

IC Compiler User Guide: Implementation

Version B-2008.09, September 2008

Comments?

Send comments on the documentation by going to <http://solvnnet.synopsys.com>, then clicking "Enter a Call to the Support Center."

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2008 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

"This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, Design Compiler, DesignWare, Formality, HDL Analyst, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, Direct Silicon Access, Discovery, Eclipse, Encore, EPIC, Galaxy, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM, HSIM^{plus}, i-Virtual Stepper, IICE, In-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

All other product or company names may be trademarks of their respective owners.

Contents

What's New in This Release	xxvi
About This Guide	xxvi
Customer Support.	xxix

Part I: IC Compiler Implementation Flow

1. Introduction to IC Compiler

Supported Platforms	1-2
IC Compiler Packages	1-2
User Interfaces	1-4
Methodology Overview	1-5
Speeding Up the Basic Flow	1-8

2. Working With IC Compiler

Getting Started	2-2
Starting IC Compiler	2-2
Entering icc_shell Commands	2-4
Choosing Menu Commands in GUI Windows	2-5
Interrupting or Terminating Command Processing	2-7
Opening and Closing the GUI	2-7
Using Tcl Scripts	2-8
Using Setup Files.	2-9
Exiting IC Compiler	2-10

Getting Help in IC Compiler	2-11
Getting Help on the Command Line	2-11
Finding Menu Commands and Dialog Boxes.	2-12
Viewing Man Pages	2-14
Displaying the List of Keyboard Shortcuts	2-14
Using Online Help	2-15
Searching for Text	2-16
Configuring the Help Browser	2-17
Working With the GUI	2-18
Using GUI Windows	2-18
Menu Bar	2-19
Toolbars	2-19
Status Bar	2-20
View Windows	2-21
Panels	2-22
Working in the Main Window	2-22
Working With the Console	2-24
Viewing the Session Log	2-25
Viewing the History View	2-26
Previewing icc_shell Commands in Dialog Boxes	2-28
Viewing the Physical Layout	2-29
Setting the Menu Task Mode	2-30
Setting the Primary Design	2-30
Setting GUI Preferences	2-31
Reading and Saving Designs	2-31
Reading a Design	2-32
Setting the Current Design	2-33
Saving Designs	2-34
Closing Designs	2-34
Saving or Discarding Design Changes	2-36
Archiving Designs	2-38
Using IC Compiler Command-Line Interfaces	2-41
Using Wildcard Characters	2-42
Using Aliases	2-43
Listing and Rerunning Previously Entered Commands	2-43
Redirecting and Appending Command Output	2-43

Checking the Syntax and Semantics of Your Scripts	2-45
Installation Requirements	2-46
Running the Synopsys Syntax Checker.	2-46
Limitations of the Synopsys Syntax Checker.	2-47
Bytecode-Compiled Files.	2-48
TclPro Limitations	2-48
Working With Licenses	2-48
Listing the Licenses in Use	2-49
Getting Licenses	2-49
Enabling License Queuing.	2-50
Releasing Licenses	2-51
Getting and Releasing Licenses in the GUI.	2-51

3. Preparing the Design

Setting Up the Libraries	3-2
Setting Up the Logic Libraries	3-2
Defining the Logic Library Settings.	3-3
Validating Logic Libraries	3-4
Setting Up the Physical Libraries.	3-9
Creating a Milkyway Design Library	3-10
Converting a Milkyway Database Model.	3-12
Validating a Milkyway Design Library	3-14
Verifying the Electromigration Constraints	3-15
Opening a Milkyway Design Library	3-15
Reporting on a Milkyway Design Library	3-16
Changing Physical Library Information.	3-16
Saving Physical Library Information	3-17
Verifying Library Consistency	3-18
Reading the Design	3-19
Reading a Design in Milkyway Format.	3-20
Reading a Design in ASCII Format	3-21
Annotating the Physical Data	3-22
Reading DEF Files.	3-23
Reading Floorplan Files.	3-23
Copying Physical Data.	3-24
Validating Physical Data	3-25

Preparing for Timing Analysis and RC Calculation	3-26
Setting Up the TLUPlus Files	3-26
Back-Annotating Delay or Parasitic Data	3-28
Setting Timing Constraints	3-28
Specifying the Analysis Mode	3-30
Setting the Derating Factors	3-31
Setting the Scaling Factors	3-31
Selecting the Delay Calculation Algorithm	3-32
Saving the Design	3-33
Saving the Design in Milkyway Format	3-33
Saving the Design in ASCII Format	3-33
Saving the Design for Manufacturing	3-34

4. Placement

Defining Placement Blockages	4-2
Defining Keepout Margins	4-2
Defining Global Keepout Margins	4-3
Defining Cell-Specific Keepout Margins	4-4
Inserting Port Protection Diodes	4-5
Defining Area-Based Placement Blockages	4-5
Placement Options	4-7
Congestion Options	4-7
Move Bounds	4-8
Intercell and Boundary Spacing Rules	4-9
Buffer Strategy for Optimization	4-11
Tie Cell Insertion	4-12
Magnet Placement	4-12
Placement and Optimization Attributes	4-14
Preparing for High-Fanout Net Synthesis	4-14
Performing Clock Tree Synthesis During Placement	4-15
Performing Placement and Optimization	4-16
Using Physical Optimization	4-18

Preroute RC Estimation	4-20
Modifying the RC Coefficients	4-21
Defining Net-Based Layer Constraints	4-21
Introducing Pessimism	4-22
Enabling Via Resistance Estimation	4-23
Scaling the Via Resistance Values	4-23
Manually Specifying Via Resistance Values	4-23
Analyzing Placement	4-24
Placement Area Utilization	4-24
Timing Analysis	4-26
Power Analysis	4-31
Placement Analysis Tools in the GUI	4-33
Cell Density Map	4-34
Congestion Map	4-36
Hierarchy Visual Mode	4-37
Clock Tree Visual Mode	4-38
Net Connections in Area Visual Mode	4-38
Refining Placement	4-42
5. Design for Test	
Overview of the DFT Flow	5-2
Preparing for Physical DFT Using the read_def Flow	5-3
Performing Scan Synthesis and Generating a SCANDEF File	5-4
Matching Hierarchy in Hierarchical Flow	5-4
Reading in a SCANDEF File	5-5
Generating SCANDEF Data Using the trace_scan_chain Flow	5-5
Scan Chain Consistency Checking	5-7
Removing SCANDEF Data	5-8
Optimizing Scan Chains	5-8
Placement-Aware Scan Chain Reordering	5-8
Clock-Aware Scan Reordering	5-11
Example Optimization Flow	5-11
Viewing Scan Chains	5-11

6. Power Optimization

Low-Power Placement	6-2
Reading Switching Activity Information	6-3
Leakage Power Optimization	6-3
Dynamic Power Optimization	6-5

7. Clock Tree Synthesis

Prerequisites for Clock Tree Synthesis	7-2
Design Prerequisites	7-2
Library Prerequisites	7-3
Analyzing the Clock Trees.	7-3
Defining the Clock Trees.	7-4
Cascaded Clocks.	7-5
Cascaded Generated Clocks.	7-5
Identifying the Clock Tree Endpoints	7-7
Defining the Clock Root Attributes.	7-8
Specifying Clock Tree Exceptions.	7-9
Specifying Nonstop Pins	7-11
Specifying Exclude Pins	7-12
Specifying Float Pins	7-13
Specifying Stop Pins	7-15
Specifying Don't Touch Subtrees.	7-16
Specifying Don't Buffer Nets	7-17
Specifying Don't Size Cells	7-17
Specifying the Clock Tree References	7-18
Specifying Clock Tree Synthesis Options	7-20
Specifying the Clock Tree Synthesis Goals	7-23
Setting Clock Tree Design Rule Constraints.	7-24
Setting Clock Tree Timing Goals	7-26
Setting Level Restrictions	7-26
Setting Clock Tree Routing Options.	7-27
Specifying Routing Rules	7-27
Shielding Clock Nets.	7-28
Specifying Routing Layers	7-31

Inserting Boundary Cells	7-32
Selecting the Clock Tree Clustering Algorithm.	7-33
Enabling On-Chip-Variation-Aware Clustering	7-33
Enabling Logic-Level Balancing.	7-34
Specifying Clock Tree Optimization Options.	7-36
Controlling Embedded Clock Tree Optimization	7-37
Controlling Preroute Clock Tree Optimization	7-38
Controlling Postroute Clock Tree Optimization.	7-38
Inserting User-Specified Clock Trees	7-39
Reading a Clock Configuration File	7-39
Reducing Skew Variation by Using RC Constraint-Based Clustering	7-39
Saving a Clock Configuration File	7-39
Defining the Clock Tree Structure	7-40
Handling Specific Design Characteristics	7-43
Handling Hard Macro Cells	7-44
Handling Existing Clock Trees	7-45
Identifying Existing Clock Trees	7-45
Preserving Portions of an Existing Clock Tree	7-46
Removing Clock Trees	7-46
Handling Non-Unate Gated Clocks	7-47
Handling Integrated Clock Gating Cells.	7-48
Optimizing for Clock Tree QoR	7-48
Optimizing for Power.	7-50
Balancing Multiple Clocks	7-52
Defining the Delay Balancing Buffers	7-53
Defining a Clock Balance Group.	7-53
Defining the Interlock Delay Requirements.	7-53
Hierarchical Designs Using Interface Logic Models.	7-55
Multivoltage Designs	7-56
Multicorner-Multimode Designs	7-56
Verifying the Clock Trees	7-57
Implementing the Clock Trees.	7-58
Stand-Alone Clock Tree Synthesis Capabilities.	7-64
Performing Clock Tree Power Optimization.	7-65
Performing Clock Tree Synthesis	7-65
High-Fanout Net Synthesis.	7-66

Performing Clock Tree Optimization	7-67
Running Interlock Delay Balancing	7-70
Adjusting the I/O Timing	7-70
Performing Multicorner Clock Tree Optimization.	7-71
Specifying Corners for Optimization	7-71
Specifying Corners Using Multicorner-Multimode Scenarios	7-73
Specifying Constraints for Multicorner Optimization	7-73
Setting Float Pin Constraints	7-73
Setting Skew Target Goals	7-74
Generating Multicorner Optimization Report	7-74
Performing Clock Routing After Multicorner Optimization	7-75
A Sample Script.	7-76
Using Batch Mode to Reduce Runtime.	7-77
Analyzing the Clock Tree Results	7-78
Generating Clock Tree Reports	7-79
Analyzing Clock Timing	7-81
Analyzing Clock Trees in the GUI	7-81
Viewing Clock Trees in the Layout Window	7-82
Using the Interactive Clock Tree Synthesis Window.	7-84
Using the Clock Tree Option Viewer	7-85
Viewing the Clock Tree Hierarchy	7-85
Viewing the Clock Tree Arrival Time Histogram	7-86
Viewing the Fanout or Fanin Schematic	7-87
Fine-Tuning the Clock Tree Synthesis Results	7-87
Using the Useful Skew Technique	7-88
skew_opt Details.	7-88
skew_opt Flow	7-90
Resynthesizing the Clock Trees.	7-91
Validating the Setup	7-91
Resynthesizing the Clock Trees	7-92
Modifying the Nondefault Routing Rule	7-92
Modifying Clock Trees in the GUI.	7-93
Inserting a Buffer	7-94
Removing a Buffer	7-95
Moving a Buffer or Gate	7-95
Reparenting a Subtree	7-95
Sizing a Buffer or Gate	7-96

