing Requirements & Options**. Click **More Settings** and turn on **Enable Clock Latency**. You should typically enable this option if your design results in timing violations for paths that pass between PLL clock domains. The TimeQuest Timing Analyzer performs this analysis by default.

Paths that cross clock domains are generally protected with synchronization logic (for example, FIFOs or double data-sync registers) to allow asynchronous interaction between the two clock domains. In such cases, you can to ignore the timing paths between registers in the two clock domains while running timing analysis, even if the clocks are related.



For more details about how to do this, refer to the *Classic Timing Analyzer* chapter or the *TimeQuest Timing Analyzer* chapter in volume 3 of the Quartus II handbook.

Evaluate the clock skew between the source clock and the destination clock to determine if that is reducing the available setup time. You can check the shortest and longest clock path reports to see what is causing any clock skew. Avoid using combinational logic in clock paths because it contributes to clock skew. Differences in the logic or in its routing between the source and destination can cause clock skew problems and result in warnings during compilation.

Global Routing Resources

Check the global signal utilization in your design to ensure that appropriate signals have been placed on global routing resources. In the Compilation Report, open the Fitter report and click the **Resource Section**. Analyze the **Global & Other Fast Signals** and **Non-Global High Fan-out Signals** reports to see if any changes are required.

You may be able to reduce clock skew for high fan-out signals by placing them on global routing resources. Conversely, you can reduce the insertion delay of low fan-out signals by removing them from global routing resources. Doing so can improve clock enable timing and control signal recovery/removal timing, but increases clock skew. You can also use the **Global Signal** setting in the **Assignment Editor** to control global routing resources. If the signal fan-out is low and it feeds locations in one area of the device, you can use any regional or fast regional clocks available.

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 also includes physical synthesis if you turned on those options. The **Flow Elapsed Time** section of the Compilation Report shows how much time is spent running the Analysis & Synthesis and Fitter modules. The **Fitter Messages** report in the **Fitter** section of the Compilation Report shows specifically how much time was spent in placement and how much time was spent in routing.



The applicable messages are indicated as follows, with each time increment in two-digit format:

```
Info: Fitter placement operations ending:
elapsed time = <hour:min:sec>
```

```
Info: Fitter routing operations ending: elapsed
time = <hour:min:sec>
```

Placement is the process of finding optimum locations for the logic in your design. Routing is 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 easier to route.

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. Ensure that you have already set the basic constraints described in [“Initial Compilation” on page 8-5](#) before proceeding with the resource utilization optimizations discussed in this section. If a design does not fit into a specified device, use the techniques in this section to achieve a successful fit. After you optimize resource utilization and your design fits in the desired target device, optimize I/O timing as described in [“I/O Timing Optimization Techniques \(LUT-Based Devices\)” on page 8-40](#).

Resolving Resource Utilization Issues Summary

Resource utilization issues can be divided into the following three categories:

- Issues relating to I/O pin utilization or placement, including dedicated I/O blocks such as PLLs or LVDS transceivers ([“I/O Pin Utilization or Placement” on page 8-24](#)).

- Issues relating to logic utilization or placement, including logic cells containing registers and look-up tables as well as dedicated logic such as memory blocks and DSP blocks (“[Logic Utilization or Placement](#)” on page 8–25).
- Issues relating to routing (“[Routing](#)” on page 8–36).

I/O Pin Utilization or Placement

Use the suggestions in the following sections to help you resolve I/O resource problems.

Use I/O Assignment Analysis

On the Processing menu, point to **Start** and click **Start I/O Assignment Analysis** to help with pin placement. The **Start I/O Assignment Analysis** command allows you to check your I/O assignments early in the design process. You can use this command to check the legality of pin assignments before, during, or after compilation 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.

Common issues with I/O placement relate to the fact that differential standards have specific pin pairings, and certain I/O standards may be supported only on certain I/O banks.

If your compilation or I/O assignment analysis results in specific errors relating to I/O pins, follow the recommendations in the error message. Right-click on the message in the Messages window and click **Help** to open the Quartus II Help topic for this message.

Modify Pin Assignments or Choose a Larger Package

If a design that has pin assignments fails to fit, compile the design without the pin assignments to determine whether a fit is possible for the design in the specified device and package. You can use this approach if a Quartus II error message indicates fitting problems due to pin assignments.

If the design fits when all pin assignments are ignored or when several pin assignments are ignored or moved, you may have 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 (which could be the same device density) that has more available user I/O pins.



For more information about I/O assignment analysis, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

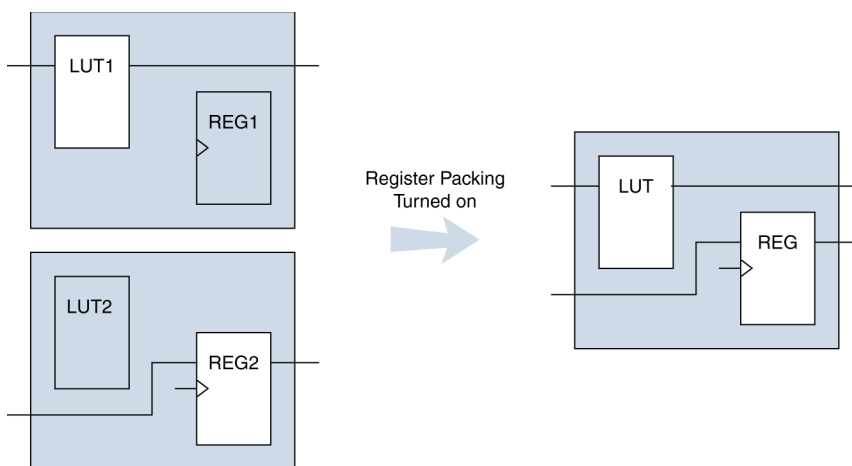
Logic Utilization or Placement

Use the suggestions in the following subsections to help you resolve logic resource problems, including logic cells containing registers and look-up tables (LUTs) as well as dedicated logic such as memory blocks and DSP blocks.

Use Register Packing

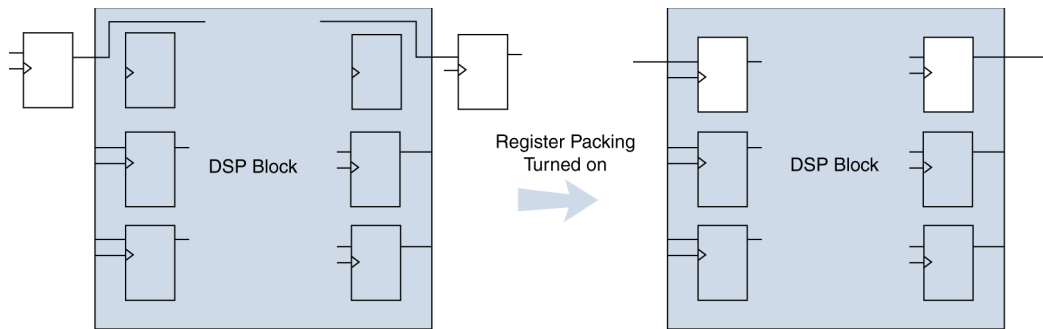
The **Auto Packed Registers** option implements the functions of two cells into one logic cell by combining the register of one cell in which only the register is used with the LUT of another cell in which only the LUT is used. [Figure 8-8](#) shows register packing and the gain of one logic cell in the design.

Figure 8-8. Register Packing



Registers can also be packed into DSP blocks (Figure 8–9).

Figure 8–9. Register Packing in DSP Blocks



The following list shows the most common cases in which register packing helps 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 and Cyclone series devices, 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 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 and Cyclone series devices, and MAX II devices.
- **Sparse**—In this mode, the combinational (LUT) and sequential (register) functions are combined such that the combined logic has either a combinational output or a sequential output but not both. This mode is available only for Cyclone II, Stratix II, and Stratix III devices. This option results in a higher LAB usage, but might give you better timing performance because of reduced routing congestion.
- **Sparse Auto**—In this mode, the Quartus II Fitter starts with sparse mode packing, and then attempts to achieve best performance while maintaining a fit for the specified device. Later optimizations are carried out in a way similar to the **Auto** mode. This mode is available only for Cyclone II, Stratix II, and Stratix III devices.

For the Cyclone and Stratix series of devices and MAX II devices, the default register packing mode is **Auto**. For other devices, the default is **Normal**.

Turning on register packing decreases the number of logic elements (LEs) or adaptive logic modules (ALMs) in the design, but could also decrease performance in some cases. On the Assignments menu, click **Settings**. In the Category list, select **Fitter Settings**, and then click **More Settings**. Turn on **Auto Packed Registers** to turn on register packing.

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 8-1](#) shows typical results for Stratix II and Stratix III devices. [Table 8-2](#) shows typical results for Cyclone II devices, and [Table 8-3](#) shows typical results for Stratix, Stratix GX, and Cyclone devices.

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

Table 8–1. Typical Register Packing Results for Stratix II & Stratix III Devices

Register Packing Setting	Relative f_{MAX}	Relative ALM 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 8–2 shows typical results for Cyclone II devices.

Table 8–2. 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 8-3 shows 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 conflicting constraints or constraints that are difficult to meet may not fit the targeted device. This can occur 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 or Chip Planner (for supported devices) to locate routing problems in the floorplan, then remove any location 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. You can delete specific location assignments in the Assignment Editor or Timing Closure Floorplan or the Chip Planner (for supported devices). Remove LogicLock assignments in the Timing Closure Floorplan (or in the Chip Planner), in the **LogicLock Regions Window**, or, on the Assignments menu, click **Remove Assignments**. Turn on the assignment categories you want to remove from the design in the **Available assignment categories** list.



For more information about the **Routing Congestion** view in the Timing Closure Floorplan, refer to the Quartus II Help.

Perform WYSIWYG Resynthesis with Balanced or Area Setting

If you use another EDA synthesis tool and want to determine if the Quartus II software can remap the circuit to use fewer LEs or ALMs, follow these steps:

1. On the Assignments menu, click **Settings**. In the Category list, select **Analysis & Synthesis Settings**, and turn on **Perform WYSIWYG primitive resynthesis (using optimization techniques specified in Analysis & Synthesis settings)** on the **Synthesis Netlist Optimizations** page. Or, on the Assignments menu, click **Assignment Editor**, and apply the **Perform WYSIWYG Primitive Resynthesis** logic option to a specific module in your design.
2. On the Assignments menu, click **Settings**. In the Category list, select **Analysis & Synthesis Settings**, and choose **Balanced** or **Area** under Optimization Technique. Or, on the Assignments menu, click **Assignment Editor**. Set the Optimization Technique to **Balanced** or **Area** for a specific module in your design.
3. Recompile the design.



The **Balanced** setting typically produces utilization results that are very similar to the **Area** setting, with better performance results. The **Area** setting may give better results in some unusual cases. Performing WYSIWYG resynthesis for area in this way typically reduces f_{MAX} .

Optimize Synthesis for Area, Not Speed

If your design fails to fit because it uses too much logic, resynthesize the design to improve the area utilization. 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 can result in higher device resource usage if the constraints are too aggressive.

If resource utilization is an important concern, some synthesis tools offer an easy way to optimize for area instead of speed. If you are using Quartus II integrated synthesis, choose **Balanced** or **Area** for the **Optimization Technique**. 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 using the **Area** setting (potentially at the expense of f_{MAX} 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**. You can also use the **Speed Optimization Technique for Clock Domains** logic option to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.

In some synthesis tools, not specifying an f_{MAX} requirement may result in less resource utilization.



In the Quartus II software, the **Balanced** setting typically produces utilization results that are very similar to those produced by the **Area** setting, with better performance results. The **Area** setting may give better results in some unusual cases.



For information about setting timing requirements and synthesis options in Quartus II integrated synthesis and other synthesis tools, refer to the appropriate chapter in the *Synthesis* section in volume 1 of the *Quartus II Handbook*, or your synthesis software's documentation.

Other attributes or options can also help improve the quality of synthesis results, including the recommendations in the following sections.

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 resource utilization. The effect of state machine encoding varies 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. When using Quartus II Integrated Synthesis, go to the Assignments menu and click **Settings**. In the Category list, select **Analysis & Synthesis Settings** and turn on **Minimal Bits for State Machine Processing**. You also can specify this logic option for specific modules or state machines in your design with the Assignment Editor.

Flatten the Hierarchy During Synthesis

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 can reduce area. Therefore, to achieve the best results, flatten your design hierarchy whenever possible. If you are using Quartus II integrated synthesis,

ensure that the **Preserve Hierarchical Boundary** logic option is turned off, that is, make sure that you have not turned on the option in the Assignment Editor or with Tcl assignments. If you are using Quartus II incremental compilation, you cannot flatten your design across design partitions. Incremental compilation always preserves the hierarchical boundaries between design partitions. Follow Altera's recommendations for design partitioning, such as registering partition boundaries to reduce the effect of cross-boundary optimizations.



For more information about using incremental compilation and recommendations for design partitioning, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*. If you are using an incremental synthesis flow that requires separate hierarchy blocks, you can find additional recommendations for design partitioning in the *Design Recommendations for Altera Devices* chapter in volume 1 of the *Quartus II Handbook*.

Restructure Multiplexers

Multiplexers form a large portion of the logic utilization in many FPGA designs. By optimizing your multiplexed logic, you can achieve a more efficient implementation in your Altera device.

The Quartus II software provides the **Restructure Multiplexers** logic option, which can extract and optimize buses of multiplexers during synthesis. This option is available on the **Analysis & Synthesis Settings** page of the **Settings** dialog box and is useful if your design contains buses of fragmented multiplexers. This option restructures multiplexers more efficiently for area, allowing the design to implement multiplexers with a reduced number of LEs or ALMs. Using the Restructure Multiplexers logic option can reduce your design's f_{MAX} . This option is turned on automatically when you set the **Quartus II Analysis & Synthesis Optimization Technique** option to **Area** or **Balanced**. To change the default setting, on the Assignments menu, click **Settings**. In the Category list, select **Analysis & Synthesis Settings**, and click the appropriate option from the **Restructure Multiplexers** list to set the option globally.



For design guidelines to achieve optimal resource utilization for multiplexer designs, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*. For more information about the Restructure Multiplexers option in the Quartus II software, refer to the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*.

Retarget Memory Blocks

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

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

ROM and RAM memory blocks can also be inferred from your HDL code, and your synthesis software can place large shift registers into memory blocks by inferring the `altshift_taps` megafunction. This inference can be turned off in your synthesis tool to cause the memory to be placed in logic instead of in memory blocks. To disable inference when using Quartus II Integrated Synthesis, on the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis**, and turn off the **Auto RAM Replacement**, **Auto ROM Replacement**, or **Auto Shift Register Replacement** logic option as appropriate for your project. Or, disable the option for a specific entity in the Assignment Editor.

Depending on your synthesis tool, you can also set the RAM block type for inferred memory blocks. In Quartus II integrated synthesis, set the **ramstyle** attribute to the desired memory type for the inferred RAM blocks, or set the option to **logic** to implement the memory block in standard logic instead of a memory block.

Consider the resource utilization by hierarchy in the report file, and determine whether there is an unusually high register count in any of the modules. Some coding styles may prevent the Quartus II software from inferring RAM blocks from the source code because of their architectural implementation, and the software must implement the logic in flip-flops, instead. As an example, a function such as an asynchronous reset on a register bank might make it incompatible with the RAM blocks in the device architecture, and the register bank is implemented in flip-flops. Often it is possible to move a large register bank into RAM by slight modification of associated logic.



For more information about memory inference control in other synthesis tools, refer to the appropriate chapter in the *Synthesis* section in volume 1 of the *Quartus II Handbook*, or your synthesis software's documentation. For more information about coding styles and HDL examples that ensure memory inference, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*.

Retarget or Balance DSP Blocks

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

If the DSP function was created with the MegaWizard Plug-In Manager, open the MegaWizard Plug-In Manager and edit the function so it targets logic cells instead of DSP blocks. The Quartus II software uses the `DEDICATED_MULTIPLIER_CIRCUITRY` megafunction parameter to control the implementation.

DSP blocks also can be inferred from your HDL code for multipliers, multiply-adders, and multiply-accumulators. This inference can be turned off in your synthesis tool. When you are using Quartus II integrated synthesis, you can disable inference by turning off the **Auto DSP Block Replacement** logic option for your whole project. On the Assignments menu, click **Settings**. In the Category list, select **Analysis & Synthesis Settings**, and turn off **Auto DSP Block Replacement**. Alternatively, you can disable the option for a specific block with the Assignment Editor.



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

The Quartus II software also offers the **DSP Block Balancing** logic option, which implements DSP block elements in logic cells or in different DSP block modes. The default **Auto** setting allows DSP block balancing to convert the DSP block slices automatically as appropriate to minimize the area and maximize the speed of the design. You can use other settings for a specific node or entity, or on a project-wide basis, to control how the Quartus II software converts DSP functions into logic cells and DSP blocks. Using any value other than **Auto** or **Off** overrides the `DEDICATED_MULTIPLIER_CIRCUITRY` parameter used in megafunction variations.



For more details about the Quartus II logic options described in this section, refer to the Quartus II Help.

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 of the design, modify the design at the source to achieve the desired results. You can often improve logic significantly by making design-specific changes to your source code. This is typically the most effective technique for improving the quality of your results.

If your design does not fit into available LEs or ALMs, but you have unused memory or DSP blocks, check to see if 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 dedicated memory or DSP resources in the target device.

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. 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 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*. 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*.

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.

Routing

Use the suggestions in the following subsections to help you resolve routing resource problems.

Set Auto Register Packing to Auto

This option is useful for reducing LE or ALM count in a design. This option is available for the Cyclone and Stratix series of devices. On the Settings menu, select **Fitter Settings**. Click **More Settings**. From the options list, select **Auto Register Packing** and select the **Auto** option from the drop-down menu.

When you choose a register packing setting to perform more register packing than the **Auto** setting, the extra register packing may affect the routability of the design as an unintended result. The **Minimize the area with chains** setting restricts placement and reduces routability significantly more than using the **Minimize Area** setting. For more information about register packing, refer to [“Use Register Packing” on page 8–25](#).

Set Fitter Aggressive Routability Optimizations to Always

If routing resources are resulting in no-fit errors, use this option to reduce routing wire utilization. On the Assignments menu, click **Settings**. In the Category list, select **Fitter Settings**. Click **More Settings**. In the More Fitter Settings dialog box, set **Fitter Aggressive Routability Optimizations** to **Always** and click **OK**.

On average, in Stratix II devices, this option saves approximately 3% wire utilization but can hurt performance by approximately 1%. In Stratix III devices, this option saves approximately 6% wire utilization, at the same time degrading the performance by approximately 3%.

These optimizations are used automatically when the Fitter performs more than one fitting attempt, but turning the option on increases the optimization effort on the first fitting attempt.

Increase Placement Effort Multiplier

Increasing the placement effort can improve the routability of the design, allowing the software to route a design that otherwise requires too many routing resources. On the Assignments menu, click **Settings**. In the Category list, select **Fitter Settings**. Click **More Settings**. In the More Fitter Settings dialog box, increase the value of the **Placement Effort Multiplier** to increase placement effort. The default value is 1.0. Legal values must be greater than 0 and can be non-integer values. Higher

numbers increase compilation time but may improve placement quality. For example, a value of 4 increases fitting time by approximately 2 to 4 times but may increase the quality of results.

Increased effort is used automatically when the Fitter performs more than one fitting attempt. Setting a multiplier higher than one (before compilation) increases the optimization effort on the first fitting attempt. The second and third fitting loops increase the Placement Effort Multiplier to 4 and then to 16. These loops result in increased compilation times, with possible improvement in the quality of placement.

Increase Router Effort Multiplier

The Router Effort Multiplier controls how quickly the router tries to find a valid solution. The default value is 1.0, and legal values must be greater than 0. Numbers higher than 1 (up to 3 is generally reasonable) may improve routing quality at the expense of run-time on difficult-to-route circuits. Numbers closer to 0 (for example, 0.1) can reduce router runtime, but usually reduce routing quality slightly. Experimental evidence shows that a multiplier of 3.0 reduces overall wire usage by about 2%. There is usually no gain in performance beyond a multiplier value of 3.

You can use the Router Effort Multiplier to a higher than default value for difficult-to-route designs. To set the Router Effort Multiplier, from the Assignments menu, click **Settings**, and then click **Fitter Settings**. Click the **More Settings** button. From the options available, select **Router Effort Multiplier** and edit the value in the subsequent dialog box that appears.

Remove Fitter Constraints

A design with conflicting constraints or constraints that are difficult to meet may not fit the targeted device. This can occur when 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 or LogicLock region assignments from 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. You can delete specific location assignments in the Assignment Editor or Timing Closure Floorplan. On the Assignments menu, click **LogicLock Regions Window** to remove LogicLock

assignments. Or, on the Assignments menu, click **Remove Assignments** and turn on the assignment categories you want to remove from the design in the **Available assignment categories** list.



For more information about the **Routing Congestion** view in the Timing Closure Floorplan, refer to the Quartus II Help. The Routing Congestion view is available on the View menu if you enable **Field View**.

Set Maximum Router Optimization Level

To improve routability in cases in which the router did not pick up the optimal routing lines, set the Router Optimization Level to **Maximum**. This setting determines how aggressively the router tries to meet timing requirements. Setting this option to Maximum can increase design speed slightly, at the cost of increased compilation time. Setting this option to **Minimum** can reduce compilation time, at the cost of slightly reduced design speed. The default value is **Normal**.

To modify the Router Optimization level, on the Assignments menu, click **Settings**. The Settings dialog box appears. In the Category list, click **Fitter Settings**. Click on the **More Settings** tab. From the available settings, select **Router Optimization Level** and choose the required setting from the drop-down menu.

Optimize Synthesis for Area, Not Speed

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

If resource utilization is important to improving the routing results in your design, some synthesis tools offer an easy way to optimize for area instead of speed. If you are using Quartus II integrated synthesis, on the Assignments menu, click **Settings**. In the Category list, select **Analysis & Synthesis Settings**, and choose **Balanced** or **Area** under **Optimization Technique**.

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 using the **Area** setting (potentially at the expense of f_{MAX} performance). You can apply the setting to specific modules while leaving the default Optimization Technique setting at **Balanced** (for the best trade-off

between area and speed for certain device families) or **Speed**. You can also use the **Speed Optimization Technique for Clock Domains** logic option to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.



In the Quartus II software, the **Balanced** setting typically produces utilization results that are very similar to those obtained with the **Area** setting, with better performance results. The Area setting may give better results in some unusual cases.

In some synthesis tools, not specifying an f_{MAX} requirement may result in less resource utilization which may improve routability.



For information about setting timing requirements and synthesis options in Quartus II integrated synthesis and other synthesis tools, refer to 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 routing problems, and the methods described in the preceding sections do not sufficiently improve the routability of the design, modify the design at the source to achieve the desired results. You can often improve results significantly by making design-specific changes to your source code, such as duplicating logic or changing the connections between blocks that require significant routing resources.

Use a Larger Device

If a successful fit cannot be achieved because of a shortage of routing resources, you may need to use a larger device.

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 8–5, and that the resource utilization is satisfactory, before proceeding with I/O timing optimization. Because changes to the I/O paths affect the internal f_{MAX} , complete this stage before proceeding to the f_{MAX} timing optimization stage as described in the “ [\$f_{MAX}\$ Timing Optimization Techniques \(LUT-Based Devices\)](#)” on page 8–47.

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.

Improving Setup & Clock-to-Output Times Summary

Table 8–4 shows the recommended order in which to use techniques to reduce t_{SU} and t_{CO} times. Check marks indicate which timing parameters are affected by each technique. Reducing t_{SU} times increases hold (t_H) times.

Technique	Affects t_{SU}	Affects t_{CO}
Ensure that the appropriate constraints are set for the failing I/Os (page 8–12)	✓	✓
Use timing-driven compilation for I/O (page 8–41)	✓	✓
Use fast input register (page 8–42)	✓	—
Use fast output register and fast output enable register (page 8–42)	—	✓
Decrease the value of Input Delay from Pin to Input Register or set Decrease Input Delay to Input Register = ON (page 8–43)	✓	—
Decrease the value of Input Delay from Pin to Internal Cells , or set Decrease Input Delay to Internal Cells = ON (page 8–43)	✓	—
Decrease the value of Delay from Output Register to Output Pin , or set Increase Delay to Output Pin = OFF (page 8–44)	—	✓
Increase the value of Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations (page 8–45)	✓	—
Use PLLs to shift clock edges (page 8–45)	✓	✓
Use the Fast Regional Clock option (page 8–46)	—	✓
For MAX II devices, set Guarantee I/O paths to zero, Hold Time at Fast Timing Corner to OFF, or When t_{SU} and t_{PD} constraints permit (page 8–46)	✓	—

Table 8–4. Improving Setup & Clock-to-Output Times *Note (1) (Part 2 of 2)*

Technique	Affects t_{SU}	Affects t_{CO}
Increase the value of Delay to output enable pin or set Increase delay to output enable pin (page 8–45)	—	✓

Note to Table 8–4:

(1) These options may not apply to all device families.

Timing-Driven Compilation

To perform IOC timing optimization using the **Optimize IOC Register Placement For Timing** option, perform the following steps.

1. On the Assignments menu, click **Settings**.
2. In the Category list, select **Fitter Settings** and click **More Settings**.
3. In the More Fitter Settings dialog box, under **Existing options settings**, select **Optimize IOC Register Placement for Timing**.

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 enabled 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 IOC Register Placement for Timing** option affects only 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 clear ports of APEX 20K and APEX II devices
- Are input registers that use the synchronous load port and the value is not 1 (in device families where the port is available, other than APEX 20K, APEX II, and FLEX® 6000 devices)
- Use the asynchronous load port and the value is not 1 (in device families where the port is available)

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

Fast Input, Output & Output Enable Registers

You can place individual registers in I/O cells manually 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 for 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 IOC Register Placement for Timing** option 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 IOC 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**) also can 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 series devices, Cyclone series devices, and MAX II devices, the Quartus II software automatically adjusts the applicable programmable delays to help meet timing requirements. For APEX series devices, the default values are set to avoid any hold time problems. Programmable delays are advanced options that you should use only after you compile a project, check the I/O timing, and determine that the timing is unsatisfactory. For detailed information about the effect of these options, refer to the device family handbook or data sheet.

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 assign programmable delay options to supported nodes with the Assignment Editor. You also can view and modify the delay chain setting for the target device with the Quartus II Chip Planner and Resource Property Editor. When you use the Resource Property Editor to make changes after performing a full compilation, recompiling the entire design is not necessary; you can save changes directly to the netlist. Because these changes are made directly to the netlist, the changes are not made again automatically when you recompile the design. The change management features allow you to reapply the changes on subsequent compilations.

Users can not control the programmable delays in Stratix III devices. However, the software may use programmable delays internally during the place-and-route phase.



For more information about using the Quartus II Chip Planner and Resource Property Editor, refer to the *Design Analysis & Engineering Change Management with Chip Planner* chapter in volume 3 of the *Quartus II Handbook*.

Table 8-5 summarizes the programmable delays available for Altera devices.

Programmable Delay	Description	I/O Timing Impact	Devices
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	<ul style="list-style-type: none"> ● 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	<ul style="list-style-type: none"> ● Stratix II ● Stratix II GX ● Cyclone II

Table 8–5. Programmable Delays for Altera Devices (Part 2 of 3)

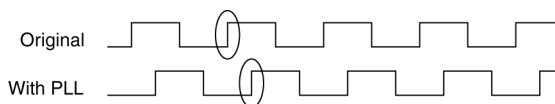
Programmable Delay	Description	I/O Timing Impact	Devices
Decrease input delay to internal cells	Decreases the propagation delay from an input or bidirectional pin to logic cells and embedded cells in the device. Applied to input/bidirectional pin or register it feeds.	Decreases t_{SU} Increases t_H	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Stratix II • Stratix II GX • 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}	<ul style="list-style-type: none"> • 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}	<ul style="list-style-type: none"> • 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}	<ul style="list-style-type: none"> • Stratix II • Stratix II GX
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}	<ul style="list-style-type: none"> • 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}	<ul style="list-style-type: none"> • Stratix II • Cyclone II • Stratix II GX
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	<ul style="list-style-type: none"> • Stratix • Stratix GX • APEX II • APEX 20KE • APEX 20KC

Table 8–5. Programmable Delays for Altera Devices (Part 3 of 3)

Programmable Delay	Description	I/O Timing Impact	Devices
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	<ul style="list-style-type: none"> ● Cyclone II
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	<ul style="list-style-type: none"> ● Stratix ● Stratix GX ● APEX II ● APEX 20KE ● APEX 20KC
Increase output enable clock delay	Increases the propagation delay from the interior of the device to the clock enable input of an output enable register.	N/A	<ul style="list-style-type: none"> ● 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}	<ul style="list-style-type: none"> ● Stratix ● Stratix GX ● APEX II ● Mercury

Use PLLs to Shift Clock Edges

Using a PLL typically improves I/O timing automatically. If the timing requirements are still not met, most devices allow the PLL output 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 t_{SU} , while shifting it forward gives a better t_{SU} at the expense of t_{CO} and t_H . Refer to [Figure 8–10](#). This technique can be used only in devices that offer PLLs with the phase shift option.

Figure 8–10. Shift Clock Edges Forward to Improve t_{SU} at the Expense of t_{CO} 

You can achieve the same type of effect in certain devices using the programmable delay called **Input Delay from Dual Purpose Clock Pin to Fan-Out Destinations**, described in [Table 8–5](#).

Use Fast Regional Clocks in Stratix Devices & Regional Clocks in Stratix II Devices

Stratix EP1S25, EP1S20, and EP1S10 devices, and Stratix GX EP1SGX25 and EP1SGX10 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 feed these clock nets directly. Stratix fast regional clocks have less delay to I/O elements than regional or global clocks, and are used for high fan-out control signals.

Stratix II, Stratix II GX, and Stratix III devices provide 32 regional clock networks. There are eight regional clock networks, RCLK [7 . . 0] in each quadrant of the device that are driven by the dedicated input pins CLK [15 . . 0], by PLL outputs, or by internal logic. These regional clock networks provide the lowest clock delay and skew for logic contained in a single quadrant. Placing clocks on these low-skew and low-delay clock nets provides better t_{CO} performance.

Change How Hold Times are Optimized for MAX II Devices

For MAX II devices, you can use the **Guarantee I/O paths have zero hold time at Fast Timing Corner** option to control how hold time is optimized by the Quartus II software. On the Assignments menu, click **Settings**. In the Category list, select **Fitter Settings**. Click **More Settings**. In the More Fitter Settings dialog box, set the option globally. Or, on the Assignments menu, click **Assignment Editor** to set this option for specific I/Os.

The option controls whether the Fitter uses timing-driven compilation to optimize a design to achieve a zero hold time for I/Os that feed globally clocked registers at the fast (best-case) timing corner, even in the absence of any user timing assignments. When this option is set to **On** (default), the Fitter guarantees zero hold time (t_H) for I/Os feeding globally clocked registers at the fast timing corner, at the expense of possibly violating t_{SU} or t_{PD} timing constraints. When this option is set to **When tsu and tpd constraints permit**, the Fitter achieves zero hold time for I/Os feeding globally clocked registers at the fast timing corner only when t_{SU} or t_{PD} timing constraints are not violated. When this option is set to **Off**, designs are optimized to meet user timing assignments only.

By setting this option to **Off** or **When tsu and tpd constraints permit**, you improve t_{SU} at the expense of t_H .

f_{MAX} Timing Optimization Techniques (LUT-Based Devices)

The next stage of design optimization is to improve f_{MAX} timing. There are a number of options available if the performance requirements are not achieved after compilation.

Before optimizing your design, you should understand the structure of your design as well as the type of logic affected by each optimization. An optimization can decrease performance if the optimization does not benefit your logic structure.

Improving f_{MAX} Summary

The choice of options and 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 techniques, and compile the design after each:

1. Ensure that your timing assignments are complete. For details, refer to [“Timing Requirement Settings” on page 8–6](#).
2. Ensure that you have reviewed any warning messages from your initial compilation, and checked for ignored timing assignments. Refer to [“Design Analysis” on page 8–14](#) for details and fix any of these problems before proceeding with optimization.
3. Apply netlist synthesis optimization options and physical synthesis ([page 8–48](#)).
4. Modify the Fitter seed ([page 8–56](#)). You can omit this step if a large number of critical paths are failing, or if paths are failing by large amounts.
5. Apply the following synthesis options to optimize for speed:
 - Optimize Synthesis for Speed, Not Area ([page 8–52](#))
 - Flatten the Hierarchy During Synthesis ([page 8–53](#))
 - Set the Synthesis Effort to High ([page 8–54](#))
 - Change State Machine Encoding ([page 8–54](#))
 - Duplicate Logic for Fan-Out Control ([page 8–54](#))
 - Prevent Shift Register Inference ([page 8–55](#))
 - Use Other Synthesis Options Available in Your Synthesis Tool ([page 8–56](#))
6. Make LogicLock assignments ([page 8–58](#)) to control placement.
7. Make design source code modifications to fix areas of the design that are still failing timing requirements by significant amounts ([page 8–57](#)).

8. Make location assignments, or, as a last resort, perform manual placement by back-annotating the design (page 8–61).



You can use the DSE to automate the process of running several different compilations with different settings. For more information, refer to the *Design Space Explorer* chapter in volume 2 of the *Quartus II Handbook*.

If these techniques do not achieve performance requirements, additional design source code modifications may be required (page 8–57).

Synthesis Netlist Optimizations & Physical Synthesis Optimizations

The Quartus II software offers advanced netlist optimization options, including physical synthesis. Various netlist optimizations can help improve the performance of many designs 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 Quartus II integrated synthesis, these optimizations make changes to the synthesis netlist to improve either area or speed, depending on your selected optimization technique.

The following synthesis netlist optimizations are available:

- WYSIWYG primitive resynthesis (for netlists from third-party EDA synthesis tools)
- Gate-level register retiming

On the Assignments menu, click **Settings**. In the **Category** list, expand **Analysis & Synthesis Settings** and select **Synthesis Netlist Optimizations** to view and modify the synthesis netlist optimization options.

If you use another EDA synthesis tool and want to determine if the Quartus II software can remap the circuit to improve performance, you can use the **Perform WYSIWYG Primitive Resynthesis** option. This option directs the Quartus II software to unmap the LEs in an atom netlist to logic gates, and then map the gates back to Altera-specific primitives. Using Altera-specific primitives enables the Fitter to remap the circuits using architecture-specific techniques.

To turn on the **Perform WYSIWYG Primitive Resynthesis** option, on the Assignments menu, click **Settings**. In the **Category** list, expand **Analysis & Synthesis Settings** and select **Synthesis Netlist Optimizations**. Turn on **Perform WYSIWYG primitive resynthesis (using optimization techniques specified in Analysis & Synthesis settings)**.

The Quartus II technology mapper optimizes the design for **Speed**, **Area**, or **Balanced**, according to the setting of the **Optimization Technique** option. To change this setting, on the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**, and choose **Speed** or **Balanced** under **Optimization Technique**.

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. You can use this option with Quartus II Integrated synthesis, or if you are using a third-party EDA synthesis tool, you can use this option if you have turned on **Perform WYSIWYG primitive resynthesis (using optimization techniques specified in Analysis & Synthesis settings)**.

The physical synthesis optimizations occur 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
- Automatic asynchronous signal pipelining
- Physical synthesis for registers
 - Register duplication
 - Register retiming

On the Assignments menu, click **Settings**. In the **Category** list, select **Fitter Settings**, and specify the physical synthesis optimization options on the **Physical Synthesis Optimizations** page. You can also specify the **Physical synthesis effort**, which sets the level of physical synthesis optimization that you want the Quartus II software to perform.

The **Perform physical synthesis for combinational logic** option allows the Quartus II Fitter to resynthesize the combinational logic in a design to reduce delay along the critical path and improve design performance.

The **Perform automatic asynchronous signal pipelining** allows the Quartus II Fitter to insert pipeline stages for asynchronous clear and asynchronous load signals automatically during fitting to increase circuit

performance. You can use this option if asynchronous control signal recovery and removal times do not meet your requirements. The option improves performance for designs in which asynchronous signals in very fast clock domains cannot be distributed across the chip quickly enough (because of long global network delays).



The **Perform automatic asynchronous signal pipelining** option adds registers to nets driving the asynchronous clear or asynchronous load ports of registers. This adds register delays (and latency) to the reset, adding the same number of register delays for each destination using the reset. Therefore the option should be used only when adding latency to reset signals does not violate any design requirements. This option also prevents the promotion of signals to use global routing resources.

The **Perform register duplication** fitter option allows the Quartus II Fitter to duplicate registers based on fitter placement information to improve design performance. The Fitter can also duplicate combinational logic when this option is enabled.

The **Perform register retiming** fitter option allows the Quartus II Fitter to move registers across combinational logic to balance timing. This option turns on algorithms similar to the Perform gate-level register retiming option. This option applies to registers and combinational logic that have already been placed into logic cells, and it compliments the synthesis gate-level option.



For more information and detailed descriptions of these netlist optimization options, refer to the *Netlist Optimizations & Physical Synthesis* chapter in volume 2 of the *Quartus II Handbook*.

Because performance results are design-dependant, try these options in different combinations until you achieve the best results. Generally, turning on all the options gives the best results but significantly increases compilation time. This section provides typical benchmark results using various designs and amounts of logic with synthesis netlists from leading third-party synthesis tools and compiled with Quartus II software. These results use the default Balanced setting for the Optimization Technique for WYSIWYG resynthesis. Changing the setting to Speed or Area can affect your results.

Tables 8–6 through 8–8 show the average results for different device families, using the following measures for the quality of results:

- **f_{MAX} Gain**—The percentage gain in clock f_{MAX} performance
- **Win Ratio**—The percentage of designs that showed better performance with the option on than without the option on

- **Winner's f_{MAX} Gain**—The average percentage improvement for the designs that showed better performance with these settings (the designs considered a win)
- **Logic Area Change**—The percentage gain in logic utilization. Negative values mean reduced area, and positive values mean increased area
- **Compile Time Change**—The multiplication factor for the compilation time when the option is used

Table 8–6. Average Results of Synthesis Netlist & Physical Synthesis Optimizations for Stratix & Cyclone Designs

Optimization Method	f _{MAX} Gain (%)	Win Ratio (%)	Winner's f _{MAX} Gain (%)	Logic Area Change (%)	Compile Time Change (x)
WYSIWYG primitive resynthesis	3	60	6	–8	1.0
Physical synthesis for combinational logic and registers					
Using physical synthesis Fast effort level	10	86	12	4	1.4
Using physical synthesis Normal effort level	15	86	16	4	2.2
Using physical synthesis Extra effort level	17	86	18	4	3.7

Table 8–7. Average Results of Synthesis Netlist & Physical Synthesis Optimizations for Stratix II & Cyclone II Designs

Optimization Method	f _{MAX} Gain (%)	Win Ratio (%)	Winner's f _{MAX} Gain (%)	Logic Area Change (%)	Compile Time Change (x)
WYSIWYG primitive resynthesis	3	60	6	–8	1.0
Physical synthesis for combinational logic and registers					
Using physical synthesis Fast effort level	11	82	14	2.5	1.3
Using physical synthesis Normal effort level	14	88	17	4	2.0
Using physical synthesis Extra effort level	15	88	18	4.3	2.3

Table 8–8. Average Performance of Synthesis Netlist & Physical Synthesis Optimizations for Stratix III Designs

Optimization Method	f_{MAX} Gain (%)	Win Ratio (%)	Winner's f_{MAX} Gain (%)	Logic Area Change (%)	Compile Time Change (x)
WYSIWYG primitive resynthesis	3	60	6	–8	1.0
Physical synthesis for combinational logic and registers					
Using physical synthesis Fast effort level	8	75	11	2	1.2
Using physical synthesis Normal effort level	10	79	14	3	1.7
Using physical synthesis Extra effort level	11	79	15	3	2.0

Turn Off Extra-Effort Power Optimization Settings

If PowerPlay Power Optimization settings are set to **Extra Effort**, your design performance may be affected. If improving timing performance is more important than reducing power use, set the Power Optimization setting to **Normal**.