Analyzing and Refining the Design	7-96
8. Routing	
Prerequisites for Routing	8-2
Checking Routability	8-2
Setting Up for Routing	8-3
Specifying Route Guides	8-3
Setting the Preferred Routing Direction	8-5
Using Nondefault Routing Rules	8-6
Defining Nondefault Routing Rules	8-6
Applying Nondefault Routing Rules	8-7
Reporting Nondefault Routing Rules	8-8
Specifying the Routing Layers	8-8
Specifying the Ignored Layers	8-9
Specifying Routing Layers for Specific Nets	8-9
Setting Net Aggressors	8-10
Setting Routing Types	8-11
Setting Routing Options	8-11
Setting Signal Integrity Options	8-14
Enabling Distributed Routing	8-15
Setting Up the Distributed Routing Network	8-16
Reporting Distributed Route Jobs	8-18
Cancelling Distributed Route Jobs	8-18
Routing Critical Nets	8-19
Shielding Clock Nets	8-20
Routing Signal Nets	8-21
Routing Signal Nets by Using route_opt	8-22
Setting the Routing and Optimization Strategy	8-22
Running route_opt	8-23
Routing Signal Nets by Using Automatic Routing	8-28
Running Individual Routing Steps	8-29
Global Routing	8-29
Track Assignment	8-31
Detail Routing	8-31
Search and Repair	8-32
Wire and Via Optimization	8-32
Shielding Nets	8-33

Performing ECO Routing	8-33
Reporting Cell Placement and Routing Statistics	8-35
Verifying the Routed Design	8-35
Using the signoff_drc Command	8-36
Setting Up the Hercules Environment.	8-37
Setting Up The Physical Signoff Options	8-37
Running the signoff_drc Command	8-38
Using the verify_drc Command	8-40
Using the verify_route Command	8-43
Using the Calibre Interface	8-44
Analyzing DRC Violations	8-44
Routing Nets and Buses in the GUI	8-45
Routing Single Nets	8-45
Creating Physical Buses	8-47
Manual Bus Creation	8-47
Automatic Bus Creation	8-48
Modifying Buses	8-49
Routing Buses	8-51
Creating Custom Wires	8-54
Postroute RC Extraction	8-57
Reliability Analysis Using PrimeRail	8-59

9. Chip Finishing and Design for Manufacturing

Overview	9-2
Inserting Tap Cells	9-3
Adding Tap Cell Arrays	9-3
Fixing Tap Spacing Violations	9-4
Removing Tap Cells.	9-6
Preventing Antenna Problems	9-6
Setting the Antenna Mode	9-8
Defining Antenna Rules	9-11
The Basic Antenna Rules	9-11
The Advanced Antenna Rules	9-13
Reporting Antenna Rules	9-18
Removing Antenna Rules	9-18

Checking Antenna Rules	9-18
Fixing Antenna Violations	9-19
Running Search and Repair	9-19
Inserting Diodes	9-20
Connecting Spare Diodes	9-22
Setting Detail Route Options to Control Antenna Fixing	9-24
Reducing Critical Areas	9-26
Analyzing Critical Areas	9-26
Reporting Critical Areas	9-26
Displaying Critical Area Maps	9-28
Performing Wire Spreading	9-28
Inserting Redundant Vias	9-30
Inserting Filler Cells	9-31
Inserting Standard Cell Fillers	9-31
Defining the Filler Rules	9-32
Inserting the Filler Cells	9-32
Removing Filler Cells	9-33
Inserting End Caps	9-34
Inserting Well Fillers	9-34
Inserting Pad Fillers	9-36
Performing Metal Density Filling	9-37
IC Compiler Metal Fill	9-39
Hercules Metal Fill	9-41
Performing Notch and Gap Filling	9-42
Analyzing CMP Thickness Variation Effects	9-43
Loading the Thickness Data	9-44
Displaying Chemical-Mechanical Polishing Maps	9-44
Lithography Compliance Checking	9-46
Detecting LCC Hotspots	9-47
Fixing LCC Hotspots	9-48

Part II: Advanced IC Compiler Features

10. Signal Integrity

Analyzing and Reducing Crosstalk	10-2
Setting the Signal Integrity Options	10-3
Preventing Crosstalk During Placement	10-5
Preventing Crosstalk During Clock Tree Synthesis	10-5
Preventing and Fixing Crosstalk During Routing	10-6
Preventing Crosstalk.	10-6
Fixing Crosstalk Violations	10-7
Analyzing Crosstalk	10-8
Preparing for Crosstalk Analysis.	10-8
Running Crosstalk Analysis	10-10
Preventing Crosstalk During Sign-Off Optimization	10-12
Sample Scripts.	10-13
Analyzing and Reducing Signal Electromigration	10-14
Preparing the Design for Signal Electromigration Analysis	10-15
Verifying the Electromigration Constraints	10-15
Verifying the Current Unit	10-16
Validating the Design	10-16
Loading Net Switching Information	10-16
Updating the Timing	10-17
Performing Signal Electromigration Analysis	10-18
Understanding the Violation Report.	10-18
Repairing Electromigration Violations	10-21

11. Interface Logic Modeling

Overview of Interface Logic Models (ILMs).	11-2
Description of an ILM.	11-2
Benefits of Using ILMs.	11-3
General Considerations When Using ILMs	11-4
Creating ILMs	11-7
Guidelines for Creating an ILM	11-9
Automatic Checking as Part of Creating ILMs	11-10
Controlling the Logic Included in an ILM	11-10
Controlling the Fanin and Fanout of Chip-Level Networks for Noncompact ILMs.	11-11

Controlling the Number of Latch Levels for Noncompact ILMs	11-12
Controlling Side-Load Cells	11-13
Rotated and Mirrored ILMs	11-14
Nested ILM	11-15
Creating Nested ILM	11-15
Design Information Used With an ILM	11-16
Summary of the create_ilm Command Options	11-17
Summary of the create_ilm_models Command Options	11-18
Reporting Information About ILMs	11-19
Reporting Design Information	11-20
Reporting Area Information	11-20
Propagation of ILM Information to the Top-Level Design	11-22
Creating and Propagating ILM Keepouts in IC Compiler	11-22
Manually Creating ILM Keepouts	11-23
Automatically Derived ILM Keepouts	11-24
Creating Keepout Margins	11-24
Storing Parasitic Information in the ILM View	11-25
Handling Rotated and Mirrored ILMs in the Top-Level Design	11-27
Instantiating and Using ILMs in the Top-Level Design	11-31
Procedure for Using ILMs at the Top Level	11-32
Viewing ILMs in the IC Compiler GUI	11-33
How the place_opt Command Handles ILMs	11-33
How the clock_opt Command Handles ILMs	11-34
How the route_opt Command Handles ILMs	11-36
Hierarchical Signal Integrity Flow With ILMs	11-36
Multicorner-Multimode Scenarios and ILMs	11-37
Creating ILMs for Multicorner-Multimode Usage	11-38
Using ILMs in Multicorner-Multimode Scenarios	11-39
Summary of Commands for ILMs	11-40

12. Physical Datapath With Relative Placement

Introduction to Physical Datapath With Relative Placement	12-3
Benefits of Relative Placement	12-4
Limitations of Relative Placement	12-4

Flow for Relative Placement	12-5
Methodology for the Relative Placement Flow.	12-6
Sample Script for a Relative Placement Flow	12-7
Considerations for Using Relative Placement.	12-8
Creating Relative Placement Groups	12-9
Anchoring Relative Placement Groups	12-11
Adding Objects to a Group	12-13
Adding Leaf Cells.	12-14
Specifying Orientation for Leaf Cells	12-14
Aligning Leaf Cells Within a Column.	12-15
Automatic Orientation of Relative Placement Groups Based on Data Flow . . .	12-19
Adding Relative Placement Groups.	12-21
Benefits of Hierarchical Relative Placement Groups	12-22
Including Relative Placement Groups.	12-22
Instantiating Relative Placement Groups	12-24
Using Hierarchical Relative Placement for Straddling	12-26
Using Hierarchical Relative Placement for Compression	12-28
Effect of Ungrouping on Hierarchical Relative Placement	12-29
Effect of Uniquifying on Hierarchical Relative Placement.	12-31
Adding Keepouts	12-32
Movement Control When Legalizing Relative Placement Groups	12-34
Placement of Relative Placement Groups	12-34
Preserving Relative Placement Cells During Optimization	12-35
Performing Relative Placement in a Design Containing Obstructions.	12-35
Performing Relative Placement in a Design Containing Tap Cells.	12-36
Using Move Bounds to Constrain Relative Placement	12-38
Adding Incremental Relative Placement Groups.	12-38
Postplacement Optimization of Relative Placement Groups.	12-39
Analyzing the Relative Placement Results	12-40
Checking Relative Placement Constraints.	12-40
Locating a Relative Placement Group	12-41
Working With Relative Placement Groups in the GUI.	12-42
Using the Relative Placement Hierarchy Browser	12-42
Using the Layout View	12-43
Querying Relative Placement Groups	12-43

Saving Relative Placement Information	12-45
Ignoring Relative Placement Constraints	12-46
Removing Relative Placement Groups	12-46
Changing Relative Placement Information	12-47
Reporting the Attributes of a Relative Placement Group	12-47
Changing the Attributes of a Relative Placement Group	12-48
Setting Relative Placement Group Attributes	12-48
Removing Relative Placement Group Attributes.	12-50
Renaming a Group	12-51
Removing Objects From a Group	12-51
Changing Specific Objects Within a Group	12-52
Deriving Relative Placement Groups	12-52
Extracting a Relative Placement Group	12-53
Ordering Extracted Relative Placement Groups	12-54
Summary of Relative Placement Commands	12-55

13. Multivoltage Design Flow

Multivoltage Designs	13-2
Power Domains	13-2
Defining Power Domains	13-3
Voltage Areas	13-4
Associating Power Domains and Voltage Areas	13-4
Level-Shifter and Isolation Cells	13-5
Basic Library Requirements for Multivoltage Designs	13-6
Physical Synthesis.	13-6
High-Level Design Flow for Multivoltage Designs	13-6
Target Library Subsetting.	13-8
UPF Flow for Multivoltage Designs	13-9
Defining Power Domains and Supply Networks	13-11
Power Domains	13-12
Hierarchy and Scope	13-12
Creating Power Domains	13-13
Power and Ground Supply Network.	13-15
Creating Supply Ports.	13-15

Creating Supply Nets	13-17
Connecting Supply Nets	13-17
Specifying the Primary Supply Net for a Power Domain.	13-18
Creating a Power Switch	13-18
Adding Port State Information to Supply Ports	13-19
Creating a Power State Table	13-19
Defining the States of the Supply Nets	13-20
Connecting Power and Ground Nets	13-21
Defining and Using Voltage Areas	13-22
Creating Voltage Areas	13-24
Updating Voltage Areas in a Design	13-25
Including Guard Bands With Voltage Areas.	13-26
Handling Nested Voltage Areas.	13-28
Voltage Area Support for Macros.	13-29
Defining Always-On Power Wells Within Voltage Areas	13-30
Removing Voltage Areas From the Design	13-30
Reporting on the Voltage Areas of a Design	13-30
Handling Level Shifters and Isolation Cells.	13-31
Level-Shifter and Isolation Cell Requirements.	13-31
Level-Shifter Commands	13-32
Level-Shifter Threshold (Automatic and User-Specified).	13-33
Inserting Buffer-Type Level Shifters.	13-34
Inserting Isolation Cells and Enable-Type Level Shifters.	13-37
Placing Level Shifters	13-37
Handling Always-On Logic	13-38
Marking Always-On Cells of Pins.	13-38
Rules for Marking Always-On Pins	13-39
Marking Always-On Library Pins.	13-39
Marking Pass-Gate Library Pins.	13-39
Marking Leaf Cell Instance Pins As Always-On	13-40
Marking Library Cells for Always-On Optimization	13-40
Reporting Commands	13-40
Sample IC Compiler Script for UPF Flow	13-42