To change the PowerPlay Power Optimization level, on the Assignments menu, choose **Settings**. The Setting dialog box appears. From the Category list, select **Analysis & Synthesis Settings**. From the drop-down menu, select the appropriate level of PowerPlay Power Optimization level.



For more information about reducing power use, refer to the *Power Optimization* chapter in volume 2 of the *Quartus II Handbook*.

Optimize Synthesis for Speed, Not Area

The manner in which the design is synthesized has a large impact on design performance. Design performance varies depending on the way the design is coded, which synthesis tool is used, and which options are specified when synthesizing. Change your synthesis options if a large number of paths are failing, or specific paths are failing by a large amount and have many levels of logic.

Set your device and timing constraints in your synthesis tool. Synthesis tools are timing-driven and optimize to meet specified timing requirements. If you do not specify target frequency, some synthesis tools optimize for area.

Some synthesis tools offer an easy way to instruct the tool to focus on speed instead of area.

For Quartus II integrated synthesis, on the Assignments menu, click **Settings**. In the Category list, select **Analysis & Synthesis Settings**, and specify **Speed** as the **Optimization Technique** option. 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). You can also use the **Speed Optimization Technique for Clock Domains** option to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.

To achieve best performance with push-button compilation, follow the recommendations in the following sections for other synthesis settings. You can use DSE to experiment with different Quartus II synthesis options to optimize your design for the best performance.



For information about setting timing requirements and synthesis options in Quartus II integrated synthesis and third-party synthesis tools, refer to the appropriate chapter in the *Synthesis* section in volume 1 of the *Quartus II Handbook*, or refer to your synthesis software documentation.

Flatten the Hierarchy During Synthesis

Synthesis tools typically let you preserve hierarchical boundaries, which can be useful for verification or other purposes. However, the best optimization results generally occur when the synthesis tool optimizes across hierarchical boundaries because doing so often allows the synthesis tool to perform the most logic minimization, which can improve performance. Whenever possible, flatten your design hierarchy to achieve the best results. If you are using Quartus II integrated synthesis, ensure that the **Preserve Hierarchical Boundary** option is turned off. If you are using Quartus II incremental compilation, you cannot flatten your design across design partitions. Incremental compilation always preserves the hierarchical boundaries for design partitioning. Follow Altera's recommendations for design partitioning such as registering partition boundaries to reduce the effect of cross-boundary optimizations.



For more information about using incremental compilation and recommendations for design partitioning, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*. If you are using an incremental synthesis flow that requires separate hierarchy blocks, you can find additional recommendations for design partitioning in the *Design Recommendations for Altera Devices* chapter in volume 1 of the *Quartus II Handbook*.

Set the Synthesis Effort to High

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 when applicable.

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, on the Assignments menu, click **Settings**. In the Category list, select **Analysis & Synthesis Settings**, and for **State Machine Processing**, choose **One-Hot**. You also can specify this logic option for specific modules or state machines in your design with the Assignment Editor.

In some cases (especially in Stratix II and Stratix III devices), encoding styles other than the default offer better performance. 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 that specify the maximum fan-out of a register. When using 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.



Logic duplication using Maximum Fan-Out assignments normally increases resource utilization, and can potentially increase compilation time depending on the placement and the total resource usage within the selected device. The improvement in timing performance that results because of Maximum Fan-Out assignments is very design-specific.

If you are using Maximum Fan-Out assignments, Altera recommends benchmarking your design with and without these assignments to evaluate whether they give the expected improvement in timing performance, and use the assignments only when you get improved results.

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.

The manual logic duplication option also accepts wildcards. This is an easy and powerful duplication technique that you can use without editing your source code. You can use this technique, for example, to make a duplicate of a large fan-out node for all of its destinations in a certain design hierarchy, such as `hierarchy_A`. To apply such an assignment in the Assignment Editor, make an entry such as the one shown in [Table 8-9](#):

From	To	Assignment Name	Value
<code>My_high_fanout_node</code>	<code>*hierarchy_A*</code>	Manual Logic Duplication	<code>high_fanout_to_A</code>



For more information about the manual logic duplication option, refer to the Quartus II Help.

Prevent Shift Register Inference

In some cases, turning off the inference of shift registers increases performance. Doing so forces the software to use logic cells to implement the shift register instead of implementing the registers in memory blocks using the `altshift_taps` megafunction. If you implement shift registers in logic cells instead of memory, logic utilization is increased.

Use Other Synthesis Options Available in Your Synthesis Tool

With your synthesis tool, experiment with the following options if they are available:

- Turn on register balancing or retiming
- Turn on register pipelining
- Turn off resource sharing

These options may increase performance. They typically increase the resource utilization of your design.

Fitter Seed

The Fitter seed affects the initial placement configuration of the design. Changing the seed value changes the Fitter results because the fitting results change whenever there is a change in the initial conditions. Because each seed value results in a somewhat different fit, you can experiment with several different seeds to attempt to obtain better fitting results and timing performance.

When there are changes in your design, there is some random variation in performance between compilations. This variation is inherent in placement and routing algorithms—there are too many possibilities to try them all and get the absolute best result, so the initial conditions change the compilation result.

Note that any design change that directly or indirectly affects the Fitter has the same type of random effect as changing the seed value. This includes any change in source files, Analysis & Synthesis settings, Fitter settings, or Timing Analyzer settings. The same effect can appear if you use a different computer processor type or different operating system because different systems can change the way floating point numbers are calculated in the Fitter.

If your design is finalized, you can compile your design with different seeds to obtain one optimal result. If any design or setting changes occur, they can make a previously optimal seed value no longer optimal.

If a design optimization slightly changes the f_{MAX} timing or number of failing paths, you can't always be certain that your change caused the improvement or degradation or whether it could be due to random effects in the Fitter. If your design is still changing, running a seed sweep (compiling your design with multiple seeds) determines whether the average result has improved after an optimization change and whether a setting that increases compilation time has benefits worth the increased

time (such as turning the physical synthesis effort to extra). The sweep also shows the amount of random variation you should expect for your design.

On the Assignments menu, select **Fitter Settings** to control the initial placement with the Seed. You can use the Design Space Explorer (DSE) to perform a seed sweep easily.



For more information about compiling with different seeds using the DSE script, refer to the *Design Space Explorer* chapter in volume 2 of the *Quartus II Handbook*.

Optimize Source Code

If the methods described in the preceding sections do not sufficiently improve timing of the design, modify your design files to achieve the desired results. Try restructuring the design to use 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 typically the most effective technique for improving the quality of your results, and is often a better choice than using LogicLock or location assignments.

If the critical path in your design involves memory or DSP functions, check whether you have code blocks in your design that describe memory or functions that are not being inferred and placed in dedicated logic. You may be able to modify your source code to cause these functions to be placed into high-performance dedicated memory or resources in the target device.

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. 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 coding style guidelines including examples of HDL code for inferring memory, and functions and guidelines and sample HDL code for state machines, refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*.

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, and recoding the design does not seem to be necessary. LogicLock assignments can help if routing delays form a large portion of your critical path delay, and placing logic closer together in the device improves the routing delay. This technique is most beneficial for devices with hierarchical routing structures such as the APEX 20K device family.



Improving fitting results with LogicLock assignments, especially for larger devices such as the Stratix series of devices, can be difficult. The LogicLock feature is intended to be used for performance preservation, therefore LogicLock assignments do not always improve the performance of the design. In many cases you cannot improve upon results from the Fitter by making location assignments.

If there are existing LogicLock assignments in your design, remove the assignments if your design methodology permits it. Recompile the design to see if the assignments are making the performance worse.

When making LogicLock assignments, it is important to consider how much flexibility to give 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, refer to 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 help you recognize patterns.

The following sections describe cases in which LogicLock regions can help to optimize a design.



For more information about using LogicLock regions, refer to the *LogicLock Design Methodology* chapter in volume 2 of the *Quartus II Handbook*.

Hierarchy Assignments

For a design with the hierarchy shown in Figure 8–11, which has failing paths in the timing analysis results similar to those shown in Table 8–10, `mod_A` is probably a problem module. In this case, a good strategy to fix the failing paths is to place the `mod_A` hierarchy block in a LogicLock region so that all the nodes are closer together in the floorplan.

Figure 8–11. Design Hierarchy

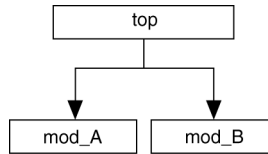


Table 8–10 shows the failing paths connecting two regions together within `mod_A` listed in the timing analysis report.

Table 8–10. Failing Paths in a Module Listed in Timing Analysis

From	To
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

Hierarchical LogicLock regions are also important if you are using an incremental compilation flow. Each design partition for incremental compilation should be placed in a separate LogicLock region to reduce conflicts and ensure good quality of results as the design develops. In this case, you should not use soft LogicLock regions because they allow the fitter to move nodes away from the region. You can use auto size and floating location regions to find a good design floorplan, but you should then fix the size and placement to achieve best results in future compilations.



For more information about using incremental compilation and recommendations for creating a design floorplan using LogicLock regions, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

Path Assignments

If you see a pattern such as the one shown in Figure 8–12 and Table 8–11, it often indicates paths with a common problem. In this case, a path-based assignment can be made from all **d_reg** registers to all **memaddr** registers. You can make a path-based assignment to place all source registers, destination registers, and the nodes between them in a LogicLock region with the wildcard characters “*” and “?”.

You also can explicitly place the nodes of a critical path in a LogicLock region. However, using this method instead of path assignments can result in alternate paths between the source and destination registers becoming critical paths.

Figure 8–12. Failing Paths in Timing Analysis

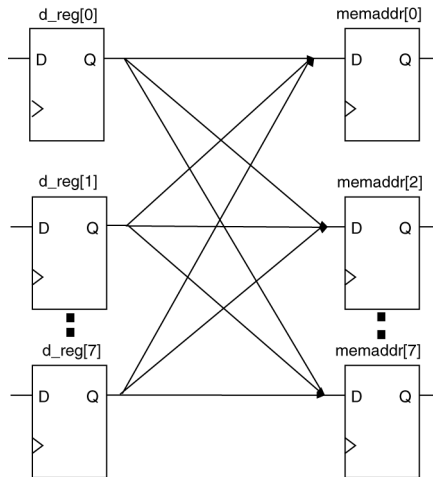


Table 8–11 shows the failing paths listed in the timing analysis report.

Table 8–11. 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]



For more information about path-based LogicLock assignments, refer to the *LogicLock Design Methodology* chapter in volume 2 of the *Quartus II Handbook*.

Location Assignments & Back-Annotation

If a small number of paths are failing to meet their timing requirements, 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 can enter location constraints in a way that produces better results.



Improving fitting results, especially for larger devices such as the Stratix series of devices, can be difficult. Location assignments do not always improve the performance of the design. In many cases you cannot improve upon the results from the Fitter by making location assignments.

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, which are constrained in the region's boundaries. If any portion of a block in the device floorplan, such as an M-RAM block, overlaps a custom region, 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 & Manual Placement

Assigning the location of nodes in a design to the locations to which they were assigned during the last compilation is called "back-annotation". When nodes are locked to their assigned locations in a back-annotated design, you can manually move specific nodes without affecting other back-annotated nodes. The process of manually moving and reassigning specific nodes is called manual placement.



Back-annotation is very restrictive to the compiler, so you should back-annotate only when the design has been finalized and no further changes are expected. Assignments can 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. In many cases, you cannot improve upon the Fitter's results.

Illegal or unroutable location constraints can cause "no fit" errors.

Before making location assignments, determine whether to back-annotate to lock down the assigned locations of all nodes in the design. When you are using a hierarchical design flow, you can lock down node locations in one LogicLock region only, while other node locations are left floating in a fixed LogicLock region. By implementing a hierarchical approach, you can use the LogicLock design methodology to reduce the dependence of logic blocks on other logic blocks in the device.

Consistent node names are required to perform back-annotation. If you use Quartus II integrated synthesis or any Quartus II optimizations, such as the WYSIWYG primitive resynthesis netlist optimization or any physical synthesis optimizations, you must create an atom netlist before you back-annotate to lock down the placement of any nodes. This creates consistent node names.




Physical synthesis optimizations are placement-specific as well as design-specific. Unless you back-annotate the design before recompilation, the physical synthesis results can differ. This happens because the atom netlist creates different placement results. By back-annotating the design, the design source and the atom netlist use the same placement when the design is recompiled. When you are using an atom netlist and you want to maintain the same placement results as a previous compilation, use LogicLock regions and back-annotate the placement of all nodes in the design. Not back-annotating the design can result in the design source and the atom netlist having different placement results and therefore different synthesis results.



For more information about creating atom netlists for your design, refer to the *LogicLock Design Methodology* chapter in volume 2 of the *Quartus II Handbook*.

When you back-annotate a design, you can choose whether to assign the nodes either to LABs (this is preferred because of increased flexibility) or LEs/ALMs. You also can 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 can result in a routing failure.



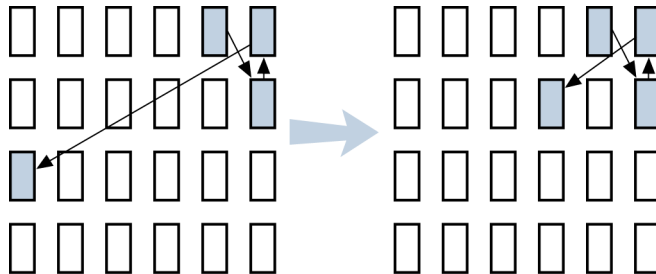
For more information about back-annotating routing, refer to the Quartus II Help.

When performing manual placement at 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 placement and routing is more difficult when you move individual logic cells. This is because LEs with different control signals are put into the same LAB may not have any unused control signals available and the design may not fit.

In general, when you are performing manual placement and routing, fix all I/O paths first 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 is consistent with 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 8–13](#), moving the destination node closer to the other nodes reduces the delay and helps meet your timing requirements.

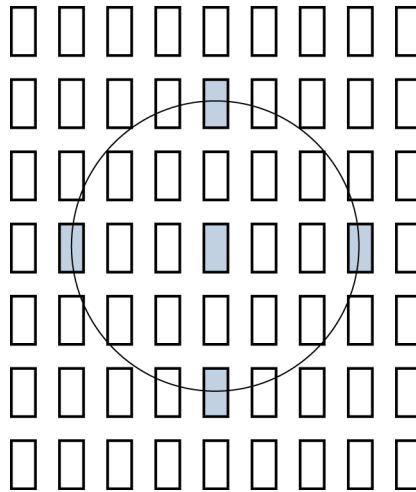
Figure 8–13. Reducing Delay of Critical Path



Optimizing Placement for Stratix, Stratix II & Cyclone II Devices

In the Stratix series of devices and Cyclone II devices, 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 8–14](#), if the source is located in the center, any of the shaded LABs should give approximately the same delay.

Figure 8–14. Possible Optimal Placement Ellipse



In addition, you should avoid crossing any M-RAM memory blocks for node-to-node routing, 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 about using the Timing Closure Floorplan, refer to 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. Long routes 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 device families, the delay for paths is 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 placed in the same MegaLAB structure to reduce the delay, then place them 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.

Resource Utilization Optimization Techniques (Macrocell-Based CPLDs)

The following recommendations 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 8–5](#). 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 can be preserved for primary logic. Also, because each signal has a dedicated path into the LAB, global signals also can 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 (such as 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 following list shows the disadvantages to using logic-generated control signals:

- More resources are required (logic cells, interconnect).
- More data skew is introduced.
- 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 control signals to dedicated input pins in one of the following ways:

- In the Assignment Editor, choose one of the two following methods:
 - Assign pins to dedicated pin locations.
 - Assign a **Global Signal** setting to the pins.
- On the Assignments menu, click **Settings**. In the **Category** list, select **Register Control Signals** in the **Auto Global Options** section of the **Analysis & Synthesis Settings** page.
- Insert a GLOBAL primitive after the pins.
- If you have already assigned pins for the design in the MAX+PLUS® II software, on the Assignments menu, click **Import Assignments**.

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 can increase the difficulty of fitting the design, 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 for macrocell-based CPLDs increases the chance that the Quartus II software can 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 the appropriate LAB.
- Assign output pins that require parallel expanders to macrocells numbered 4 to 16.



Altera recommends that you allow the Quartus II software to choose pin assignments automatically when possible.

Control Signal Pin Assignments

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).

Output Enable Pin Assignments

Occasionally, because the total number of required output enable pins is more than the dedicated input pins, output enable signals must be assigned to I/O pins. Therefore, to minimize possible fitting errors, refer to the pin tables on the Literature page of Altera's website when assigning the output enable pins for MAX 7000 and MAX 3000 devices.

Estimate Fan-In When Assigning Output Pins

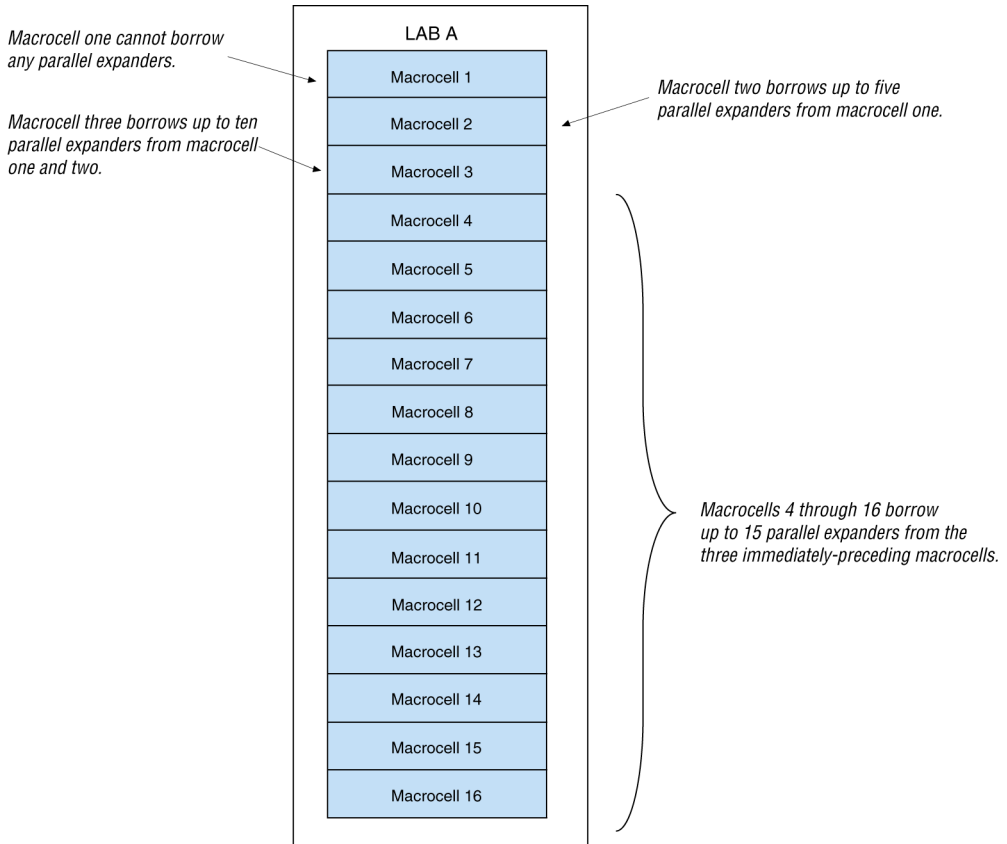
Macrocells with high fan-in can cause more placement problems for the Quartus II Fitter than those with low fan-in. The maximum fan-in per LAB should not exceed 36 in MAX 7000 and MAX 3000 devices. Therefore, estimate the fan-in of logic (such as an x -input AND gate) that feeds each output pin. If the 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 8-15 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 8–15, only macrocells 2 through 16 can borrow parallel expanders. Therefore, 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 8–15. LAB Macrocells & Parallel Expander Associations



Resolving Resource Utilization Problems

There are two common Quartus II compilation fitting issues that cause errors: excessive macrocell usage and lack of 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 are insufficient to implement the design. Check the Message Window 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** for more information.

Resolving Macrocell Usage Issues

Occasionally, a design requires more macrocell resources than are available in the selected device, which results in the design not fitting. The following list provides tips for resolving macrocell usage issues as well as tips to minimize the number of macrocells used.

- On the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**, and turn off **Auto Parallel Expanders**. If the design's clock frequency (f_{MAX}) is not an important design requirement, turn off parallel expanders for all or part of the project. The design usually requires more macrocells if parallel expanders are turned on.
- Change Optimization Technique from **Speed** to **Area**. Selecting **Area** instructs the compiler to give preference to area utilization rather than speed (f_{MAX}). On the Assignments menu, click **Settings**. In the **Category** list, change the **Optimization Technique** option in the **Analysis & Synthesis Settings** page.
- Use D-type flipflops instead of latches. Altera recommends that you always use D-type flipflops instead of latches in your design because D-type flipflops can 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 MAX 7000 and MAX 3000 macrocells contain D-type flipflops 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 produces macrocells and pins with higher fan-out.



After following the suggestions in this section, if your project still does not fit the targeted device, consider using a larger device. When upgrading to a different density, the vertical-package-migration feature of the MAX 7000 and MAX 3000 device families allows pin assignments to be maintained.

Resolving Routing Issues

Routing is another resource that can cause design fitting issues. For example, if the total fan-in into a LAB exceeds the maximum allowed, a no-fit error can occur 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. 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 macrocell usage issues. Refer to the same suggestion in [“Resolving Macrocell Usage Issues” on page 8–69](#).
- Reduce the fan-in per cell. If you are not limited by the number of macrocells used in the design, you can use the **Fan-in per cell (%)** option to reduce the fan-in per cell. The allowable values are 20–100%; 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 or under More Settings in the **Analysis & Synthesis Settings** page of the **Settings** dialog box.
- On the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**, and turn off **Auto Parallel Expanders**. By turning off the parallel expanders, the Quartus II software has more fitting flexibility for each macrocell, allowing macrocells to be relocated. For example, each macrocell (previously grouped together in the same LAB) can be moved to a different LAB to reduce routing constraints.

- Insert logic cells. Inserting logic cells reduces fan-in and shared expanders used per macrocell, increasing routability. By default, the Quartus II software automatically inserts logic cells when necessary. Otherwise, **Auto Logic Cell** can be disabled as follows. On the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**. Under More Settings, turn off **Auto Logic Cell Insertion**. Refer to [“Using LCELL Buffers to Reduce Required Resources” on page 8-71](#) for more information.
- Change pin assignments. If you are willing to discard your pin assignments, you can let the Quartus II Fitter ignore some or all the assignments.



If you prefer reassigning pins to increase routing efficiency, refer to [“Pin Assignment Guidelines & Procedures” on page 8-66](#).

Using LCELL Buffers to Reduce Required Resources

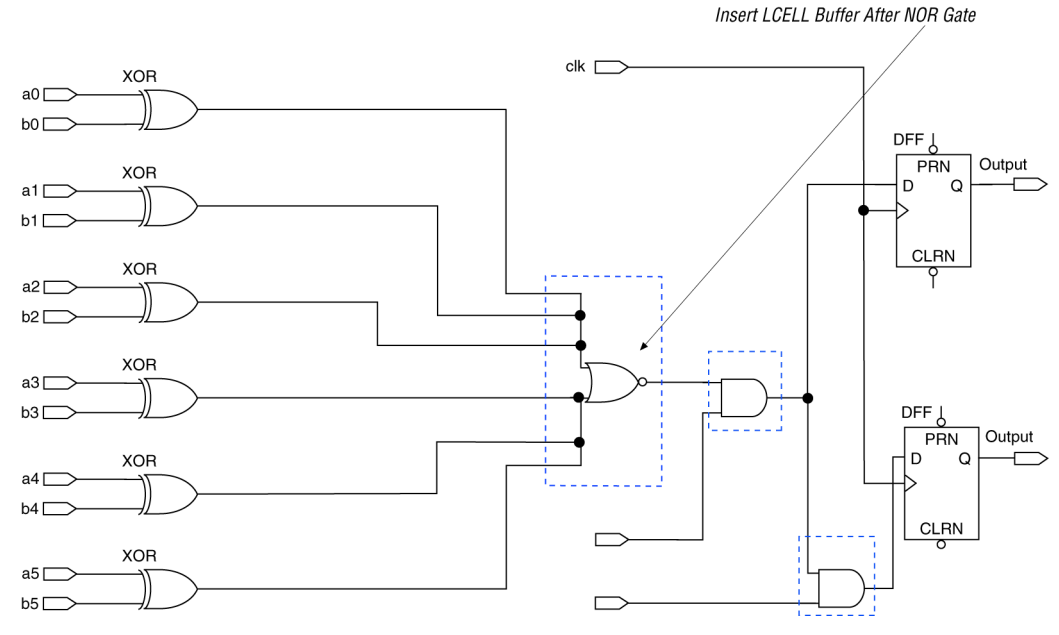
Complex logic, such as multilevel XOR gates, are often implemented with more than one macrocell. When this occurs, the Quartus II software automatically allocates shareable expanders—or additional macrocells (called synthesized logic cells)—to supplement the logic resources that are available in a single macrocell. You also can break down complex logic by inserting logic cells in the project to reduce the average fan-in and the 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 duplicate 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, refer to [Figure 8–16](#). The [Figure 8–16](#) 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 macrocells. However, with the LCELL buffer, the design requires three macrocells, eight shareable expanders, and the average fan-in is just 6.33 macrocells.

Figure 8–16. Reducing the Average Fan-In by Inserting LCELL Buffers



Timing Optimization Techniques (Macrocell-Based CPLDs)

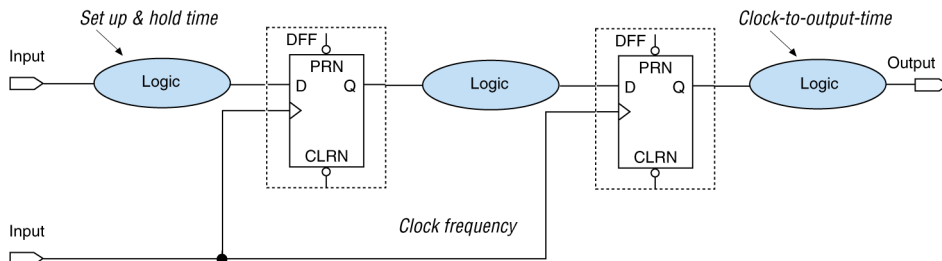
After resource optimization, design optimization focuses on timing. Ensure that you have made the appropriate assignments as described in “Initial Compilation” on page 8–5, and that the resource utilization is satisfactory before proceeding with timing optimization.

Maintaining system performance at or above certain timing requirements is an important goal of circuit designs. The following five timing parameters are primarily responsible for a design’s performance:

- Setup time (t_{SU}), the propagation time for input data signals
- Hold time (t_H), the propagation time for input data signals
- Clock-to-output time (t_{CO}), the propagation time for output signals.
- Pin-to-pin delays (t_{PD}), the time required for a signal from an input pin to propagate through combinational logic and appear at an external output pin
- Maximum clock frequency (f_{MAX}), the internal register-to-register performance.

This section provides guidelines to improve the timing if the timing requirements are not met. Figure 8–17 shows the parts of the design that determine the t_{SU} , t_H , t_{CO} , t_{PD} , and f_{MAX} timing parameters.

Figure 8–17. 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 for the classic timing analyzer, as discussed in “Design Analysis” on page 8–14.

When you are analyzing a design to improve performance, be sure to consider the two major contributors to long delay paths:

- Excessive levels of logic
- Excessive loading (high fan-out)

When a MAX 7000 or MAX 3000 device signal drives 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, 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 does not meet your timing requirements. To improve the t_{SU} timing, refer to the following guidelines:

- Turn on the **Fast Input Register** option using the Assignment Editor. 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 useful when a pin drives a D-type flipflop that does not have 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 causes more delays. 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. To improve the setup time, minimize the fan-out.

Improving Clock-to-Output Time

To improve a design's clock-to-output time, minimize the register-to-output-pin delay. To improve the t_{CO} timing, refer to the following guidelines.

- Use the global clock. In addition to minimizing the delay from a register to an output pin, minimizing the delay from the clock pin to the register also can improve t_{CO} timing. 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 causes more delay. Always minimize the amount of logic between the register and output pin for faster clock-to-output time.

Table 8–12 shows the 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} delays increase.

Table 8–12. EPM7064AETC100-4 Device Timing Results

Number of Registers	t_{SU} (ns)	t_H (ns)	t_{CO} (ns)	Global Clock Used	Fast Input Register Option	D Input Location	Q Output Location	Additional Logic Between:	
								D Input Location & Register	Register & Q Output Location
1	1.3	1.2	4.3	No	On	LAB A	LAB A	No	No
1	1.6	0.3	4.3	No	Off	LAB A	LAB A	No	No
1	2.5	0	3.1	Yes	On	LAB A	LAB A	No	No
1	2.8	0	3.1	Yes	Off	LAB A	LAB A	No	No
1	3.6	0	3.1	Yes	Off	LAB A	LAB A	Yes	No
1	2.8	0	7.0	Yes	Off	LAB D	LAB A	No	Yes
16 with the same D and clock inputs	2.8	0	All 6.2	Yes	Off	LAB D	LAB A, B	No	No
32 with the same D and clock inputs	2.8	0	All 6.4	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.

- On the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**, and turn on **Auto Parallel Expanders**. Turning on the parallel expanders for individual nodes or sub-designs 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 increases and results in a no-fit error during compilation if the device resources are limited. For more information about turning the **Auto Parallel Expanders** option on, refer to [“Resolving Macrocell Usage Issues” on page 8–69](#).
- 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. Reset the **Optimization Technique** option to **Speed** only if you previously set it to **Area**. On the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**, and turn on **Speed** under **Optimization Technique**.

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 inputs of the registers 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 following guidelines to improve your design's clock speed (f_{MAX}).

- On the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings** and turn on **Auto Parallel Expanders**. 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 also increases and can result in a no-fit error during compilation if the device's resources are limited. For more information about using the **Auto Parallel Expanders** option, refer to ["Resolving Macrocell Usage Issues"](#) on page 8–69.
- Use global signals or dedicated inputs. Altera MAX 7000 and MAX 3000 devices have dedicated inputs that provide low skew and high speed for high fan-out signals. 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. Reset the **Optimization Technique** option to **Speed** only if you have previously set it to **Area**. You can reset the **Optimization Technique** option. In the **Category** list, choose **Analysis & Synthesis Settings**, and turn on **Speed** under **Optimization Technique**.
- Pipeline the design. Pipelining, which increases clock frequency (f_{MAX}), refers to dividing large blocks of combinational logic by inserting registers. For more information about pipelining, refer to ["Optimizing Source Code—Pipelining for Complex Register Logic"](#) on page 8–77.

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 4-bit numbers. The decoder is based on five 2-to-4 pipelined decoders with outputs that are registered using D-type flipflops. Figure 8–18 shows one of the 2-to-4 pipelined decoders. The function 2TO4DEC is the 2-to-4 decoder that feeds all four decoded outputs (out1, out2, out3, and out4) to the D-type flipflops in 4REG.

Figure 8–18. A 2- to 4-Pipelined Decoder

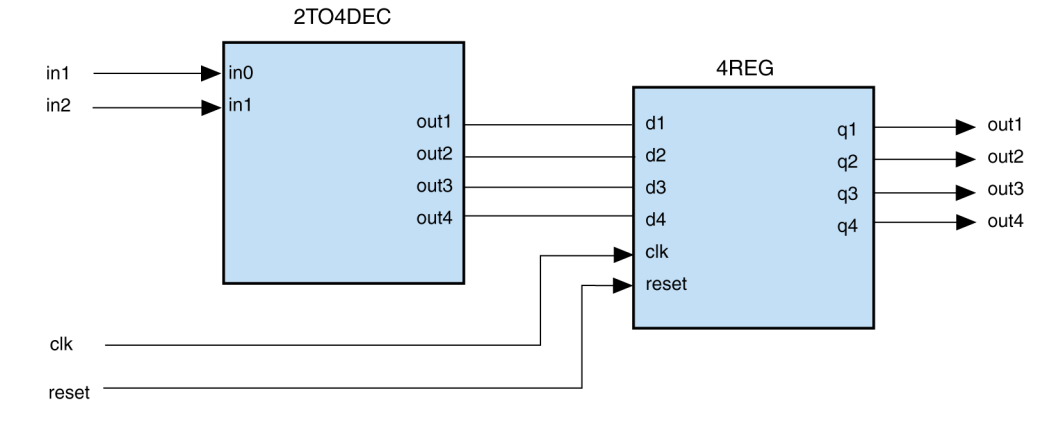
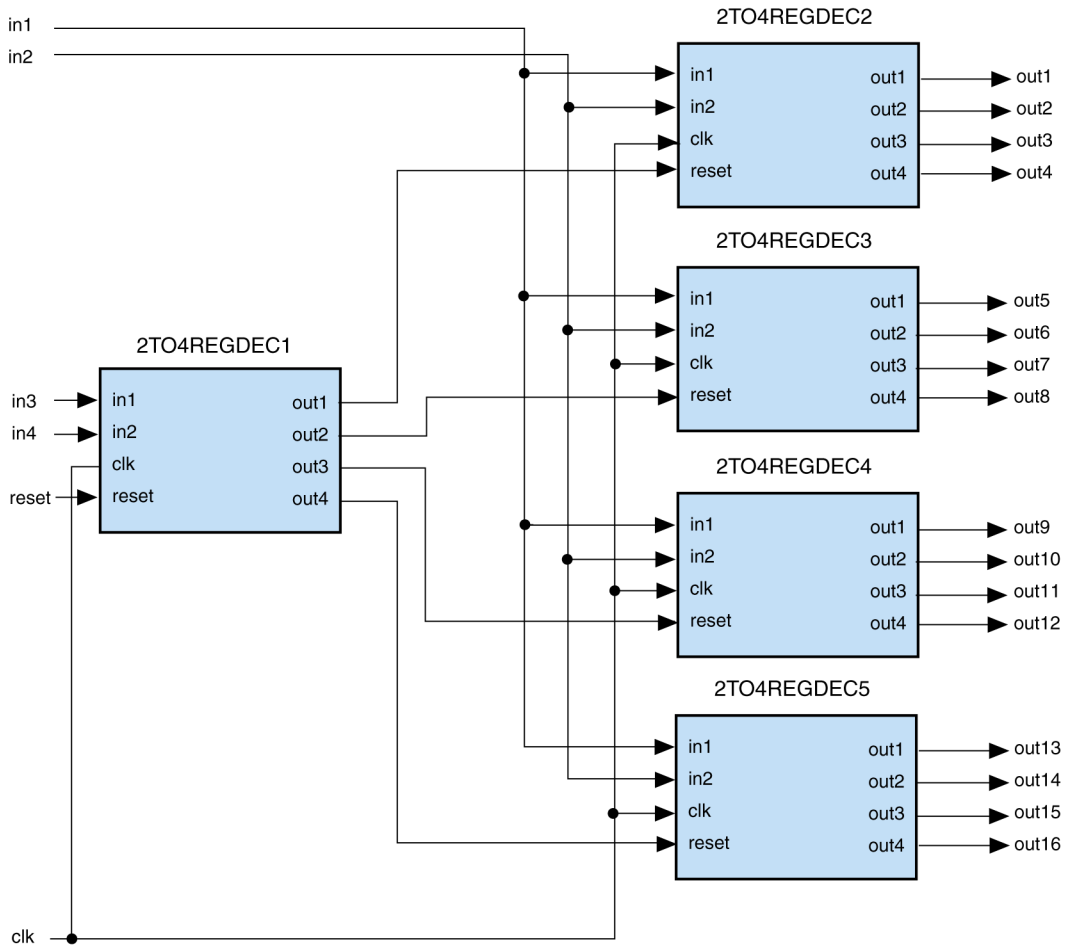


Figure 8–19 shows five 2-to-4 decoders (2TO4REGDEC) that are combined to form a 4-to-16 pipelined decoder. The first decoder (2TO4REGDEC1) decodes the two most significant bits (MSB) (in3 and in4) of the 4-to-16 decoder. The decoded output from the 2TO4REGDEC1 decoder enables only one of the rest of the 2-to-4 decoders (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 reduces the time consumed to generate the decoded output, thus improving the maximum frequency. In Figure 8–19, the MSBs (in3 and in4) are decoded in the first clock cycle, while the other bits (in1 and in2) are decoded in the following clock cycle.

Figure 8–19. 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 some of these techniques can reduce the overall quality of results.

Incremental Compilation

You can speed up design iteration time by an average of 60% when making changes to the design and reach design timing closure more efficiently with the incremental compilation feature. Using incremental compilation allows you to organize your design into logical and physical partitions for design synthesis and fitting. Design iterations can be made dramatically faster by recompiling only a particular design partition and merging results with previous compilation results of other partitions. You can also use optimization techniques such as physical synthesis for specific design partitions while leaving other modules untouched to preserve performance.

When making changes to the design, use the incremental synthesis feature (part of incremental compilation) to save synthesis time. Incremental synthesis allows you to set design partitions to ensure that only those sections of a design that have been updated are resynthesized when the design is compiled, which reduces synthesis time and run-time memory usage.

If you are using a third-party synthesis tool, you can create separate atom netlist files for parts of your design that you already have synthesized and optimized so that you update only the parts of the design that change.

Regardless of your synthesis tool, you can use full incremental compilation along with LogicLock regions to preserve your placement and routing results for unchanged partitions while working on other partitions. This ability provides the most reduction in compilation time and run-time memory usage because neither synthesis nor fitting must be performed for unchanged partitions in the design.

You can also perform a bottom-up compilation in which parts of the design are compiled completely independently in separate Quartus II projects, and then exported into the top-level design. This flow is useful in team-based designs or when incorporating third-party IP.



For information about the full incremental compilation flow in the Quartus II software, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*. For information about using the Quartus II incremental synthesis feature alone, refer to the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*. For information about creating multiple netlist files in third-party tools for use with incremental compilation, refer to the appropriate chapter in the *Synthesis* section in volume 1 of the *Quartus II Handbook*.

Reduce Synthesis Time & Synthesis Netlist Optimization Time

You can reduce synthesis time by reducing your use of netlist optimizations and by using incremental synthesis. For more ideas on reducing synthesis time in third-party EDA synthesis tools, refer to your tool's documentation.

Synthesis Netlist Optimizations

You can use Quartus II integrated synthesis to synthesize and optimize HDL designs, and you can use synthesis netlist optimizations to optimize netlists that were synthesized by third-party EDA software. Using these netlist optimizations can cause the Analysis & Synthesis module to take much longer to run. Look at the Analysis & Synthesis messages to find out how much time these optimizations take. 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 off the optimizations to save time. If you need to turn on synthesis netlist optimizations to meet performance, you can optimize parts of your design hierarchy separately to reduce the overall time spent in analysis and synthesis.

Check Early Timing Estimation before Fitting

The Quartus II software allows you to get an estimate of your timing results after synthesis, before the design is fully processed by the Fitter. In cases where you want a quick estimate of your design results before proceeding with further design or synthesis tasks, this feature can save you significant compilation time. For more information, refer to [“Early Timing Estimation” on page 8–12](#).

In the Processing menu, point to **Start**, and click **Start Early Timing Estimate** after you perform analysis and synthesis in the Quartus II software.

Reduce 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 and 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.
- Use incremental compilation to preserve the placement for parts of the design.

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 negate the overall time reduction.

Fitter Effort Setting

On the Assignments menu, click **Settings**. In the **Category** list, select **Fitter Settings**, and use the **Fitter effort** setting to shorten runtime by changing the effort level to **Auto Fit** or **Fast Fit**.