Voltage-Area-Aware Capabilities	13-44
Automatic High-Fanout Synthesis	13-45
Virtual Hierarchy Routing	13-45
Maximum Net Length Optimization	13-45
Voltage-Area-Based Optimization	13-46
Voltage-Area-Based Clock Tree Synthesis and Optimization	13-46
Voltage-Area-Based Global Routing	13-46
Voltage-Area-Based Power and Ground Net Associations	13-47
Voltage-Area-Based Standard-Cell Filler Cell Insertion	13-47
Interface Logic Model Hierarchical Flow	13-47
Multivoltage Relative Placement Capability	13-48
Hierarchical Signal Integrity Flow	13-48
Multicorner-Multimode Designs	13-49
Placing and Optimizing the Design	13-49
Handling Isolation Cells and Always-On Clock Paths During Clock Tree Synthesis	13-50
Checks and Reporting Functions for Multivoltage Designs	13-51
Non-UPF Flow For Multivoltage Designs	13-52
Using Power Domains	13-52
Power Domain Commands	13-53
Connecting Power and Ground Nets	13-53
Sample IC Compiler Script For Non-UPF Flow	13-54

14. Using Multicorner-Multimode Technology

Basic Multicorner-Multimode Concepts	14-2
Scenario Definition	14-2
Concurrent Multicorner-Multimode Optimization and Timing Analysis	14-2
Basic Multicorner-Multimode Flow	14-3
Setup Considerations	14-4
Multicorner Library Considerations	14-5
Link Libraries With Equal Nominal PVT Values	14-5
Link Library Example	14-6
Distinct PVT Requirements	14-7
Unsupported k-factors	14-8
Automatic Detection of Driving Cell Library	14-8

Relating the Minimum Library to the Maximum Library	14-9
Unique Identification of Libraries Based on File Names	14-10
Using Multicorner-Multimode Commands.	14-10
Scenario Management.	14-10
Multicorner-Multimode Optimization	14-13
Scaling Resistance and Capacitance	14-14
Filtering Coupling Capacitances	14-15
Using Scenario Reduction	14-15
Preroute Hold Fixing	14-16
Using the Clock Tree Synthesis Scenario	14-17
Using Interface Logic Models With Multicorner-Multimode Scenarios.	14-17
Reporting Commands for Multicorner-Multimode	14-18
report_scenarios Command	14-18
Reporting Commands That Support the -scenario Option	14-19
Commands That Report the Current Scenario	14-19
Reporting Examples	14-21
Multicorner-Multimode Script Example	14-24

15. ECO Flow

ECO Flow Overview	15-2
ECO Key Features	15-2
ECO Limitations	15-3
Inserting Spare Cells	15-3
Inserting Spare Cells Using a Verilog Netlist	15-4
Automatically Distribute the Spare Cells During Placement.	15-5
Manually Distribute the Spare Cells After Placement	15-6
Manually Place the Spare Cells Before Placement	15-7
Inserting Spare Cells Using insert_spare_cells	15-8
Creating ECO Changes and Updating the Design	15-8
Updating With a Verilog Netlist	15-8
Power and Ground (PG) Support	15-10
ECO on Physical-Only Cells	15-10

Updating With Tcl Commands	15-10
Port Operations.	15-10
Net Operations	15-11
Connection Operations	15-11
Instance Operations	15-11
Updating Placement	15-12
Updating Routing	15-13
16. Sign-Off Driven Design Closure	
IC Compiler Sign-Off Driven Design Closure Overview	16-2
Key Features.	16-2
IC Compiler Sign-Off Driven Design Closure Methodology	16-4
Automatic IC Compiler Sign-Off Driven Design Closure	16-4
Manual IC Compiler Sign-Off Driven Design Closure	16-6
Usage Guidelines	16-9
PrimeTime and Star-RCXT Version Requirements	16-10
Entry Requirements for IC Compiler Sign-Off Driven Design Closure	16-10
General Guidelines	16-11
Scenario Differences Between IC Compiler and PrimeTime	16-13
Multicorner-Multimode Guidelines	16-14
PrimeTime and Star-RCXT Customizing Options	16-17
Limitations.	16-21
Troubleshooting and Problem Solving	16-21
Appendix A. Using GUI Tools	
Viewing a Design	A-2
Selecting a Mouse Tool	A-3
Previewing Objects in a Layout View	A-5
Selecting Objects.	A-6
Highlighting Objects.	A-8
Viewing Object Information	A-11
Magnifying and Traversing the Design.	A-12
Following Selected Objects	A-14
Reusing Zoom and Pan Settings	A-14

Drawing Rulers	A-15
Redrawing the Active View	A-17
Examining Design Objects	A-17
Selecting Objects by Name	A-18
Selecting Timing Paths	A-20
Viewing and Editing Object Properties	A-21
Exploring the Design Hierarchy	A-22
Viewing Design and Path Schematics	A-23
Viewing Reports.	A-25
Visualizing the Physical Design	A-26
Navigating Through Layout Views	A-27
Displaying Grid Lines	A-28
Setting and Saving View Properties	A-28
Adjusting the Color Brightness	A-32
Enhancing Image Accuracy	A-32
Displaying Cell Orientations.	A-33
Examining ILM Blocks	A-34
Viewing Design Overlays	A-34
Saving an Image of the Layout View	A-36
Using Stroke Activated Commands	A-36
Analyzing the Physical Layout	A-38
Using Visual Analysis Modes.	A-38
Displaying Flylines	A-39
Displaying Net Connectivity	A-40
Using Visual Modes	A-41
Using Map Modes.	A-43
Exploring Relative Placement Groups	A-45
Analyzing Signal Integrity	A-46
Examining Routing and Verification Errors	A-47
Filtering Errors in the Error List.	A-50
Selecting Errors in the Layout View	A-51
Examining Nets Associated With Detail Routing Errors	A-52
Examining Net Nodes Associated With Open Locator Errors.	A-52
Examining Errors in an Area.	A-53
Saving the Error List Information in a File.	A-53

Editing the Physical Layout	A-54
Editing a Design.	A-54
Fixing and Unfixing Objects for Edits	A-60
Changing the Way Objects Snap to a Grid.	A-61
Undoing and Redoing Edit Changes	A-62
Creating Objects	A-62
Analyzing Clock Tree Synthesis Results.	A-63
Opening an Interactive Clock Tree Synthesis Window.	A-64
Exploring Clock Tree Structures	A-65
Expanding or Collapsing Fanout Levels	A-66
Highlighting a Selected Object	A-67
Changing the Timing Mode.	A-67
Highlighting the Critical Paths	A-68
Analyzing Timing Paths	A-68
Opening a Timing Analysis Window	A-69
Viewing Timing Path Details	A-69
Using Timing Analysis Views.	A-70
Viewing Histograms	A-70
Examining Path Profiles	A-72
Inspecting a Timing Path	A-72

Appendix B. Defining and Analyzing Clocks

Clock Overview	B-2
Specifying Clock Characteristics.	B-2
Creating a Clock	B-3
Creating a Virtual Clock.	B-4
Creating Internally Generated Clocks	B-4
Creating a Divide-By Generated Clock.	B-5
Shifting the Edges of a Generated Clock	B-9
Restricting Clock Source Latency Calculation To Combinational Paths	B-11
Creating Generated Clocks Based on a Non-Unate Master Clock.	B-12
Multiple Clocks at the Source Pin.	B-15
Using Pulse Clocks	B-15
Reporting Clock Information	B-18
Selecting Clock Objects.	B-18
Applying Commands to All Clocks.	B-19

Removing Clock Objects	B-19
Specifying Clock Network Effects	B-20
Setting Clock Latency	B-20
Setting Clock Uncertainty	B-23
Limitation When Specifying Negative Uncertainty	B-25
Removing Clock Uncertainty	B-25
Estimating Clock Pin Transition Time	B-25
Removing Estimated Clock Pin Transition Time	B-25
Using Multiple Clocks	B-26
Synchronous Clocks	B-27
Asynchronous Clocks	B-29
Exclusive Clocks	B-29
Logically Exclusive Clocks	B-30
Physically Exclusive Clocks	B-33
Working With Clock Groups	B-35
Propagating Clock Paths Through a Non-Unate Clock Network.	B-36
Specifying Clock Sense	B-38
Defining Data-to-Data Checks	B-39
Specifying Data-to-Data Checks	B-40
Generating Timing Reports for Data Checks.	B-43
Removing Data Checks	B-44
Data Checks and Clock Domains	B-44
Library-Based Data Checks	B-44
Specifying Clock-Gating Setup and Hold Checks.	B-45
Disabling or Restoring Clock-Gating Checks.	B-48

Index

Preface

This preface includes the following sections:

- [What's New in This Release](#)
- [About This Guide](#)
- [Customer Support](#)

What's New in This Release

Information about new features, enhancements, and changes, along with known problems and limitations and resolved Synopsys Technical Action Requests (STARs), is available in the *IC Compiler Release Notes* in SolvNet.

To see the *IC Compiler Release Notes*,

1. Go to the Download Center on SolvNet located at the following address:

<http://solvnet.synopsys.com/DownloadCenter>

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

2. Select IC Compiler, and then select a release in the list that appears.

About This Guide

The IC Compiler tool provides a complete netlist-to-GDSII or netlist-to-clock-tree-synthesis design solution, which combines proprietary design planning, physical synthesis, clock tree synthesis, and routing for logical and physical design implementations throughout the design flow.

This guide describes the IC Compiler implementation and integration flow; the companion volume, *IC Compiler User Guide: Design Planning*, describes the design planning flow.

Audience

This user guide is for design engineers who use IC Compiler to implement designs.

To use IC Compiler, you need to be skilled in physical design and design synthesis and be familiar with the following:

- Physical design principles
- The Linux or UNIX operating system
- The tool command language (Tcl)

Related Publications

For additional information about IC Compiler, see Documentation on the Web, which is available through SolvNet at the following address:

<http://solvnet.synopsys.com/DocsOnWeb>

You might also want to refer to the documentation for the following related Synopsys products:

- Design Compiler
- Milkyway Environment

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
<code>Courier</code>	Indicates command syntax.
<i>Courier italic</i>	Indicates a user-defined value in Synopsys syntax, such as <i>object_name</i> . (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
Courier bold	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[]	Denotes optional parameters, such as <code>pin1 [pin2 ... pinN]</code>
	Indicates a choice among alternatives, such as <code>low medium high</code> (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
—	Connects terms that are read as a single term by the system, such as <code>set_annotated_delay</code>
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

<http://solvnet.synopsys.com>

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to <http://solvnet.synopsys.com> (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
 - E-mail support_center@synopsys.com from within North America.
 - Find other local support center e-mail addresses at http://www.synopsys.com/support/support_ctr
- Telephone your local support center.
 - Call (800) 245-8005 from within the continental United States.
 - Call (650) 584-4200 from Canada.
 - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr

Preface

xxx

www.cadfamily.com EMail:cadserv21@hotmail.com

The document is for study only,if tort to your rights,please inform us,we will delete

Part I: IC Compiler Implementation Flow

1

Introduction to IC Compiler

IC Compiler is a single, convergent netlist-to-GDSII or netlist-to-clock-tree-synthesis design tool for chip designers developing very deep submicron designs. It takes as input a gate-level netlist, a detailed floorplan, timing constraints, physical and timing libraries, and foundry-process data, and it generates as output either a GDSII-format file of the layout or a Design Exchange Format (DEF) file of placed netlist data ready for a third-party router. IC Compiler can also output the design at any time as a binary Synopsys Milkyway database for use with other Synopsys tools based on Milkyway or as ASCII files (Verilog, DEF, and timing constraints) for use with tools not from Synopsys.