Placement Effort Multiplier Settings

You can control the amount of time the Fitter spends in placement by reducing one aspect of placement effort with the **Placement Effort Multiplier** option. On the Assignments menu, click **Settings**. Select **Fitter Settings**, and click **More Settings**. Under **Existing Option Settings**, select **Placement Effort Multiplier**. The default is 1.0. Legal values must be greater than 0 and can be non-integer values. Numbers between 0 and 1 can reduce fitting time, but also can reduce placement quality and design performance. Numbers higher than 1 increase placement time and placement quality, but may reduce routing time for designs with routing congestion. For example, a value of 4 increases placement time by approximately 2 to 4 times, but may increase quality.

Final Placement Optimization Levels

The **Final Placement Optimization Level** option specifies whether the Fitter performs final placement optimizations. This can be set to **Always**, **Never**, and **Automatically**. Performing optimizations may improve f_{MAX} timing and fitting, but may require longer compilation times. The default setting of **Automatically** can be used with the **Auto Fit Fitter Effort Level** (also the default) to let the fitter decide whether these optimizations should run based on the routability and timing requirements of the design.

Setting the Final Placement Optimization to **Never** often reduces your compilation time, but typically affects routability negatively and reduces timing performance.

To change the PowerPlay Power Optimization level, on the Assignments menu, choose **Settings**. The Setting dialog box appears. From the Category list, select **Fitter Settings**. Click the **More Settings** button. Select **Final Placement Optimization Level**, and then from the drop-down menu, select the required setting.

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 8–6 on page 8–51](#) for detailed results.

If your design meets your performance requirements without physical synthesis options, turn them off to save time. You also can use the **Physical synthesis effort** setting on the **Physical Synthesis Optimizations** page under **Fitter Settings** in the **Category** list to reduce the amount of extra compilation time that these optimizations use. The **Fast** setting directs the Quartus II software to use a lower level of physical synthesis optimization that, compared to the normal level, can cause a smaller increase in compilation time. However, the lower level of optimization can result in a smaller increase in design performance.

Limit to One Fitting Attempt

This option causes the software to quit after one fitting attempt option, instead of looping through placement and routing with increased effort. Preventing multiple fitter loops controls compilation time. The option increases the fitting effort on this one fitting attempt so it increases the compilation time as compared to a compilation that requires only one fitting attempt.

If your design is not routable, using the **Limit to one fitting attempt option** results in a fitting error due to routing. The option might also be useful in this case, because finishing fitting more quickly allows you to analyze the results more quickly and optimize your design appropriately to improve routability.

From the Assignments menu, select **Settings**. On the Fitter Settings page, turn on **Limit to one fitting attempt**.

Preserving Placement, Incremental Compilation, & LogicLock Regions

Preserving information about previous placements can make future placements take less time. The incremental compilation provides an easy-to-use methodology for preserving placement results. For more information, refer to [“Incremental Compilation” on page 8–80](#) and the references listed in the section.

Reduce 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 incremental compilation to preserve routing information for parts of your design

Identify Routing Congestion in the Timing Closure Floorplan

To identify areas of congested routing in your design, open the Timing Closure Floorplan. On the Assignments menu, click **Timing Closure Floorplan**, and turn on **Show Routing Congestion**. This feature is available only when you choose the **Field View** on the View menu. Routing resource usage above 90% indicates routing congestion. You can change the connections in your design to reduce routing congestion. If the area with routing congestion is in a LogicLock region or between LogicLock regions, change or 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, consider changing the size, location, or contents of LogicLock regions to reduce congestion and decrease routing time.

Router Effort Multiplier Setting

To control how quickly the router finds a fit, you can change the value of the **Router Effort Multiplier** option by performing the following steps:

1. On the Assignments menu, click **Settings**.
2. In the Category list, select **Fitter Settings**, and click **More Settings**.
3. Select the **Router Effort Multiplier** option from the drop-down menu.

The default value is 1.0. Legal values must be greater than 0. Numbers closer to 0 (for example, 0.1) can reduce router run-time, but usually reduce router quality slightly, which also may reduce design performance.

Preserve Routing Incremental Compilation & LogicLock Regions

Preserving information about the previous routing results for part of the design can make future routing efforts take less time. The use of LogicLock regions with incremental compilation provides an easy-to-use methodology that preserves placement and routing results. For more information, refer to [“Incremental Compilation” on page 8–80](#) and the references listed in the section.

Use Multiple Processors for Multi-Threaded Compilation

The Quartus II software can run some algorithms in parallel to take advantage of multiple processors and reduce compilation time when more than one processor is available to compile the design. You can specify the maximum number of processors that the software can use. The software may not necessarily use all the processors that you specify during a given compilation, but it never uses more than the specified number of processors to ensure other jobs running on the same machine are not affected.

By allowing the Quartus II software to use two processors, you may be able to reduce the fitting time by up to 10%. Four processors can reduce fitting time by up to 15%. There is no marked reduction beyond this when more than four processors are in use, however. The actual reduction in compilation time depends on the design and the specific settings used for compilation. For example, compilations with fast-corner optimization turned on benefit more from using multiple processors than do compilations that do not use fast-corner optimization. The runtime requirement is not reduced for some other compilation stages, such as Analysis and Synthesis.

With certain design flows in which timing analysis runs alone, using multiple processors can reduce the time required for timing analysis by an average of 12% when using two processors. This reduction can reach an average of 15% when using four processors.



Do not consider processors with Intel Hyper-Threading to be more than one processor. If you have a single processor with Intel Hyper-Threading enabled, you should set the number of processors to one. Altera recommends that you do not use the Intel Hyper-Threading feature for Quartus II compilations.

The Quartus II software does not check the number of processors physically available, and configures its algorithms to use the specified number of processors. Therefore, you should not specify more processors than those actually available. Doing so could have an adverse effect, resulting in increased compilation time.

Using multiple processors does not affect the quality of the fit. For a given fitter seed on a specific design, the fit is exactly the same, regardless of whether it uses one processor or multiple processors. The only difference between such compilations using different number of processors is the compilation times.

To set the number of processors available for Quartus II compilation, on the Assignments menu, select **Settings**. From the **Settings** dialog box, under Category, click **Compilation Process Settings**. In the dialog box that appears, specify the **Maximum processors for multithreaded Quartus II use**. The default value for the number of processors is 1.

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 ←
```

The *Quartus II Scripting Reference Manual* includes the same information in PDF form.



For more information about Tcl scripting, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. Refer to the *Quartus II Settings File Reference Manual* for information about all settings and constraints in the Quartus II software. For more information about command-line scripting, refer to the *Command-Line Scripting* chapter 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

The Quartus Settings File 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.



This chapter refers to timing settings and analysis in the Quartus II Classic Timing Analyzer. For equivalent settings and analysis in the TimeQuest Timing Analyzer, refer to the *TimeQuest Timing Analyzer* or the *Switching to the TimeQuest Timing Analyzer* chapters in volume 3 of the *Quartus II Handbook*.

Table 8–13 lists the Quartus Settings File variable name and applicable values for the settings discussed in “Initial Compilation” on page 8–5.

Setting Name	Quartus Settings File Variable Name	Values	Type
Device Setting	DEVICE	<device part number>	Global
Use Smart Compilation	SPEED_DISK_USAGE_TRADEOFF	SMART, NORMAL	Global
Optimize IOC Register Placement For Timing	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
Router Effort Multiplier	ROUTER_EFFORT_MULTIPLIER	Any positive, non-zero value	Global
Router Timing Optimization level	ROUTER_TIMING_OPTIMIZATION_LEVEL	NORMAL, MINIMUM, MAXIMUM	Global
Final Placement Optimization	FINAL_PLACEMENT_OPTIMIZATION	ALWAYS, AUTOMATICALLY, NEVER	Global

Resource Utilization Optimization Techniques (LUT-Based Devices)

Table 8–14 lists the Quartus Settings File variable name and applicable values for the settings discussed in “Resource Utilization Optimization Techniques (LUT-Based Devices)” on page 8–23. 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 8–14. Resource Utilization Optimization Settings

Setting Name	QSF Variable Name	Values	Type
Auto Packed Registers (1)	AUTO_PACKED_REGISTERS_ <device family name>	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
Speed Optimization Technique for Clock Domains	SYNTH_CRITICAL_CLOCK	ON, OFF	Instance
State Machine Encoding	STATE_MACHINE_PROCESSING	AUTO, ONE-HOT, MINIMAL BITS, USER-ENCODE	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 Block Replacement	AUTO_DSP_RECOGNITION	ON, OFF	Global, Instance

Notes to Table 8–14:

(1) Allowable values for this setting depend on the device family that is selected.

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

Table 8–15 lists the QSF variable name and applicable values for the settings discussed in “I/O Timing Optimization Techniques (LUT-Based Devices)” on page 8–40. 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 8–15. I/O Timing Optimization Settings

Setting Name	Quartus Settings File Variable Name	Values	Type
Optimize IOC 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 8–16 lists the QSF variable name and applicable values for the settings discussed in “[f_{MAX} Timing Optimization Techniques \(LUT-Based Devices\)](#)” on page 8–47. 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 8–16. F_{MAX} Timing Optimization Settings (Part 1 of 2)

Setting Name	Quartus Settings File 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 t _{SU} /t _{CO} 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
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

Table 8–16. F_{MAX} Timing Optimization Settings (Part 2 of 2)

Setting Name	Quartus Settings File Variable Name	Values	Type
Manual Logic Duplication	DUPLICATE_ATOM	<node name>	Instance
Optimize Power during Synthesis	OPTIMIZE_POWER_DURING_SYNTHESIS	NORMAL, OFF EXTRA_EFFORT	Global
Optimize Power during Fitting	OPTIMIZE_POWER_DURING_FITTING	NORMAL, OFF EXTRA_EFFORT	Global

Duplicate Logic for Fan-Out Control

The manual logic duplication option accepts wildcards. This is an easy and powerful duplication technique that you can use without editing your source code. You can use this technique, for example, to make a duplicate of a large fan-out node for all of its destinations in a certain design hierarchy, such as `hierarchy_A`. To make such an assignment with Tcl, use a command similar to [Example 8–1](#).

Example 8–1. Duplication Technique

```
set_instance_assignment -name DUPLICATE_ATOM \
    high_fanout_to_A -from high_fanout_node \
    -to *hierarchy_A*
```

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 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.

Document Revision History

The table below shows the revision history for this chapter.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Updated for the Quartus II software version 6.1.0: <ul style="list-style-type: none"> ● Added references to the Power Optimization Advisor and Incremental Compilation Advisor ● Updated text to include TimeQuest Timing Analyzer ● Added <code>check_timing</code> for illegal & ignored constraints ● In the “Resource Utilization” section, modified note about ALM counts ● In the “Use Register Packing” section, updated for Stratix II support; added new benchmarking tables ● Added new sections to the “Routing” section: <ul style="list-style-type: none"> • Set Auto Register Packing to Auto • Set Fitter Aggressive Routability Optimizations to Always • Increase Router Effort Multiplier • Set Maximum Router Optimization Level ● To the “Increase Placement Effort Multiplier” section, added that second and third fitting loops increase the multiplier to 4 and then 16 ● Added relevant information for Stratix III ● In the “Synthesis Netlist Optimizations & Physical Synthesis Optimizations” section, updated benchmarking information in tables; reorganized information for clarity ● Added new section: “Turn Off Extra-Effort Power Optimization Settings” 	Updates for TimeQuest support, Stratix III devices, and updated benchmarking tables.
May 2006 v6.0.0	Updated for the Quartus II software version 6.0.0: <ul style="list-style-type: none"> ● Added Optimization advisors. ● Added initial compilation information. ● Added design analysis information. ● Added f_{MAX} timing optimization techniques. 	
December 2005 v5.1.1	Minor typographic corrections.	
October 2005 v5.1.0	Chapter 8 was formerly Chapter 7 in version 5.0.	
May 2005 v5.0.0	Chapter 7 was formerly Chapter 6 in version 4.2.	

Table 8–17. Documentation Revision History (Part 2 of 2)

Date & Document Version	Changes Made	Summary of Changes
Dec. 2004 v2.1	Updated for Quartus II software version 4.2: <ul style="list-style-type: none"> ● Re-organized chapter. ● Added Early Timing Estimation segment. ● Removed Incremental Fitting segment. ● Updated Optimization Advisors. ● Updated Resource Utilization Optimization Techniques (LUT-Based Devices) segment. ● Added the DSP Block Balancing logic option to Retarget or Balance DSP Blocks segment. ● Updated Duplicate Logic for Fan-Out Control segment. ● Updates to tables, figures. 	
June 2004 v2.0	<ul style="list-style-type: none"> ● Updates to tables, figures. ● New functionality in the Quartus II software version 4.1. 	
Feb. 2004 v1.0	Initial release.	

Introduction

The Quartus® II software offers power-driven compilation to fully optimize device power consumption. Power-driven compilation focuses on reducing your design's total power consumption using power-driven synthesis and power-driven place-and-route. This chapter describes the power-driven compilation feature and flow in detail, as well as low power design techniques that can further reduce power consumption in your design. The techniques primarily target Stratix® III, Stratix II, Stratix II GX, Cyclone™ II, and HardCopy® II devices. These devices utilize a low-k dielectric material that dramatically reduces dynamic power and improves performance. Stratix III and Stratix II devices include new, more efficient, logic structures called adaptive logic modules (ALMs) that obtain maximum performance while minimizing power consumption. Cyclone II devices offer the optimal blend of high performance and low power in a low-cost FPGA.



For more information on Stratix III architecture, refer to the *Stratix III Device Handbook*.

Altera® provides the Quartus II PowerPlay Power Analyzer to aid you during the design process by delivering fast and accurate estimations of power consumption. You can minimize power consumption, while taking advantage of the industry's leading FPGA performance, by using the tools and techniques described in this chapter.



For more information on the PowerPlay Power Analyzer, refer to the *PowerPlay Power Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Total FPGA power consumption is comprised of I/O power, core static power, and core dynamic power. This chapter focuses on design optimization options and techniques that help reduce core dynamic power and I/O power. In addition to these techniques there are additional power optimization techniques available for Stratix III devices. These techniques include:

- Selectable Core Voltage
- Programmable Power Technology
- Device Speed Grade selection



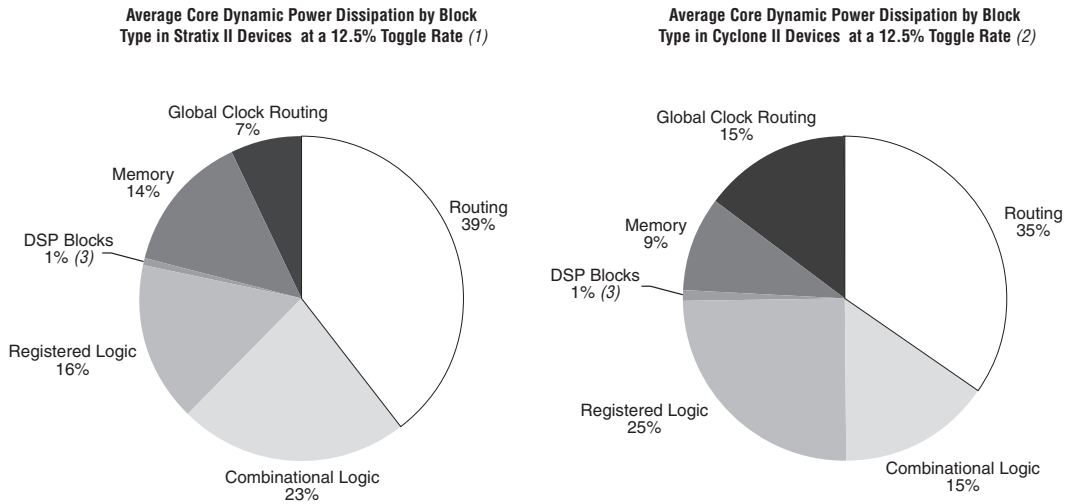
For more information on power optimization techniques available for Stratix II devices, refer to application note *AN: 437 Power Optimization in Stratix III FPGAs*.

Power Dissipation

This section describes the sources of power dissipation in Stratix II and Cyclone II devices. You can refine techniques that reduce power consumption in your design by understanding the sources of power dissipation.

Figure 9-1 shows the power dissipation of Stratix II and Cyclone II devices in different designs. All designs were analyzed at a fixed clock rate of 200 MHz and exhibited varied logic resource utilization across available resources.

Figure 9-1. Average Core Dynamic Power Dissipation



Notes to Figure 9-1:

- (1) 112 different designs were used to obtain these results.
- (2) 93 different designs were used to obtain these results.
- (3) In designs using DSP blocks, DSPs consumed 5% of core dynamic power.

As shown in Figure 9-1, a significant amount of the total power is dissipated in routing for both Stratix II and Cyclone II devices, with the remaining power dissipated in logic, clock, and RAM blocks.

In Stratix II and Cyclone II devices, a series of column and row interconnect wires of varying lengths provide signal interconnections between logic array blocks (LABs), memory block structures, and digital signal processing (DSP) blocks or multiplier blocks. These interconnects dissipate the largest component of device power.

FPGA combinational logic is another source of power consumption. The basic building block of logic in Stratix II devices is the ALM, and in Cyclone II devices, it is the logic element (LE).



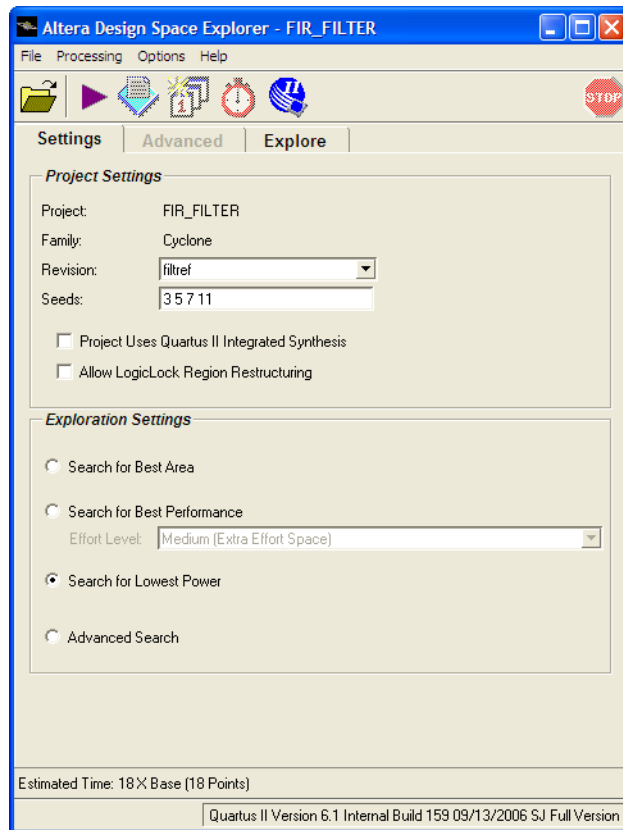
For more information on ALMs and LEs in Stratix III, Stratix II, and Cyclone II devices, refer to the *Stratix III Device Handbook*, *Stratix II Device Handbook*, and the *Cyclone II Device Handbook*, respectively.

Memory and clock resources are other major consumers of power in FPGAs. Stratix II devices feature the TriMatrix memory architecture. TriMatrix memory includes 512-bit M512 blocks, 4-Kbit M4K blocks, and 512-Kbit M-RAM blocks, which are each configurable to support many features. Cyclone II devices have 4-Kbit M4K memory blocks.

Design Space Explorer

The Design Space Explorer (DSE) is a simple, easy-to-use, design optimization utility that is included in the Quartus II software. The DSE explores and reports optimal Quartus II software options for your design, targeting either power optimization, design performance, or area utilization improvements. You can use the DSE to implement the techniques described in this chapter.

Figure 9–2 shows the DSE user interface. The **Settings** tab is divided into **Project Settings** and **Exploration Settings**.

Figure 9–2. Design Space Explorer User Interface

The **Search for Lowest Power** option, under **Exploration Settings**, uses a predefined exploration space that targets overall design power improvements. This setting focuses on applying different options that specifically reduce total design thermal power. You can also set the **Optimization Goal** option for your design using the **Advanced** tab in the DSE window. You can select your design optimization goal, such as optimize for power, from the list of available settings in the **Optimization Goal** list. The DSE then uses the selection from the **Optimization Goal** list, along with the **Search for Lowest Power** selection, to determine the best compilation results.

By default, the Quartus II PowerPlay Power Analyzer is run for every exploration performed by the DSE when the **Search for Lowest Power** option is selected. This helps you debug your design and determine trade-offs between power requirements and performance optimization.



For more information on the DSE, refer to the *Design Space Explorer* chapter in volume 2 of the *Quartus II Handbook*.

Power-Driven Compilation

The standard Quartus II compilation flow consists of Analysis and Synthesis, Fitter, Assembler, and Timing Analysis. Power-driven compilation takes place at the analysis and synthesis and fitter levels. Power-driven compilation settings are divided in the **PowerPlay power optimization** list on the **Analysis & Synthesis Settings** page, and **PowerPlay power optimization** on the **Fitter Setting** page. The following section describes these power optimization options at the analysis and synthesis and fitter levels.

Power-Driven Synthesis

Synthesis netlist optimization occurs during the synthesis stage of the compilation flow. The optimization technique makes changes to the synthesis netlist to optimize your design according to the selection of area, speed, or power optimization. This section describes power optimization techniques at the synthesis level.

The **Analysis & Synthesis Settings** page allows you to specify logic synthesis options. The **PowerPlay power optimization** option is available for Stratix III, Stratix II, Stratix II GX, Stratix, Stratix GX, Cyclone II, Cyclone, and MAX[®] II devices (Figure 9-3).

To perform power optimization at the synthesis level in the Quartus II software, perform the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Category** list, select **Analysis & Synthesis**. The **Analysis & Synthesis** page is shown.
3. In the **PowerPlay power optimization** list, select your preferred setting. This option determines how aggressively Analysis and Synthesis optimizes the design for power.

Figure 9–3. Analysis & Synthesis Settings Page

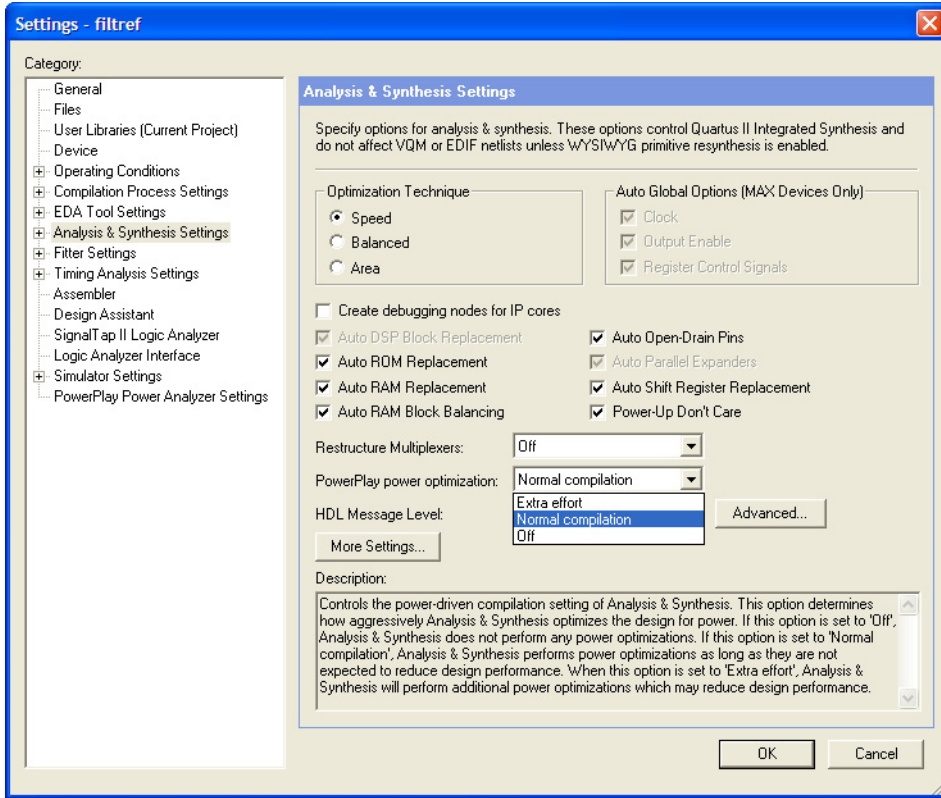


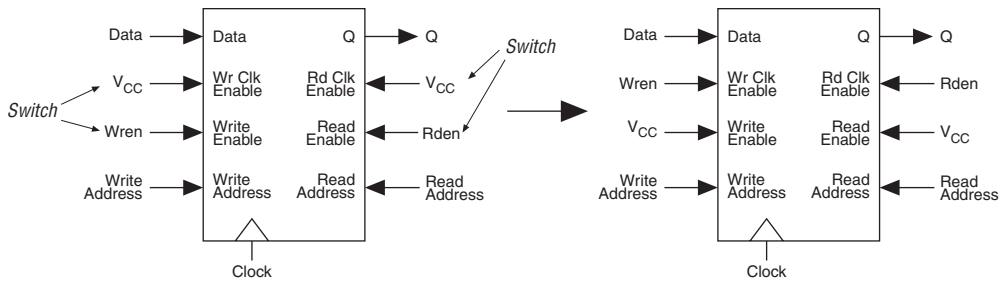
Table 9–1 shows the settings in the **PowerPlay power optimization** list. You can apply these settings on a project or entity level.

Settings	Description
Off	No power optimizations are performed
Normal compilation (Default)	Enables power optimizations as long as they are not expected to reduce design performance
Extra effort	Enables you to perform additional power optimizations which can reduce design performance

The **Normal compilation** setting is turned on by default. This setting performs memory optimization and power-aware logic mapping during synthesis.

Memory blocks can represent a large fraction of total design dynamic power as described in “[Reducing Memory Power Consumption](#)” on [page 9–22](#). Minimizing the number of memory blocks accessed during each clock cycle can significantly reduce memory power. Memory optimization involves effective movement of user-defined read/write enable signals to associated read-and-write clock enable signals for all memory types ([Figure 9–4](#)).

Figure 9–4. Memory Transformation



[Figure 9–4](#) shows a default implementation of a simple dual-port memory block in which write-clock enable and read-clock enable signals are connected to V_{CC} , making both read-and-write memory ports active during each clock cycle. Memory transformation effectively moves the read-enable and write-enable signals to the respective read-clock enable and write-clock enable signals. By using this technique, memory ports are shut down when they are not accessed. This significantly reduces your design’s memory power consumption. For more information on clock enable signals, refer to “[Reducing Memory Power Consumption](#)” on [page 9–22](#).

The other type of power optimization that takes place with the **Normal compilation** setting is power-aware logic mapping. The power-aware logic mapping reduces power by rearranging the logic during synthesis to eliminate nets with high toggle rates.

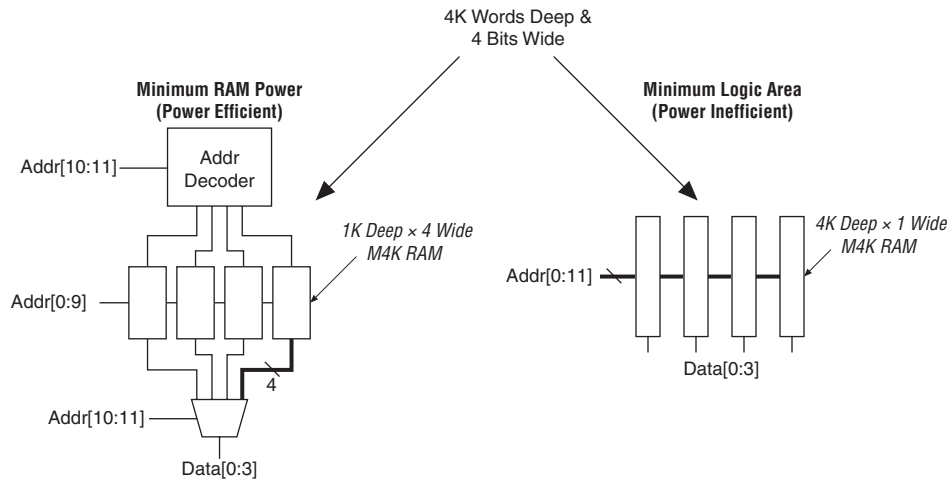
The **Extra effort** setting performs the functions of the **Normal compilation** setting and other memory optimizations to further reduce memory power by shutting down memory blocks that are not accessed. This level of memory optimization may require extra logic which can reduce design performance.

The **Extra effort** setting also performs power-aware memory balancing. Power-aware memory balancing automatically chooses the best memory configuration for your memory implementation and provides optimal power saving by determining the number of memory blocks, decoder, and multiplexer circuits needed. If you have not previously specified target-embedded memory blocks for your design's memory functions, the power-aware balancer automatically selects it during memory implementation.

Figure 9–5 shows an example of a $4K \times 4$ (4K deep and 4 bit wide) memory implementation in two different configurations using M4K memory blocks available in Stratix II devices. The minimum logic area implementation uses M4K blocks configured as $4K \times 1$. This implementation is the default in the Quartus II software because it has the minimum logic area (0 logic cells) and the highest speed. However, all four M4K blocks are active on each memory access in this implementation, which increases RAM power. The minimum RAM power implementation is created by selecting **Extra effort** in the **PowerPlay power optimization** list. This implementation automatically uses four M4K blocks configured as $1K \times 4$ for optimal power saving. An address decoder is implemented by the `altsyncram` megafunction to select which of the four M4K blocks should be activated on a given cycle, based on the state of the top two user address bits. The `altsyncram` megafunction automatically implements a multiplexer to feed the downstream logic by choosing the appropriate M4K output. This implementation reduces RAM power because only one M4K block is active on any cycle, but it requires extra logic cells, costing logic area and potentially impacting design performance.

There is a trade-off between power saved by accessing fewer memories and power consumed by the extra decoder and multiplexor logic. The Quartus II software automatically balances the power savings against the costs to choose the lowest power configuration for each logical RAM.

Figure 9–5. 4K × 4 Memory Implementation Using Multiple M4K Blocks



Memory optimization options can also be controlled by the `Low_Power_Mode` parameter in the **Default Parameters** page of the **Settings** dialog box. The settings for this parameter are **None**, **Auto**, and **ALL**. **None** corresponds to the **Off** setting in the **PowerPlay power optimization** list. **Auto** corresponds to the **Normal compilation** setting and **ALL** corresponds to the **Extra effort** setting, respectively. You can apply PowerPlay power optimization either on a compiler basis or on individual entities. The `Low_Power_Mode` parameter always takes precedence over the Optimize Power for Synthesis option for power optimization on memory.

You can also set the `MAXIMUM_DEPTH` parameter manually to configure the memory for low power optimization. This technique is the same as the power-aware memory balancer, but it is manual rather than automatic like the **Extra effort** setting in the **PowerPlay power optimization** list. You can set the `MAXIMUM_DEPTH` parameter for memory modules manually in the megafunction instantiation or in the MegaWizard® Plug-In Manager for power optimization as described in [“Reducing Memory Power Consumption” on page 9–22](#). The `MAXIMUM_DEPTH` parameter always takes precedence over the Optimize Power for Synthesis options for power optimization on memory optimization.

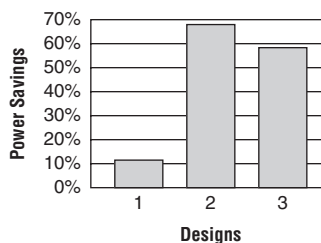
Power-Driven Synthesis Experiment for Stratix II Devices

In this experiment for Stratix II devices, three designs are compiled with the Quartus II software using **Normal compilation** and **Extra effort** settings in the **PowerPlay power optimization** list. The default setting for Fitter is **Normal compilation**. Table 9–2 shows resources used in the power-driven synthesis experiment.

Design Name	Settings	ALUT	Register	Memory Bits
Design 1	Normal compilation	8,941	9,150	293,856
	Extra effort	8,954	9,151	293,856
Design 2	Normal compilation	28,169	12,148	1,009,920
	Extra effort	28,817	12,297	1,009,920
Design 3	Normal compilation	5,376	2,809	153,864
	Extra effort	5,559	2,813	153,864

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–6 shows that the power-driven synthesis reduces memory power consumption by as much as 68% in Stratix II devices.

Figure 9–6. Memory Blocks Power Saving Using the Power-Driven Synthesis for Stratix II Devices



Power-Driven Fitter

The **Fitter Settings** page enables you to specify options for fitting (Figure 9–7). The **PowerPlay power optimization** option is available for Stratix III, Stratix II, Stratix II GX, Cyclone II, HardCopy II, and MAX II devices.

To perform power optimization at the fitter level, perform the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Category** list, select **Fitter Settings**. The **Fitter Settings** page is shown.
3. In the **PowerPlay power optimization** list, select your preferred setting. This option determines how aggressively the Fitter optimizes the design for power.

Figure 9–7. Fitter Settings Window

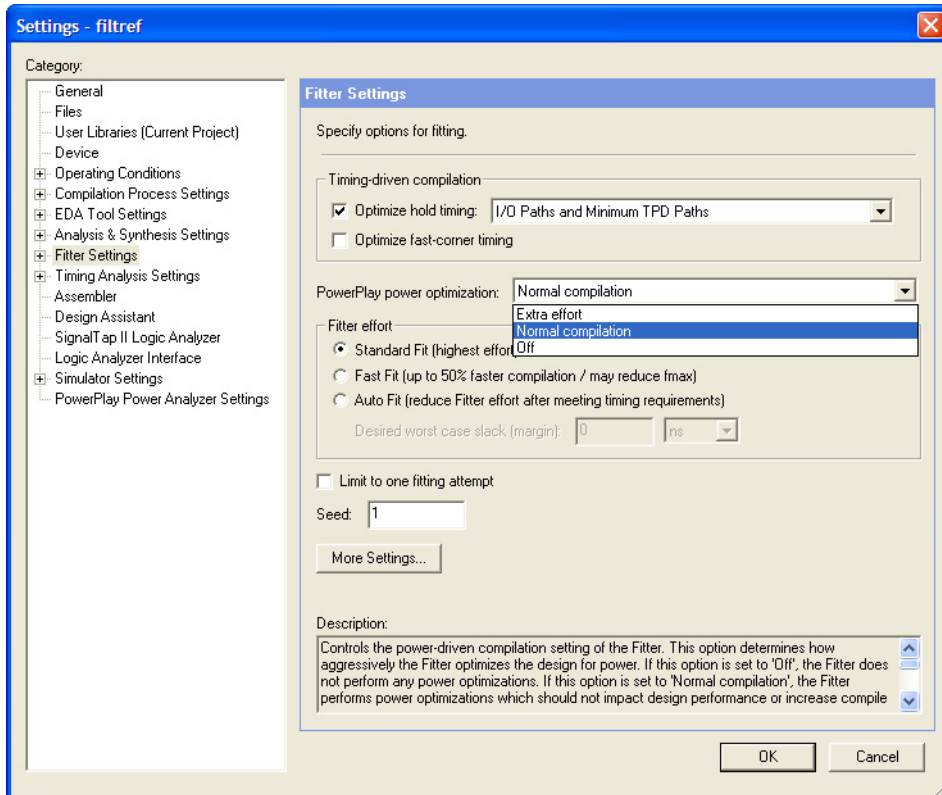


Table 9–3 lists the settings in the **PowerPlay power optimization** list. These settings can only be applied on a project-wide basis. The **Extra effort** setting for the Fitter requires extensive effort to optimize the design for power and can increase the compilation time.

Settings	Description
Off	No power optimizations are performed
Normal compilation (Default)	Enables power optimizations as long as they are not expected to reduce design performance
Extra effort	Enables you to perform additional power optimizations that can reduce design performance

The **Normal compilation** setting is selected by default and performs DSP optimization by creating power-efficient DSP block configurations for your DSP functions. For Stratix III devices, this setting, which is based on timing constraints entered for the design, enables the Programmable Power Technology to configure tiles as high-speed mode or low-power mode. Tiles are the combination of LAB and MLAB pairs (including the adjacent routing associated with LAB and MLAB) which can be configured to operate in high-speed or low-power mode. This level of power optimization will not have any affect on the fitting, timing results, or the compile time.



For more information on Stratix III power optimization, refer to *Power Optimization in Stratix III FPGAs*.

The **Extra effort** setting performs the functions of the **Normal compilation** setting and other place-and-route optimizations during fitting to fully optimize the design for power. The Fitter applies an extra effort to minimize power even after timing requirements have been met by effectively moving the logic closer during placement to localize high-toggling nets, and using routes with low capacitance. However, this effort can increase the compilation time.

The **Extra effort** setting uses a Signal Activity File (.saf) or Verilog Value Change Dump File (.vcd) that guides the Fitter to fully optimize the design for power based on the signal activity of the design. The best power optimization during fitting results from using the most accurate signal activity information. Signal activities from full post-fit netlist (timing) simulation provide the highest accuracy because all node activities reflect the actual design behavior, provided that supplied input vectors are representative of typical design operation. If you do not have a Signal Activity File (from simulation or other source), then the

Quartus II software uses assignments, clock assignments, and vectorless estimation values (PowerPlay Power Analyzer Tool settings) to estimate the signal activities. This information is used to optimize your design for power during fitting.



Only the **Extra effort** setting in the **PowerPlay power optimization** list for the Fitter option uses the signal activities (from Value Change Dump File or SAF) during fitting. The settings made in the **PowerPlay Power Analyzer Settings** page in the **Settings** dialog box are used to calculate the signal activity of your design.



For more information on Signal Activity Files and Verilog Value Change Dump Files, and how to create them, refer to the *PowerPlay Power Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Power-Driven Fitter Experiment for Stratix II Devices

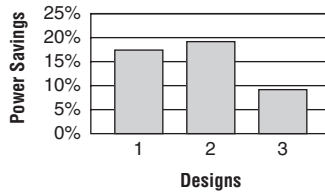
In this experiment for Stratix II devices, three designs are compiled with the Quartus II software using the **Normal compilation** and **Extra effort** settings in the Fitter for the PowerPlay power optimization list. The default setting for Analysis and Synthesis is **Normal compilation**.

Table 9–4 shows resources used in the power-driven fitter experiment.

Design Name	ALUTs (Normal Compilation)	ALUTs (Extra Effort)
Design 1	21,435	21,363
Design 2	19,035	18,970
Design 3	5,335	5,328

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–8 shows that the power-driven fitter technique reduces power consumption by as much as 19% in Stratix II devices.

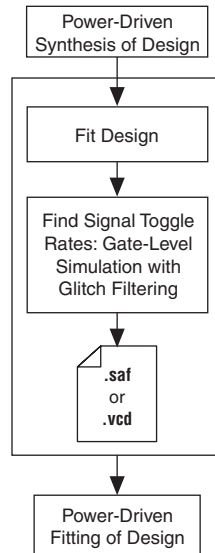
Figure 9–8. Power Savings Using the Power-Driven Fitter for Stratix II Devices



Recommended Flow for Power-Driven Compilation

Figure 9–9 shows the recommended design flow to fully optimize your design for power during compilation. This flow utilizes the power-driven synthesis and power-driven fitter options. On average, you can reduce core dynamic power by 16% with the extra effort synthesis and extra effort fitting settings, as compared to off settings in both synthesis and fitter options for power-driven compilation.

Figure 9–9. Recommended Flow for Power-Driven Compilation



Area-Driven Synthesis

Using of area optimization rather than timing or delay optimization during synthesis saves power because you use fewer logic blocks. Using less logic usually means less switching activity.

Area-Driven Synthesis Experiment for Stratix II Devices

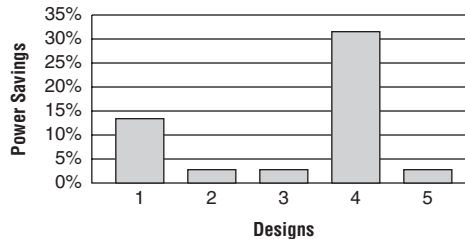
In this experiment for Stratix II devices, five designs are compiled with the Quartus II software in two ways. First, the designs are compiled optimizing for area. The same designs are then compiled optimizing for speed. The power optimization settings for synthesis and fitting are set to Off.

Table 9-5 shows ALUT usage in the area-driven synthesis experiment.

Design Name	ALUTs (Area Mapping)	ALUTs (Speed Mapping)
Design 1	5,682	8,553
Design 2	16,986	17,783
Design 3	36,554	36,312
Design 4	4,717	5,820
Design 5	15,947	15,978

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9-10 shows that the area-driven technique reduces power consumption by as much as 31% in Stratix II devices.

Figure 9-10. Power Savings Using Area-Driven Synthesis for Stratix II Devices



Area-Driven Synthesis Experiment for Cyclone II Devices

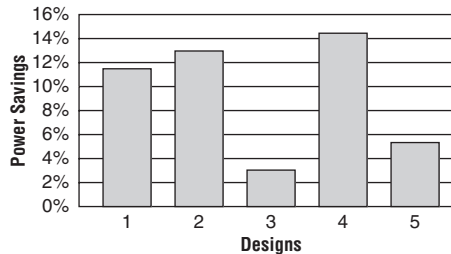
In this experiment for Cyclone II devices, five designs are compiled with the Quartus II software in two ways. First, the designs are compiled optimizing for area. The same designs are then compiled optimizing for speed.

Table 9–6 shows LE usage in the area-driven synthesis experiment.

Design Name	LEs (Area Mapping)	LEs (Speed Mapping)
Design 1	13,020	16,429
Design 2	13,317	13,636
Design 3	5,384	5,690
Design 4	33,640	40,008
Design 5	21,409	22,988

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–11 shows that the area-driven technique reduces power consumption by as much as 15% in Cyclone II devices.

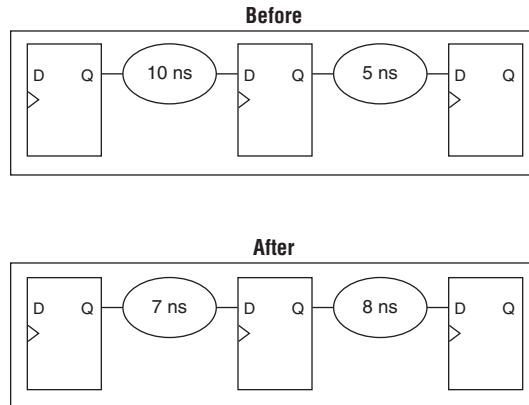
Figure 9–11. Power Savings Using Area-Driven Synthesis for Cyclone II Devices



Gate-Level Register Retiming

You can also use gate-level register retiming to reduce circuit switching activity. Retiming shuffles registers across combinational blocks without changing design functionality. The **Perform gate-level register retiming** option in the Quartus II software enables the movement of registers across combinational logic to balance timing, allowing the software to trade off the delay between timing critical and non-critical timing paths.

Retiming uses fewer registers than pipelining. Figure 9–12 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, resulting in the reduction of data depth and switching activity.

Figure 9–12. Gate-Level Register Retiming

Gate-level register retiming makes changes at the gate level. If you are using an atom netlist from a third-party synthesis tool, you must also select the **Perform WYSIWYG primitive resynthesis** option to undo the atom primitives to gates mapping (so that register retiming can be performed), and then to remap gates to Altera primitives. When using the Quartus II integrated synthesis, retiming occurs during synthesis before the design is mapped to Altera primitives.



For more information on register retiming, refer to the *Netlist Optimizations & Physical Synthesis* chapter in volume 2 of the *Quartus II Handbook*.

Gate-Level Register Retiming Experiment for Stratix II Devices

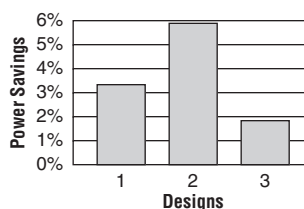
In this experiment for Stratix II devices, three designs are compiled with the Quartus II software in two ways. First, a netlist from a third-party synthesis tool is compiled. Then, the same netlist is compiled after selecting the **Perform WYSIWYG primitive resynthesis** and **Perform gate-level register retiming** options.

Table 9–7 shows resource usage results.

<i>Table 9–7. Resources Used in the Gate-Level Register Retiming Experiment for Stratix II Devices</i>					
Design Name	WYSIWYG & Register Retiming	ALUTs	Registers	DSP Blocks	Memory
Design 1	No	2,051	691	0	16
	Yes	1,882	731	0	16
Design 2	No	123,909	40,070	0	0
	Yes	95,593	39,816	0	0
Design 3	No	6,354	6,019	64	3,584
	Yes	7,496	5,970	64	3,584

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–13 shows that the combination of WYSIWYG remapping and gate-level register retiming reduces power consumption by nearly 6% in Stratix II devices.

Figure 9–13. Power Savings Using Retiming for Stratix II Devices



Gate-Level Register Retiming Experiment for Cyclone II Devices

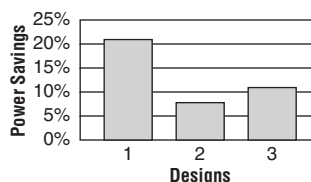
In this experiment for Cyclone II devices, three designs are compiled with the Quartus II software in two ways. First, a netlist from a third-party synthesis tool is compiled. Then, the same netlist is compiled by selecting the **Perform WYSIWYG primitive resynthesis** and **Perform gate-level register retiming** options.

Table 9–8 shows resource usage results.

Design Name	WYSIWYG & Register Retiming	LEs	Registers	Multiplier Blocks	Memory
Design 1	No	385	137	0	0
	Yes	278	143	0	0
Design 2	No	14,758	1,683	0	0
	Yes	13,079	1,683	0	0
Design 3	No	31,727	29,097	96	3,120
	Yes	27,038	24,272	96	3,120

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–14 shows that the combination of WYSIWYG remapping and gate-level register retiming reduces power consumption by as much as 21% in Cyclone II devices.

Figure 9–14. Power Savings Using Retiming for Cyclone II Devices



Design Guidelines

Several low-power design techniques can reduce power consumption when applied during FPGA design implementation. This section provides detailed design techniques for Stratix III, Stratix II, and Cyclone II devices that affect overall design power. The results of these techniques may be different from design to design.

Clock Power Management

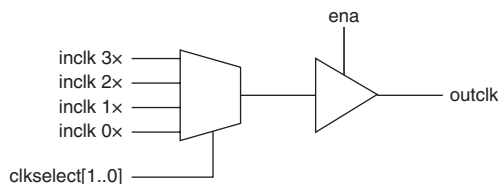
Clocks represent a significant portion of dynamic power consumption due to their high switching activity and long paths. Figure 9–1 shows a 7% average contribution to power consumption for global clock routing in Stratix II devices and 15% in Cyclone II devices. Actual clock-related

power consumption is higher than this because the power consumed by local clock distribution within logic, memory, and DSP or multiplier blocks is included in the power consumption for the respective blocks.

Clock routing power is automatically optimized by the Quartus II software, which only enables those portions of the clock network that are required to feed downstream registers. Power can be further reduced by gating clocks when they are not needed. It is possible to build clock gating logic, but this approach is not recommended because it is difficult to generate a glitch-free clock in FPGAs using ALMs or LEs.

Stratix III, Stratix II, and Cyclone II devices use clock control blocks that include an enable signal. A clock control block is a clock buffer that lets you dynamically enable or disable the clock network and dynamically switch between multiple sources to drive the clock network. You can use the Quartus II MegaWizard Plug-In Manager to create this clock control block with the `altclkctrl` megafunction. Stratix II and Cyclone II devices provide clock control blocks for global clock networks. In addition, Stratix II devices have clock control blocks for regional clock networks. The dynamic clock enable feature lets internal logic control the clock network. When a clock network is powered down, all the logic fed by that clock network does not toggle, thereby reducing the overall power consumption of the device. Figure 9–15 shows a 4-input clock control block diagram.

Figure 9–15. Clock Control Block Diagram



The enable signal is applied to the clock signal before being distributed to global routing. Therefore, the enable signal can either have a significant timing slack (at least as large as the global routing delay) or it can reduce the f_{MAX} of the clock signal.

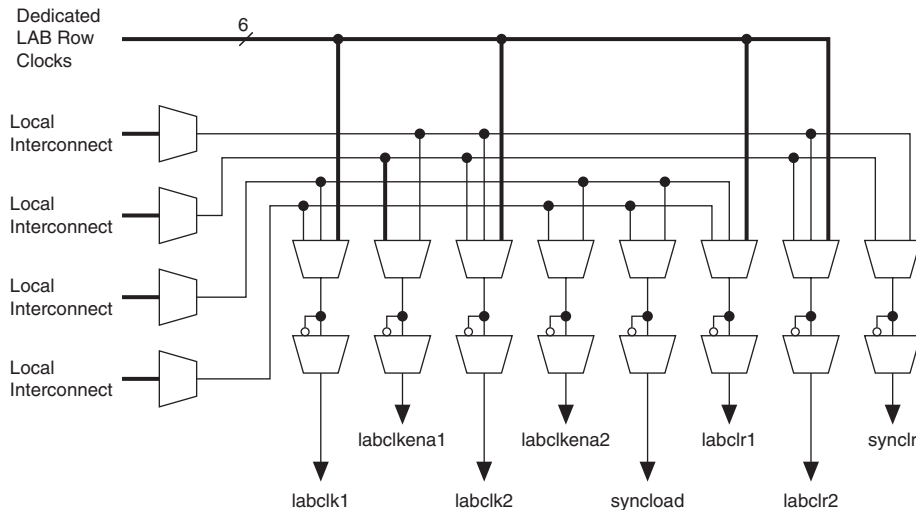


For more information about using clock control blocks, refer to the *altclkctrl Megafunction User Guide*.

Another contributor to clock power consumption is the LAB clock that distributes a clock to the registers within a LAB. LAB clock power can be the dominant contributor to overall clock power. For example, in

Cyclone II devices, each LAB can use two clocks and two clock enable signals as, shown in Figure 9–16. Each LAB's clock signal and clock enable signal are linked. For example, an LE in a particular LAB using the `labclk1` signal also uses the `labckena1` signal.

Figure 9–16. LAB-Wide Control Signals



To reduce LAB-wide clock power consumption without disabling the entire clock tree, use the LAB-wide clock enable to gate the LAB-wide clock. The Quartus II software automatically promotes register-level clock enable signals to the LAB-level. All registers within an LAB that share a common clock and clock enable are controlled by a shared gated clock. To take advantage of these clock enables, use a clock enable construct in the relevant HDL code for the registered logic.

LAB-Wide Clock Enable Example

This VHDL code makes use of a LAB-wide clock enable. This clock-gating logic is automatically turned into an LAB-level clock enable signal.