This chapter contains the following sections:

- [Supported Platforms](#)
- [IC Compiler Packages](#)
- [User Interfaces](#)
- [Methodology Overview](#)

Supported Platforms

You can use IC Compiler on the following platforms and operating systems:

- AMD64
- sparc64

Note:

If you run the platform-independent version of `icc_shell` (`$icc_root/bin/icc_shell`), the default behavior is to run the 32-bit binary. Because a 32-bit binary does not exist for the sparc platform, `icc_shell` terminates with an error. To prevent this problem, you must explicitly specify the `-64bit` option when running the platform-independent version of `icc_shell` on a sparc64 platform.

This requirement does not apply to the platform-specific version of `icc_shell` (`$icc_root/sparc64/syn/bin/icc_shell`).

- Linux32
- Suse 32
- Suse 64

IC Compiler Packages

IC Compiler has four packages: IC Compiler, IC Compiler-XP, IC Compiler-PC, and IC Compiler-DP. If you are using IC Compiler for a netlist-to-clock-tree-synthesis design flow and will perform routing with another tool, you need the IC Compiler-PC package. If you are also performing routing (netlist-to-GDSII flow), you need the IC Compiler package. If you are designing at process nodes of 130 nm and above and need a complete netlist-to-GDSII flow, you might consider the IC Compiler-XP package. If you are using IC Compiler only to perform design planning, use the IC Compiler-DP package.

Note:

The design planning functionality is described in the *IC Compiler User Guide: Design Planning*.

Low-power design capabilities, including multivoltage and multithreshold-CMOS (MTCMOS), are available only in the IC Compiler, IC Compiler-PC, and IC Compiler-DP packages.

The following advanced features are available only in the IC Compiler and IC Compiler-PC packages: relative placement, power-aware placement, low power clock tree synthesis, clock-sink clustering based on on-chip variation (OCV) analysis, and concurrent multicorner-multimode (MCMM) optimization.

The following advanced features are available only in the IC Compiler package: sign-off driven closure with incremental PrimeTime and Star-RCXT, Zroute, advanced design rules support for process nodes below 130nm, and advanced design-for-yield (DFY) capabilities.

Each package offers a different set of licenses. Each license allows you to perform specific tasks within the IC Compiler product. [Table 1-1](#) lists the licenses provided with each package.

Table 1-1 Licenses Required for Each IC Compiler Package

Licenses	IC Compiler package	IC Compiler-XP package	IC Compiler-PC package	IC Compiler-DP package
Galaxy-ICC	X	X	X	X
Galaxy-Common	X	X	X	X
Milkyway-Interface	X	X	X	X
Galaxy-FP	X	X	X	X
Galaxy-FP-Hier				X
Galaxy-FP-MV				X
Galaxy-FP-AdvCTS				X
Galaxy-FP-AdvTech				X
Galaxy-PSYN	X	X	X	
Galaxy-MultiRoute4	X ¹	X ¹		
Galaxy-MultiRoute8	X ²	X ²		
Galaxy-Power	X	X		
Galaxy-PNR	X	X		
Galaxy-AdvCTS	X		X	
Galaxy-AdvTech	X		X	
Galaxy-MV	X		X	
Galaxy-Prototype	X		X	
Galaxy-AdvRules	X			
Galaxy-DFT	X			
Galaxy-DFY	X			
Galaxy-IU	X			

Table 1-1 Licenses Required for Each IC Compiler Package (Continued)

Licenses	IC Compiler package	IC Compiler-XP package	IC Compiler-PC package	IC Compiler-DP package
Galaxy-Zroute	X			
Power Compiler			X ³	

1. This optional license is required to perform distributed routing with three or four CPUs.
2. This optional license is required to perform distributed routing with more than four CPUs.
3. This optional license is required to use power optimization features.

User Interfaces

IC Compiler provides two user interfaces:

- Shell interface (icc_shell) – The IC Compiler command-line interface is used for scripts, batch mode, and push-button operations.
- Graphical user interface (GUI) – The IC Compiler graphical user interface is an advanced analysis and physical editing tool. IC Compiler can perform certain tasks, such as very accurately displaying the design and providing visual analysis tools, only from the GUI. The look and feel of the IC Compiler GUI is consistent with the look and feel of other Synopsys GUIs.

Help is available for both interfaces (see [“Viewing Man Pages” on page 2-14](#) and [“Getting Help in IC Compiler” on page 2-11](#)).

IC Compiler uses the tool command language (Tcl), which is used in many applications in the EDA industry. Using Tcl, you can extend the IC Compiler command language by writing reusable procedures and scripts (see the *Using Tcl With Synopsys Tools* manual).

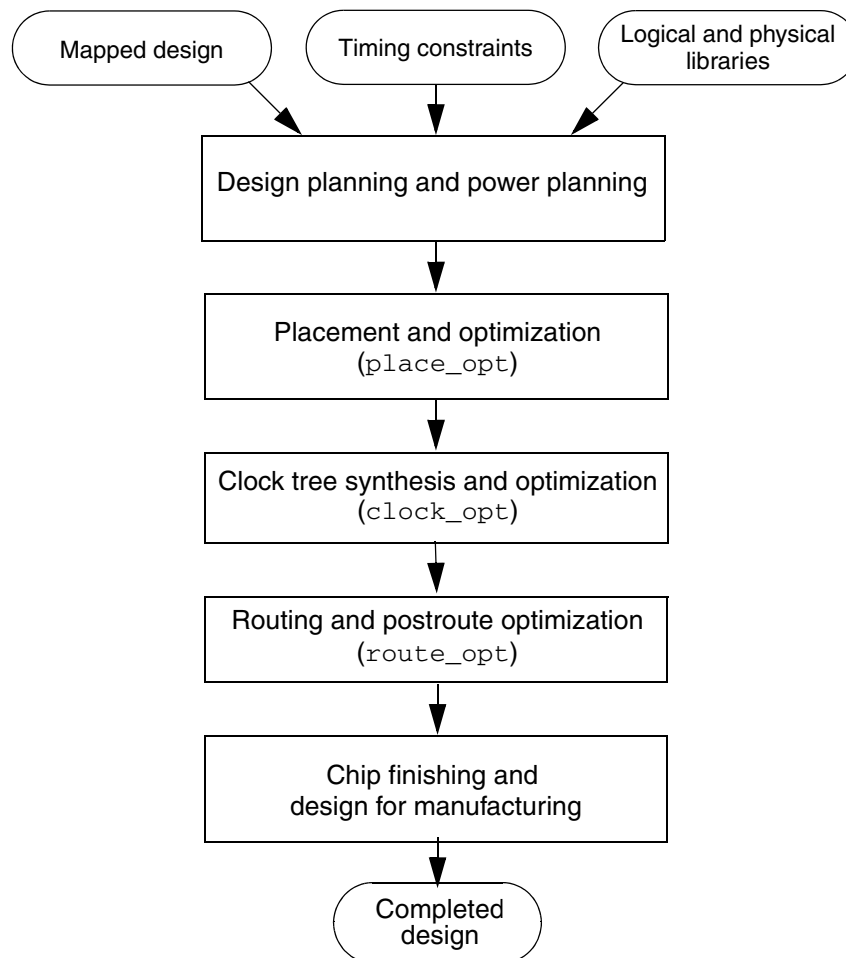
Methodology Overview

The IC Compiler design flow is an easy-to-use, single-pass flow that provides convergent timing closure. [Figure 1-1](#) shows the basic IC Compiler design flow, which is centered around three core commands that perform placement and optimization (`place_opt`), clock tree synthesis and optimization (`clock_opt`), and routing and postroute optimization (`route_opt`).

Note:

If you have the IC Compiler-PC package, you cannot perform routing and postroute optimization or chip finishing and design for manufacturing tasks in IC Compiler.

Figure 1-1 IC Compiler Design Flow



For most designs, the `place_opt`, `clock_opt`, and `route_opt` steps are preset for optimal results. IC Compiler also provides additional placement, clock tree synthesis, and routing technologies, as well as extended physical synthesis tools, that you can use to further improve the quality of results for your design.

To run the IC Compiler design flow,

1. Set up the libraries and prepare the design data, as described in [Chapter 3, “Preparing the Design.”](#)

2. Perform design planning and power planning.

When you perform design planning and power planning, you create a floorplan to determine the size of the design, create the boundary and core area, create site rows for the placement of standard cells, set up the I/O pads, and create a power plan.

For more information about design planning and power planning, see the *IC Compiler User Guide: Design Planning*.

3. Perform placement and optimization.

To perform placement and optimization, use the `place_opt` core command (or choose Placement > Core Placement and Optimization in the GUI).

The `place_opt` core command addresses and resolves timing closure for your design. This iterative process uses enhanced placement and synthesis technologies to generate legalized placement for leaf cells and an optimized design. You can supplement this functionality by optimizing for power, recovering area for placement, minimizing congestion, and minimizing timing and design rule violations.

For more information about placement and optimization, see [Chapter 4, “Placement.”](#)

4. Perform clock tree synthesis and optimization.

To perform clock tree synthesis and optimization, use the `clock_opt` core command (or choose Clock > Core CTS and Optimization in the GUI).

IC Compiler clock tree synthesis and embedded optimization solve complicated clock tree synthesis problems, such as blockage avoidance and the correlation between preroute and postroute data. Clock tree optimization improves both clock skew and clock insertion delay by performing buffer sizing, buffer relocation, gate sizing, gate relocation, level adjustment, reconfiguration, delay insertion, dummy load insertion, and balancing of interclock delays.

For more information about clock tree synthesis and optimization, see [Chapter 7, “Clock Tree Synthesis.”](#)

5. Perform routing and postroute optimization.

To perform routing and postroute optimization, use the `route_opt` core command (or choose Route > Core Routing and Optimization in the GUI).

As part of routing and postroute optimization, IC Compiler performs global routing, track assignment, detail routing, search and repair, topological optimization, and engineering change order (ECO) routing. For most designs, the default routing and postroute optimization setup produces optimal results. If necessary, you can supplement this functionality by optimizing routing patterns and reducing crosstalk or by customizing the routing and postroute optimization functions for special needs.

For more information about routing and postroute optimization, see [Chapter 8, “Routing.”](#)

6. Perform chip finishing and design for manufacturing tasks, as described in [Chapter 9, “Chip Finishing and Design for Manufacturing.”](#)

IC Compiler provides chip finishing and design for manufacturing and design for yield capabilities that you can apply throughout the various stages of the design flow to address process design issues encountered during chip manufacturing.

7. Save the design.

Save your design in the Milkyway format. This format is the internal database format used by IC Compiler to store all the logical and physical information about a design. For more information, see [“Saving the Design in Milkyway Format” on page 3-33.](#)

The IC Compiler reference methodology (SolvNet article 021624) provides scripts that implement this flow.

Speeding Up the Basic Flow

If runtime is a concern and a slight degradation in quality of results is acceptable, you can run a low-effort quick flow that uses a low effort for placement and optimization and a low effort for routing and postroute optimization. This quick flow can save 30 to 50 percent in runtime, but it can also slightly degrade the quality of results.

To run a quick flow, run the `set_fast_mode true` command before running the core commands. When you enable fast mode,

- The `place_opt` command uses low effort (and ignores the `-effort` option) and invokes congestion removal.

To further reduce runtime, you can disable congestion removal by specifying the `-no_congestion` option.

Note:

The quality of results after running `place_opt` with low effort can be disappointing, but it should improve after running clock tree synthesis.

- There is no impact on the `clock_opt` command.
- The `route_opt` command uses low effort (and ignores the `-effort` option).

2

Working With IC Compiler

IC Compiler provides a flexible working environment with both a shell command-line interface and a graphical user interface. The shell command-line interface, `icc_shell`, is always available during an IC Compiler session. You can start or exit a session in either the GUI or `icc_shell`, and you can open or close the GUI at any time during a session. This chapter describes standard tasks for working in the IC Compiler environment.

This chapter contains the following sections:

- [Getting Started](#)
- [Getting Help in IC Compiler](#)
- [Working With the GUI](#)
- [Reading and Saving Designs](#)
- [Archiving Designs](#)
- [Using IC Compiler Command-Line Interfaces](#)
- [Checking the Syntax and Semantics of Your Scripts](#)
- [Working With Licenses](#)

Getting Started

The `icc_shell` command-line interface is a text-only environment in which you enter commands at the command-line prompt. The IC Compiler GUI provides both menu commands and a command-line interface. The GUI also provides tools you can use to visualize design data and analyze results. You can perform any task in the GUI that you can perform in `icc_shell`. You can use menus and dialog boxes to preview or run `icc_shell` commands.

For information about using `icc_shell` and the IC Compiler GUI, see the following sections:

- [Starting IC Compiler](#)
- [Entering `icc\_shell` Commands](#)
- [Choosing Menu Commands in GUI Windows](#)
- [Interrupting or Terminating Command Processing](#)
- [Opening and Closing the GUI](#)
- [Using Setup Files](#)
- [Using Tcl Scripts](#)
- [Exiting IC Compiler](#)

Starting IC Compiler

IC Compiler operates in the X Windows environment on UNIX or Linux. Before starting IC Compiler, make sure your `$SYNOPSYS` variable is set and the path to the bin directory is included in your `$PATH` variable. Before starting the GUI, make sure your `$DISPLAY` environment variable is set to the name of your UNIX or Linux system display.

Note:

When you start IC Compiler, it automatically executes commands in the IC Compiler setup files. Setup commands perform basic tasks, such as initializing parameters and variables, declaring design libraries, and setting GUI options. For more details, see [“Using Setup Files” on page 2-9](#).

You start IC Compiler by entering the `icc_shell` command in a UNIX or Linux shell:

```
% icc_shell
```

By default, this command starts the tool in the command-line interface (`icc_shell`) in Unified Power Format (UPF) mode.

- You can start the tool in the GUI by specifying the `-gui` option.

```
% icc_shell -gui
```

- If you are not using UPF commands, you must indicate this by specifying the `-non_upf_mode` option.

```
% icc_shell -non_upf_mode
```

When you start the tool in a package other than the IC Compiler package, you must indicate which package you are using. [Table 2-1](#) shows the commands for starting the various versions of IC Compiler. For more information about these IC Compiler packages, see [“IC Compiler Packages” on page 1-2](#).

Table 2-1 Commands for Starting IC Compiler

IC Compiler package	IC Compiler startup command
IC Compiler	<code>icc_shell [-gui] [-non_upf_mode]</code>
IC Compiler-XP	<code>icc_shell -xp_mode [-gui] [-non_upf_mode]</code>
IC Compiler-PC	<code>icc_shell -psyn_mode [-gui] [-non_upf_mode]</code>
IC Compiler-DP	<code>icc_shell -dp_mode [-gui] [-non_upf_mode]</code>

When you start the command-line interface, the `icc_shell>` prompt appears in the UNIX or Linux shell. If you need to use the GUI after starting the command-line interface, enter the `gui_start` command at the `icc_shell>` prompt.

When you start the GUI, the `icc_shell>` prompt appears in the UNIX or Linux shell, and the IC Compiler main window opens. By default, the console appears attached (docked) to the main window above the status bar.

You can include other options on the command line when you start IC Compiler. For example, you can use

- `-f script_file_name` to execute a script
- `-x command` to execute an IC Compiler shell command
- `-display terminal_name` to display the GUI on a different terminal from the one you set with the `DISPLAY` environment variable
- `-h` to display a list of the available options (without starting IC Compiler)

For a complete list of startup options, see the *IC Compiler Quick Reference*.

At startup, IC Compiler performs the following tasks:

1. Creates a command log file
2. Reads and executes the setup files
3. Executes any script files or commands specified by the `-x` and `-f` options, respectively, on the command line
4. Displays the program header and `icc_shell>` prompt in the shell in which you started `icc_shell`

The program header lists all IC Compiler packages for which your site is licensed.

Entering `icc_shell` Commands

You interact with IC Compiler by using `icc_shell` commands, which are based on the tool command language (Tcl) and include certain command extensions needed to implement specific IC Compiler functionality. The IC Compiler command language provides capabilities similar to UNIX command shells, including variables, conditional execution of commands, and control flow commands. You can

- Enter individual commands interactively at the `icc_shell>` prompt in `icc_shell`
- Enter individual commands interactively on the console command line in the GUI
- Run one or more Tcl command scripts, which are text files containing `icc_shell` commands

You enter commands in `icc_shell` the same way you enter commands in a standard UNIX or Linux shell. When the GUI is open, you can enter commands on the console command line to supplement the IC Compiler commands available through the menu interface. For details about running Tcl scripts, see [“Using Tcl Scripts” on page 2-8](#).

To enter a command in `icc_shell`,

1. Type the command on the command line.
2. Press Return.

To enter a command on the console command line,

1. Click anywhere on the console to make sure the command line is active.
2. Type the command.
3. Click the `icc_shell>` prompt button, or press Return.

IC Compiler echoes the command output (including processing messages and any warnings or error messages) in both `icc_shell` and the console log view.

You can display the options used with an `icc_shell` command by entering the command name and `-help` on the `icc_shell` command line. For example, to see the options used with the `place_opt` command, enter the following command:

```
icc_shell> place_opt -help
```

For more details, see [“Getting Help on the Command Line” on page 2-11](#).

When entering a command, an option, or a file name, you can minimize your typing by pressing the Tab key when you have typed enough characters to specify a unique name, IC Compiler provides the remaining characters. If the characters you typed could be used for more than one name, IC Compiler lists the qualifying names, from which you can use the arrow keys and the Enter key to select the appropriate name.

If you need to reuse a command from the output for a command-line interface, you can copy and paste the portion by selecting it, moving the pointer to the `icc_shell` command line, and clicking with the middle mouse button.

For more details about entering `icc_shell` commands, see [“Using IC Compiler Command-Line Interfaces” on page 2-41](#).

Choosing Menu Commands in GUI Windows

The IC Compiler GUI provides menu commands and dialog boxes for most graphic features, such as viewing, selecting, and highlighting objects in layout views. In addition, the GUI provides menu and dialog box equivalents for many `icc_shell` commands. Menu commands are grouped by function on the menus in each GUI window. You can add commands to these menus and even create new menus.

To choose a command on a menu bar menu,

- Click the menu name to open the menu, and click the command name on the menu.

Some frequently used menu commands are also available on pop-up menus for individual views.

To choose a command on a pop-up menu,

- Move the pointer over the object of interest, right-click to display the menu, and click the command name.

A menu command can perform an immediate operation, display a submenu, or display a dialog box.

- Menu commands that display a submenu are followed by a right-pointing arrow.
- Menu commands that open a dialog box that requires a response before performing an operation are followed by an ellipsis (...). These commands open a dialog box to prompt you for the information.

Dialog boxes that require a response before performing an operation contain OK and Cancel buttons and sometimes an Apply button. After selecting options or entering information in the dialog box, you respond by clicking OK or Apply.

- Menu commands without an arrow or ellipsis either perform an immediate operation or open a dialog box that performs immediate operations.

Dialog boxes that perform immediate operations usually display options and contain a Close button. You select options that perform operations and click Close to close the dialog box.

When you choose a command or select a dialog box option that performs an immediate action, or when you click OK or Apply in a dialog box that requires a response, IC Compiler displays command output (including processing messages and any warnings or error messages) in both `icc_shell` and the console log view.

This manual identifies commands with their menus in the following formats:

- *Menu > Command*
- *Menu > Submenu > Command*

where

- *Menu* represents a menu title on the menu bar.
- *Submenu* represents a menu command that displays a submenu. Some submenus contain commands that open other submenus.
- *Command* represents a command that performs an operation or displays a dialog box.

Each menu command can also be activated by a shortcut key, which is indicated on the menu by an underscore (_) below a letter in the command and, if needed, the name of the modifier key (Shift or Control) to the right of the command name. You can view a list of shortcut keys by choosing Help > Report Hotkey Bindings. For details, see [“Displaying the List of Keyboard Shortcuts” on page 2-14](#).

Interrupting or Terminating Command Processing

If you enter the wrong options for a command or enter the wrong command, you can interrupt command processing and remain in `icc_shell`. To interrupt or terminate a command, press Control-c.

Some commands and processes, such as `update_timing`, cannot be interrupted. To stop these commands or processes, you must terminate `icc_shell` at the system level. When you terminate a process or the shell, no data is saved.

When you use Control-c, keep the following points in mind:

- If a script file is being processed and you interrupt one of its commands, the script processing is interrupted and no further script commands are processed.
- If you press Control-c three times before a command responds to your interrupt, `icc_shell` is interrupted and exits with this message:

```
Information: Process terminated by interrupt.
```

This behavior has a few exceptions, which are documented in the man pages for the applicable commands.

Opening and Closing the GUI

If you start `icc_shell` without the GUI, you can open the GUI by entering the `gui_start` command at the `icc_shell>` prompt.

To open the GUI,

- Enter the `gui_start` command.

```
icc_shell> gui_start
```

If you want to run a setup script before opening the GUI, use the `-file` option. For example,

```
icc_shell> gui_start -file my_gui_setup.tcl
```

If you open a design in `icc_shell` and then open the GUI, IC Compiler automatically opens a layout window and displays the design in the layout view. This is the primary design in the window. If you have multiple designs open when you open the GUI, IC Compiler opens a separate layout window for each design. The active window (the window that has the mouse focus) is the window with the current design.

You can open or close the GUI at any time during the session. For example, if you need to conserve system resources, you can close the GUI and continue the session in `icc_shell`.

To close the GUI, do one of the following:

- Choose File > Close.
- Enter the `gui_stop` command.

```
icc_shell> gui_stop
```

When you are ready to use the GUI again, enter the `gui_start` command on the command line.

Using Tcl Scripts

You can use Tcl scripts to accomplish routine, repetitive, or complex tasks. You can define scripts in your setup file or in separate script files.

To run a script defined in a setup file,

1. Define the script in your setup file.
2. Start the IC Compiler GUI.
3. Enter the script name on the `icc_shell` command line.

IC Compiler executes the script and displays status messages in the console log view and the status bar.

Note:

You can also run scripts when you start the IC Compiler GUI. For details, see [“Starting IC Compiler” on page 2-2](#).

You can run all the scripts defined in your setup file or in another script file by selecting the file name in the Execute Script File dialog box.

To run scripts from a file,

1. Create the script file with a `.tcl` extension.
2. Choose File > Execute Scripts.

The Execute Script File dialog box is a standard file browser.

3. Navigate to the directory that contains the script file you want to run.

4. If you need to display file names that do not have a script file extension, select “All Files (*)” in the “File type” list.
5. Select the file name.
6. Click Open.

IC Compiler executes the script and displays status messages on the status bar and in the console log view.