```

IF clk'event AND clock = '1' THEN
    IF logic_is_enabled = '1' THEN
        reg <= value;
    ELSE
        reg <= reg;
    END IF;
END IF;

```



For more information on LAB-wide control signals, refer to the *Stratix II Architecture* or *Cyclone II Architecture* chapters in the respective device handbook.

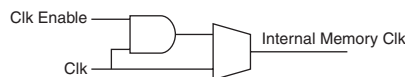
Reducing Memory Power Consumption

The memory blocks in FPGA devices can represent a large fraction of typical core dynamic power. Memory represents 14% of the core dynamic power in a typical Stratix II device design and 9% in a Cyclone II device design. Memory blocks are unlike most other blocks in the device because most of their power is tied to the clock rate, and is insensitive to the toggle rate on the data and address lines.

When a memory block is clocked, there is a sequence of timed events that occur within the block to execute a read or write. The circuitry controlled by the clock consumes the same amount of power regardless of whether or not the address or data has changed from one cycle to the next. Thus, the toggle rate of input data and the address bus have no impact on memory power consumption.

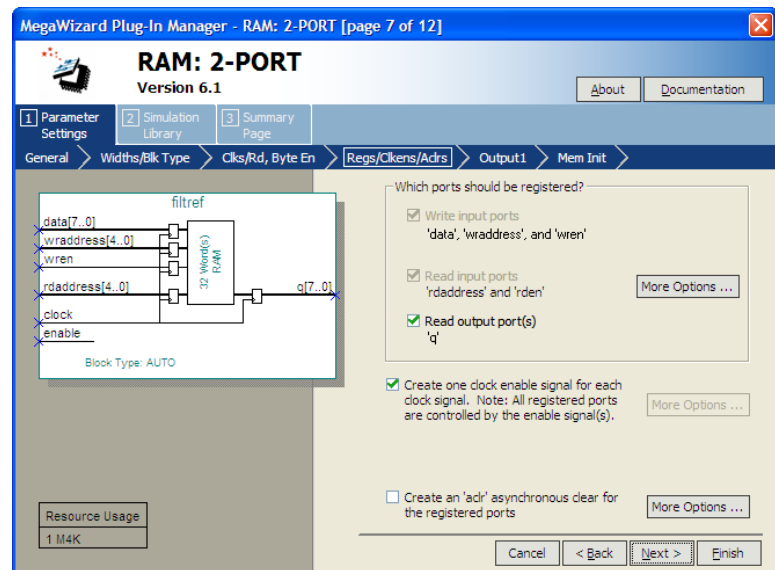
The key to reducing memory power consumption is to reduce the number of memory clocking events. You can achieve this through clock network-wide gating described in “[Clock Power Management](#)” on [page 9–19](#), or on a per-memory basis through use of the clock enable signals on the memory ports. [Figure 9–17](#) shows the logical view of the internal clock of the memory block. Use the appropriate enable signals on the memory to make use of the clock enable signal instead of gating the clock.

Figure 9–17. Memory Clock Enable Signal



Using the clock enable signal enables the memory only when necessary and shuts it down for the rest of the time, reducing the overall memory power consumption. You can use the Quartus II MegaWizard Plug-In Manager to create these enable signals by selecting the **Clock enable signal** option for the appropriate port when generating the memory block function ([Figure 9–18](#)).

Figure 9–18. MegaWizard Plug-In Manager RAM 2-Port Clock Enable Signal Selectable Option

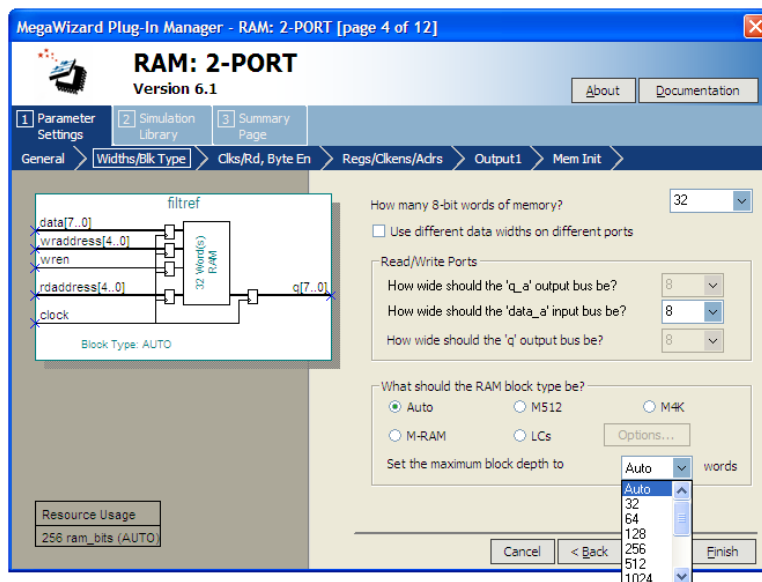


For example, consider a design that contains a 32-bit-wide M4K memory block in ROM mode that is running at 200 MHz. Assuming that the output of this block is only needed approximately every four cycles, this memory block will consume 8.45 mW of dynamic power according to the demands of the downstream logic. By adding a small amount of control logic to generate a read clock enable signal for the memory block only on the relevant cycles, the power can be cut 75% to 2.15 mW.

You can also use the `MAXIMUM_DEPTH` parameter in your memory megafunction to save power in Stratix II and Cyclone II devices; however, this approach may increase the number of LEs required to implement the memory and affect design performance.

You can set the `MAXIMUM_DEPTH` parameter for memory modules manually in the megafunction instantiation or in the MegaWizard Plug-In Manager (Figure 9–19). The Quartus II software can automatically choose the best design memory configuration for optimal power as described in “Power-Driven Compilation” on page 9–5.

Figure 9–19. MegaWizard Plug-In Manager RAM 2-Port Maximum Depth Selectable Option



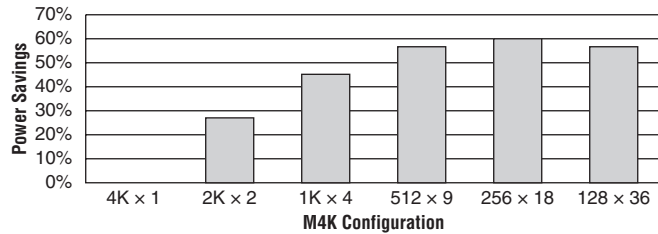
Memory Power Reduction Example

Table 9–9 shows power usage measurements for a 4K × 36 simple dual-port memory implemented using multiple M4K blocks in a Stratix II EP2S15 device. For each implementation, the M4K blocks are configured with a different memory depth.

M4K Configuration	Number of M4K Blocks	ALUTs
4K × 1 (Default setting)	36	0
2K × 2	36	40
1K × 4	36	62
512 × 9	32	143
256 × 18	32	302
128 × 36	32	633

Figure 9–20 shows the amount of power saved using the `MAXIMUM_DEPTH` parameter. For all implementations, a user-provided read enable signal is present to indicate when read data is needed. Using this power saving technique can reduce power consumption by as much as 60%.

Figure 9–20. Power Savings Using `MAXIMUM_DEPTH` Parameter



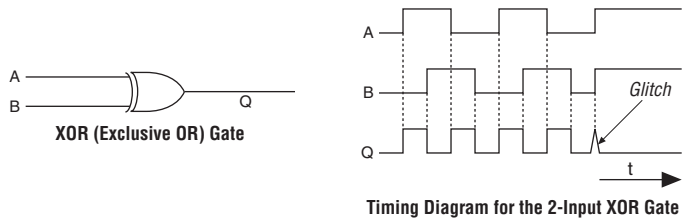
As the memory depth becomes more shallow, memory dynamic power decreases because unaddressed M4K blocks can be shut off using a decoded combination of address bits and the read enable signal. For a 128-deep memory block, power used by the extra LEs starts to outweigh the power gain achieved by using a more shallow memory block depth. The power consumption of the memory blocks and associated LEs depends on the memory configuration.

Pipelining & Retiming

Designs with many glitches consume more power because of faster switching activity. Glitches cause unnecessary and unpredictable temporary logic switches at the output of combinational logic. A glitch usually occurs when there is a mismatch in input signal timing leading to unequal propagation delay.

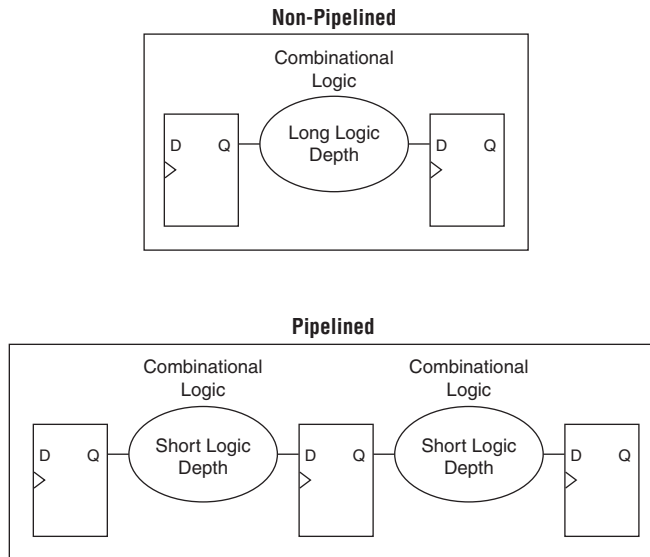
For example, consider an input change on one input of a 2-input XOR gate from 1 to 0, followed a few moments later by an input change from 0 to 1 on the other input. For a brief moment of time, both inputs become 1 (high) during the state transition, resulting in 0 (low) at the output of the XOR gate. Subsequently, when the second input transition takes place, the XOR gate output becomes 1 (high). During signal transition, a glitch is produced before the output becomes stable, as shown in Figure 9–21. This glitch can propagate to subsequent logic and create unnecessary switching activity, increasing power consumption. Circuits with many XOR functions, such as arithmetic circuits or cyclic redundancy check (CRC) circuits, tend to have many glitches if there are several levels of combinational logic between registers.

Figure 9–21. XOR Gate Showing Glitch at the Output



Pipelining can reduce design glitches by inserting flipflops into long combinational paths. Flipflops do not allow glitches to propagate through combinational paths. Therefore, a pipelined circuit tends to have less glitching. Pipelining has the additional benefit of generally allowing higher clock speed operations, although it does increase the latency of a circuit (in terms of the number of clock cycles to a first result). [Figure 9–22](#) shows an example where pipelining is applied to break-up a long combinational path.

Figure 9–22. Pipelining Example



Pipelining is very effective for glitch-prone arithmetic systems because it reduces switching activity, resulting in reduced power dissipation in combinational logic. Additionally, pipelining allows higher-speed

operation by reducing logic-level numbers between registers. The disadvantage of this technique is that if there are not many glitches in your design, pipelining may increase power consumption by adding unnecessary registers. Pipelining can also increase resource utilization.

Pipelining Experiment for Stratix II Devices

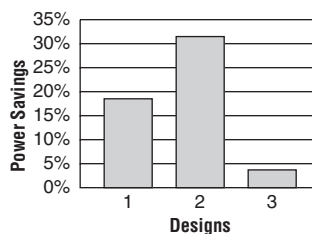
In this experiment, three designs are implemented in Stratix II devices with and without pipelining. These three designs use arithmetic heavily (based on XOR functions) that may result in significant glitching.

Table 9–10 shows the resource utilization for the designs used in the experiment.

Design Name	Pipelined	ALUTs	Registers
Multiplier (Design 1)	No	9,726	448
	Yes	9,772	1,109
Accumulator multipliers (Design 2)	No	13,719	1,120
	Yes	14,007	2,260
Fir filter (Design 3)	Yes (level 1)	1,048	949
	Yes (level 2)	932	929

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–23 shows that pipelining reduces dynamic power consumption by as much as 31% in Stratix II devices.

Figure 9–23. Power Savings Using Pipelining for Stratix II Devices



Pipelining Experiment for Cyclone II Devices

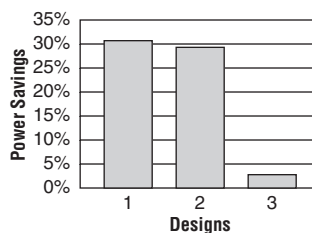
In this experiment, three designs are implemented in Cyclone II devices with and without pipelining. These three designs heavily use arithmetic (based on XOR functions) that may result in significant glitching.

Table 9–11 shows resource utilization for the designs used in the experiment.

Design Name	Pipelined	LEs	Registers
Accumulator Multipliers (Design 1)	No	6,870	320
	Yes	13,071	3,719
Adder (Design 2)	No	7,392	1,076
	Yes	7,343	752
Divider (Design 3)	No	6,659	320
	Yes	6,735	520

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–24 shows that pipelining reduces dynamic power by as much as 31% in Cyclone II devices.

Figure 9–24. Power Savings Using Pipelining for Cyclone II Devices



Architectural Optimization

You can use design-level architectural optimization by taking advantage of specific device architecture features. These features include dedicated memory and DSP or multiplier blocks available in FPGA devices to perform memory or arithmetic-related functions. You can use these

blocks in place of LUTs to reduce power consumption. For example, you can build large shift registers from RAM-based FIFO buffers instead of building the shift registers from the LE registers.

The Stratix II device family allows you to efficiently target small, medium, and large memories with the TriMatrix memory architecture. Each TriMatrix memory block is optimized for a specific function. The M512 memory blocks are useful for implementing small FIFO buffers, DSP, and clock domain transfer applications. M512 memory blocks are more power-efficient than the distributed memory structures in some competing FPGAs. The M4K memory blocks are used to implement buffers for a wide variety of applications, including processor code storage, large look-up table implementation, and large memory applications. The M-RAM blocks are useful in applications where a large volume of data must be stored on-chip. Effective utilization of these memory blocks can have a significant impact on power reduction in your design.

The Cyclone II device family has configurable M4K memory blocks that provide various memory functions such as RAM, FIFO buffers, and ROM.



For more information on using DSP and memory blocks efficiently, refer to the *Area & Timing Optimization* chapter in volume 2 of the *Quartus II Handbook*.

Architectural Optimization Experiment for Stratix II Devices

In this experiment, three designs are implemented in Stratix II devices in three ways to illustrate the power-reducing capabilities of dedicated blocks. The first two designs use logic elements and DSP blocks. The third design uses M4K and M-RAM blocks. In the third design, you can see that using MRAM blocks is more power efficient than using M4K blocks for large memory applications. The power optimization options for synthesis and fitting are turned off in this experiment.

Table 9–12 shows relative resource usage results.

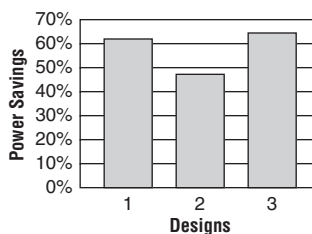
Design Name	Implementation	ALUT	Register	DSP Blocks	Memory
Design 1	Regular implementation	9,726	448	0	0
	Dedicated resource implementation	1,124	448	121	0

Table 9–12. Designs Used in the Architectural Optimization Experiment for Stratix II Devices (Part 2 of 2)

Design Name	Implementation	ALUT	Register	DSP Blocks	Memory
Design 2	Regular implementation	13,719	1,120	0	0
	Dedicated resource implementation	2,880	896	212	0
Design 3	M4K	286	228	0	1,835,008 (M4K)
	M-RAM	224	224	0	1,835,008 (M-RAM)

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–25 shows that the architectural optimization technique has power savings of over 60% in Stratix II devices.

Figure 9–25. Power Savings Using Dedicated Blocks for Stratix II Devices



Architectural Optimization Experiment for Cyclone II

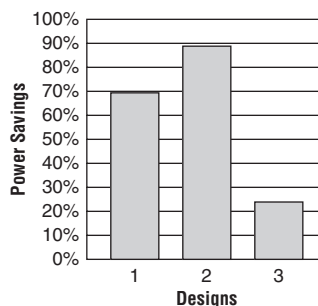
In this experiment, three designs are implemented in Cyclone II devices in three ways to illustrate the power-reducing capabilities of dedicated blocks. The first two designs use LEs and multiplier blocks. The third design uses LEs and M4K blocks.

Table 9–13 shows relative resource usage results.

Design Name	Implementation	LEs	Register	Multiplier Blocks	Memory
Design 1	Regular implementation	6,870	320	0	0
	Dedicated resource implementation	1,130	320	49	0
Design 2	Regular implementation	7,343	752	0	0
	Dedicated resource implementation	1,401	608	44	0
Design 3	Regular implementation	1,550	1,265	0	0
	M4K	72	72	0	1,152

The PowerPlay Power Analyzer estimates the power using a gate-level simulation output file. Figure 9–26 shows that the architectural optimization technique has power savings of as much as 88% in Cyclone II devices.

Figure 9–26. Power Savings Using Dedicated Blocks for Cyclone II Devices



I/O Power Guidelines

Non-terminated I/O standards such as LVTTTL and LVCMOS have a rail-to-rail output swing. The voltage difference between logic-high and logic-low signals at the output pin is equal to the V_{CCIO} supply voltage. If the capacitive loading at the output pin is known, the dynamic power consumed in the I/O buffer can be calculated as:

$$P = 0.5 \times F \times C \times V^2$$

In this equation, F is the output transition frequency and C is the total load capacitance being switched. V is equal to V_{CCIO} supply voltage. Because of the quadratic dependence on V_{CCIO} , lower voltage standards consume significantly less dynamic power. In addition, lower pin capacitance is an important factor in considering I/O power consumption. Hardware and simulation data show that Stratix II device I/O pins have half the pin capacitance of the nearest competing FPGA. Cyclone II devices exhibit 20% less I/O power consumption than competitive, low-cost, 90 nm FPGAs.

Transistor-to-transistor logic (TTL) I/O buffers consume very little static power. As a result, the total power consumed by a LVTTTL or LVCMOS output is highly dependent on load and switching frequency.

When using resistively terminated I/O standards like SSTL and HSTL, the output load voltage swings by a small amount around some bias point. The same dynamic power equation is used, where V is the actual load voltage swing. Because this is much smaller than V_{CCIO} , dynamic power is lower than for nonterminated I/O under similar conditions. These resistively terminated I/O standards dissipate significant static (frequency-independent) power, because the I/O buffer is constantly driving current into the resistive termination network. However, the lower dynamic power of these I/O standards means they often have lower total power than LVCMOS or LVTTTL for high-frequency applications. Use the lowest drive strength I/O setting that meets your speed and waveform requirements to minimize I/O power when using resistively terminated standards.

You can save a small amount of static power by connecting unused I/O banks to the lowest possible V_{CCIO} voltage of 1.2 V.

Table 9–14 shows the total supply and thermal power consumed by outputs using different I/O standards for Stratix II devices. The numbers are for an I/O pin transmitting random data clocked at 200 MHz with a 10 pF capacitive load.

Standard	Total Supply Current Drawn from V_{CCIO} Supply (mA)	Total On-Chip Thermal Power Dissipation (mW)
3.3-V LVTTTL	2.42	9.87
2.5-V LVCMOS	1.9	6.69
1.8-V LVCMOS	1.34	4.18
1.5-V LVCMOS	1.18	3.58
3.3-V PCI	2.47	10.23
SSTL-2 class I	6.07	4.42
SSTL-2 class II	10.72	5.1
SSTL-18 class I	5.33	3.28
SSTL-18 class II	8.56	4.06
HSTL-15 class I	6.06	3.49
HSTL-15 class II	11.08	4.87
HSTL-18 class I	6.87	4.09
HSTL-18 class II	12.33	5.82

For this specific configuration, non-terminated standards generally use less power, but this is not always the case. If the frequency or the capacitive load is increased, the power consumed by non-terminated outputs increases faster than the power of terminated outputs.



For more information on I/O Standards, refer to the *Selectable I/O Standards in Stratix II Devices* chapter in volume 2 of the *Stratix II Device Handbook* or the *Selectable I/O Standards in Cyclone II Devices* chapter in the *Cyclone II Device Handbook*.

When calculating I/O power, the PowerPlay Power Analyzer uses the default capacitive load set for the I/O standard in the **Capacitive Loading** tab of the **Device & Pin Options** dialog box. If **Enable Advanced I/O Timing** is turned on, I/O power is measured using an equivalent load calculated as the sum of the near capacitance, the transmission line distributed capacitance, and the far end capacitance as defined in the **Board Trace Model** tab of the **Device & Pin Options** dialog box or the

Board Trace Model view in the Pin Planner. Any other components defined in the board trace model are not taken into account for the power measurement.



For information on using Advanced I/O Timing and configuring a board trace model, refer to the *I/O Management* chapter in volume 2 of the *Quartus II Handbook*.

Power Optimization Advisor

The Quartus II software includes the Power Optimization Advisor which provides specific power optimization advice and recommendations based on the current design project settings and assignments. The advisor covers many of the suggestions listed in this chapter. The following example shows how to reduce your design power with the Power Optimization Advisor.

Power Optimization Advisor Example

After compiling your design, run the PowerPlay Power Analyzer to determine your design power and to see where power is dissipated in your design. Based on this information, you can run the power optimization advisor to implement recommendations that can reduce design power. [Figure 9-27](#) shows the Power Optimization Advisor after compiling a design that is not fully optimized for power.

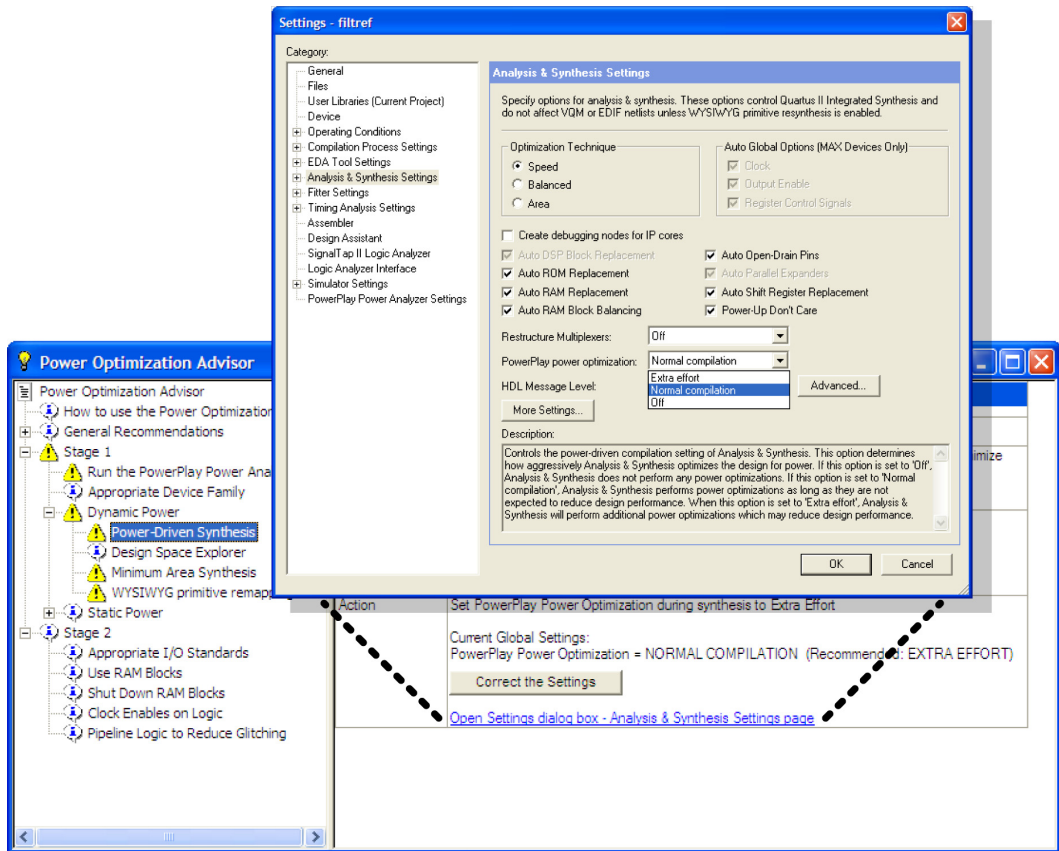
Figure 9–27. Power Optimization Advisor

Power-Driven Synthesis	
Recommendation	Set PowerPlay Power Optimization during synthesis to Extra Effort.
Description	Extra effort power-driven synthesis will choose logic and RAM implementations which minimize design dynamic power. More Info
Summary	The following areas will be affected by the recommended changes: <ul style="list-style-type: none"> · Delay may increase (fmax may decrease) · Logic element usage may increase = Compilation time is unaffected
Action	Set PowerPlay Power Optimization during synthesis to Extra Effort No action is needed for this recommendation. The recommended setting has been made. Current Global Settings: PowerPlay Power Optimization = EXTRA EFFORT (Recommended: EXTRA EFFORT) Correct the Settings Open Settings dialog box - Analysis & Synthesis Settings page

The Power Optimization Advisor shows the recommendations that can reduce power in your design. The recommendations are split into stages to show the order in which you should apply the recommended settings. The first stage shows the options that are easy to implement, as it has to do mostly with CAD settings, and are highly effective in reducing design power. An icon indicates whether each recommended setting is made in the current project. In Figure 9–27, the checkmark icon for Stage 1 shows the recommendations that are already implemented. The warning icons indicate recommendations that are not followed for this compilation. The information icon shows the general suggestions. Each recommendation includes the description, summary of the affect of the recommendation, and the action required to make the appropriate setting.

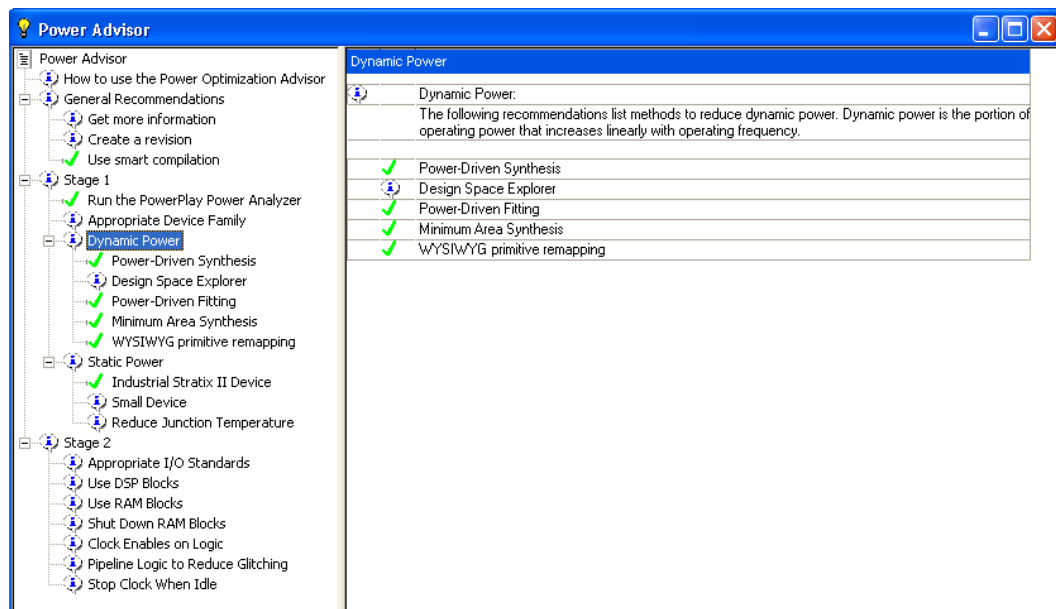
There is a link from each recommendation to the appropriate location in the Quartus II user interface where you can change the setting, such as the **Power-Driven Synthesis** setting. You can change the **Power-Driven Synthesis** setting by clicking **Open Settings dialog box - Analysis & Synthesis Settings page** (Figure 9–28). The **Setting** dialog box is shown with the **Analysis & Synthesis Settings** page selected, where you can change the **PowerPlay power optimization** settings.

Figure 9–28. Analysis & Synthesis Settings Page



After making the recommended changes, recompile your design. The Power Optimization Advisor indicates with green checkmarks that the recommendations were implemented successfully (Figure 9–29). You can use the PowerPlay Power Analyzer to verify your design power results.

Figure 9–29. Implementation of Power Optimization Advisor Recommendations



The recommendations listed in Stage 2 generally involve design changes, rather than CAD settings changes as in Stage 1. You can use these recommendations to further reduce your design power consumption. Altera recommends that you implement Stage 1 recommendations first, then the Stage 2 recommendations.

Conclusion

The combination of a smaller process technology, the use of low-k dielectric material, and reduced supply voltage, significantly reduces dynamic power consumption in the latest FPGAs. In order to further reduce your dynamic power, you should use the design recommendations presented in this chapter to optimize resource utilization and minimize power consumption.

Document Revision History

Table 9–15 shows the revision history for this document.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Updated figures to accomodate GUI changes in the software.	Added information about Stratix III support. Changes in procedures were made for Quartus II enhancements to new user functionality.
May 2006 v6.0.0	Updated for the Quartus II software version 6.0.0: <ul style="list-style-type: none"> ● Updated device support. ● Added multi-VCD/SAF support information. ● Updated achieved power reductions values. 	
October 2005 v5.1.0	Chapter 9 was formerly Chapter 7 in Volume 1: Stratix II Low Power Design Techniques.	

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. Beginning with version 6.1, the Timing Closure Floorplan Editor has also been integrated with the Chip Planner tool. This chapter explains how to use the Timing Closure Floorplan to enhance your FPGA design analysis.



For more information about the Chip Planner, refer to the *Design Analysis & Engineering Change Management with Chip Planner* chapter in volume 3 of the *Quartus II Handbook*.

Table 10–1 lists the device families supported by the Timing Closure Floorplan Editor, the Chip Planner (Timing Closure Floorplan and Chip Editor), or both.



For device families that support both the Timing Closure Floorplan and the Chip Planner, Altera recommends using the Chip Planner for design analysis.

Table 10–1. Timing Closure Floorplan Support (Part 1 of 2)

Device Family	Timing Closure Floorplan	Chip Planner (Floorplan and Chip Editor)
Stratix® III	—	✓
HardCopy® II	—	✓
Stratix	✓	✓
Stratix GX	✓	✓
Stratix II	✓	✓
Stratix II GX	✓	✓
Cyclone®	✓	✓
Cyclone II	✓	✓
MAX® II	✓	✓
MAX 7000	✓	—
ACEX®	✓	—

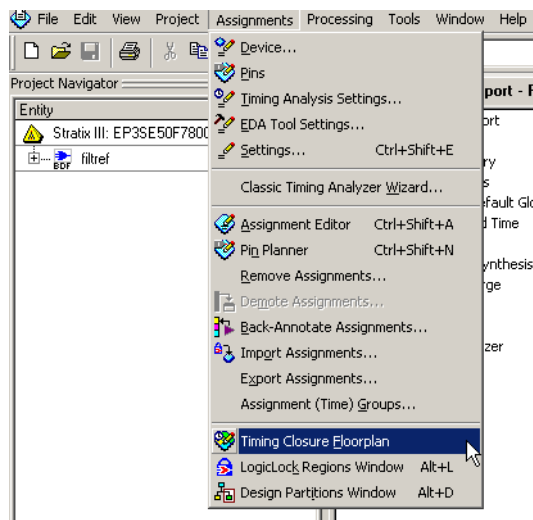
Table 10–1. Timing Closure Floorplan Support (Part 2 of 2)

Device Family	Timing Closure Floorplan	Chip Planner (Floorplan and Chip Editor)
FLEX 10K® FLEX® 10KA FLEX 10KE FLEX 6000	✓	—
APEX™ II APEX 20KC APEX 20KE	✓	—

Invoking the Timing Closure Floorplan Editor

To invoke the Timing Closure Floorplan Editor, on the Assignments menu, click **Timing Closure Floorplan** (Figure 10–1).

Figure 10–1. Invoking the Timing Closure Floorplan



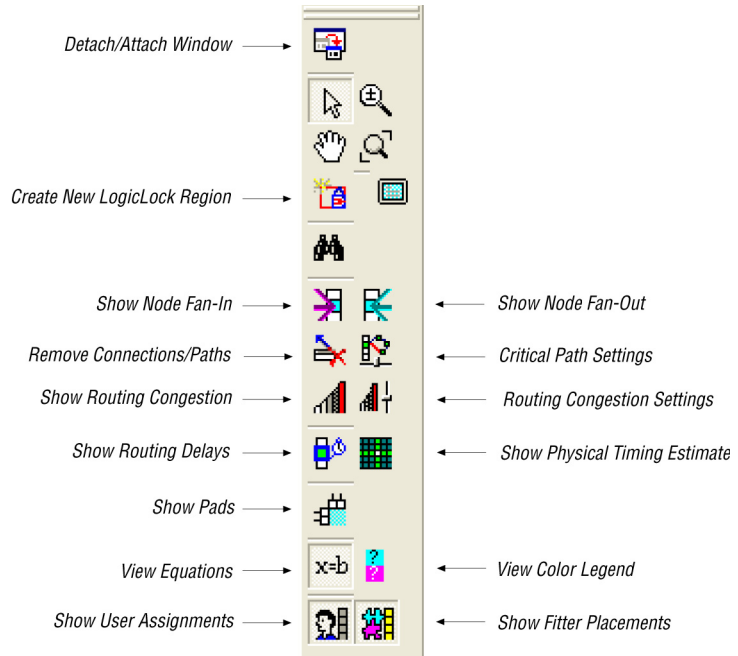
If the device in your project is a HardCopy II or Stratix III device, the following message appears: “Can’t display a floorplan: the current device family is only supported by Chip Planner.” To create and see the floorplan for these devices, you must use the Chip Planner.

You can also invoke the Timing Closure Floorplan tool by right-clicking on the following sources, pointing to **Locate**, and clicking **Locate in Timing Closure Floorplan**:

- Compilation Report
- Node Finder
- Project Navigator
- RTL source code
- RTL Viewer
- Simulation Report
- Timing Report

Figure 10–2 shows the icons in the Timing Closure Floorplan toolbar.

Figure 10–2. Timing Closure Floorplan Icons



Design Analysis Using the Timing Closure Floorplan

The Timing Closure Floorplan Editor allows you to analyze your designs visually before and after performing a full design compilation in the Quartus II software. This floorplan editor, used in conjunction with the Quartus II timing analyzer, provides a powerful method for performing design analysis.

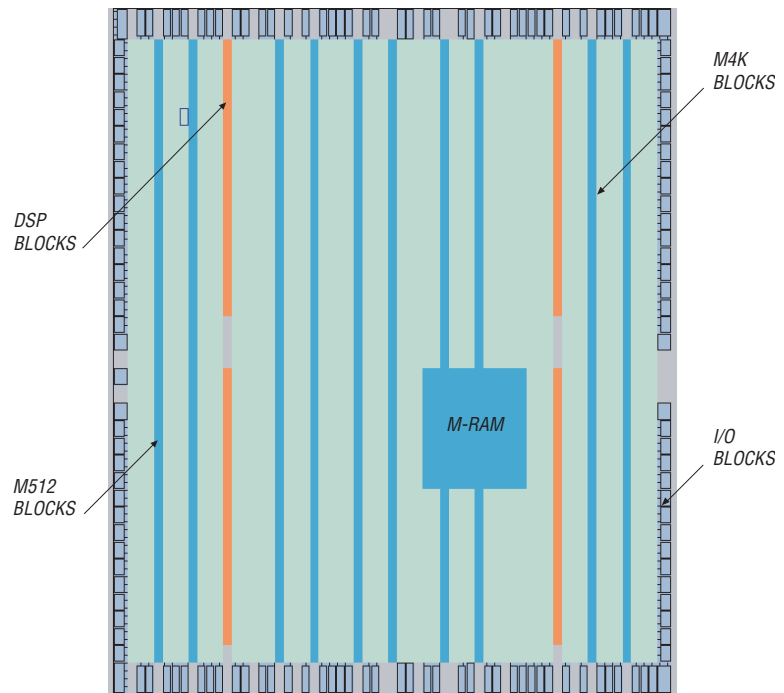
Timing Closure Floorplan Views

The Timing Closure Floorplan Editor incorporates five ways to view your design:

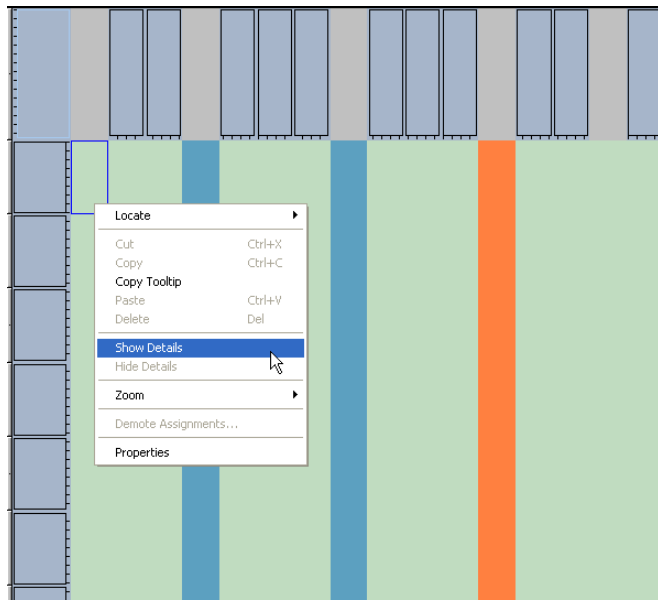
- Field view
- Interior Cells view
- Interior Labs view
- Package Top view
- Package Bottom view

Field View

The Field view provides a color-coded, high-level view of the resources used in the device floorplan. All device resources, such as embedded system blocks (ESBs) and MegaLAB blocks, are outlined. [Figure 10–3](#) shows the Field view of a Stratix II device.

Figure 10–3. Field View of a Stratix II Device

To view the details of a resource in the Field view, select the resource, right-click, and click **Show Details**. To hide the details, select all the resources, right-click, and click **Hide Details** (Figure 10–4).

Figure 10–4. Show Details & Hide Details of a Logic Array Block in Field View

Other Views

You can also view your design in the Timing Closure Floorplan Editor with the traditional Interior Cells, Interior Labs, Package Top, and Package Bottom views. Use the View menu to display the various floorplan views. The Interior Cells view provides a detailed view of device resources, including device pins and individual logic elements within a MegaLAB.

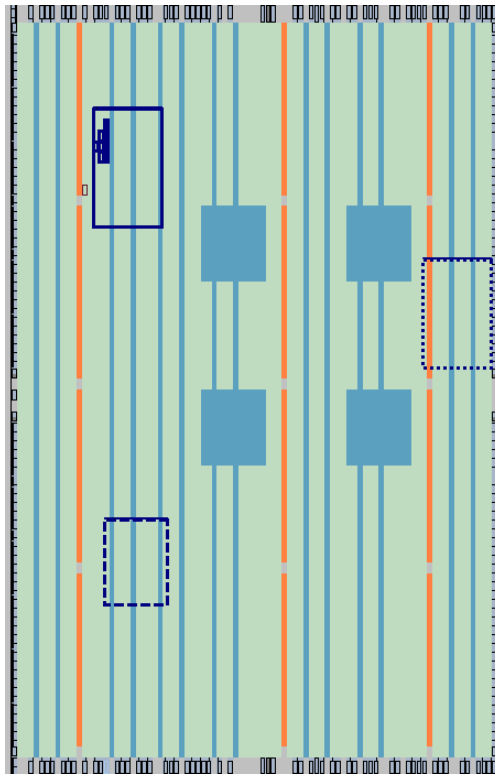
Viewing Assignments

The Timing Closure Floorplan editor differentiates between user assignments and Fitter placements. User assignments include LogicLock™ regions and are made by a user. Assignments are directives to the Quartus II Fitter for placing certain logic nodes at desired locations. They are used to create the floorplan of your device. Fitter placements are the locations where the Quartus II software places unconstrained (or unassigned) nodes during compilation. You can view both user and Fitter placements at the same time.

If the device is changed after a compilation, the user assignment and Fitter placement options cannot be used together. When this situation occurs, the Fitter placement displays the last compilation result and the user assignment displays the floorplan of the newly selected device.

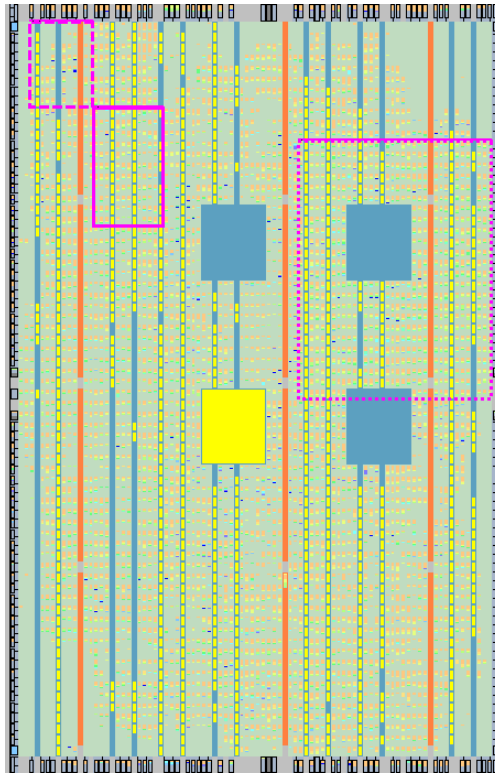
To see the user assignments, click the **Show User Assignments** icon in the Floorplan Editor toolbar, or, on the View menu, point to Assignments and click **Show User Assignments**. Figure 10-5 shows the user assignments.

Figure 10-5. User Assignments



To see the Fitter placements, click the **Show Fitter Placements** icon in the Floorplan Editor toolbar, or, on the View menu, point to Assignments and click **Show Fitter Placements**. Figure 10-6 shows the Fitter placements.

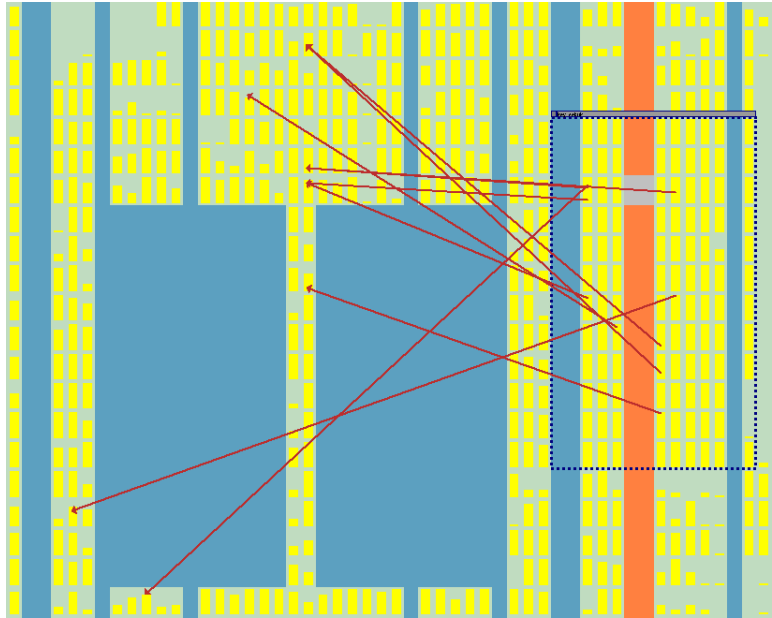
Figure 10–6. Fitter Placements



Viewing Critical Paths

The View Critical Paths feature displays routing paths in the floorplan, as shown in [Figure 10–7](#). The criticality of a path is determined by its slack and is also shown in the timing analysis report.

Figure 10–7. Critical Paths



To view critical paths in the Timing Closure Floorplan, click the **Critical Path Settings** icon on the toolbar, or, on the View menu, point to Routing and click **Critical Paths Settings**.

When viewing critical paths, you can specify the clock in the design to be viewed. You can determine which paths to display by specifying the slack threshold in the slack field.



Timing settings must be made and a timing analysis performed for paths to be displayed in the floorplan.



For more information about performing static timing analyses of your design with a timing analyzer, refer to the *Classic Timing Analyzer* and the *TimeQuest Timing Analyzer* chapters in volume 3 of the *Quartus II Handbook*.

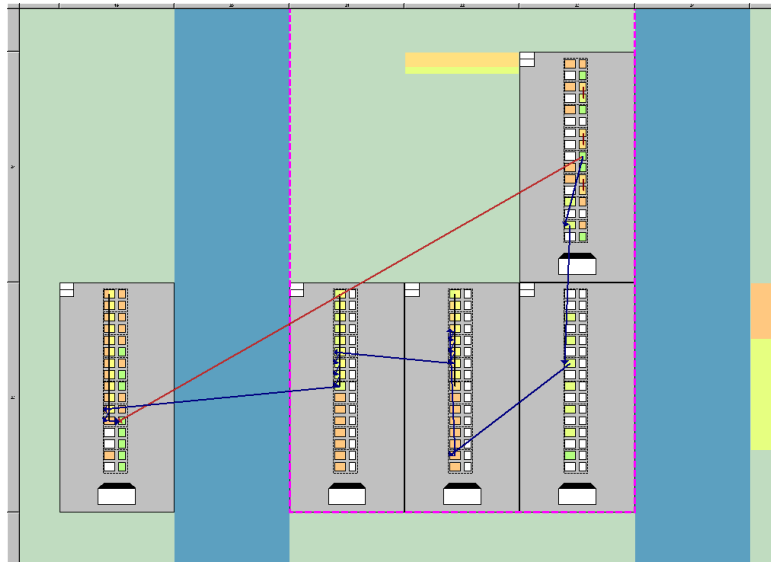
Viewing the critical paths is extremely useful for determining the criticality of nodes based on placement. There are a number of ways to view the details of the critical path.

The default view in the Timing Closure Floorplan shows the path with the source and destination registers displayed. You can also view all the combinational nodes along the worst-case path between the source and destination nodes. To view the full path, click on the delay label to select the path, right-click, and select **Show Path Edges**. Figure 10–8 shows a critical path through combinational nodes. To hide the combinational nodes, select the path, right-click, and select **Hide Path Edges**.



You must view the routing delays to select a path.

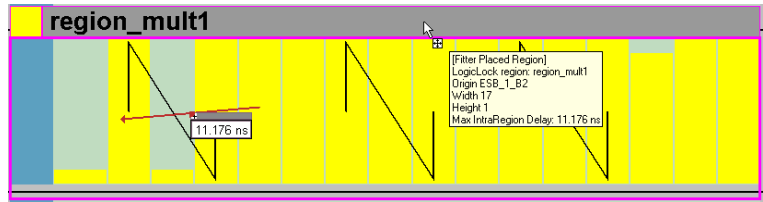
Figure 10–8. Worst-Case Combinational Paths of Critical Paths



To assign the path to a LogicLock region using 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, on the View menu, point to Routing and click **Show Intra-region Delay**. Place your cursor over a Fitter-placed LogicLock region to see the maximum delay. Figure 10–9 shows the maximum routing delay of a LogicLock region.

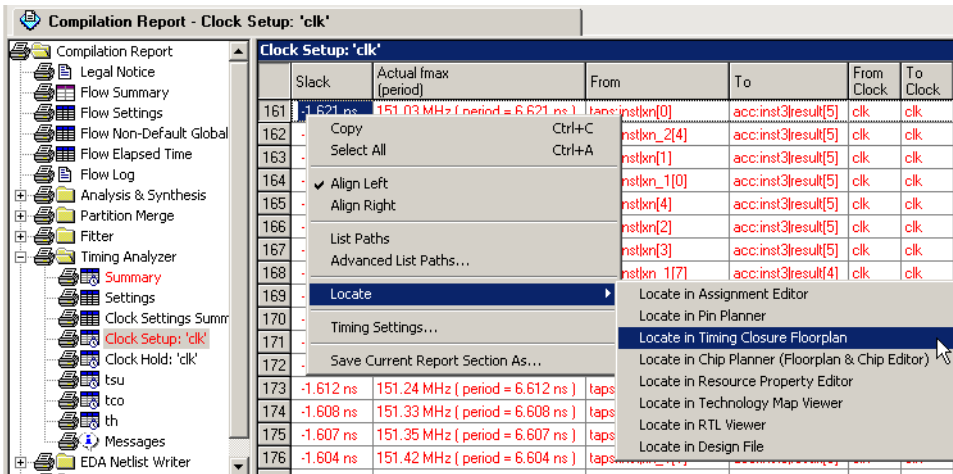
Figure 10–9. Maximum Intra-Region Delay



For more information about making path assignments with the Paths dialog box, refer to the *LogicLock Design Methodology* chapter in volume 2 of the *Quartus II Handbook*.

After running timing analysis, you can locate timing paths from the timing reports file produced. Right-click on any row in the report file, point to **Locate**, and click **Locate in Timing Closure Floorplan** (Figure 10–10). The Timing Closure Floorplan is invoked with the timing path highlighted.

Figure 10–10. Invoking the Timing Closure Floorplan from the Classic Timing Analyzer Report



For more information about viewing the critical timing path and other timing paths in the Chip Planner after running timing analysis, refer to the *Design Analysis & Engineering Change Management with Chip Planner* chapter in volume 3 of the *Quartus II Handbook*.

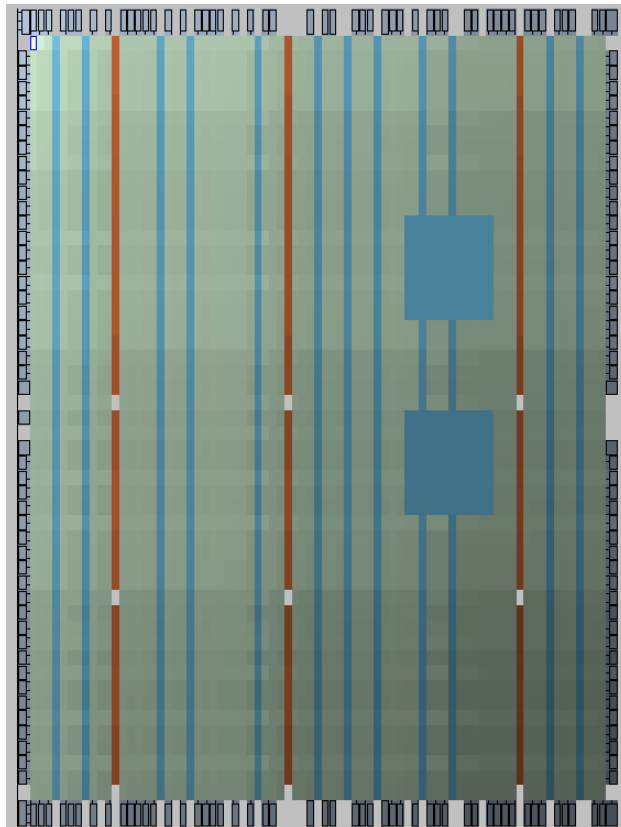


For more information about optimizing your design in the Quartus II software, refer to the *Area & Timing Optimization* chapter in volume 2 of the *Quartus II Handbook*. With the options and tools available in the Timing Closure Floorplan and the techniques described in that chapter, the Quartus II software can help you achieve timing closure in a more time-efficient manner.

Physical Timing Estimates

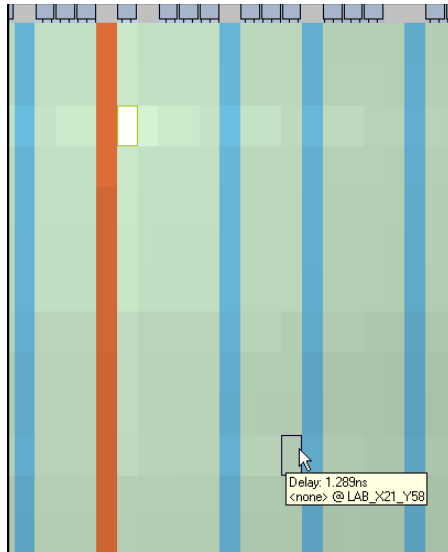
In the Timing Closure Floorplan Editor, you can select a resource and see the approximate delay to any other resource on the device. After you select a resource, the delay is represented by the color of potential destination resources. The darker the color of the resource, the longer the delay (Figure 10–11).

Figure 10–11. Physical Timing Estimates View



You can also obtain an approximation of the delay between two points by selecting a source and holding your cursor over a potential destination resource (Figure 10-12).

Figure 10-12. Delay for Physical Timing Estimate in the Timing Closure Floorplan



The delays represent an estimate based on probable best-case routing. The delay may be 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, on the View menu, point to Routing and click **Show Physical Timing Estimates**.

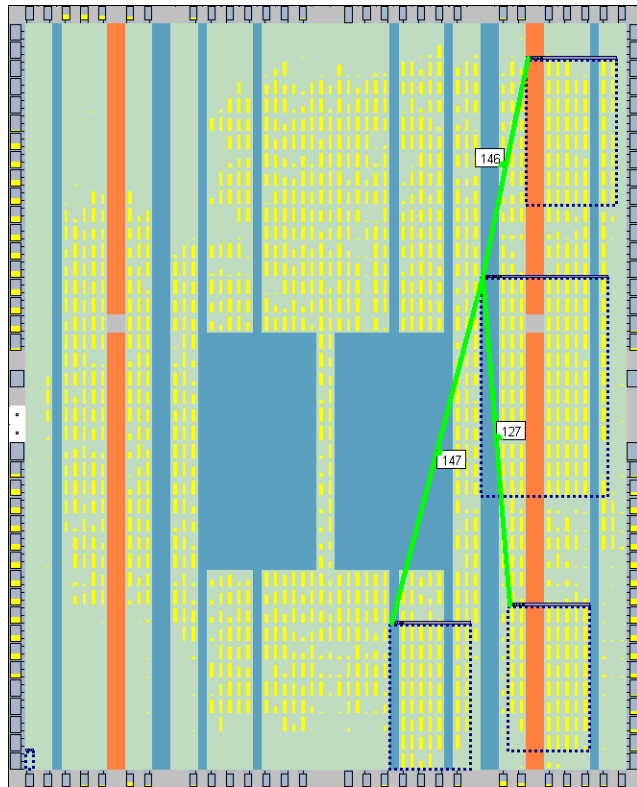
You can use the physical timing estimate information when manually placing logic in a device. This information allows you to place critical nodes and modules closer together, and non-critical or unrelated nodes and modules further apart, reducing the routing congestion between critical and non-critical entities and modules. This placement enables the Quartus II Fitter to meet the timing requirements.

LogicLock Region Connectivity

To see how logic in LogicLock regions interfaces, view the connectivity between LogicLock regions. This capability is extremely useful when entities are assigned to LogicLock regions. You can also see the fan-in and fan-out of selected LogicLock regions.

To view the connections in the timing closure floorplan, on the View menu, point to Routing and click **Show LogicLock Regions Connectivity**. Figure 10–13 shows standard LogicLock region connections.

Figure 10–13. LogicLock Region Connections with Connection Count



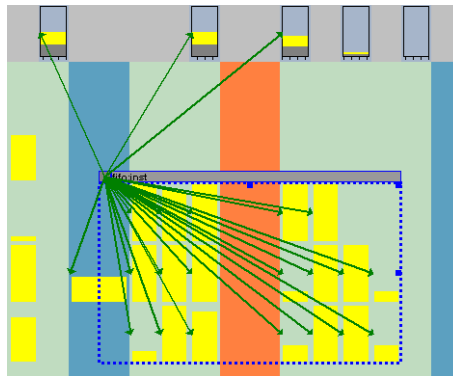
As shown in Figure 10–13, the thickness of the connection line indicates how many connections exist between regions. To see the number of connections between regions, on the View menu, point to Routing and click **Show Connection Count**.

LogicLock region connectivity is applicable only when the user assignments are enabled in the Timing Closure Floorplan. When you use floating LogicLock regions, the origin of the user-assigned region is not necessarily the same as the Fitter-placed region. You can change the origin of your floating LogicLock regions to that of the last compilation origin in the LogicLock Regions window 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 in the Timing Closure Floorplan, 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, on the View menu, point to Routing and click **Show Node Fan-In**. To set the fan-out option, click the **Show Node Fan-Out** icon, or, on the View menu, point to Routing and click **Show Node Fan-Out**. Only the nodes that have user assignments are visible when viewing fan-in or fan-out of LogicLock regions. Figure 10–14 shows the fan-out of a selected LogicLock region.

Figure 10–14. Fan-Out of a LogicLock Region



For more information about the LogicLock incremental design capability, refer to the *LogicLock Design Methodology* chapter in volume 2 of the *Quartus II Handbook*.

Viewing Routing Congestion

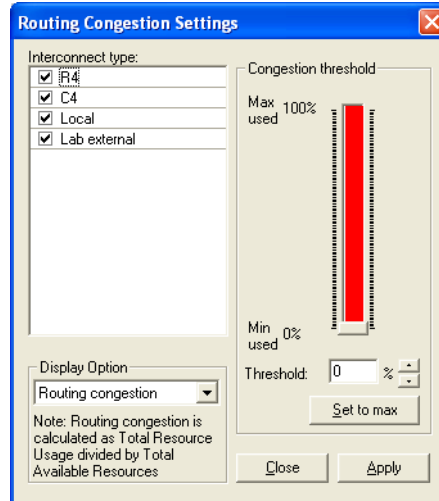
The View Routing Congestion feature allows you to determine the percentage of routing resources used after a compilation. This feature identifies where there is a lack of routing resources.

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.

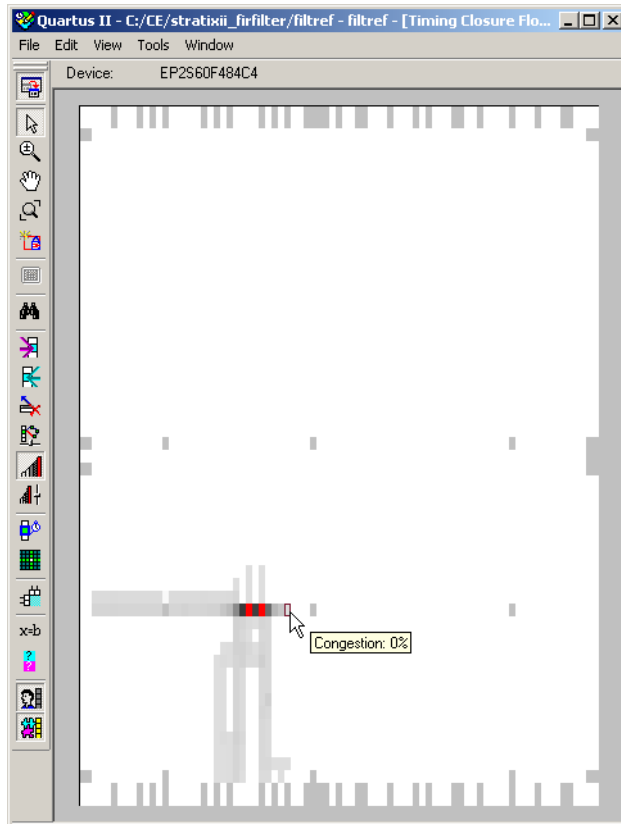
To view routing congestion in the floorplan, click the **Show Routing Congestion** icon, or, on the View menu, point to Routing and click **Show Routing Congestion**. To set the criteria for the critical path you want to view, click the **Routing Congestion Settings** icon, or, on the View menu, point to Routing and click **Routing Congestion Settings**.

In the Routing Congestion Settings dialog box, you can choose the routing resource (Interconnect type) 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 (Figure 10–15).

Figure 10–15. Routing Congestion Settings Dialog Box



If you use the routing congestion viewer to determine where there is a lack of routing resources, examine each routing resource individually to determine which ones use close to 100% of the available resources (Figure 10–16).

Figure 10–16. Routing Congestion of a Sample Design in a Stratix II Device

Conclusion

Design analysis for timing closure is a fundamental requirement for optimal performance in highly complex designs. The Quartus II Timing Closure Floorplan Editor helps you close timing quickly on complex designs.

Document Revision History

Table 10–2 shows the revision history for this document.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Updated for the Quartus II software version 6.1.0: <ul style="list-style-type: none"> ● Added Table 10–1 on page 10–1 that shows which device families support the Timing Closure Floorplan tool, the Chip Planner tool, or both ● Consolidated section on I/O timing Analysis Report File under “Viewing Critical Paths” section 	The Chip Planner integrates the Timing Closure Floorplan and Chip Editor tools into one tool. The Timing Closure Floorplan is the only floorplan tool available for some device families, as listed in Table 10–1 on page 10–1.
May 2006 v6.0.0	Updated for the Quartus II software version 6.0.0: <ul style="list-style-type: none"> ● Updated device support. 	
October 2005 v5.1.0	Chapter 10 was formerly Chapter 8 in version 5.0.	
May 2005 v5.0.0	Chapter 8 was formerly Chapter 7 in version 4.2.	
December 2004 v2.1	Updated for Quartus II software version 4.2: <ul style="list-style-type: none"> ● Removed By Delay and Show Routing Delays options from the Viewing Critical Paths segment. ● Updates to figures. 	
June 2004 v2.0	<ul style="list-style-type: none"> ● Updates to tables, figures. ● New functionality in the Quartus II software version 4.1. 	
February 2004 v1.0	Initial release.	

Introduction

The Quartus® II software offers advanced netlist optimization options, including physical synthesis, to optimize your design beyond the optimization performed in the course of the standard Quartus II compilation flow. 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. Device support for these optimizations varies; see the appropriate section for details.

Netlist optimization options work with your design's atom netlist, which describes a design in terms of Altera®-specific primitives. An atom netlist file can take the form of an Electronic Design Interchange Format file (.edf) or a Verilog Quartus Mapping file (.vqm) generated by a third-party synthesis tool, or a netlist used internally by 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. The synthesis netlist optimizations make changes to the synthesis netlist output from a third-party synthesis tool or make changes as an intermediate step in Quartus II integrated synthesis (one of the optimizations applies only to third-party synthesis netlists). These netlist changes are beneficial in terms of area or speed, depending on your selected optimization technique.

Physical synthesis optimizations take place during the fitter stage of the Quartus II compilation flow. These 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 11-3 and "[Physical Synthesis Optimizations](#)" on page 11-11 explain how the available optimizations work. This chapter also provides information about 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 the SignalTap® II logic analyzer or formal verification. If your design flow requires fixed node names, you may need to turn off the synthesis netlist optimization and physical synthesis options.

Primitive node names are specified during synthesis. When netlist optimizations are applied, node names may change as primitives are created and removed. Hardware description language (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.

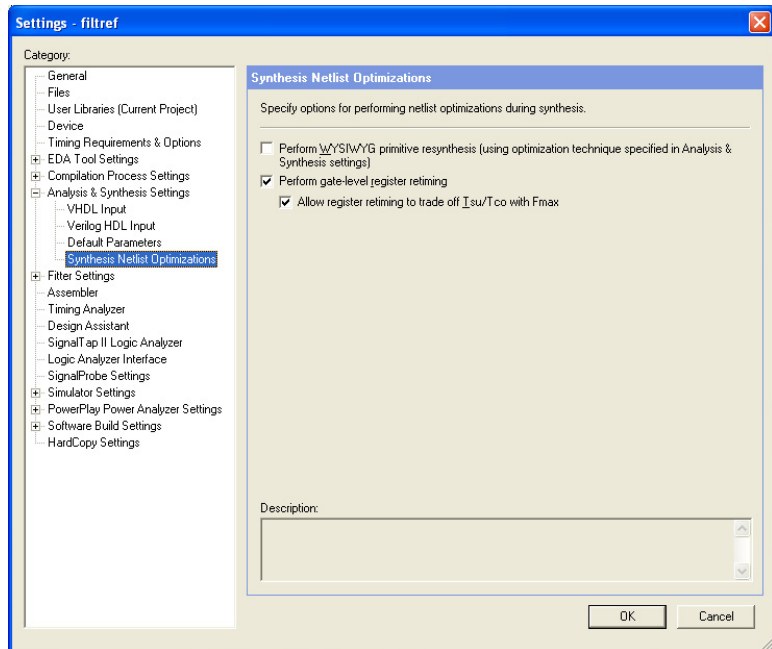


For more information about using these attributes during synthesis in the Quartus II software, refer to the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*.

Synthesis Netlist Optimizations

To view and modify the synthesis netlist optimization options, on the Assignments menu, click **Settings**. In the **Category list**, select **Analysis & Synthesis Settings**, select **Synthesis Netlist Optimizations**, and specify the options for performing netlist optimization during synthesis, as shown in [Figure 11–1](#).

Figure 11–1. Synthesis Netlist Optimizations Page



The sections [“WYSIWYG Primitive Resynthesis”](#) and [“Gate-Level Register Retiming”](#) on page 11–5 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 take the form of either an Electronic Design Interchange Format file or a Verilog Quartus Mapping file generated by a third-party synthesis tool. To select this option, on the Assignments menu, click **Settings**. In the **Category list**, select **Analysis & Synthesis Settings**, select **Synthesis Netlist**

Optimizations, and turn on **Perform WYSIWYG primitive resynthesis (using optimization technique specified in Analysis & Synthesis settings)**. If you want to perform WYSIWYG resynthesis on only a portion of your design, you can use the Assignment Editor to assign the **Perform WYSIWYG primitive resynthesis** logic option to a lower-level entity in your design. This option can be used with the HardCopy® series, Stratix® series, Cyclone™ series, MAX® II, or APEX™ series 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. This feature re-maps the design using the Optimization Technique specified for your project.

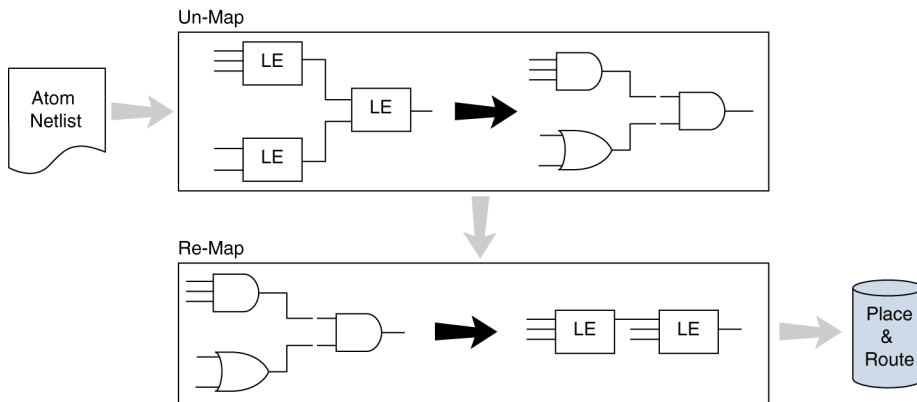
To turn on this option, on the Assignments menu, click **Settings**. In the **Category list**, select **Analysis & Synthesis Settings**. In the **Analysis & Synthesis Settings** page, under **Optimization Technique**, select **Speed**, **Area**, or **Balanced** to specify how the Quartus II technology mapper optimizes the design. The Balanced setting is the default for many Altera device families; this setting optimizes the timing critical parts of the design for speed and the rest for area.



Refer to the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook* for details on the Optimization Technique option.

Figure 11–2 shows the Quartus II software flow for this feature.

Figure 11–2. WYSIWYG Primitive Resynthesis



The **Perform WYSIWYG primitive resynthesis** option is not applicable if you are using Quartus II integrated synthesis. With the Quartus II synthesis, you do not have 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 un-maps and re-maps only logic cell, also referred to as LCELL or LE primitives, and regular I/O primitives (which may contain registers). Double data rate (DDR) 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 Verilog Quartus Mapping file or an Electronic Design Interchange Format 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 Verilog Quartus Mapping file or Electronic Design Interchange Format file from your third-party synthesis tool, because the primitives in the atom 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. To apply this logic option, on the Assignments menu, click **Assignment Editor**. This option disables WYSIWYG resynthesis for parts of your design.

Gate-Level Register Retiming

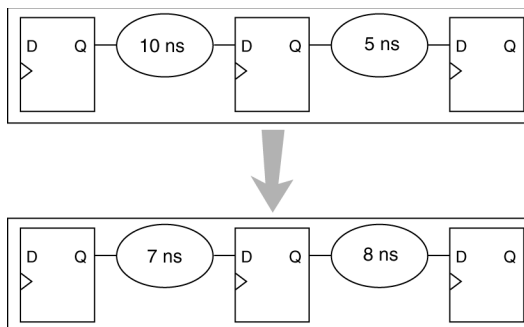
The **Perform gate-level register retiming** option enables movement of registers across combinational logic to balance timing, allowing the Quartus II software to trade off the delay between timing-critical paths and non-critical paths. See [Figure 11-3 on page 11-6](#) for an example. This option can be used with the HardCopy® series, Stratix series, Cyclone series, MAX II, and APEX series device families. To set this option, on the Assignments menu, click **Settings**. In the **Category list**, select **Analysis & Synthesis Settings**, select **Synthesis Netlist Optimizations**. In the **Synthesis Netlist Optimizations** page, turn on **Perform gate-level register retiming**.

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. To change the default setting for this option, on the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**. In the **Analysis & Synthesis Settings** page, click **More Settings**.

You can set the **Power-Up Don't Care** logic option for individual registers or entities using the Assignment Editor. You can also specify a power-up value for individual registers or entities with the **Power-Up Level** logic option. Registers that are explicitly assigned power-up values are not combined with registers that have been explicitly assigned other values.

Figure 11-3 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.

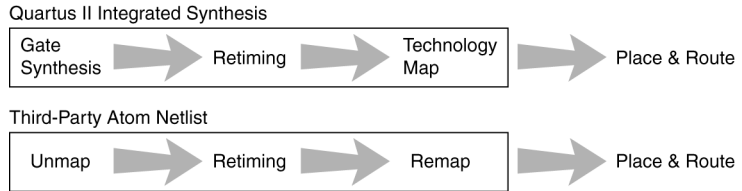
Figure 11-3. Gate-Level Register Retiming Diagram



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 11-4](#).

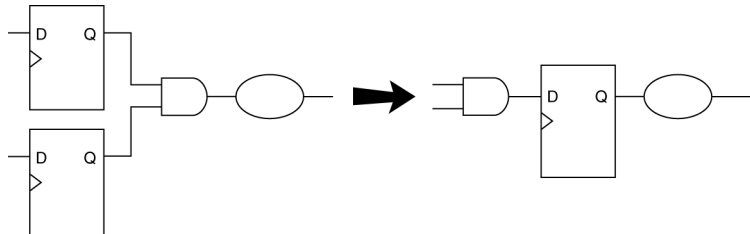
Figure 11-4. Gate-Level Synthesis



The gate-level register retiming option 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 benefit 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, as shown in [Figure 11-5](#).

Figure 11-5. Combining Registers with Register Retiming



The register retiming option 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. To access this report, on the Processing menu, click **Compilation Report**. In the **Analysis & Synthesis** list, select **Optimization Results**, select **Netlist Optimizations**, and click **Gate-level Retiming** (Figure 11-6).


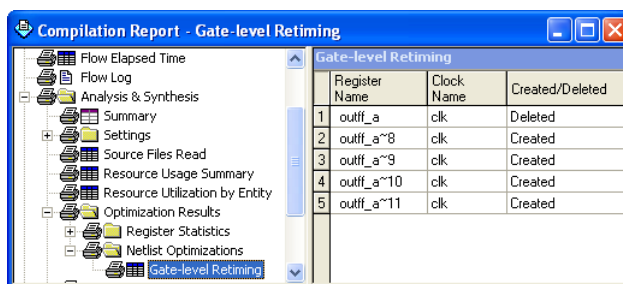
 The node names for these registers change during the retiming process.

Figure 11-6. Gate-Level Retiming Report



Register Name	Clock Name	Created/Deleted
1 outff_a	clk	Deleted
2 outff_a~8	clk	Created
3 outff_a~9	clk	Created
4 outff_a~10	clk	Created
5 outff_a~11	clk	Created

You can set the **Netlist Optimizations** logic option to **Never Allow** to prevent register movement during register retiming. This option can be applied either to individual registers or entities in the design using the Assignment Editor.

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 direct connection from input pin-to-register-to-register if both registers have the same clock and the first register does not fan out to anywhere else. These registers are considered synchronization registers.
- Both registers in a direct connection from register-to-register if both registers have the same clock, the first register does not fan out to anywhere else, and the first register is fed by another register in a different clock domain (directly or through combinational logic). These registers are considered synchronization registers.

You can change the retiming behavior for a sequence of synchronization or meta-stability registers by changing the value of the **Retiming Meta-Stability Register Sequence Length** logic option. The value of this option indicates the number of synchronization registers that will not be moved during gate-level register retiming. The default value is 2. To set the value to any number greater than 0, on the Assignments menu, click Settings. In the Settings dialog box, select **Analysis & Synthesis Settings** and click **More Settings**. A value of 1 means that any registers connected to the first register in a register-to-register connection can be moved during retiming. A value of $n > 1$ means that any registers in a sequence of length 1, 2, ... n are not moved during gate-level register retiming as long as all of the following are true:

- The first register is fed either directly by a pin or by a register in another clock domain (directly or through combinational logic)
- All registers in the sequence have the same clock
- All but the last register feed the next register in the sequence directly and do not fan out to anywhere else

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** for a given set of registers.

Allow Register Retiming to Trade-Off t_{SU}/t_{CO} with f_{MAX}

To determine whether the Quartus II compiler should attempt to increase f_{MAX} at the expense of t_{SU} or t_{CO} times, on the Assignments menu, click **Settings**. In the **Category** list, select **Analysis & Synthesis Settings**, and select **Synthesis Netlist Optimizations**. In the **Synthesis Netlist Optimizations** page, turn on **Allow register retiming to trade off T_{su}/T_{co} with F_{max}** . 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 Tsu/Tco with Fmax** 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 Synthesis Netlist Optimization Results

The Quartus II software generates the same results on every compilation for the same source code and settings on a given system. 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 some cases, if you avoid running **Analysis & Synthesis**, or **quartus_map**, and run the Fitter or another desired Quartus II executable instead, you can skip the synthesis stage of the compile.

You can use the incremental compilation feature to preserve synthesis results for a particular partition of your design by choosing a netlist type of post-synthesis.



You should use the incremental compilation flow to preserve compilation results instead of the LogicLock back-annotation flow described here.



For information about the incremental compilation design methodology, refer to the *Quartus II Incremental Compilation* chapter in volume 1 of the *Quartus II Handbook*.

If you wish, you may preserve the nodes resulting from netlist optimizations. Preserving the nodes may be required if you use the LogicLock flow to back-annotate placement and/or import one design into another. (Note that this is not needed if you use the incremental compilation design flow along with the LogicLock feature).

If you are using any Quartus II synthesis netlist optimization options, you can save your optimized results. To do so, on the Assignments menu, click **Settings**. In the **Category** list, select **Compilation Process Settings**. In the **Compilation Process Settings** page, turn on **Save a node-level netlist of the entire design into a persistent source file**. This option saves your final results as an atom-based netlist in Verilog Quartus Mapping file format. By default, the Quartus II software places the Verilog Quartus Mapping file in the **atom_netlists** directory under the current project directory. If you want to create a different Verilog Quartus Mapping file

using different Quartus II settings, on the Assignments menu, click **Settings**. In the **Category** list, select **Compilation Process Settings**. In the **Compilation Process Settings** page, change the **File name** setting.

If you are using the synthesis netlist optimizations (and not any physical synthesis optimizations), generating a Verilog Quartus Mapping file is optional. To lock down the location of all logic and device resources in the design with or without a Quartus II-generated Verilog Quartus Mapping file, on the Assignments menu, click **Back-Annotate Assignments** and specify the desired options. You should use 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 Verilog Quartus Mapping file and wish to recompile the design, use the new Verilog Quartus Mapping 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 cells 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 Stratix and Cyclone series device families, as well as with HardCopy II devices.

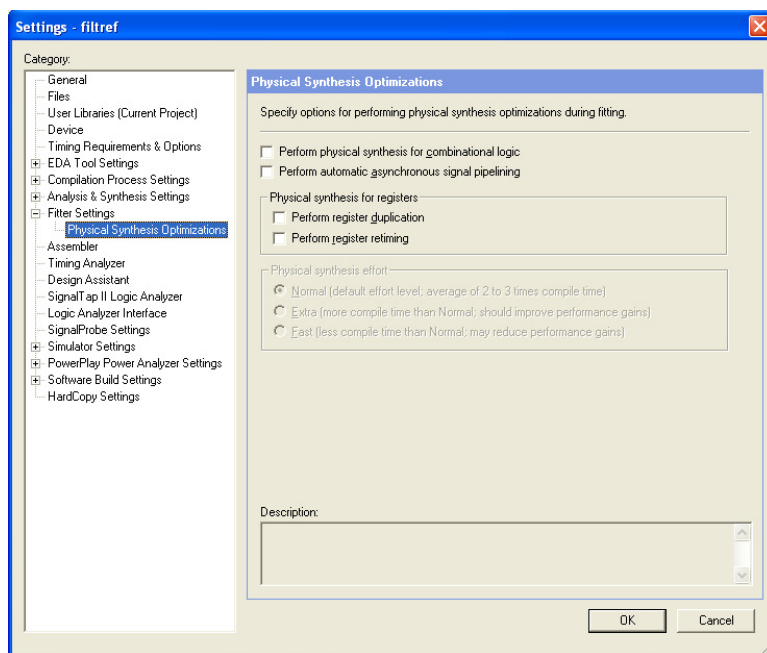
If you are migrating your design to a HardCopy II device, you can target physical synthesis optimizations to the FPGA architecture in the FPGA-first flow or to the HardCopy II architecture in the HardCopy-first flow. The optimizations are mapped to the other device architecture during the migration process. Note that you cannot target optimizations to optimize for both device architectures individually because doing so would result in a different post-fitting netlist for each device.



For more information about using physical synthesis with HardCopy devices, refer to the *Quartus II Support of HardCopy Series Devices* chapter in volume 1 of the *Quartus II Handbook*.

To view and modify the physical synthesis optimization options, on the Assignments menu, click **Settings**. In the **Category** list, select **Fitter Settings** and select **Physical Synthesis Optimizations** as shown in Figure 11-7.

Figure 11-7. Physical Synthesis Optimization Settings



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 you to keep your registers intact for formal verification or other reasons.

The following physical synthesis optimizations are available:

- Physical synthesis for combinational logic
- Automatic asynchronous signal pipelining
- 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 compilation time to try to achieve extra circuit performance, while the **Fast** effort setting uses less compilation time than **Normal** but may not achieve the same gains.

All Physical Synthesis optimizations write results to the **Netlist Optimizations** report. To access this report, on the Processing menu, click **Compilation Report**. In the **Category** list, select **Fitter** and select **Compilation Report**. This report provides a list of atom netlist files that were modified, created, and deleted during physical synthesis.

The node names for these atoms change during the physical synthesis process.

Nodes or entities that have the **Netlist Optimizations** logic option set to **Never Allow** are not affected by the physical synthesis algorithms. To access this logic option, on the Assignments menu, click **Assignment Editor**. Use this option to disable physical synthesis optimizations for parts of your design.

Automatic Asynchronous Signal Pipelining

The **Perform automatic asynchronous signal pipelining** option on the **Physical Synthesis Optimizations** page in the **Fitter Settings** section of the **Settings** dialog box allows the Quartus II fitter to perform automatic insertion of pipeline stages for asynchronous clear and asynchronous load signals during fitting when these signals negatively affect performance. You can use this option if asynchronous control signal recovery and removal times are not achieving their requirements.

This option improves performance for designs in which asynchronous signals in very fast clock domains cannot be distributed across the chip fast enough due to long global network delays. This optimization performs automatic pipelining of these signals, while attempting to minimize the total number of registers inserted.



The **Perform automatic asynchronous signal pipelining** option adds registers to nets driving the asynchronous clear or asynchronous load ports of registers. This adds register delays (adds latency) to the reset, adding the same number of register delays for each destination using the reset, changing the behavior of the signal in the design. Therefore this option should only be used when adding latency to reset signals does not violate any design requirements. This option also prevents the promotion of signals to global routing resources.

The Quartus II software performs automatic asynchronous signal pipelining only if **Recovery/Removal Analysis** is enabled. Pipelining is allowed only on asynchronous signals that have the following properties:

- The asynchronous signal is synchronized to a clock (a synchronization register drives the signal)
- The asynchronous signal fans-out only to asynchronous control ports of registers

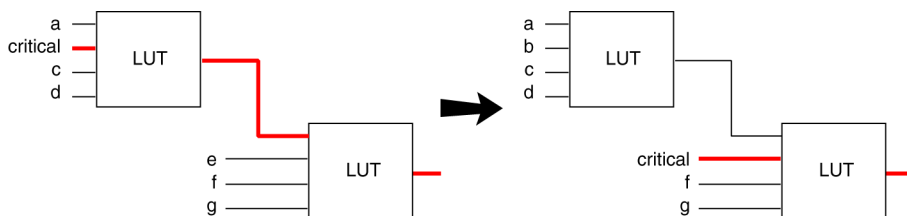
To access the **Recovery/Removal Analysis** option, on the Assignments menu, click **Settings**. In the **Category** list, select **Timing Requirements & Options**. On the Timing Requirements & Options page, click **More Settings**.

The Quartus II software does not perform automatic asynchronous signal pipelining on asynchronous signals that have the **Netlist Optimization** logic option set to **Never Allow**.

Physical Synthesis for Combinational Logic

To resynthesize the design and reduce delay along the critical path using the Quartus II fitter, on the Assignments menu, click **Settings**. In the **Category** list, select **Fitter Settings** and select **Physical Synthesis Optimizations**. In the **Physical Synthesis Optimizations** page, click **Perform physical synthesis for combinational logic**. The software can accomplish 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 11–8](#) for an example. This option also allows the duplication of LUTs to enable further optimizations on the critical path.

Figure 11–8. Physical Synthesis for Combinational Logic



In the 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 affects only 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 (IOEs) are not swapped.

The Quartus II software does not perform combinational optimization on logic cells that have the following properties:

- Are part of a chain
- Drive global signals
- Are constrained to a single logic array block (LAB) location
- Have the **Netlist Optimizations** option set to **Never Allow**

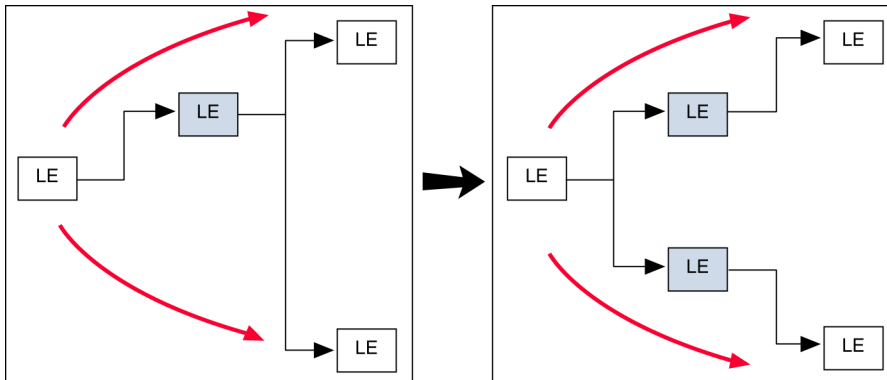
If you want to consider logic cells with any of these conditions for physical synthesis, you can override these rules by setting the **Netlist Optimizations** logic option to **Always Allow** on a given set of nodes.

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 11–9 shows an example of register duplication.

Figure 11–9. Register Duplication



The Quartus II software does not perform register duplication on logic cells that have the following properties:

- Are part of a 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 about 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 that meet 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 11–5). This option applies to the atom level (registers and combinational logic have already been placed into logic cells), and it complements 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 SERDES
- Are considered virtual I/O pins
- Registers that have the **Netlist Optimizations** logic option set to **Never Allow**



For more information about virtual I/O pins, refer to the *LogicLock Design Methodology* chapter in volume 2 of the *Quartus II Handbook*.

If you want to consider logic cells that meet 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.

Preserving Your Physical Synthesis Results

Given the same source code and settings on a given system, the Quartus II software generates the same results for 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.

You can use the incremental compilation feature to preserve fitting results for a particular partition of your design by choosing a netlist type of post-fit.



You should use the incremental compilation flow to preserve compilation results instead of the LogicLock back-annotation flow described here.



For information about the incremental compilation design methodology, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

If you wish, you can preserve the nodes resulting from physical synthesis. Preserving the nodes may be required if you use the LogicLock flow to back-annotate placement and/or import one design into another. (Note that this is not needed if you use the incremental compilation design flow along with the LogicLock feature).

If you are using any Quartus II physical synthesis optimization options, you can save the nodes in your optimized result using the **Save a node-level netlist into a persistent source file (Verilog Quartus Mapping File)** option on the **Compilation Process Settings** page in the **Settings** dialog box. This option saves your final results as an atom-based netlist in Verilog Quartus Mapping file format. By default, the Quartus II software places the Verilog Quartus Mapping file in the **atom_netlists** directory under the current project directory. If you want to create a different Verilog Quartus Mapping file using different Quartus II settings, you may do so by changing the **File name** setting on the **Compilation Process Settings** page in the **Settings** dialog box.

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, a Verilog Quartus Mapping file 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. To access this option, on the Assignments menu, click **Back-Annotate Assignments**.

You should not use a Quartus II-generated Verilog Quartus Mapping file 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 Verilog Quartus Mapping file.

To back-annotate logic locations for a design that was compiled with physical synthesis optimizations, first create a Verilog Quartus Mapping file. When recompiling the design with the hard logic location assignments, use the new Verilog Quartus Mapping file as the input source file and turn off the physical synthesis optimizations for the new compilation.

If you are importing a Verilog Quartus Mapping file 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 invalid.



You should use the incremental compilation flow to preserve compilation results instead of using logic back-annotation.

Applying Netlist Optimization Options

Netlist optimization 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 the SignalTap II logic analyzer or formal verification that requires fixed or known node names. On average, 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 also 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 you are using a third-party atom netlist (Verilog Quartus Mapping file or Electronic Design Interchange Format file), the **WYSIWYG Primitive Resynthesis** option must be turned on in order to use the **Gate-level Register Retiming** option.

The Design Space Explorer (DSE) tool command language (Tcl)/Tk script is provided with the Quartus II software to automate the application of various sets of netlist optimization options.



For more information about using the DSE script to run multiple compilations, refer to the *Design Space Explorer* chapter in volume 2 of the *Quartus II Handbook*. For information about typical performance results using combinations of netlist optimization options and other optimization techniques, refer to the *Area & Timing Optimization* 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. 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 ←
```