For more information about writing scripts and script files, see the *Using Tcl With Synopsys Tools* manual. For information about using the Synopsys Syntax Checker to find syntax and semantic errors in your Tcl scripts, see [“Checking the Syntax and Semantics of Your Scripts” on page 2-45](#).

Using Setup Files

When you start IC Compiler, it automatically executes commands in three synthesis setup files, named `.synopsys_dc.setup`, and three `icc_shell` setup files, named `.synopsys_icc.tcl`. When you open the GUI, it automatically executes commands in three GUI setup files named `.synopsys_icc_gui.tcl`. These files reside in the installation (Synopsys root) directory, your home directory, and the project directory (the current working directory in which you start IC Compiler). For more information about the `.synopsys_dc.setup` files, see the *Design Compiler User Guide*.

The setup files can contain commands that perform basic tasks, such as initializing parameters and variables, declaring logic libraries, and setting GUI options. You can add commands and Tcl procedures to the setup files in your home and project directories. For example,

- To set variables that define your IC Compiler working environment, create setup files in your home directory.
- To set project- or design-specific variables that affect the optimizations or analysis of a design, create setup files in the design directory.

The default setup files in the installation directory contain system variables defined by Synopsys and general IC Compiler setup information for all users at your site. You cannot edit these files.

If settings conflict between files with the same name, IC Compiler gives precedence to settings in your project directory over settings in your home directory and to settings in your home directory over settings in the installation directory. This order of precedence allows you to set generic settings in the home directory and specific design-related settings in each project directory.

Note:

In addition to executing setup files when you open the GUI, IC Compiler loads application preferences from a file named `.synopsys_icc_prefs.tcl`. For details, see [“Setting GUI Preferences” on page 2-31](#).

For more information about using IC Compiler setup files, see IC Compiler online Help.

Exiting IC Compiler

You can end the session and exit IC Compiler at any time. If you have modified a design and have not saved the changes, IC Compiler prompts you to save the design. You can save or discard the changes before you exit.

Note:

IC Compiler does not save view settings when you exit. If you have changed view settings for layout or schematic views and want to save them, do so before exiting the tool, as explained in [“Setting and Saving View Properties” on page A-28](#).

To exit IC Compiler,

- Choose File > Exit.

If you do not have any unsaved design changes, an IC Compiler message box appears. Do one of the following:

- To exit, click OK.
- To continue the session, click Cancel.

If you have unsaved design changes, the Exit IC Compiler dialog box appears with a list of designs that have unsaved changes. Do one of the following:

- To save all design changes, click Save All.
- To discard all design changes, click Discard All.
- To save or discard changes for individual designs, select “Show advanced options” to display additional options in the list of designs, set options as needed, and click OK or Apply.

For details about setting advanced options, see [“Saving or Discarding Design Changes” on page 2-36](#).

Alternatively, you can exit IC Compiler by entering the `exit` or `quit` command on the command line in the console or in `icc_shell`. IC Compiler displays the message box but does not prompt you to save design changes. You can also exit IC Compiler by pressing Control-c three times in the shell (IC Compiler exits immediately, without displaying the message box).

Getting Help in IC Compiler

IC Compiler provides a variety of user-assistance tools. You can view command-line Help and man pages for `icc_shell` commands and variables, find the equivalent GUI menu commands and dialog boxes for `icc_shell` commands, display a list of keyboard shortcuts that you can use in the GUI, view contextual Help pages for interactive editing and visualization tools, and view an online Help system with information about using IC Compiler with the GUI.

For information about these tools, see the following sections:

- [Getting Help on the Command Line](#)
- [Finding Menu Commands and Dialog Boxes](#)
- [Viewing Man Pages](#)
- [Displaying the List of Keyboard Shortcuts](#)
- [Using Online Help](#)

Getting Help on the Command Line

The following online information resources are available while you are using the IC Compiler tool:

- Command help, which is a list of options and arguments used with a specified `icc_shell` command, displayed in the IC Compiler shell and in the console log view when the GUI is open
- Man pages displayed in the IC Compiler shell and in the console log view when the GUI is open
- A man page viewer that displays command, variable, and error message man pages that you request while using the GUI

To display a description of an `icc_shell` command, variable, or variable group,

- Enter `help` followed by the command or variable name:

```
icc_shell> help command_or_variable
```

To display a list of command options and arguments,

- Enter the command name followed by the `-help` option:

```
icc_shell> command_name -help
```

To display a man page in `icc_shell` and in the console log view if the GUI is open,

- Enter `man` followed by the command or variable name in `icc_shell`:

```
icc_shell> man command_or_variable_name
```

To display a man page in the GUI man page viewer, do either of the following:

- Enter `man` followed by the command or variable name on the console command-line in the GUI:

```
icc_shell> man command_or_variable_name
```

- Enter `gui_show_man_page` followed by the command or variable name in `icc_shell` or on the console command-line in the GUI:

```
icc_shell> gui_show_man_page command_or_variable_name
```

Note:

If you enter the `gui_show_man_page` command in `icc_shell` when the GUI is closed, the tool automatically opens the GUI and displays the man page in the man page viewer.

To view man pages interactively in the GUI man page viewer,

1. Choose Help > Man Pages.

The man page viewer opens.

2. Select the type of man pages to view: Commands, Variables, or Messages.

A list of man pages appears.

3. Select the man page to view.

For more information about using the man page viewer, see [“Viewing Man Pages” on page 2-14](#).

Finding Menu Commands and Dialog Boxes

You can quickly find the equivalent menu command and open the dialog box for any `icc_shell` command by using the `gui_show_form` command. You can specify a single `icc_shell` command or use wildcard characters (`?` or `*`) to specify commands with similar names. The `gui_show_form` command has the following syntax:

```
gui_show_form command_name_or_name_pattern
```


When you specify a command that is used by only one dialog box, the GUI displays the menu command and the menu task mode in the console log view, and opens the dialog box.

```
icc_shell> gui_show_form open_mw_cel  
Info: GUI access for open_mw_cel  
Menu : File->Open Design...  
Task : all
```

Task indicates the menu task mode. For menu commands in the layout window, the task mode can be Block Implementation, Design Planning, or All. For menu commands in other windows, the task mode is always All. To learn more about menu task modes in the layout window, see [“Setting the Menu Task Mode” on page 2-30](#).

When you specify an `icc_shell` command that is used by more than one dialog box, or specify a name pattern that matches more than one `icc_shell` command name, the GUI displays an information table in the man page viewer. For each `icc_shell` command, the table displays information in the following columns:

- Command – displays the `icc_shell` command names (with keywords for additional menu commands)
- Menu – displays the equivalent menu command names
- Window – displays the names of the windows in which the menu commands can be found
- Task – displays the menu task modes in which the menu commands are available

You can click links in the man page viewer to open dialog boxes and view man pages.

To open the dialog box for a menu command,

- Click the link on the menu command name in the man page viewer.

To display the man page for an `icc_shell` command,

- Click the link on the `icc_shell` command name in the man page viewer.

If you specify an `icc_shell` command used by a menu command that performs an immediate action without opening a dialog box, the GUI displays the command information in the man page viewer. If you enter a name for an `icc_shell` command that does not exist, the GUI displays an error message in the console log view.

Viewing Man Pages

You can use the man page viewer to view, search, and print man pages for commands, variables, and error messages.

To view a man page in the man page viewer,

1. Choose Help > Man Pages.

The man page viewer appears and displays a list of links for the different man page categories.

2. Click the category link for the type of man page you want to view: Commands, Variables, or Messages.

The contents page for the category displays a list of title links for the man pages in that category.

3. Click the title link for the man page you want to view.

You can also display man pages in the man page viewer by using the `man` command (or the `gui_show_man_page` command) on the console command line. For more information about using the man page viewer, see IC Compiler online Help.

Displaying the List of Keyboard Shortcuts

You can view a report of the keyboard shortcuts for Tcl commands and commands on menus in the active window. IC Compiler displays the hotkeys report in a report view.

To display the report of keyboard shortcuts for the active window,

- Choose Help > Report Hotkey Bindings.

The report consists of three columns:

- Hot Key lists the shortcut keys or key combinations.
- Type indicates whether the key is a shortcut for a menu command or a Tcl command.
- Function lists the commands that the shortcut keys launch.

Using Online Help

The IC Compiler GUI provides the following type of online Help:

- Contextual Help pages that display information about the current interactive editing or visualization tool in GUI
- A Web browser-based online Help system that explains how to use IC Compiler with the GUI

Contextual Help pages are available for most of the interactive mouse tools and for certain other tools, such as the error browser. These Help pages provide usage information for the tool, explanations of the tool's options, and descriptions of the Tcl command options used to log the tool operations.

To view the Help page for the tool you are currently using,

- If the tool displays options on the Mouse Tool Options toolbar, click the  button.
- If the tool displays options on a panel or in a dialog box, click the Help button.

The IC Compiler online Help system is a browser-based HTML Help system, which provides detailed information and instructions for using the IC Compiler GUI. You can use it to

- Learn about tool features, including windows, views, toolbars, panels, menus, and dialog boxes
- Learn how to use the interactive visualization and editing tools
- Learn how to use IC Compiler to perform design tasks

To open the IC Compiler Help system in your Web browser,

1. Choose Help > Online Help.

The online Help viewer appears in a Web browser window and displays the Welcome topic for the IC Compiler Help system.

2. Use the navigation frame (leftmost frame) to find the information you need in one of the following ways:
 - Browse for the topic in the hierarchical table of contents by clicking Contents and expanding the appropriate books until you find the information you need.
 - Find the topic by its subject by clicking Index and looking for the subject in the alphabetical listing.

- Search for text found in topics by clicking Search and entering the text.

If more than one topic has the words you are searching for, you must select the appropriate topic from a list of topics.

For more information, see the following sections:

- [Searching for Text](#)
- [Configuring the Help Browser](#)

Searching for Text

IC Compiler online Help lets you search for text in any Help topic. By default, the Search mechanism in the navigation frame performs a boolean AND search for one or more text strings separated by spaces. To search for an exact phrase containing two or more words, enclose the words within double quotation marks (" ").

A legal search string has the following characteristics:

- Contains letters and numbers
- Can begin with a period (.) or hyphen (-)
- Can contain underscores (_), but only in a string that begins with a hyphen

However, you can find a string that contains underscores by replacing them with blank spaces and enclosing the string in double quotation marks.

- Can contain a slash (/), backslash (\), or colon (:), but only when preceded and followed by whole words or in a string that begins with a hyphen
- Cannot contain a comma (,) or semicolon (;) in any position
- Cannot contain quotation marks (" , ') or any other special characters in any position
- Is three or more characters long
- Is not case-sensitive

Search results are ranked according to the location of the matched term and the number of matches. Matches in headings always rank near the top. To locate the search string in a Help topic, you can use the quick search box on the button bar above the topic frame or the Find command in your Web browser.

Configuring the Help Browser

You can use the following Web browsers to view the IC Compiler online Help system:

- Firefox version 1.5 or 2.0
- Mozilla 1.7

The default online Help browser is Firefox. You can use the `gui_online_browser` variable to control which browser IC Compiler uses to display the online Help.

To set the browser for the current GUI session,

- Enter the following command after you start the IC Compiler GUI:

```
set gui_online_browser "browser-name"
```

The *browser-name* must be one of the following:

```
firefox  
mozilla
```

Alternatively, you can set the default Help browser for any GUI session by including the `gui_online_browser` variable in the `.synopsys_icc_gui.tcl` file in your home directory.

You can open the Help system from within the IC Compiler GUI (choose Help > IC Compiler Online Help) or stand-alone in your Web browser. To open the online Help from within the GUI, the browser executable file must be specified in your UNIX or Linux path variable.