The *Scripting Reference Manual* includes the same information in PDF form.



For more information about Tcl scripting, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. Refer to the *Quartus II Settings File Reference Manual* for information about all settings and constraints in the Quartus II software. For more information about command-line scripting, refer to the *Command-Line Scripting* chapter 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 11–1 lists the Quartus II Settings File (.qsf) variable name and applicable values for the settings discussed in “[Synthesis Netlist Optimizations](#)” on page 11–3. The Quartus II Settings File 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 11–1. Synthesis Netlist Optimizations & Associated Settings

Setting Name	Quartus II Settings File 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 Fmax	ADV_NETLIST_OPT_RETIME_CORE_AND_IO	ON, OFF	Global
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 11–2 lists the Quartus II Settings File variable name and applicable values for the settings discussed in “Physical Synthesis Optimizations” on page 11–11. The Quartus II Settings File 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 11–2. Physical Synthesis Optimizations & Associated Settings (Part 1 of 2)

Setting Name	Quartus II Settings File Variable Name	Values	Type
Physical Synthesis for Combinational Logic	PHYSICAL_SYNTHESIS_COMBO_LOGIC	ON, OFF	Global
Automatic Asynchronous Signal Pipelining	PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING	ON, OFF	Global

Table 11–2. Physical Synthesis Optimizations & Associated Settings (Part 2 of 2)

Setting Name	Quartus II Settings File Variable Name	Values	Type
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, Instance
Power-Up Level	POWER_UP_LEVEL	HIGH, LOW	Instance
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>	

Incremental Compilation

For information about scripting and command line usage for incremental compilation as mentioned in [“Preserving Synthesis Netlist Optimization Results”](#) on page 11–10 or [“Preserving Your Physical Synthesis Results”](#) on page 11–17, refer to the *Quartus II Incremental Compilation* chapter in volume 1 of the *Quartus II Handbook*.

Back-Annotating Assignments

You can 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 about back-annotating assignments, see [“Preserving Synthesis Netlist Optimization Results”](#) on page 11–10 or [“Preserving Your Physical Synthesis Results”](#) on page 11–17.

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.

Document Revision History

Table 11–3 shows the revision history for this document.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history for document.	
May 2006 v6.0.0	Minor updates for the Quartus II software version 6.0.0.	
October 2005 v5.1.0	Chapter 11 was formerly Chapter 9 in version 5.0.	
May 2005 v5.0.0	Chapter 9 was formerly Chapter 8 in version 4.2.	
Dec. 2004 v2.1	Updated for Quartus II software version 4.2: <ul style="list-style-type: none"> ● General formatting and editing updates. ● Additional description about fixed and primitive node names for synthesis netlist optimization and physical synthesis options. ● Updates to figures. ● Clarified APEX support. ● Added information about node name changes for atoms during physical synthesis. ● Deleted Physical Synthesis Report section. 	
June 2004 v2.0	<ul style="list-style-type: none"> ● Updates to tables and figures. ● New functionality in the Quartus II software version 4.1. 	
Feb. 2004 v1.0	Initial release.	

Introduction

The Quartus® II software includes many advanced optimization algorithms to help you achieve timing closure and reduce dynamic power. The various settings and parameters control the behavior of the algorithms. These options provide complete control over the Quartus II software optimization and power techniques.

Each FPGA design is unique. There is no standard set of options that always results in the best performance or power utilization. Each design requires a unique set of options to achieve optimal performance. This chapter describes the Design Space Explorer (DSE), a utility written in Tcl/Tk that automates 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 explains the concepts and terminology used by DSE.

Exploration Space & Exploration Point

Before DSE explores a design, DSE creates an exploration space, which consists of Synthesis and Fitter settings available in the Quartus II software. Each group of settings in an exploration space is referred to as a *point*. An exploration space contains one or more points. DSE traverses the points in the exploration space to determine optimal settings for your design.

Seed & Seed Sweeping

The Quartus II Fitter uses a seed to specify the starting value that randomly determines the initial placement for the current design. The seed value can be any non-negative integer value. Changing the starting value may or may not produce better fitting. However, varying the value of the seed or seed sweeping allows the Quartus II software to determine an optimal value for the current design.

DSE extends Fitter seed sweeping in exploration spaces by providing a method for sweeping through general compilation and Fitter parameters to find the best options for your design. You can run DSE in various exploration space modes, ranging from an exhaustive try-all-options-and-values mode to a mode that focuses on one parameter.

DSE Exploration

DSE compares all exploration point results with the results of a base compilation, generated from the initial settings that you specify in the original Quartus II project files. As DSE traverses all points in the exploration space, all settings, not explicitly modified by DSE, default to the base compilation setting. For example, if an exploration point turns on register retiming but does not modify the register packing setting, the register packing setting defaults to the value you specified in the base compilation.



DSE performs the base compilation with the settings you specified in the original Quartus II project. These settings are restored after DSE traverses all points in the exploration space.

General Description

You can use DSE in either the graphical user interface (GUI) or from a command line. To run DSE with the GUI, either click **Design Space Explorer** on the Tools menu in the Quartus II software, or at the command prompt, type:

```
quartus_sh --dse ←
```

To run DSE from a command line, type the following command at the command prompt:

```
quartus_sh --dse -nogui [<options>] ←
```

You can run DSE with the following options:

```
-archive
  -concurrent-compiles [0..6]
  -custom-file <filename>
  -decision-column <"column name">
  -exploration-space <"space">
  -ignore-failed-base
  -ignore-signalprobe
  -ignore-signaltap
  -llr-restructuring
  -lower-priority
  -lsf-queue <queue name>
  -nogui
  -optimization-goal <"goal">
  -project <project name>
  -revision <revision name>
  -run-power
  -search-method <"method">
  -seeds <seed list>
  -skip-base
  -slaves <"slave list">
  -stop-after-time <dd:hh:mm>
  -stop-after-zero-failing-paths
  -use-lsf
```

The DSE script is in the default Quartus II software installation in *<Quartus II installation directory>/common/tcl/apps/dse/dse.tcl* on the PC, Solaris, HP-UX, and Linux platforms. You can launch DSE using one of the following methods:

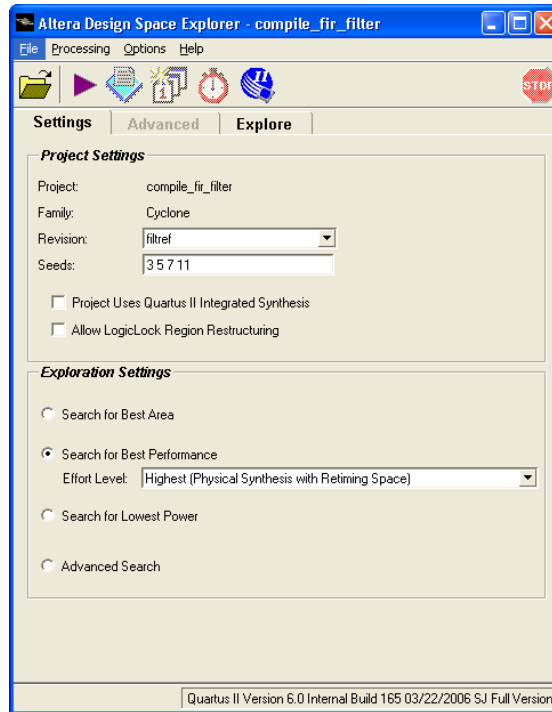
- On the Tools menu, click **Launch Design Space Explorer**.
- On Windows, select **Start > Programs > Altera > Design Space Explorer** or **Quartus II <version number>**.



For more information on DSE, launch the DSE GUI. On the Help menu, click **Contents** or press the **F1** key.

Figure 12–1 shows the DSE user interface. The **Settings** tab is divided into two sections: **Project Settings** and **Exploration Settings**.

Figure 12–1. DSE User Interface



Timing Analyzer Support

DSE supports both the Classic Timing Analyzer and the TimeQuest Timing Analyzer. You must set the timing analyzer prior to opening the project in DSE. Once the timing analyzer is set, DSE performs the design exploration with the selected timing analyzer that guides the fitter.



TimeQuest is launched directly from DSE if you set the default timing analyzer to TimeQuest.

DSE Flow

You can run DSE at any point in the design process. However, Altera recommends that you run DSE late in your design cycle when you are focusing on optimizing performance and power. The results gained from different combinations of optimization options may not persist over large changes in a design. Running DSE in signature mode (refer to “[Signature Mode](#)” on page 12–13) at the midpoint of your design cycle shows you the affect of various parameters such as the register packing logic option on your design.

DSE runs the Quartus II software for every compilation specified in the **Exploration Settings** options. DSE selectively determines the best settings for your design based on the **Optimization Goal** selected for the exploration. The Quartus II software always attempts to achieve all your timing requirements regardless of the Optimization Goal set in DSE. The Optimization Goal changes the metrics that DSE evaluates to determine if one compilation is better than another. Design Space Explorer does not change the behavior of the Quartus II software.

DSE reports the compilation that has the smallest slack. Specifying all timing requirements before you use DSE to explore your design is very important to ensure that DSE finds the optimal set of parameters for your design based on design criteria you set in your initial design.

You can change the initial placement configuration used by the Quartus II Fitter by varying the **Fitter Seed** value. You can enter seed values in the **Seeds** field of the DSE user interface.



You can set the seed value on the Assignments menu, click **Fitter Settings** in the **Settings** dialog box.

Compilation time increases as DSE exploration spaces become more comprehensive. Increased compilation time results from running several compilations and comparing the generated results with the original base 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

DSE setting support varies across device families. To see the range of settings DES supports, click the **Advanced Search** radio button on the **Settings** tab, then select the **Advanced** tab to access the settings listed in the following categories:

- Exploration Space
- Optimization Goal
- Search Method

The following device families support all **Advanced** setting types:

- Stratix® III
- Stratix II
- Stratix
- Stratix GX
- Cyclone™ II
- Cyclone
- MAX® II

The following device families support only the **Advanced Exploration Space** and **Optimization Goal** settings shown in [Table 12–1](#):

- APEX™ 20K
- APEX 20KC
- APEX 20KE
- APEX II
- FLEX® 10K
- FLEX 10KA
- FLEX 10KE

Click the **Advanced Search** radio button on the **Settings** tab before you select the **Advanced** tab to access the settings in [Table 12–1](#).

Table 12–1. Advanced Exploration Space Support for APEX 20K, APEX II & FLEX 10K Devices

Seed sweep	Area optimization space
Signature fitting effort level	Extra effort space
Extra effort for Quartus II Integrated Synthesis Projects	Custom space

DSE Project Settings

This section provides the following information about DSE project settings:

- Setting up the DSE work environment
- Specifying the revision
- Setting the initial seed
- Quartus II integrated synthesis
- Restructuring LogicLock regions

Setting Up the DSE Work Environment

From the DSE user interface, you can open a Quartus II project for a design exploration with either of the following actions:

- On the File menu, click **Open Project** and browse to your project.
- Use the **Open** icon to open a project.

Specifying the Revision

You can specify the revision to be explored with the **Revision** field in the DSE user interface. The **Revision** field is populated after the Quartus II project has been opened.



If no revisions were 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*.

Setting the Initial Seed

To specify the seed that DSE uses for an exploration, specify a non-negative integer value in the **Seed** box under **Project Settings** on the **Settings** tab. The seed value determines your design's initial placement in a Quartus II compilation.

To specify a range of seeds, type the low end of the range followed by a hyphen, followed by the high end of the range. For example, 2-5-DSE uses every seed in the range.

Restructuring LogicLock Regions

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. In addition, DSE can remove LogicLock regions that negatively affect the performance of the design.

Use the **Exploration Settings** list to select the type of exploration to perform: **Search for Best Area**, **Search for Best Performance**, **Search for Lowest Power**, or **Advanced Search**.



The “[Exploration Space](#)” on page 12–10 describes the type of explorations you can perform.

Search for Best Performance, Search for Best Area Options, or Search for Lowest Power Option

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 increases, causing compilation time to increase.

The **Search for Lowest Power** option uses a predefined exploration space that targets overall power improvements for your design. When **Search for Lowest Power** is selected, DSE automatically runs the PowerPlay Power Analyzer for each point in the space. You must ensure that the PowerPlay Power Analyzer is configured correctly to ensure accurate results. DSE issues a warning if the confidence level for any power estimate is low.

The **Search for Best Area** option uses a predefined exploration space that targets device utilization improvements for your design.

Advanced Search Option

The **Advanced Search** option provides full control over the exploration space, the optimization goal for your design, and the search method used in a design exploration. Refer to “[Performing an Advanced Search in Design Space Explorer](#)” on page 12–9 for detailed information on how to set up and perform an **Advanced Search** in DSE.



You can use **Advanced Search** to define exploration spaces that are equivalent to the **Search for Best Area**, **Search for Lowest Power**, and **Search for Best Performance** options.

Quartus II Integrated Synthesis

The **Project Uses Quartus II Integrated Synthesis** option works only for designs that have been synthesized with Quartus II integrated synthesis. With this option turned on, DSE explores options that affect the synthesis stage of compilation.



For more information on integrated synthesis options, refer to the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*.

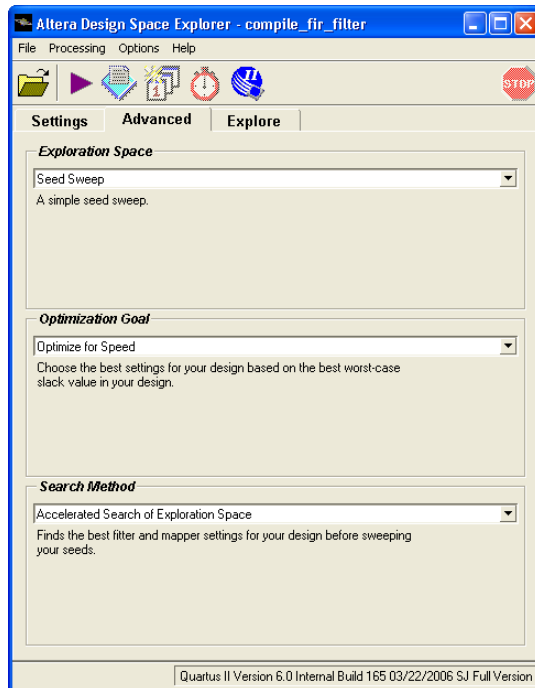
Performing an Advanced Search in Design Space Explorer

You must make three exploration settings in the **Advanced Search** dialog box before exploring a design. These three settings, **Exploration Space**, **Optimization Goal**, and **Search Method**, are described in the following sections. [Figure 12–2](#) shows the **Advanced Search** dialog box.



You can access the **Advanced** tab only after you open a Quartus II project in DSE and select **Advanced Search** on the **Settings** tab.

Figure 12–2. DSE Advanced Search Dialog Box



Exploration Space

The **Exploration Space** list controls the types of explorations that DSE performs on your design. DSE traverses the points in the exploration space, applying the settings to the design and comparing compilation results to determine the best settings for your design. DSE offers the following exploration space types:

- Seed Sweep
- Extra Effort Spaces
- Physical Synthesis Spaces
- Retiming Spaces
- Area Optimization Space
- Custom Space
- Signature mode—Power Optimization Spaces



Not all **Advanced** exploration space types are available for every device family. Refer to [“DSE Support for Altera Device Families” on page 12-6](#) for **Advanced** exploration space support for various device families.

Compilation time increases proportionally to the breadth of the explorations. The exploration space compilation time increases with the number and type of exploration spaces DSE explores, especially with exploration space types that have more optimization options and parameters.

On the Options menu, click **Advanced**, and turn on **Save Exploration Space to File** to save an XML file representing the exploration space. DSE writes the exploration space 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, refer to [“Creating Custom Spaces for DSE” on page 12-21](#).

Seed Sweep

Enter the seed values in the **Seeds** field in the DSE user interface. There are no “magic” seeds. The variation between seeds is truly random, any non-negative 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 change your netlist.



The **Seeds** field accepts individual seed values, for example, 2, 3, 4, and 5, or seed ranges, for example, 2-5.

Compilation time increases 1× for every seed value you specify. For example, if you enter five seeds, the compilation time increases to 5× the initial compilation time.

Extra Effort Spaces

The **Extra Effort Space** exploration space adds the **Register Packing** option to the exploration space done by the **Seed Sweep**. The **Extra Effort Space** exploration space also increases the Quartus II Fitter effort during placement and routing. However, the **Extra Effort Space** exploration space does not change your netlist.

Physical Synthesis Spaces

The **Physical Synthesis Space** exploration space adds physical synthesis options such as register retiming and physical synthesis for combinational logic to the options included in the **Extra Effort Space** exploration space. These netlist optimizations move registers in your design. Look-up tables (LUTs) are modified by these options. However, the design behavior is not affected by these options.



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

The **Physical Synthesis for Quartus II 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 II Integrated Synthesis Projects** exploration space works only for designs that have been synthesized using Quartus II integrated synthesis software.

Retiming Space

The **Physical Synthesis with Retiming Space** exploration space includes all the options in the **Physical Synthesis Space** exploration space and explores register retiming. Register retiming can move registers in your design.

The **Physical Synthesis Retiming Space for Quartus II Integrated Synthesis Projects** exploration space includes all the options in **Physical Synthesis with Retiming Space** exploration space, and also explores various Quartus II integrated synthesis optimization options. The **Physical Synthesis with Retiming Space for Quartus II Integrated Synthesis Projects** exploration space works only for designs that have been synthesized using the Quartus II integrated synthesis.

Area Optimization Space

The **Area Optimization Space** exploration space explores options that affect logic cell utilization for your design. These options include register packing and **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 space 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 a summary of the settings adjusted by each exploration space, refer to [Table 12-2](#).

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 12-2:

(1) For exploration spaces that includes Quartus II Integrated Synthesis Projects, DSE increases the synthesis effort.

For more information about using custom exploration spaces with DSE, refer to [“Creating Custom Spaces for DSE”](#) on page 12-21.

Signature Mode

In **Signature** mode, DSE analyzes the f_{MAX} , slack, compilation time, and area trade-offs of a single parameter. Running the single parameter over multiple seeds, DSE reports the average of the resulting 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 in exploring your design, whether the metric is performance, logic utilization, or a combination of both. These metrics allow you to determine which compilation is best, based on the design requirements. By specifying options in the **Optimization Goal** settings, you specify your optimization design goals. DSE then uses the **Optimization Goal** settings to determine the best compilation results. [Table 12–3](#) summarizes the six available optimization settings.

Table 12–3. Optimization Goal Settings

Setting	Description
Optimize for Speed	The exploration point containing the smallest worst-case slack value is selected as the best run.
Optimize for Area	The exploration point containing the lowest logic cell count is selected as the best run
Optimize for Power	The exploration point containing the lowest thermal power dissipation, and, if possible, a positive worst-case slack value, is selected as the best run.
Optimize for Negative Slack and Failing Path	The exploration point containing the best average negative worst-case slack and lowest number of failing paths is selected as the best run.
Optimize for Average Period	The exploration point containing the highest average period in a multiclock design is selected as the best run.
Optimize for Quality Fit	The exploration point containing the highest quality of fit is selected as the best run.

Quality of Fit (QoF)

Quality of Fit (QoF) is a better evaluation of fit than traditional worst-case slack metrics, because QoF considers all timing domains. QoF is not susceptible to the common mistake of accepting a fit because it has marginally better worst-case slack than other marginal timing domains with much worse slack. For example, the traditional worst-case slack metric favors a fit that achieves -2 ns slack for clock A and -5 ns slack for clock B, over a fit that achieves 1 ns slack for clock A and -5.5 ns slack for clock B. By applying a piece-wise linear function to each domain slack value, QoF ensures that large improvements in domains with ample slack do not unnecessarily skew the overall quality assessment of the fit.

To achieve a representative QoF value, ensure that slack values from domains that are easily meeting timing requirements do not offset the slack values from domains that are marginally meeting timing requirements. To correlate these values correctly, DSE applies a piece-wise linear function to the individual slack values before they are added together. This function reduces the improvement per unit of additional slack in a domain, as the domain slack improves. For example, the improvement of 100 ps in a domain that begins with 0 ns of slack is weighted more significantly than a 100 ps improvement in a domain that begins with 10 ns of slack.

To calculate the QoF for a design, use the sum of worst- case slack values for all timing domains reported by timing analysis. Timing domains include: Clock Setup, Clock Hold, t_{SU} , t_{CO} , t_{PD} , t_H , $\min t_{CO}$, $\min t_{PD}$, and other timing parameters. For example, if clock A has a Clock Setup slack of -500 ps, and clock B has a Clock Setup slack of 200 ps, the QoF for these two domains is -700 ps. The higher the QoF value reported, the better the QoF.

The QoF can be calculated for every design by entering the following Tcl command in the Tcl console:

```
source [file join $::quartus(binpath) tcl_scripts dse  
calculate_quality_of_fit.tcl]
```



All variables in the above statement are predefined; type the statement as shown without any variable substitution.

Search Method

The **Search Method** setting allows you to control the breadth of the search that DSE performs. DSE provides two search methods: **Exhaustive search of exploration space** and **Accelerated search of exploration space**. These search methods are described in [Table 12–4](#).

Search Method	Description
Exhaustive search of exploration space	Applies all settings available in the exploration space to all seeds specified. This search method yields optimal settings for your design, but this search requires the most time.
Accelerated search of exploration space	Finds the best exploration space for your design by first determining the best settings and then sweeping the settings across all seeds specified.

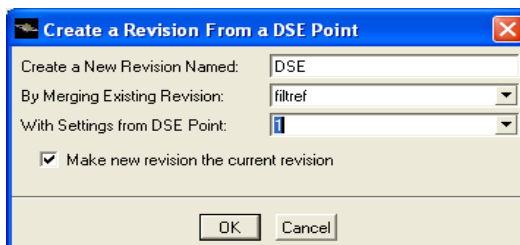
DSE Flow Options

You can control the configuration of DSE with the following options:

- Create a Revision from a DSE Point
- Stop If Zero Failing Paths are Achieved
- Continue Exploration Even If Base Compilation Fails
- Run Quartus II PowerPlay Power Analyzer During Exploration
- Archive All Compilations
- Stop Flow After Time
- Save Exploration Space to File
- Ignore SignalTap & SignalProbe Settings
- Skip Base Analysis & Compilation If Possible
- Lower Priority of Compilation Threads
- DSE Configuration File

Create a Revision from a DSE Point

After you have performed a design exploration with DSE, a Quartus II revision can be made from any exploration point. This option facilitates the creation of multiple revisions based on the same space point for further optimization within the Quartus II software. [Figure 12–3](#) shows the **Create a Revision From a DSE Point** dialog box.

Figure 12–3. Create a Revision from a DSE Point


The criteria DSE uses to determine the best space point in an exploration is known as the **Decision** column. As DSE explores a design space, the best exploration point changes according to the following inequality:

$$\langle \text{Current Decision Column Value} \rangle > \langle \text{Previous Decision Column Value} \rangle$$

By default, DSE uses worst-case slack as the **Decision** column for an exploration. The worst-case slack **Decision** column is the greatest slack value in an exploration, which can be I/O timing or clock slack values. You can change the **Decision** column on the Options menu. On the Options menu, click **Advanced**, and select **Change Decision Column**. [Table 12–5](#) lists the available **Decision** columns. The **Decision** column can be any column within the Quartus II Timing Analyzer Report.