IC Compiler online Help makes extensive use of JavaScript and style sheets. If your browser encounters problems displaying online Help, open the browser preferences and make sure that JavaScript and style sheets are enabled and that JavaScript is not blocked by your security preferences.

Note:

Although online Help might work on a stand-alone basis in Internet Explorer versions 6.0 and later, it is not tested in this browser.

Working With the GUI

The IC Compiler GUI provides a flexible working environment with multiple windows for performing different tasks. The GUI windows operate independently in your X Windows environment, but they all share the design and global selection data in the tool.

For information about how to get started working with the IC Compiler GUI, see the following sections:

- [Using GUI Windows](#)
- [Working in the Main Window](#)
- [Working With the Console](#)
- [Viewing the Physical Layout](#)
- [Setting GUI Preferences](#)

Using GUI Windows

Each top-level GUI window displays design information in view windows and provides menus and dialog boxes (with keyboard shortcuts), toolbars and panels, interactive mouse button tools, a command console (with a command-line interface), and a status bar. View windows are child windows that appear in the workspace area between the toolbars and the status bar.

The IC Compiler GUI provides the following types of windows:

- IC Compiler main window
- Layout window
- Timing analysis window
- Interactive clock tree synthesis (CTS) window

Each top-level GUI window is independent of other GUI windows. The title bar displays the product name (IC Compiler), the window name (MainWindow.1 for the window that appears when you start the GUI) and the name of the active view (the view window that has the mouse focus). Design objects you select in one window are automatically selected in the other windows.

You can open and close GUI windows, and you can open multiple instances of each window to work with different design settings. Each window instance has a unique name that includes an incremented number.

To open a new instance of a GUI window,

- Choose Window > *window-type*.

To close a GUI window,

- In the window you want to close, choose Window > Close Top Level.

Note:

If you close all the open GUI windows, you end the GUI session but you remain in the `icc_shell` and your designs remain open. You can reopen the GUI by using the `gui_start` command.

You can move, resize, minimize, or maximize GUI windows by using the window management tools on your UNIX or Linux desktop.

For more information about the top-level GUI windows, see the following sections:

- [Menu Bar](#)
- [Toolbars](#)
- [Status Bar](#)
- [View Windows](#)
- [Panels](#)

Menu Bar

The menu bar at the top of a GUI window contains menus with the commands you need to work in the window. You can display a brief message in the status bar about the action that a menu command performs by holding the pointer over the command. For menu commands that can also be used by clicking a toolbar button or pressing a keyboard shortcut, the menus show representations of those alternatives.

Note:

If a window is not wide enough to display all the menu names on the menu bar, the window displays all the menu names that fit, from left to right, followed by an overflow

button (). To access the other menus, click the overflow button.

Toolbars

Each top-level IC Compiler window provides toolbars with buttons you can use to quickly perform frequently used operations or tasks. To determine the function of a toolbar button, hold the pointer over the button. A ToolTip displays the name of the button, and the status bar displays a brief description of its use. You cannot disable these messages.

Toolbars are always attached to a window edge. If a window edge is not long enough to display all of the toolbars attached to it, the GUI displays the full toolbars that fit and shortened versions of the other toolbars. A shortened toolbar displays an overflow button

(). To access the other buttons on a shortened toolbar, click the overflow button.

You can move a toolbar to another location by dragging the double bars on one of its sides. You can also disable a toolbar, hiding it from view.

To display or hide a toolbar or panel,

- Choose View > Toolbars > *toolbar_name*.

A check mark beside the name of a toolbar on the Toolbars menu indicates that the toolbar is visible.

Status Bar

Each GUI window displays a status bar at the bottom of the window. The status bar displays the following information:

- Status information such as the number and type of selected objects
- Information about the action performed by the toolbar button, menu command, or workspace tab under the pointer

If you hold the pointer over a menu command, a toolbar button, or a tab in the workspace, the status bar displays a brief message about the action that the command, button, or tab performs.
- The relative coordinates of the pointer position over an active layout view (layout windows only)

To quickly display the list of selected objects in the Selection List dialog box, click the  button at the right end of the status bar.

You can control the visibility of the status bar in each GUI window.

To display or hide the status bar,

- Choose View > Status Bar.

A check mark next to the Status Bar command on the View menu indicates that the status bar is visible.

View Windows

IC Compiler displays design information in view windows, which are child windows that open up within the workspace area of a GUI window. View windows display specific graphic or textual design information.

Most menu commands and toolbar buttons in a GUI window operate in the active view, which is the view window that has the mouse focus. Some view windows contain two or more panes, with a view in each pane. You can drag the split bars between two panes to adjust their relative sizes within the window.

You can resize a view window or move it to another location inside the workspace area of its parent window.

To change the size of a view window,

1. Hold the pointer over an edge or corner of the window until the pointer changes shape.
2. Drag the edge or corner to resize the window.

To move a view window,

1. Move the pointer over the title bar.
2. Drag the window to a different location.

You can organize view windows by tiling or cascading them within the workspace area.

To tile the view windows in the workspace area,

- Choose Window > Tile.

To cascade the view windows in the workspace area,

- Choose Window > Cascade.

You can also minimize individual view windows, or maximize a view window to fill the workspace area.

The GUI displays a tab at the bottom of the workspace area for each open view window. When you click a tab, the view window appears on top of the other view windows in the workspace area and becomes the active view.

Note:

If a view window and a panel overlap on the screen, the panel appears on top of the view window.

In most views, you can use navigation keys (the arrow keys, Page Up, Page Down, Home, and End) to scroll through and navigate the view.

Panels

Panels are enhanced toolbars that open up within the workspace area of a top-level window to provide tools or views for performing particular tasks. Most panels are associated with a particular view window and operate on the active view (the view window that has the mouse focus). An exception is the console, which contains a command line and its own views.

Some panels contain tabs that you can click to access different tools or views. The first time that you open a panel during a session, it displays the tools or view for its default tab.

You can move a panel to another location on or off the parent window by dragging the double bars on one of its sides. You can also dock a panel (attach it to a window edge) or let it float where you leave it.

You can also disable a panel, hiding it from view. (A panel that is associated with a particular view is automatically hidden when you close the view window.)

To display or hide a toolbar or panel,

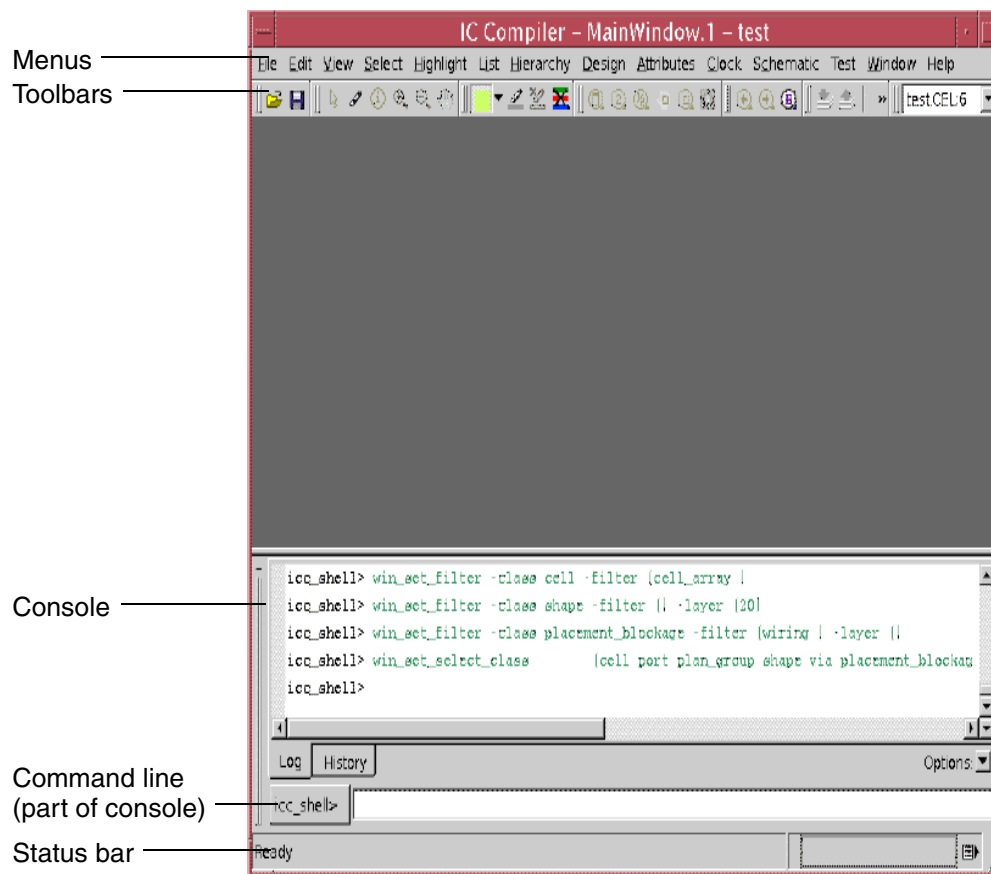
- Choose View > Toolbar > *panel_name*.

A check mark beside the name of a panel on the Toolbars menu indicates that the panel is visible.

Working in the Main Window

When you start IC Compiler and open the GUI, the IC Compiler main window appears. [Figure 2-1](#) identifies the major features of the main window. For information about these features, see IC Compiler online Help.

Figure 2-1 IC Compiler Main Window Features



In the main window, the console is docked above the status bar by default. Although the other top-level GUI windows also provide consoles, they are hidden by default.

Typically, you use the main window to

- Run Tcl scripts
- Enter icc_shell commands, monitor command processing, and view messages in the console
- Read in and save (or close) designs
- Set variables and setup options
- Open other GUI windows
- Close the GUI or exit IC Compiler

In addition, you can open view windows (in the workspace area below the toolbars), explore the logic design in the hierarchy browser, examine the logic design in design schematics, and examine design object and timing information in object list views.

Working With the Console

The console provides a command-line interface and two views, a log view that displays the session transcript (the default) and a history view that displays the command list.

You can use the console to

- Enter IC Compiler Tcl commands on the console command line
- View either the session transcript (log view) or the command history list (history view) by clicking the tabs above the command line
- Display an error message man page in the man page viewer by clicking the message number in the console log view
- Copy and edit or reuse commands
- Search for, select, and save commands or messages in the log view or the history view

You can enter any IC Compiler shell command on the console command line. For example, if you enter `get_selection`, the console log view displays a list of the names of all selected objects. When you enter a command on the console command line, make sure the console has the mouse focus.

To enter a command on the console command line,

1. Type the command.
2. Click the `icc_shell>` prompt button, or press Return.

You can open (or close) one console in each top-level window. When the console is open, you can dock it to the bottom or top of its parent window, or move it over or away from the window.

For more information about the console, see the following sections:

- [Viewing the Session Log](#)
- [Viewing the History View](#)
- [Previewing `icc\_shell` Commands in Dialog Boxes](#)

Viewing the Session Log

The console log view displays a transcript of session information that includes the commands you entered and the IC Compiler output and messages resulting from your commands. You can use this information to check on tool status after performing functions, to troubleshoot problems you encounter, and to look up information on past functions. You can even reenter commands you've already used by copying them from the log to the `icc_shell` command line.

To view the session transcript,

- Click the Log tab in the console.

The console changes to the log view, which is displayed by default whenever the console is opened.

The log view displays the commands you enter next to a boldfaced `icc_shell>` prompt. Warnings and error messages are noted with "Warning" or "Error" as the first word. If you need to see information not currently displayed in the display area of the log view, use the scroll bars to scroll through the session information.