Table 12–5. DSE Change Decision Columns

Decision Column Name	Description
Worst-case slack (default)	Determines best exploration point on worst-case slack in the exploration space.
Clock Setup: '<clock name>': Slack	Determines best exploration point on the <clock name> specified.
Clock Setup: '*': Slack	Determines best exploration point on all clocks.
Worst-case minimum t_{CO} Slack	Determines best exploration point on worst case minimum t_{CO} slack.
Worst-case t_H Slack	Determines best exploration point on worst-case t_H slack.
Worst-case t_{SU} Slack	Determines best exploration point on worst case t_{SU} slack.
<any column name>	Determines best exploration point on any column available in the Quartus II timing analysis report file.

Stop If Zero Failing Paths are Achieved

Instructs DSE to stop exploring the space after it encounters any point, including the base point, that has zero failing paths. DSE uses the failing path count reported in the **All Failing Paths report** column to make this decision.

Continue Exploration Even If Base Compilation Fails

With the **Continue Exploration Even If Base Compilation Fails** option turned on, DSE continues the exploration even when a design compilation error occurs. For example, if timing settings are not applied to your design, a DSE error occurs. To cause DSE to continue with the exploration instead of halting when an error occurs, turn on this option.

Run Quartus II PowerPlay Power Analyzer During Exploration

Turn on **Run Quartus II PowerPlay Power Analyzer During Exploration** to invoke the Quartus II PowerPlay Analyzer for every exploration performed by DSE. Using this option can help you debug your design and determine trade-offs between power requirements and performance optimization.

Archive All Compilations

Turn on **Archive All Compilations** to create a Quartus II Archive File (.qar) for each compilation. These archive files are saved to the **dse** directory in the design's working directory.

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 compilation.

Save Exploration Space to File

Turn on **Save Exploration Space to File** to write out a *<project name>.dse* file containing all options explored by DSE. You can use or modify this file to perform a custom exploration.

Ignore SignalTap & SignalProbe Settings

DSE uses advanced physical synthesis options that are not compatible with the SignalTap® II or SignalProbe™ features. As a result, DSE issues an error message when a project is opened for exploration that has either SignalTap II or SignalProbe turned on. The error message is similar to the following:

```
Error Opening Project-----  
Project is using SignalProbe. Please turn off  
SignalProbe before using this project with Design Space  
Explorer or Ignore SignalProbe Setting in your Design  
on the Options menu.
```

When the **Ignore SignalTap and SignalProbe Settings** option is turned on, DSE bypasses this check.

If you have already verified the design, you might save compilation time and improve resource utilization by turning this option on.

Skip Base Analysis & Compilation If Possible

Skip Base Analysis & Compilation If Possible allows the DSE to skip the Analysis & Elaboration stage or the compilation of the base point if base point compilation results are available from a previous Quartus II compilation.

Lower Priority of Compilation Threads

The **Lower Priority of Compilation Threads** option allows DSE to run the Quartus II executables with the `lower_priority` option. The `lower_priority` option lowers the priority of the Quartus II executable.

DSE Configuration File

Many options exist that allow you to customize the behavior of each DSE exploration. For example, you can specify seed values or a list of slave computers to be used for a distributed exploration run. Each time you close the DSE GUI it saves these values in a configuration file, **dse.conf**. The next time you launch the DSE GUI, it reads the values from **dse.conf** and restores the previous exploration settings.

Where the **dse.conf** file is stored varies based on the operating system that launches DSE. Table 12–6 specifies the locations where **dse.conf** files are stored based on operating system usage.

OS	File Location (default)	Comment
Windows	%APPDATA%/Altera/dse.conf	If the variable %APPDATA% is not defined, the configuration file is saved to /.altera.quartus/dse.conf
Unix	~/.altera.quartus/dse.conf	



Settings specified in the DSE command-line mode are not saved to a **dse.conf** configuration file.

DSE Advanced Information

This section covers advanced features that are available in DSE. These features increase the processing efficiency of design space exploration and provide further customization of the design space.

Computer Load Sharing in DSE Using Distributed Exploration

When you select **Distribute Compiles to Other Machines**, the DSE uses cluster computing technology to decrease exploration time. DSE uses multiple client computers to compile points in the specified exploration space. When you select the **Distributed DSE** option, DSE functions in one of the following operation modes:

- **Use LSF Resources:** DSE uses the Platform LSF grid computing technology to distribute exploration space points to a computing network.
- **Distribute Compiles to Other Machines** uses a Quartus II master process: DSE acts as a master and distributes exploration space points to client computers.

Distributed DSE Using LSF Resources

The easiest way to use distributed DSE technology is to submit the compilations to a preconfigured LSF cluster at your local site. For more information on LSF software, refer to www.platform.com, or contact your system administrator. Turn on **Use LSF resources** to enable this feature. You can specify an LSF queue when you select the **Configure Clients** option.

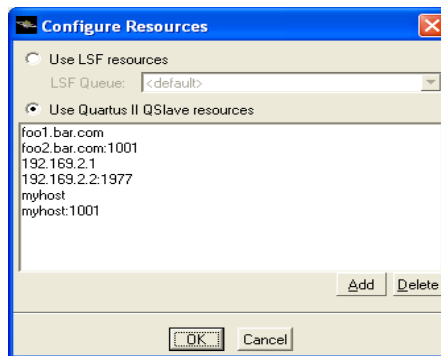
Distributed DSE Using a Quartus II Master Process

Before DSE can use computers in the local area network to compile points in the exploration space, you must create Quartus II software slave instances on the computers that will be used as clients. Type the following command at a command prompt on a client computer:

```
quartus_sh --qslave ←
```

Repeating this on several computers creates a cluster of Quartus II software slaves for DSE to use. After you have created a set of Quartus II software slaves on the network, add the names of each slave computer in the **Configure Clients** dialog box. [Figure 12–4](#) shows an example of client entries for a distributed search.

Figure 12–4. Client Entry in DSE



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, at the command prompt type:

```
quartus_sh --help=qslave ←
```

You must use the same version of the Quartus II software to run the slave processes as you use to run DSE. To determine which Quartus II software version that you are using to run DSE, select Help and click **About DSE**. Unexpected results can occur if you mix different Quartus II software versions when using the Distributed DSE search feature.

Concurrent Local Compilations

To reduce compilation time, DSE can compile exploration points concurrently. The **Concurrent Local Compilations** option allows you to specify the number of local compilations that DSE performs. For the **Concurrent Local Compilations** option, you can specify up to six concurrent compilations by choosing an integer value ranging from 1 through 6. You can use this option in conjunction with distributed processing. However, your system must have both the appropriate resources and licenses to perform concurrent compilations, and distributed processing. Multiprocessor or multicore systems are recommended for concurrent local compilations.



Concurrent Local Compilations require a separate Quartus II software license for each concurrent compilation. For example, if you compile four concurrent compilations, you need four licenses. Be sure before you choose a **Concurrent Local Compilations** value and start compilation that sufficient licenses are available.

Creating Custom Spaces for DSE

You can use custom spaces to explore combinations of options that are not in the predefined **Exploration Space** list. An exploration space is defined in an XML file. The following sections describe the tags you use to create a **Custom Space** that DSE can process.

A custom space is defined by the following three pairs of tags:

- `<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 `</DESIGNSPACE>` defines the end of the exploration space. Both of these tags are required 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 point in a custom exploration space. The end tag `</POINT>` defines the end of the exploration point. The `POINT` also allows you to specify the `<stage>` value 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**—indicates an Analysis & Synthesis setting change for that point
- **fit**—indicates a Fitter setting change for that point
- **seed**—indicates a Fitter seed change
- **llr**—indicates a LogicLock property change

The <value> value in the POINT tag can either be "1," indicating that for a specific stage the exploration point is active, or "0" for an inactive point.

An example of a POINT tag follows:

```
<POINT space="map" enabled="1">
...
</POINT>
```

The preceding point indicates a point that has Analysis & Synthesis setting changes and is active during Analysis & Synthesis.

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 a particular exploration point. The end tag </PARAM> defines the end of the parameter. The **Analysis & Synthesis** settings and the “<parameter>” values are shown in [Table 12-7](#).

Analysis & Synthesis Settings	Description	Value
STRATIX_OPTIMIZATION_TECHNIQUE	Type of optimization technique to use during the 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 the 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 12-7:

(1) Not all Analysis & Synthesis settings are available for all device families.

Table 12–8 shows the Fitter settings. An example of a PARAM tag is shown below:

```
<PARAM name="ADV_NETLIST_OPT_SYNTH_GATE_RETIME"> ON </PARAM>
```

The point in the example above indicates that the Analysis and Synthesis setting gate-level retiming is turned on for the exploration space point.

Table 12–8. Fitter Settings *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}
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 12–8:

(1) Not all Fitter settings are available for all device families.

Simple Custom Space

The custom exploration space example below shows a simple custom exploration space performing a seed sweep with settings for the Analysis & Synthesis and the Fitter compilation stages.

```
<DESIGNSPACE>
  <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 defines a custom exploration space that has two points: one map exploration point which changes synthesis settings, and one fit exploration point which 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 are turned on; register packing is turned off; and a seed value of three is used.

Custom Space XML Schema

The following example contains an XML schema describing 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.

```
<?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:complexType>
  </xs:element>
</xs:schema>
```


Document Revision History

Table 12–9 show the revision history for this document.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history to the document.	Added support information for the Stratix III device.
May 2006 v6.0.0	Updated for the Quartus II software version 6.0.0: <ul style="list-style-type: none"> • Updated with the new TimeQuest Timing Analyzer feature. 	
October 2005 v5.1.0	Chapter 12 was formerly Chapter 10 in version 5.0.	
May 2005 v5.0.0	Chapter 10 was formerly Chapter 9 in version 4.2.	
Dec. 2004 v2.1	<ul style="list-style-type: none"> • Updates to tables and figures. • New functionality in the Quartus II software version 4.2. 	
June 2004 v2.0	<ul style="list-style-type: none"> • Updates to tables and figures. • New functionality in the Quartus II software version 4.1. 	
Feb. 2004 v1.0	Initial release.	

Introduction

Available exclusively in the Altera® Quartus® II software, the LogicLock™ feature enables you to design, optimize, and lock down your design one module at a time. With the LogicLock feature, 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. The LogicLock feature also facilitates the incremental compilation flow for block-based design available in the Quartus II software.



For more information on hierarchical and team-based design, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

The Quartus II software beginning with version 4.2 supports the LogicLock block-based design flow for all of the following device families:

- Stratix® II, Stratix, Stratix GX
- Cyclone™ series
- MAX® II, APEX®, APEX II
- Excalibur™
- Mercury™ (Mercury devices support only locked and fixed regions)



This chapter assumes that you are familiar with the basic functionality of the Quartus II software.

Improving Design Performance

The LogicLock flow helps you optimize and preserve performance. You can use the LogicLock flow to place modules, entities, or any group of logic into regions in a device's floorplan. LogicLock assignments can be hierarchical, which allows you to have more control over the placement and performance of each module as well as groups of modules.

In addition to hierarchical blocks, you can apply LogicLock constraints to individual nodes; for example, you can make a wildcard path-based LogicLock assignment on a critical path. This technique is useful if the critical path spans multiple design blocks.



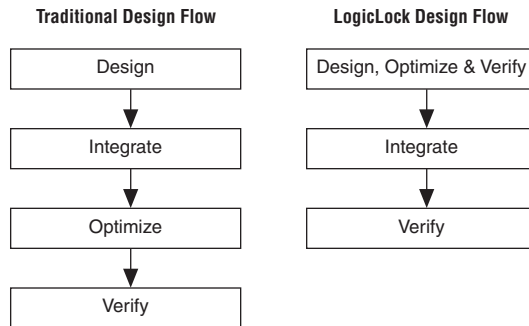
Although LogicLock constraints can improve performance, they can also degrade performance if they are not applied correctly.

The Quartus II LogicLock Methodology

The LogicLock design methodology lets you place the logic in each netlist file into a fixed or floating region in an Altera device. You can then maintain the placement and, if necessary, the routing of your blocks in the Altera device, thus retaining performance. Also, the LogicLock design methodology allows you to create design floorplans to obtain good results with the full incremental compilation flow in the Quartus II software.

Figures 13–1 compares the traditional design flow with the LogicLock design flow.

Figure 13–1. Traditional Design Flow Compared with Quartus II LogicLock Design Flow



For more information on block-based design with the LogicLock feature, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

Preserving Timing Results Using the LogicLock Flow

To preserve the timing results for a design module in the Quartus II software, you need to preserve the placement and routing information for all the logic in the design module. You can use one of two methods to preserve the placement and the routing results for a design module:

- You can use the LogicLock design methodology to back-annotate logic locations within a LogicLock region, which makes assignments to each node in the design.
- You can use the incremental compilation flow to preserve the fitting results for a design partition, and use the LogicLock design methodology to create a design floorplan that achieves good results.



For more information on block-based design with the LogicLock feature, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

When preserving logic placement in an Altera device, using LogicLock back-annotation, an atom netlist preserves the node names in subblocks of your design. An atom netlist contains design information that fully describes the submodule logic in terms of the device architecture. In the atom netlist, the nodes are fixed as Altera primitives and the node names do not change if the atom netlist does not change. If a node name changes, any placement information associated with that node, such as LogicLock assignments made when back-annotating a region, is invalid and ignored by the compiler.

If all the netlists are contained in one Quartus II project, use the LogicLock flow to back-annotate the logic in each region. If a design region changes, only the netlist associated with the changed region is affected. When you place and route the design using the Quartus II software, the software needs to re-fit only the LogicLock region associated with the changed netlist file.



Turn on the **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** option to **Never Allow** for all nodes in the region, avoiding the possibility of a node name change in the top-level design when the region is recompiled.

You may need to remove previously back-annotated assignments for a modified block because the node names may be different in the newly synthesized version. When you recompile with one new netlist file, the placement and assignments for the unchanged netlist files assigned to other LogicLock regions are not affected. Therefore, you can make changes to code in an independent block and not interfere with another designer's changes, even when all the blocks are integrated into the same top-level design.

With the LogicLock design methodology, you can develop and test submodules without affecting other areas of a design.

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 optimize the logic placed within the boundaries of a region to achieve the required performance, you must back-annotate the region's contents to lock the logic placement and routing. Locking the placement and routing preserves the performance when you integrate the region with the rest of the design.

This section explains the basics of designing with the LogicLock regions, including:

- Creating LogicLock Regions
- Timing Closure Floorplan View
- LogicLock Region Properties
- Hierarchical (Parent and/or Child) LogicLock Regions
- Assigning LogicLock Region Content
- Excluded Resources
- Tcl Scripts
- Importing and Exporting LogicLock Regions
- Additional Quartus II LogicLock Design Features


Creating LogicLock Regions

There are four ways to create a LogicLock region:

- On the Assignments menu, click **LogicLock Regions Window**.
- Using the Create New Region button in the Timing Closure Floorplan.
- On the View menu, click **Project Navigator**. Use the **Hierarchy** tab.
- Tcl scripts.

LogicLock Regions Window

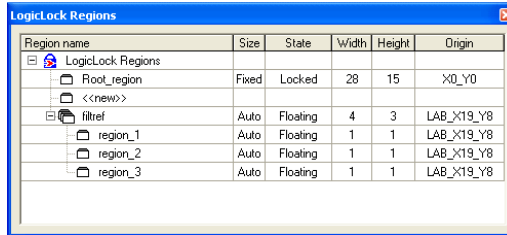
The LogicLock window is comprised of the LogicLock Regions window (Figure 13–2) and LogicLock Region Properties dialog box. Use the LogicLock Regions window to create LogicLock regions and assign nodes and entities to them. The dialog box provides a summary of all LogicLock regions in your design. In the LogicLock Regions window, you can modify a LogicLock region’s size, state, width, height, and origin as well as whether the region is soft or reserved. When the region is back-annotated, the placement of the nodes within the region are relative to the region’s origin, and the region’s node placement during subsequent compilations is maintained.

 The origin location varies based on device family. For Stratix II, Stratix, Stratix GX, Cyclone series, and MAX II 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. 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, refer to “Repair Branch” on page 13–22.

Figure 13–2. LogicLock Regions Window



Region name	Size	State	Width	Height	Origin
LogicLock Regions					
Root_region	Fixed	Locked	28	15	X0_Y0
<<new>>					
filterf	Auto	Floating	4	3	LAB_X19_Y8
region_1	Auto	Floating	1	1	LAB_X19_Y8
region_2	Auto	Floating	1	1	LAB_X19_Y8
region_3	Auto	Floating	1	1	LAB_X19_Y8

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 II, Stratix, Stratix GX, Cyclone series, and MAX II 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 13-3](#)). The **LogicLock Region Properties** dialog box shows the properties of the current selected regions.


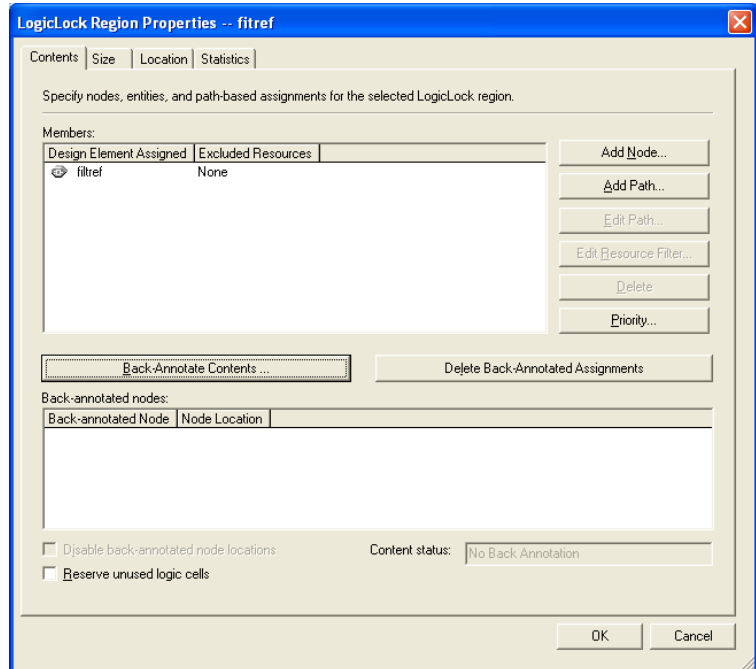
 To open the **LogicLock Region Properties** dialog box, double-click any region in the LogicLock Regions window, or right-click the region and click **Properties**.

Figure 13-3. LogicLock Region Properties Dialog Box



To back-annotate the contents of your LogicLock regions, perform these steps:

1. In the **LogicLock Region Properties** dialog box, click **Back-Annotate Contents**. The **Back-Annotate Assignments** dialog is shown.
2. In the **Back-Annotate Assignments** dialog box, in the **Back annotation type** list, select **Advanced** (Figure 13–4) and click **OK**.
3. In the **LogicLock Region Properties** dialog box, click **OK**.


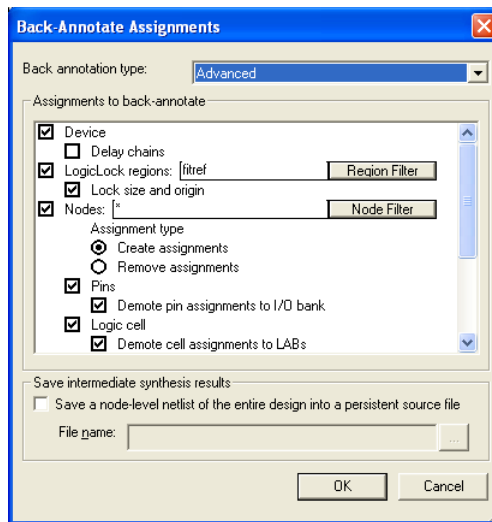

 If using the incremental compilation flow, logic back-annotation is not required. Preserve placement results using the Post-Fit Netlist Type instead of making placement assignments with back-annotation as described in this section.

Figure 13–4. Back-Annotate Assignments Dialog Box (Advanced Type)



 You also can back-annotate routing within LogicLock regions to preserve performance of the regions. For more information on back-annotating routing, refer to [“Back-Annotating Routing Information”](#) on page 13–34.

When you back-annotate a region's contents, all of the design element nodes appear under **Back-annotated nodes** with an assignment to a device resource under **Node Location**, for example, logic array block (LAB), M512, M4K, M-RAM, and digital signal processing (DSP) block. 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 module. If cell assignments are demoted, then the nodes are assigned to LABs rather than directly to logic cells.

Timing Closure Floorplan Editor

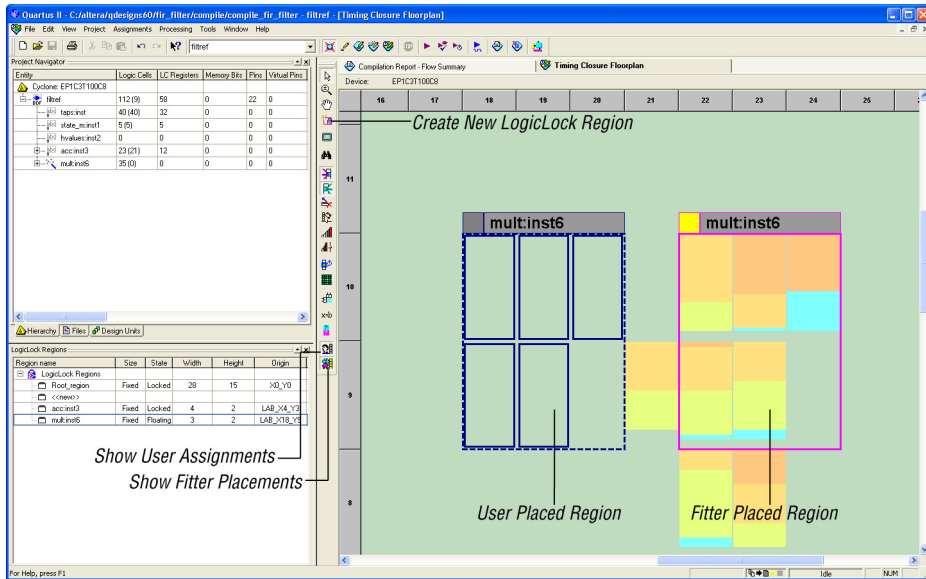
The Timing Closure Floorplan Editor has toolbar buttons that are used to manipulate LogicLock regions as shown in [Figure 13-5](#). You can use the **Create New LogicLock Region** button to draw LogicLock regions in the device floorplan.



The Timing Closure Floorplan Editor displays LogicLock regions when you select **Show User Assignments** or **Show Fitter Placements**. The type of region determines its appearance in the floorplan.

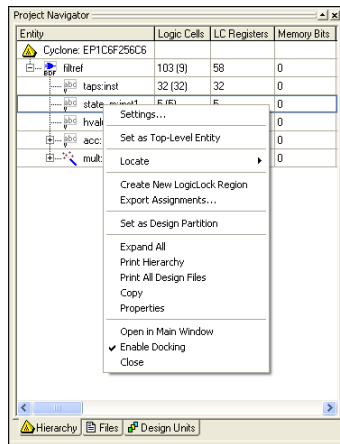
The Timing Closure Floorplan Editor differentiates between user assignments and fitter placements. When the **Show User Assignments** option is enabled in the Timing Closure Floorplan, you can see current assignments made to a LogicLock region. When the Fitter Placement option is enabled, you can see the properties of the LogicLock region after the last compilation. User-assigned LogicLock regions appear in the Floorplan Editor with a dark blue border ([Figure 13-5](#)). Fitter-placed regions appear in the Floorplan Editor with a magenta border.

Figure 13–5. Floorplan Editor Toolbar Buttons



Design Hierarchy

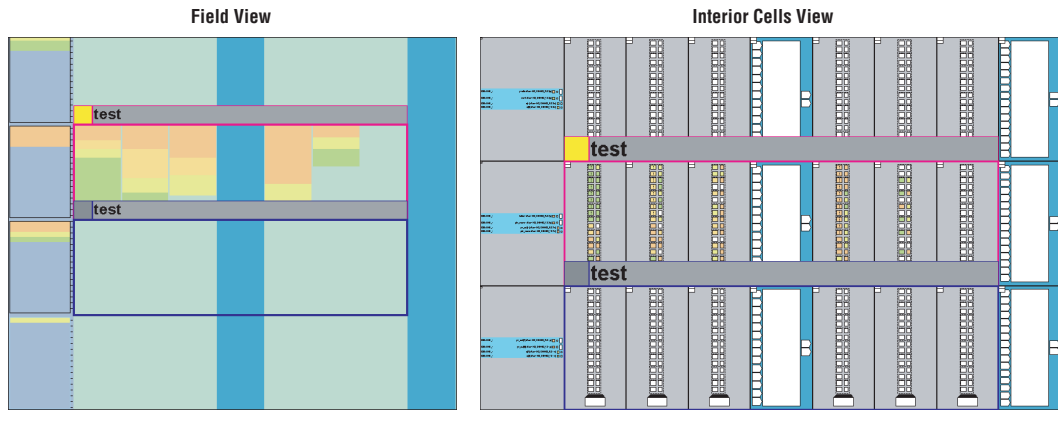
After you perform either a full compilation or analysis and elaboration on the design, the Quartus II software displays the hierarchy of the design. On the View menu, click **Project Navigator**. With the hierarchy of the design fully expanded, as shown in Figure 13–6, To create a LogicLock region, with the design fully expanded, right-click on any design entity in the design and click **Create New LogicLock Region**.

Figure 13–6. Using the Project Navigator to Create LogicLock Regions

Timing Closure Floorplan 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 13–7. The Field View provides an uncluttered view of the device floorplan in which all device resources such as embedded system blocks (ESBs) and MegaLAB™ blocks are outlined. The Interior Cells View provides a detailed view of device resources, including individual logic elements within a MegaLAB and device pins.

Figure 13–7. Timing Closure Floorplan Editor



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 the region’s contents and the module’s timing requirements. Table 13–1 describes the options for creating LogicLock regions.

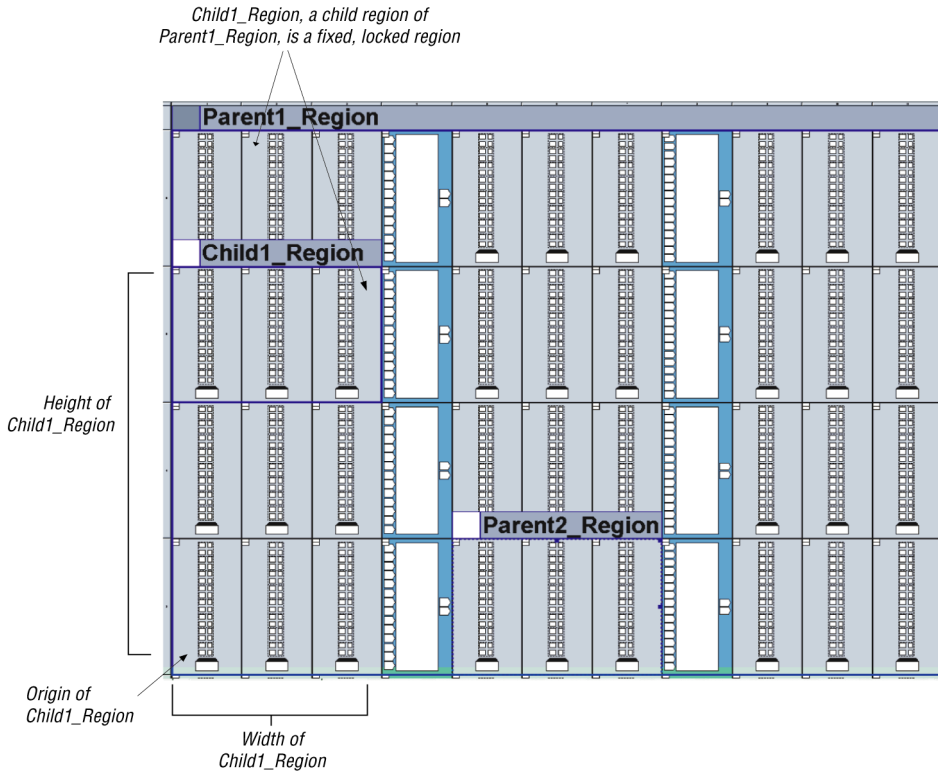
Properties	Values	Behavior
State	Floating (default), Locked	Floating regions allow the Quartus II software to determine the region’s location on the device. Locked regions represent user-defined locations for a region and are shown with a solid boundary in the floorplan. A locked region must have a fixed 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 the Fitter can use the resources within a region for entities that are not assigned to the region. If the reserved property is turned on, only items assigned to the region can be placed within its boundaries.
Soft	Off (default), On	Soft (on) 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 (off) regions do not allow contents to be placed outside of the boundaries of the region.
Origin	Any Floorplan Location	The origin is the origin of the LogicLock region’s placement on the floorplan. For Stratix, Stratix II, Stratix GX, Cyclone series, and MAX II devices, the origin is located in the lower left corner. The origin is located in the upper left corner for other device 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 support only locked and fixed regions.

The floorplan excerpt in Figure 13-8 shows the LogicLock region properties for a design implemented in a Stratix device.

Figure 13-8. LogicLock Region Properties

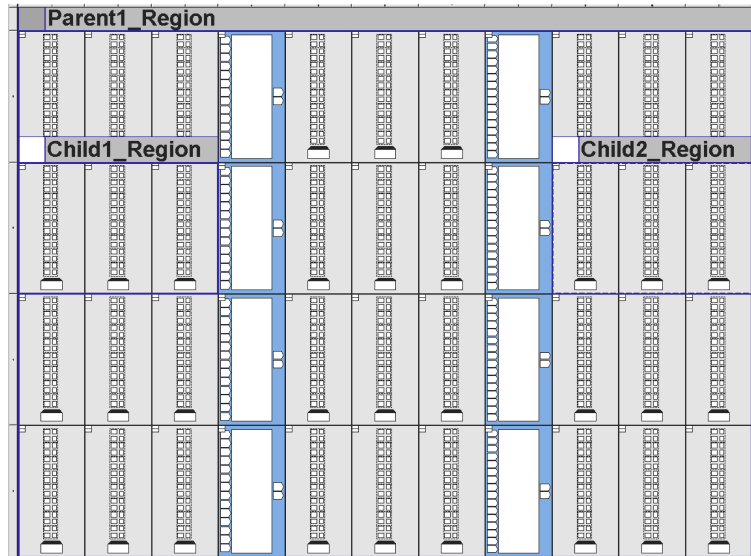


Hierarchical (Parent and/or Child) LogicLock Regions

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 13–9 shows an example child region within a parent region, including labels for a locked location and floating location in a Stratix II device.

Figure 13–9. Child Region Within a Parent 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. 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, the child region must fit entirely within the parent region.

Assigning LogicLock Region Content

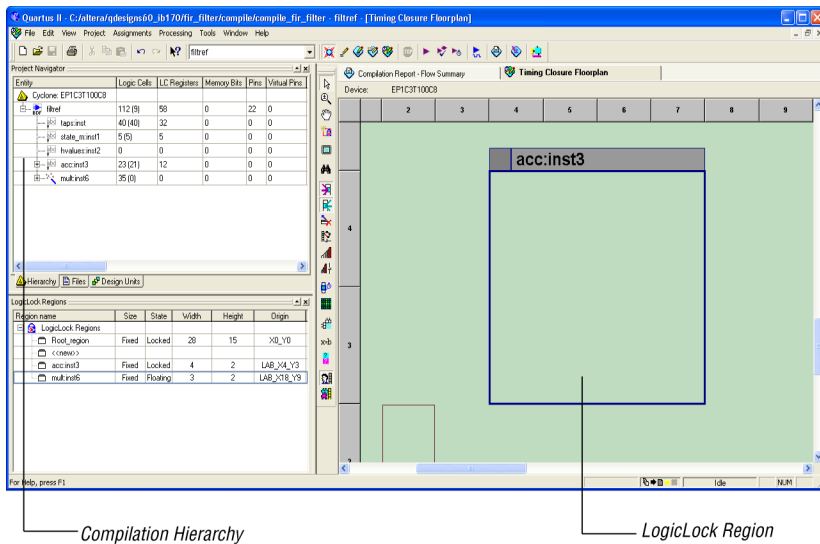
Once you have defined a LogicLock region, you must assign resources to it using the Timing Closure Floorplan, the **LogicLock Regions** dialog box, the Assignment Editor, or Tcl scripts with the Quartus II Tcl Console or the `quartus_sh` executable.

Using Drag & Drop to Place Logic

You can drag selected logic displayed in the **Hierarchy** tab of the **Project Navigator**, Node Finder, or a schematic design file and drop it into the Timing Closure Floorplan or the **LogicLock Regions** dialog box.

Figure 13–10 shows logic that has been dragged from the **Hierarchy** tab of the **Project Navigator** and dropped into a LogicLock region in the **Timing Closure Floorplan**.

Figure 13–10. Drag & Drop Logic in the Timing Closure Floorplan

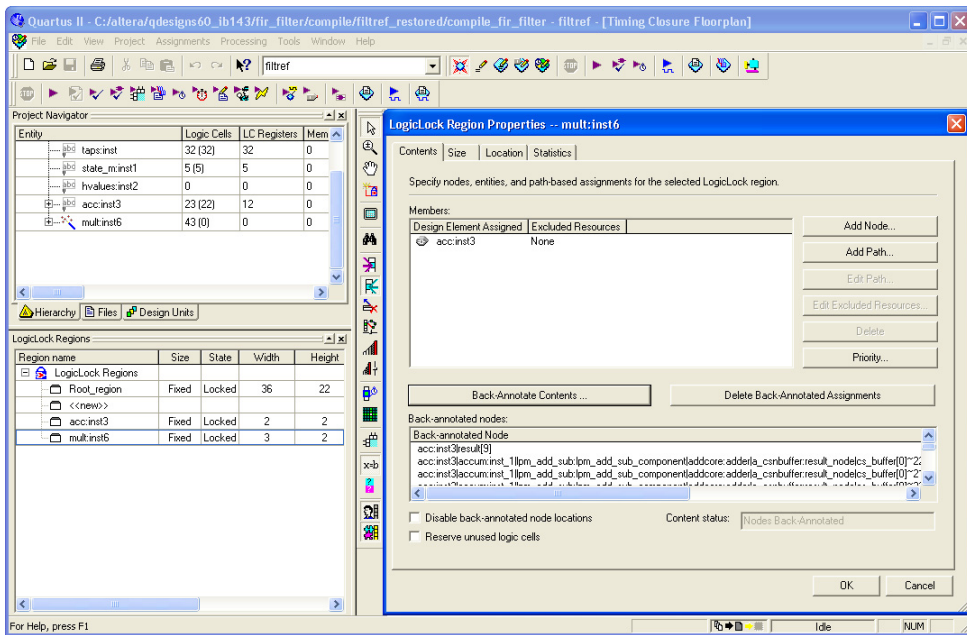


Compilation Hierarchy

LogicLock Region

Figure 13–11 shows logic that has been dragged from the **Hierarchy** tab of the **Project Navigator** and dropped into the **LogicLock Regions Properties** dialog box. Logic can also be dropped into the **Design Element Assigned** column of the **Contents** tab of the **LogicLock Region Properties** box.

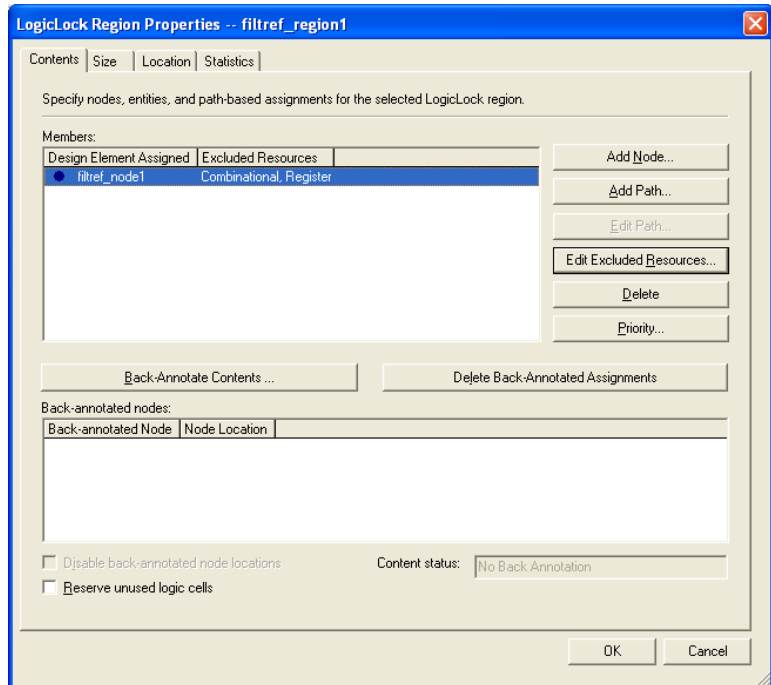
Figure 13–11. Drag & Drop Logic into the LogicLock Regions Dialog Box



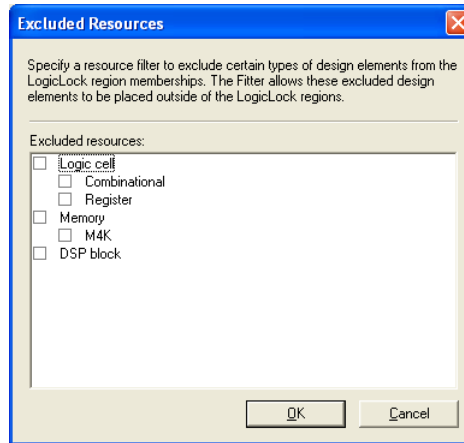
You must manually assign pins to a LogicLock region. The Quartus II software does not include pins automatically when you assign an entity to a region. The software only obeys pin assignments to locked regions that border the periphery of the device. For Stratix, Stratix II, Cyclone II, Cyclone, and MAX II devices, the locked regions must include the I/O pins as resources.

Excluded Resources

The Excluded Resources feature allows you to easily exclude specific device resources such as DSP blocks or M4K memory blocks from a LogicLock region. For example, you can specify resources that belong to a specific entity that are assigned to a LogicLock region, and specify that these resources be included with the exception of the DSP blocks. Use the Excluded Resources feature on a per-LogicLock region member basis. Figure 13–12 shows the **LogicLock Region Properties** dialog box with the Excluded Resources highlighted.

Figure 13–12. LogicLock Region Properties Dialog Box

To exclude certain device resources from an entity, in the **LogicLock Region Properties** dialog box, highlight the entity in the **Design Element Assigned** column, and click **Edit Excluded Resources**. The **Excluded Resources** dialog box is shown (Figure 13–13). In the **Excluded Resources** dialog box, you can select the device resources you want to exclude from the entity. Once you have selected the resources to exclude, the **Excluded Resources** column is updated in the **LogicLock Region Properties** dialog box to reflect the excluded resources.

Figure 13–13. Excluded Resources

The Excluded Resources feature prevents certain resource types from being included in a region, but it does not prevent the resources from being placed inside the region unless the region's "Reserved" property is set to **On**. To inform the Fitter that certain resources are not required inside a LogicLock region, define a resource filter.

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, refer to the *Command Line* and *Tcl API* topics in the Quartus II online Help or ["Scripting Support" on page 13–38](#).

Importing and Exporting LogicLock Regions



This section describes the steps required to import and export the LogicLock regions. For information on importing and exporting the assignments for lower-level design partitions using the incremental compilation flow, refer to the *Quartus II Incremental Compilation for Hierarchical & Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

For the Quartus II software to achieve optimal placement, you should make timing assignments for all clock signals in the design, including t_{SU} , t_{CO} , and t_{PD} .

To facilitate the LogicLock design flow, the **Timing Closure Floorplan** highlights resources that have back-annotated LogicLock regions.

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 II Settings File, and routing information stored in a Routing Constraints 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 Verilog Quartus Mapping Netlist File (.vqm) using the Quartus II software, because the original atom netlist may have changed as a result of these optimizations.



Turn on the **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** for 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 Verilog Quartus Mapping Netlist 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-box library of parameterized modules (LPM) functions or Altera megafunctions, you must generate a Quartus II Verilog Quartus Mapping Netlist File for the black-box modules.



For instructions on creating an atom netlist in the Quartus II software, refer to *Saving Synthesis Results for an Entity to a Verilog Quartus Mapping File* in Quartus II Help.

When you export LogicLock regions, all your design assignments are exported. Filtering is done only when the design is imported. However, you can export a subentity of the compilation hierarchy and all of its relevant regions. To do this, right-click the entity in the **Hierarchy** tab of the **Project Navigator** and click **Export Assignments**.

Placement Information

The Quartus II Settings File 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 Routing Constraints File (.rcf) contains the module's LogicLock routing information. To maintain performance, you must back-annotate the module.

Exporting the Routing Constraint File and Atom Netlist

To specify the Routing Constraint File and Atom Netlist to export, perform the following steps:

1. Run a full compilation.
2. On the Assignments menu, click **LogicLock Regions Window**.
3. Right-click the region name, and click **Properties**.
4. In the **LogicLock Region Properties** dialog box, click **Back-Annotate Contents**.
5. Enable or disable any of the advanced options such as **Prevent further netlist optimization**.
6. Turn on **Routing**, and click **OK**.
7. In the **LogicLock Region Properties** dialog box, click **OK**.
8. On the Assignments menu, click **Export Assignments**.
9. In the **Export Assignments** dialog box, turn on **Export back-annotated routing** and **Save a node-level netlist of the entire design into a persistent source file**, click **OK**.



For instructions on exporting a LogicLock region assignment in the Quartus II software, refer to *Exporting LogicLock Region Assignments & Other Entity Assignments* in Quartus II Help.

Import the Module

To specify which Quartus II Settings File for a specific instance or entity, use the **LogicLock Import File Name** option in the Assignment Editor. This option lets you specify different LogicLock region constraints for each instance of an entity and import them into the top-level design. You also can 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 Quartus II Settings File and Routing Constraints File 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 and elaboration, or compile the top-level design, thus ensuring that the Quartus II software is aware of all instances of the lower-level modules.

The following sections describe how to specify a Quartus II Settings File for a module and how to import the LogicLock assignments into the top-level design.

Importing the Routing Constraints File and the Atom Netlist File

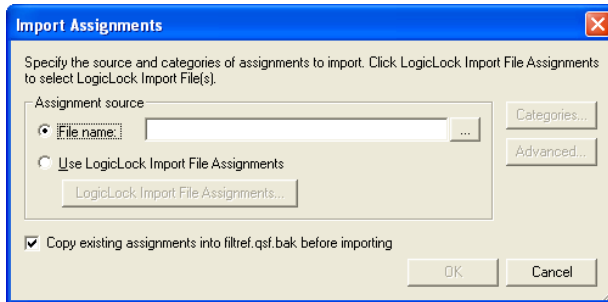
To specify the Quartus II Settings File and atom netlist to import, perform the following steps:

1. On the Assignments menu, click **Import Assignments**. In the **Import Assignments** dialog box, click **Advanced**.
2. In the **Advanced Import Settings** dialog box, turn on **Back-annotated routing**.

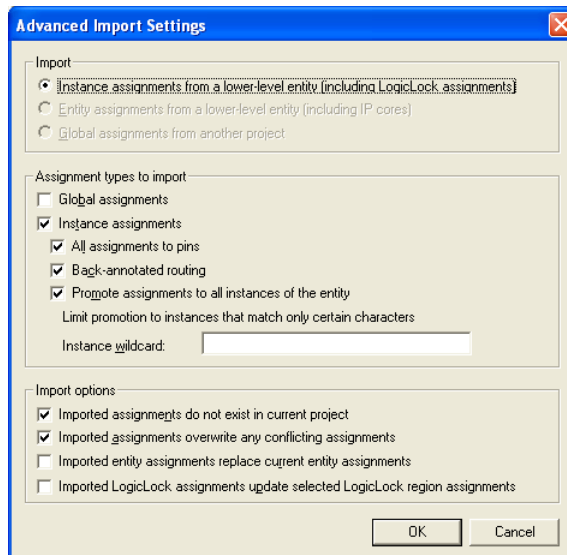
Now, when you import a LogicLock region, the routing constraint file is also be imported.

Import the Assignments

On the Assignments menu, click **Import Assignments** to import the assignments. The Import Assignments dialog box is shown (Figure 13–14).

Figure 13–14. Import Assignments Dialog Box

Use the options available in the **Advanced Import Settings** dialog box to control how you import your LogicLock regions (Figure 13–15).

Figure 13–15. Advanced Import Settings Dialog Box

To prevent spurious no-fit errors, parent, or top-level regions with multiple instances (that do not contain back-annotated routing information), are imported with their states set to floating. Otherwise, the region's state remains as specified in the Quartus II Settings File. This allows the Quartus II software to move LogicLock regions to areas on the

device with free resources. A child region is locked or floating relative to its parent region's origin as specified in the module's 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 the 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 pointer over a LogicLock region name on the **Hierarchy** tab 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 pointer over fitter-placed LogicLock regions displays the maximum routing delay within the LogicLock region. To enable this feature, on the View menu, point to Routing and click **Show Intra-region Delay**.

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. In the **LogicLock Regions** dialog box, the Quartus II software identifies and displays in red the names of illegal LogicLock regions.

To correct the illegal LogicLock region, use the **Repair Branch** command. Right click the desired LogicLock region's name and choose **Repair Branch**.

If more than one illegal LogicLock region exists, you can repair all regions. To do so, right-click the first line in the LogicLock window that contains the text "LogicLock Regions" and click **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** option excludes the specified 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 subentity or node from the region.



You can make the **Prevent Assignment to LogicLock Regions** assignment to an entity or node 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.

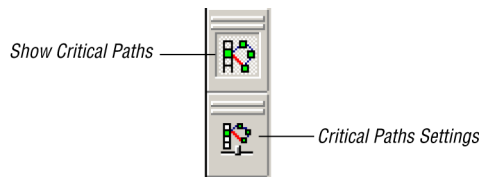
Rubber Banding

On the View menu, click Routing, and select **Rubber Banding** to show 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 in the design by turning on the **Show Critical Paths** option. Use this option with the **Critical Paths Settings** option to display paths based on the Timing Analysis report, as shown in [Figure 13–16](#).

Figure 13–16. Show Critical Paths & Critical Paths Settings



Show Connection Count

You can determine the number of connections between LogicLock regions by turning on the **Show Connection Count** option.

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 source and destination nodes can be any of the following:

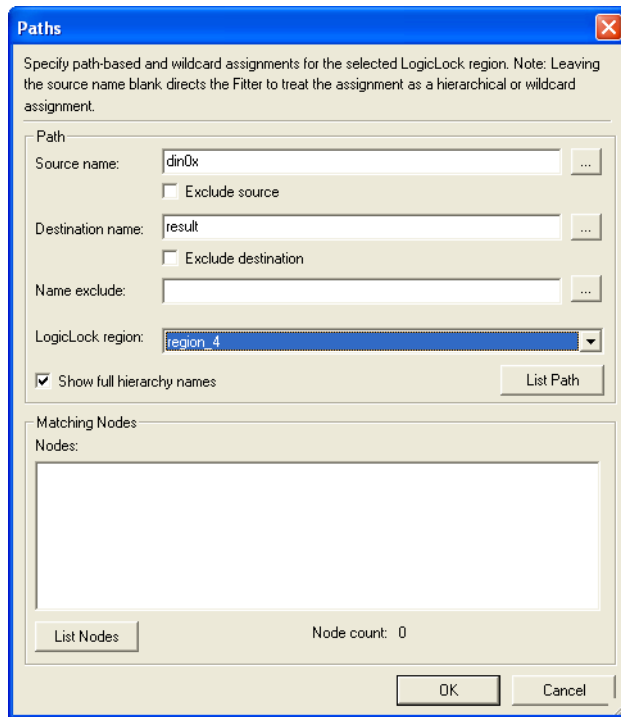
- Valid register-to-register path, meaning that the source and destination nodes must be registers
- Valid pin-to-register path, meaning the source node is a pin and the destination node is a register
- Valid register to pin path, meaning that the source node is a register and the destination node is a pin
- Valid pin-to-pin path, meaning that both the source and destination nodes are pins

Figure 13–17 shows the **Paths** dialog box.

To access the **Paths** dialog box, on the **Contents** tab of the **Logic Lock Regions** dialog box, click **Add Path** or **Edit Path**.



Both “*” and “?” wildcard characters are allowed for the source and destination nodes. When creating path-based assignments, you can exclude specific nodes using the **Name exclude** field in the **Paths** dialog box.

Figure 13–17. Paths Dialog Box

You can also use the Quartus II Timing Analysis Report to create path-based assignments by following these steps:

1. Expand the **Timing Analyzer** section in the **Compilation Report**.
2. Select any of the clocks in the section that is labeled “Clock Setup:<clock name>.”
3. Locate a path that you want 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.

On the Project menu, click **Revisions**. In the **Revisions** dialog box, 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 reside 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,” the 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 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 are not assigned outside the boundaries of the first non-soft parent region. If a non-soft parent does not exist (in a design targeting a Stratix II, Stratix GX, Stratix, Cyclone II, Cyclone, or MAX II device), the region floats within the `Root_region`, that is, 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. The I/O port mapping may create problems for a modular and hierarchical design as lower level modules may have I/O ports that exceed device pins available on the targeted device. Or, the I/O ports may not directly feed a device pin, but instead drive other internal nodes. The Quartus II accommodates this situation by supporting virtual pins.

The Virtual Pin assignment communicates to the Quartus II software which I/O ports of the design module are 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 is mapped to either an LCELL or ALM, depending on the target device. [Figure 13–19](#) shows the virtual input and output pins in the Timing Closure Floorplan Editor.



Bidirectional, registered I/O pins, and I/O pins with output enable signals cannot be virtual pins.

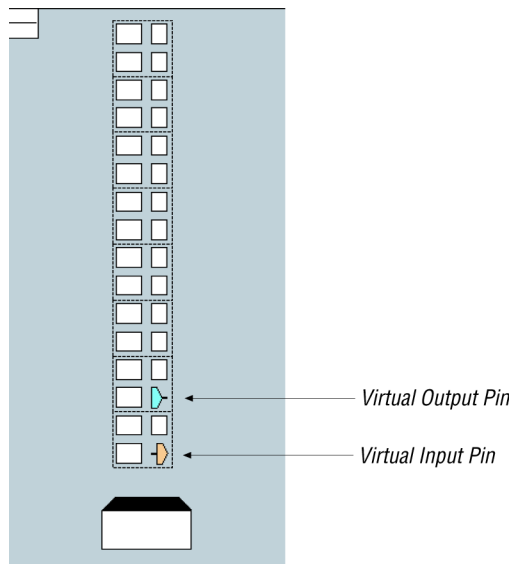
In the top-level design, these virtual pins are connected to an internal node of another module. Making assignments to virtual pins allows you to place them in the same location or region on the device as the corresponding internal nodes would exist in the top-level module. This feature has the added 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 on the **Virtual Pin Clock** option for each virtual pin.

Use the following guidelines for creating virtual pins in the Quartus II software:

- Clock pins should not be declared as virtual pins.
- Nodes/signals that drive physical device pins in the top-level design should not be declared as virtual pins.

Figure 13–18. Virtual I/O Pins in the Quartus II Floorplan Editor




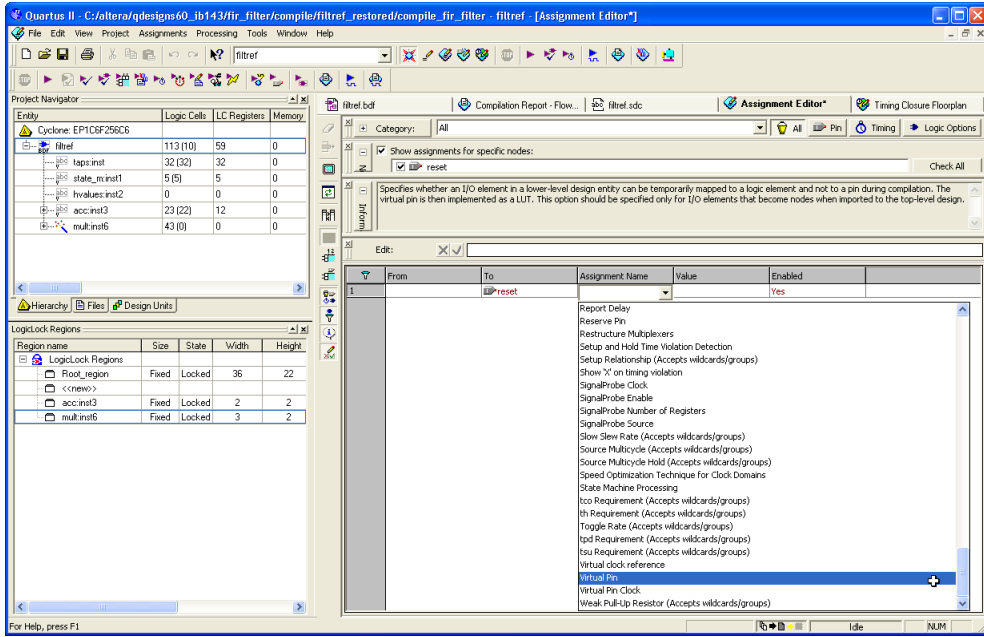
 Bidirectional, registered I/O pins, and I/O pins with output enable signals cannot be virtual pins. All virtual pins must map to device I/O pins in the top-level design.

Figure 13–19 shows assigning virtual pins using the Assignment Editor.

Figure 13–19. Using the Assignment Editor to Assign Virtual Pins



In the Node Finder, setting **Filter Type** to **Pins: Virtual** allows you to display all assigned virtual pins in the design. From the Assignment Editor, to access the Node Finder, double-click the **To** field; when the arrow appears on the right side of the field, click the arrow and select **Node Finder**.

LogicLock Restrictions

This section discusses restrictions to consider when you use 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, placing restrictions on nodes or entities in the design often is necessary. These restrictions often conflict with the node or entity assignments for a LogicLock region. To avoid conflicts, 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 automatically can 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:

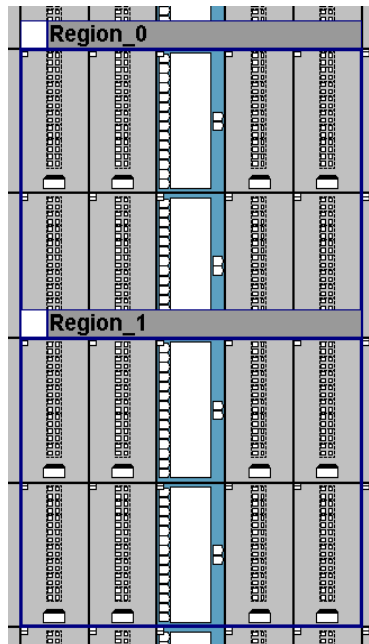
- LogicLock regions with pin assignments must be placed on the periphery of the device, adjacent to the pins. (For Stratix II, Stratix, Stratix GX, Cyclone series, and MAX II devices, you must also include the I/O block.)
- Floating LogicLock regions cannot overlap.
- Avoid creating fixed and locked 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 guidelines 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 13–20 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 of the device that have the same resources.

Figure 13–20. Floorplan of Two Instances of a LogicLock Region



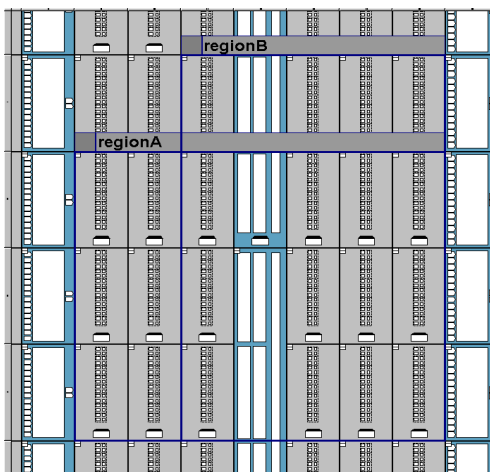
Placing Memory, Pins & Other Device Features into LogicLock Regions

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 are effective only in 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 13–21 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 13–21. Overlapping 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.

Back-annotate the routing from the Assignments menu, by choosing **Routing** from the **Back-Annotate Assignments** dialog box. Refer to [Figure 13–4](#).



If you are not using an atom netlist, you must turn on the **Save a node-level netlist of the entire design into a persistent source file** option (on the Assignments menu, click **Back-Annotate Assignments**) if back-annotation of routing is selected. Writing out a Verilog Quartus Mapping Netlist File causes the Quartus II software to enforce persistent naming of nodes when saving the routing information. The Verilog Quartus Mapping Netlist File is then 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

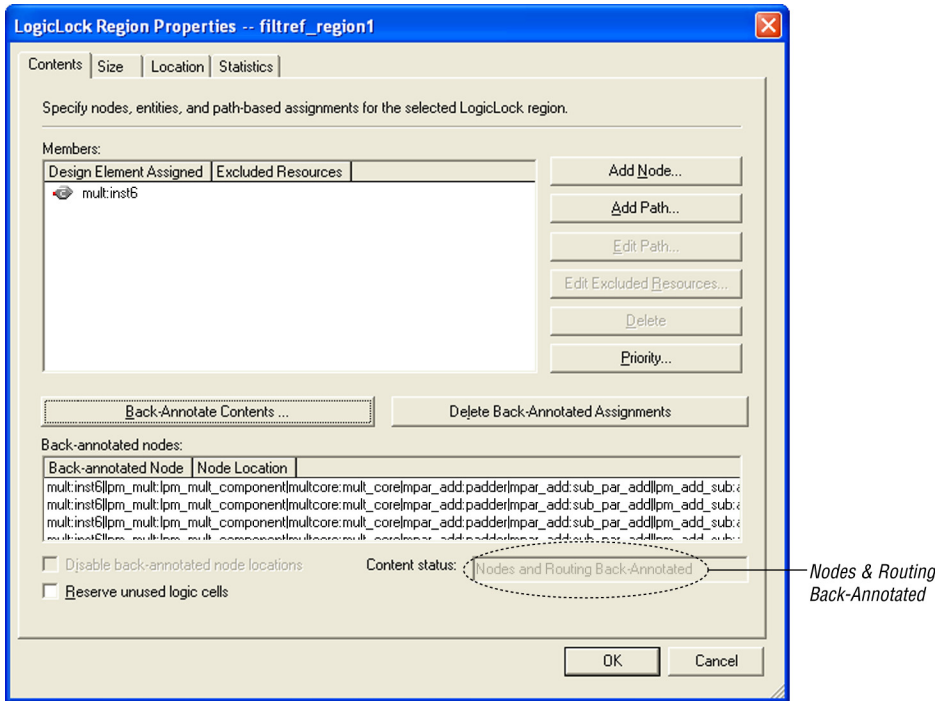
To export the LogicLock region routing information, on the Assignments menu, click **Export Assignments**, and in the **Export Assignments** dialog box, turn on **Export Back-annotated routing**. This generates a Quartus II Settings File and a Routing Constraints File in the specified directory. The Quartus II Settings File contains all LogicLock region properties as specified in the current design. The Routing Constraints File contains all the necessary routing information for the exported LogicLock regions.

This Routing Constraints File works only with the atom netlist for the entity being exported.

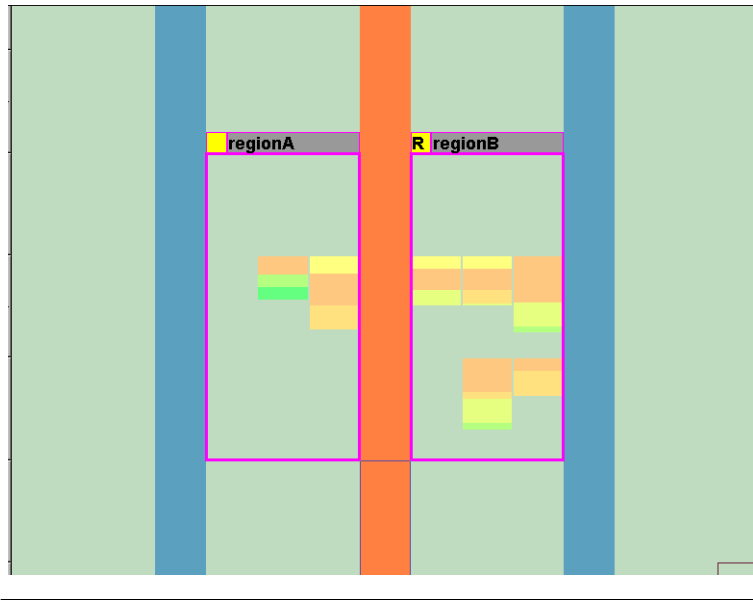
Only regions that have back-annotated routing information have their routing information exported when you export the LogicLock regions. All other regions are exported as regular LogicLock regions.

To determine if a LogicLock region contains back-annotated routing, refer to 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" (Figure 13-22).

Figure 13–22. LogicLock Status



The Quartus II software also reports whether routing information has been back-annotated in the Timing Closure Floorplan. LogicLock regions with back-annotated routing have an “R” in the top-left hand corner of the region (Figure 13–23).

Figure 13–23. Back-Annotation of Routing

Importing Back-Annotated Routing in LogicLock Regions

To import LogicLock region routing information, you must specify the instance that will have its routing information imported. This is done with the assignment LogicLock Routing Constraints File in the Assignment Editor.



A Routing Constraints File must be explicitly specified using the LogicLock **Back-annotated Routing Import File Name** assignment prior to importing any LogicLock region.

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 Routing Constraints File for a lower level entity, you must use the same atom netlist, that is, the Verilog Quartus Mapping Netlist File that was used to generate the Routing Constraints File. This ensures that the node names annotated in the Routing Constraints File 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 ↵
```

The *Scripting Reference Manual* has the same information in PDF form.



For more information about Tcl scripting, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. Refer to the *Quartus II Settings File Reference Manual* for information about all settings and constraints in the Quartus II software. For more information about command-line scripting, refer to the *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook*.

Initializing & 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 is 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, refer to [“Creating LogicLock Regions” on page 13–4](#).

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 13–13](#).

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.

Save a Node-level Netlist for the Entire Design into a Persistent Source File (.vqm)

Make the following assignments to cause the Quartus II Fitter to save a node-level netlist for the entire design into a Verilog Quartus Mapping Netlist 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 Verilog Quartus Mapping Netlist File is saved in the directory specified at the completion of a full compilation.

For more information about saving a node-level netlist, refer to [“Atom Netlist Design Information”](#) on page 13–18.

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 export.qsf
```

For more information about exporting LogicLock regions refer to [“Export the Module”](#) on page 13–18.

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 -to my_destination  
  
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, refer to [“Import the Module” on page 13–20](#).

Setting LogicLock Assignment Priority

Use the following Tcl code to set the priority for a 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, refer to [“Constraint Priority” on page 13–31](#).

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, refer to [“Virtual Pins” on page 13–29](#).

Back-Annotating LogicLock Regions

The Quartus II software provides the back-annotate Tcl package that allows you to back-annotate the contents of a LogicLock region. Use the following command line option to back-annotate a LogicLock region:

```
logiclock_back_annotate [-h | -help] [-long_help]
[-region <region_name>] [-from <source_name>]
[-to <destination_name>] [-exclude_from] [-exclude_to] [-path_exclude <path_exclude_name>]
[-no_delay_chain] [-no_contents] [-lock] [-routing]
[-resource_filter <resource_filter_value>] [-no_dont_touch]

[-remove_assignments] [-no_demote_lab] [-no_demote_mac] [-no_demote_pin] [-no_demote_ram]
```

For example, the following command back-annotates all nodes and routing in the region, `one_region`.

```
package require ::quartus::backannotate
logiclock_back_annotate -routing -lock -no_demote_lab -region one_region
```



For more information about Tcl scripting, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. For more information about command-line scripting, refer to 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 the implementation of design modules to occur independently, and also 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.

Document Revision History

Table 13–2 shows the revision history for this document.

Date & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history to the document.	
May 2006 v6.0.0	Minor updates for the Quartus II software version 6.0.0.	
October 2005 v5.1.0	Chapter 13 was formerly Chapter 11 in version 5.0.	
May 2005 v5.0.0	Chapter 11 was formerly Chapter 10 in version 4.2.	
Dec. 2004 v2.2	<ul style="list-style-type: none"> ● Updates to tables and figures. ● New functionality in the Quartus II software version 4.2. 	
August 2004 v2.1	<ul style="list-style-type: none"> ● New functionality in the Quartus II software version 4.1 Sp1. 	
June 2004 v2.0	<ul style="list-style-type: none"> ● Updates to tables and figures. ● New functionality in the Quartus II software version 4.1. 	
Feb. 2004 v1.0	Initial release.	

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, refer to *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 5.1
- Amplify Physical Optimizer, version 3.7

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.



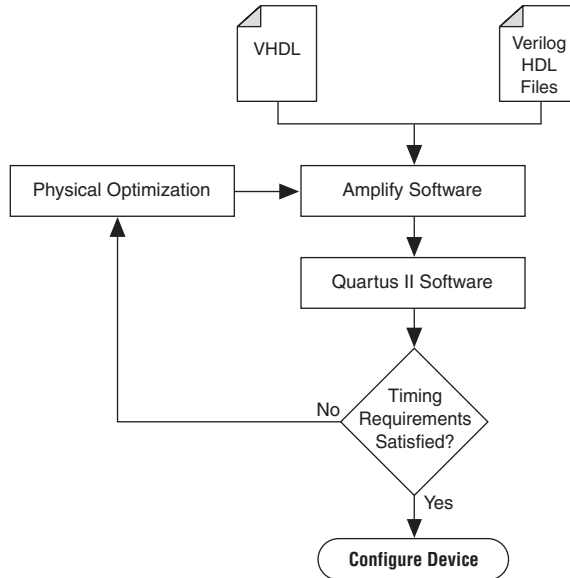
Refer to [“Forward Annotating Amplify Physical Optimizer Constraints into the Quartus II Software”](#) on page 14–12 for more information on setting up a Quartus II project with Amplify Physical Optimizer Tcl script files.

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 `<revision 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 14-1 shows the Amplify Physical Optimizer design flow.

Figure 14-1. 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.

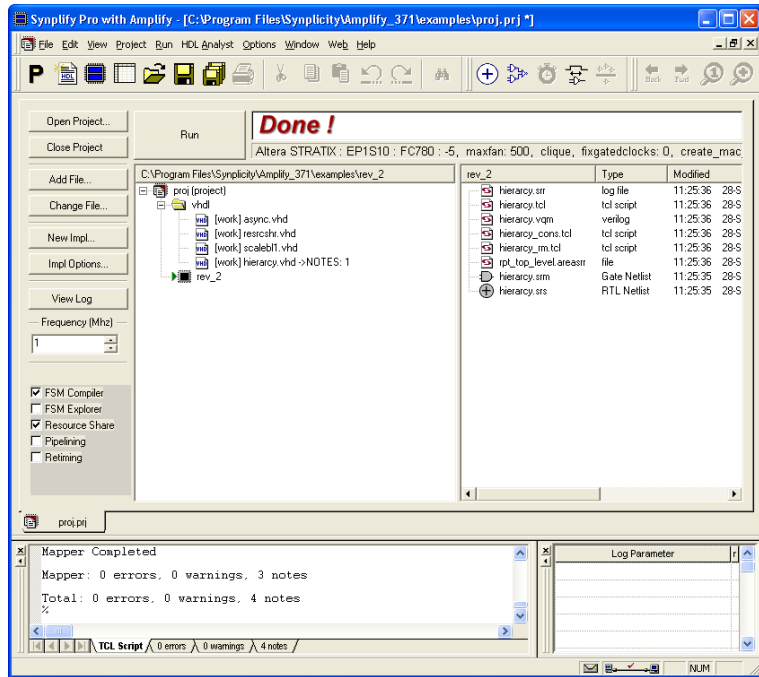
Create New Implementations

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 Cliqing** 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 14–2](#) 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 14–2. Amplify Physical Optimizer Project Window



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

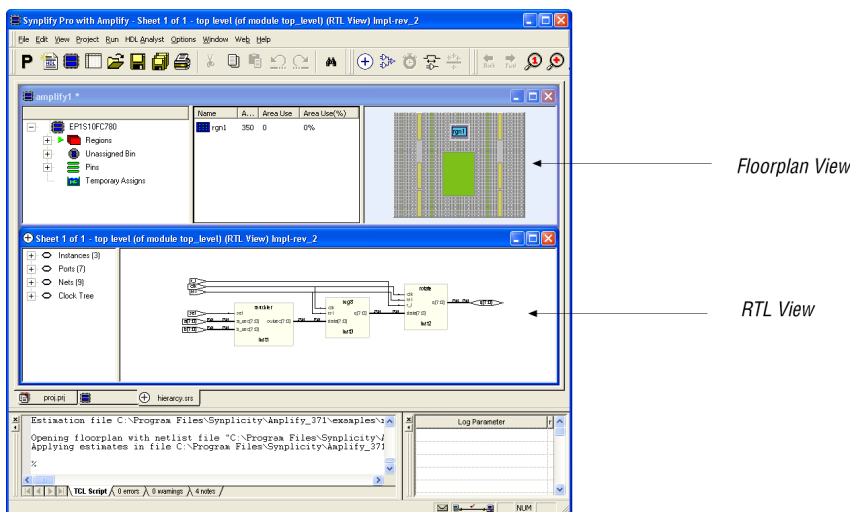
When designs do not meet performance requirements with the initial pass through the Amplify Physical Optimizer software, you can create location assignments to reduce interconnect and logic delays to improve your design's performance.

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 *<revision 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 is shown (Figure 14-3).

Figure 14-3. Stratix 1S20 Floorplan in the Amplify Physical Optimizer Software



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:

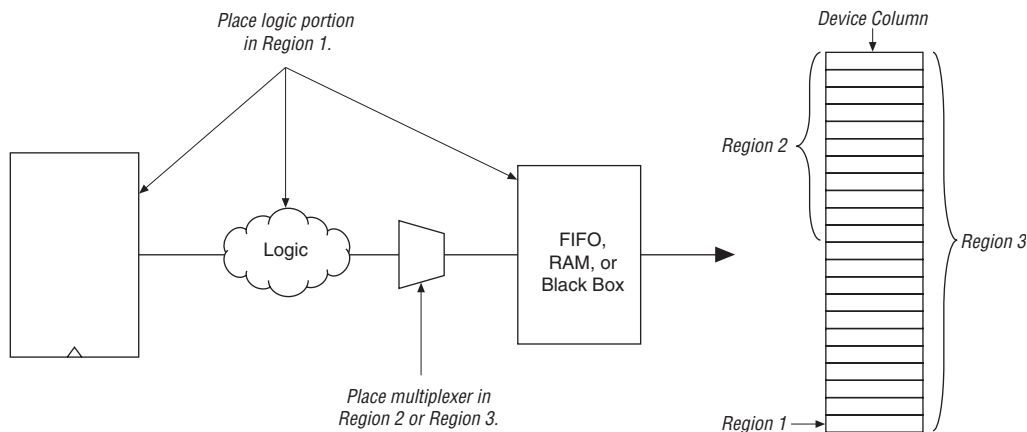
1. Create a region in the Amplify Physical Optimizer device floorplan window and select the module in the RTL view of the design.
2. Drag the module to the new region. The software will then report the utilization of the region.
3. Resynthesize the design in the software to reoptimize the critical path 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

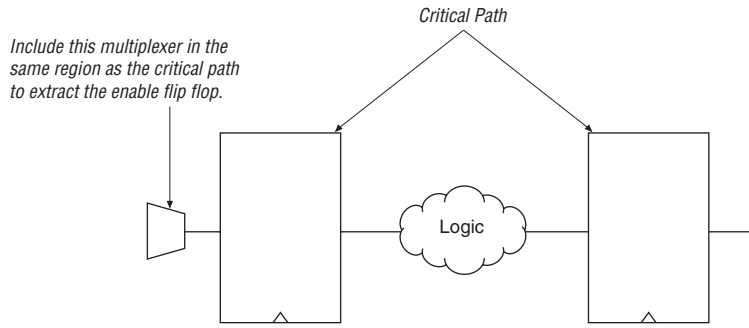
To create a floorplan for critical paths with one or more multiplexers, create multiple regions and assign the multiplexer to one region and the logic to another. [Figure 14–4](#) shows placing critical paths with multiplexers.

Figure 14–4. Placing Critical Paths with Multiplexers



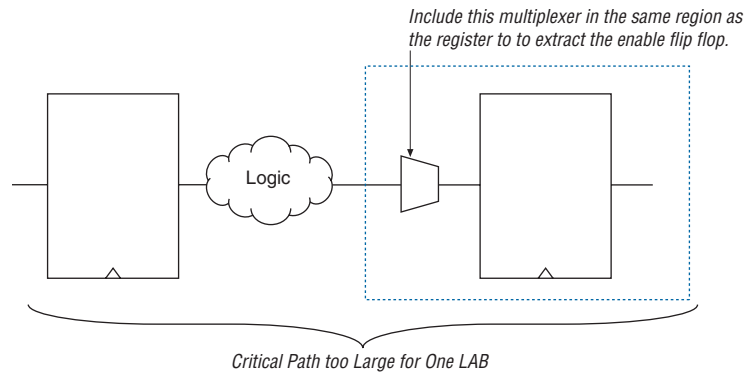
If the critical path contains a multiplexer feeding a register, create a region and place the multiplexer along with the entire critical path in the region (Figure 14-5).

Figure 14-5. Critical Paths with Multiplexers Feeding Registers



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 14-6 shows large critical paths with multiplexers feeding registers.

Figure 14-6. Large Critical Paths with Multiplexers Feeding Registers



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:

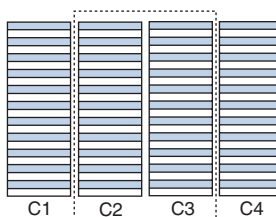
1. Create a region and assign the first critical path to that region.
2. Create another region, leaving one MegaLAB structure between the first and second regions.
3. Assign the second critical path to the second region.

Feedback Paths

If critical paths have the same start and end points, follow the steps below in the Amplify Physical Optimizer software (Figure 14-7):

1. Select the register and instance not directly connected to the register.
2. Right-click and select **Filter Schematic** twice.
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 14-7. Critical Paths with the Same Starting or Ending Points



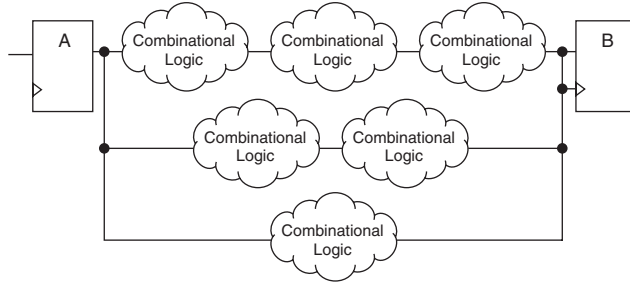
If the critical path does not include I/O pins, create region in columns C2 or C3.

Starting & Ending Points

Figure 14-8 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,

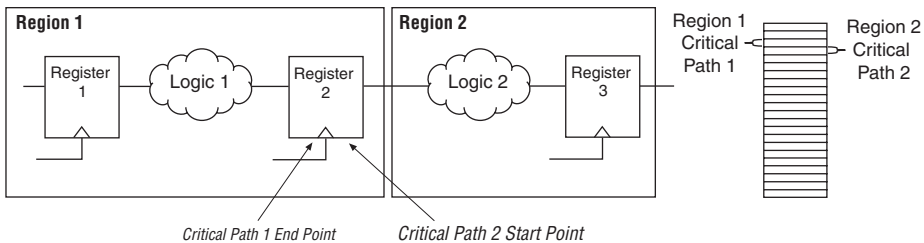
assign the starting points and ending points to the same region. Similarly, if there is unrelated logic between starting points and multiple ending points, assign the starting points and ending points to the same region.

Figure 14–8. Critical Paths with Multiple Starting or Ending Points



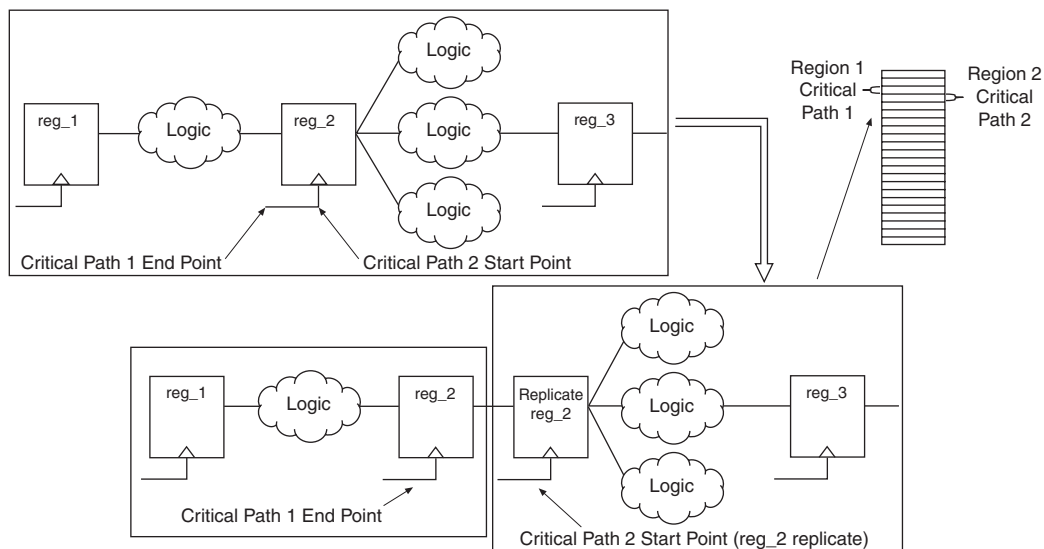
If the two critical paths share a register at the starting or ending point, assign one critical path to one region, and assign the other critical path to an adjacent region. Figure 14–9 shows two critical paths that share a register.

Figure 14–9. Two Critical Paths Sharing a Register



If the fanout is on the shared region, replicate the register and assign both registers to two regions (Figure 14–10). This is done by dragging the same register to the required regions. Entities and nodes are also replicated by performing the same procedure.

Figure 14–10. 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. On the Run menu, click **Estimate Area** to update the utilization estimates.

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 14–1 provides a description of each script’s purpose.

Tcl File	Description
<code><project name>_cons</code>	This Tcl script will create and compile a Quartus II project. The <code><project name>.tcl</code> will automatically be sourced when this script is sourced.
<code><project name></code>	This script contains forward annotation of constraint information including clock frequency, duty cycle, location, etc.
<code><project name>_rm</code>	This script removes any previous constraints from the project. The removed constraint is saved in <code><project name>_prev.tcl</code>

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.

```
<command prompt>quartus_cmd f-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.



Refer to 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 synthesizable 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 synthesizable 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.

To turn on the clear box, go to the Tools menu, and select the **MegaWizard Plug-In Manager**. Turn on the **Generate Clearbox body (for EDA tools only)** option. This option is only for certain megafunctions. If this option does not appear, then clear box models are not supported for the selected megafunction. Turning on this option causes 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 14–14](#).

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** 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, refer to 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.

Document Revision History

Table 14–2 shows the revision history for this document.

Data & Document Version	Changes Made	Summary of Changes
November 2006 v6.1.0	Added revision history to this chapter.	
May 2006 v6.0.0	Minor updates for the Quartus II software version 6.0.0.	
October 2005 v5.1.0	Chapter 14 was formerly Chapter 12 in version 5.0.	
May 2005 v5.0.0	Chapter 12 was formerly Chapter 11 in version 4.2.	
Dec. 2004 v1.1	<ul style="list-style-type: none">• Chapter 11 was formerly Chapter 12.• Updates to tables and figures.• New functionality in the Quartus II software version 4.2	
Feb. 2004 v1.0	Initial release.	