You can use commands on the Options menu (on the right side of the console above the command line) to

- Select and copy text in the transcript
- Search the transcript for commands or messages
- Save text from the transcript in a text file

To copy a used command from the log view to the command line,

1. Select the command or portion of the command that you want to copy in the log view.
2. Middle-click the console command line.

If you already have text on the command line, middle-click where you want to paste the selected information.

You can search the transcript for commands or messages, and you can set the types of text that you want the search mechanism to find.

To search the transcript,

1. (Optional) Set up the search criteria to include or exclude certain types of information by choosing commands on the Options menu.

You can include or exclude commands, error messages, warnings, or information messages. Check marks appear on the Options menu next to the commands for the types of text included in the search.

2. Search forward or backward by choosing Options > Find Next or Options > Find Previous.

The console highlights the next or previous instance of any text type you included in the search.

3. Repeat step 2 as needed to find the command or message you need.

You can save all the text in the transcript, save selected text, or save just the error and warning messages.

To save text from the transcript in a text file,

1. Do one of the following:

- To save the transcript, choose Options > Save Contents As.
- To save selected text, drag the pointer over the text to select it, and choose Options > Save Selection As.
- To save error and warning messages, choose Options > Save Errors/Warnings As.

2. In the dialog box that appears, navigate to the directory where you want to save the file.

3. Enter the file name in the "File name" box.

4. Click Save.

Viewing the History View

The console history view lists shell, menu, and stroke commands you have used in the current session. You can do the following with this list:

- See which commands you have used
- Find and reuse commands already used
- Save the list for later use

To view the command history for the current session,

- Click the History tab in the command console.

The console displays the list of commands, each with a number showing the order in which you used it.

If you need to see information not currently displayed in the display area of the history view, use the scroll bars to scroll through the list of commands. Alternatively, you can enter the `history` command on the command line. This displays the list of commands in the log view.

To reuse a command listed in the history view, you can do any of the following:

- Double-click the command in the history view.
- Select the command in the view and click either Execute (to immediately rerun the command) or Edit (to paste the command on the command line where you can edit it).
- Enter an exclamation point character followed by the number of the command on the `icc_shell` command line.

For example, to reuse the fifth command, enter the following:

```
icc_shell> !5
```

To save the command history list (or a portion of the list) in a text file,

1. (Optional) To save a portion of the list, select the commands you want to save.
2. Do one of the following:
 - If you are saving the entire list, click the Save Contents As button or choose Options > Save Contents As.
The Save Contents As dialog box appears.
 - If you are saving just a portion of the list, choose Options > Save Selection As.
The Save Selection As dialog box appears.
3. If necessary, navigate to the directory where you want to save the file.
4. Enter the file name in the "File name" box.
5. Click Save.

You can set options in the Application Preferences dialog box to control which types of GUI commands are included in the history list. For details, see ["Setting GUI Preferences" on page 2-31](#).

Previewing `icc_shell` Commands in Dialog Boxes

Many IC Compiler dialog boxes support a preview mechanism that you can use to view the commands without running them. You can view the command equivalents for various dialog box options, or combinations of options, or generate an `icc_shell` command that you want to copy into a script. When you enable the preview mechanism, it remains on for all dialog boxes that support the feature until you disable it.

When you click OK or Apply in a dialog box that supports the preview mechanism while the mechanism is enabled, the `icc_shell` command equivalents appear in the console history log preceded by a pound sign (#), but IC Compiler does not execute the commands and they do not appear in the console log view or the shell transcript.

To enable or disable the dialog box command preview mechanism,

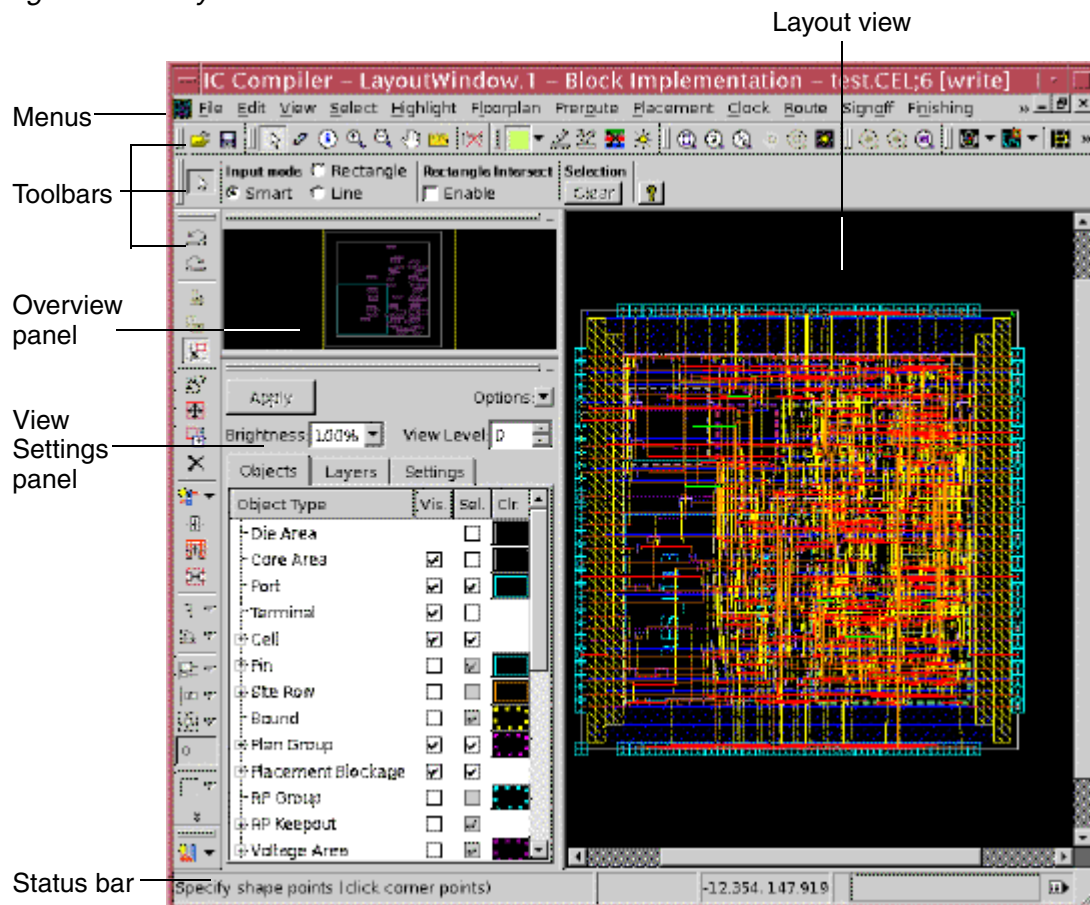
- Click the  menu button in the bottom right corner of a dialog box, and choose Preview command only.

A check mark next to “Preview command only” on the menu indicates that the preview mechanism is enabled.

Viewing the Physical Layout

When you read in a design, a new layout window appears and displays the design in the layout view. This is the primary design in the window. Each layout view you open in a layout window displays the primary design. [Figure 2-2](#) identifies the major layout window features.

Figure 2-2 Layout Window Features



The layout window provides the physical design working environment for the GUI. Layout views provide the focal points for viewing, analyzing, and editing the physical design.

You can use the layout window for each major stage in the physical design process, including the floorplan, the power network, placement, preroute, routing, finishing, and verification.

The layout window provides the following layout analysis tools:

- Visually customizable layout views
- A hierarchy browser you can use to explore the design hierarchy and highlight hierarchical cells and subblocks in the layout view
- A relative placement hierarchy view you can use to examine relative placement groups and hierarchy levels in the design
- An error browser you can use to examine routing and verification errors and highlight their locations in the layout view

Setting the Menu Task Mode

The layout window provides two overlapping sets of menu commands. By default, the menus are set to display commands for block implementation tools. You can change the menus to display commands for design planning or all commands.

To set the design task mode,

- Choose File > Task > *task_type*, where *task_type* is Design Planning, Block Implementation, or All.

The design modes simplify the menus, allowing you to more quickly choose the menu commands you need.

Setting the Primary Design

If you open a new layout window when multiple designs are open, you must select the primary design for the window.

To open a new layout window,

- Choose Window > New Layout Window.

If you have more than one design open, the Choose Design dialog box appears and displays a list of the open designs.

- a. Select the name of the design you want to make the primary design for layout views in the new layout window.
- b. Click OK.

Note:

The GUI provides special windows for timing path analysis and clock tree synthesis. For details, see [“Analyzing Timing Paths” on page A-68](#) and [“Analyzing Clock Tree Synthesis Results” on page A-63](#).

Setting GUI Preferences

At the beginning of a GUI session, IC Compiler loads GUI preferences and layout view settings from the preferences file, `.synopsys_icc_prefs.tcl`, in your home directory. You can change global preferences for GUI tools and the window environment by setting options in the Application Preferences dialog box. If you change preference settings during a GUI session, the new settings are automatically saved in your preferences file when you click OK in the Application Preferences dialog box.

The preferences you can set include

- Font options
- GUI command logging options
- Global tool default options
- Global layout view options
- Global schematic view options

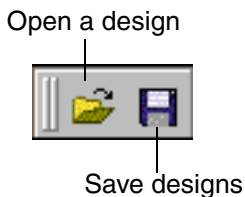
For details about setting these preferences, see the IC Compiler online Help.

Reading and Saving Designs

IC Compiler reads and saves designs in a Milkyway library. You can open a design simply by opening the Milkyway library and then opening the Milkyway design. When you make changes in a design, you can save the changes anytime during the session.

When the GUI is open, you can use the File toolbar, shown in [Figure 2-3](#), to open and save designs. By default, the File toolbar is located on the left end of the toolbar area in the main and layout windows.

Figure 2-3 File Toolbar



IC Compiler is built on the Synopsys Milkyway design database—it can directly access design data without time-consuming and error-prone translation steps.

You can also save (or discard) design changes when you close a design, close the Milkyway library, or exit IC Compiler.

For information about reading and saving designs, see the following sections:

- [Reading a Design](#)
- [Setting the Current Design](#)
- [Saving Designs](#)
- [Closing Designs](#)
- [Saving or Discarding Design Changes](#)

Before you can read a design, you must prepare the design and set up your libraries. In addition, you can use IC Compiler to import design information and save it in the Milkyway database or to export a design for use with tools that do not use Milkyway. For details, see [Chapter 3, “Preparing the Design.”](#)

Reading a Design

IC Compiler reads design information by first opening the Milkyway design library and then opening the design. You can read a design in the GUI or in `icc_shell` (with the GUI open or closed).

When you open a design, IC Compiler

- Locks the Milkyway design library

When you open a Milkyway design library, IC Compiler locks the library, which prevents other copies of it from being used and updated while you are working in it. If you want to use the same Milkyway CEL version for multiple runs, copy the design by using the Copy Design dialog box (choose File > Copy Design) or the `copy_mw_cel` command.

- Sets the design as the current design
- If the GUI is open, opens a new layout window and displays the design in the layout view

This is the primary design in the layout window; if you open multiple layout views in the same layout window, they all display the primary design. However, you can display other designs as overlays on the primary design.

To read a Milkyway design,

1. Click the  button on the File toolbar, or choose File > Open Design.

The Open Design dialog box appears.

2. If you have not already opened the Milkyway library, type the library name in the “Library name” box.

Alternatively, you can click the browse button, select the library name in the Select Library dialog box, and click OK.

The names of designs in the library appear in a list in the dialog box.

3. Select a design name in the list.
4. Click OK.

Setting the Current Design

When you open a Milkyway design library, IC Compiler sets it as the current design (the design you are actively working on). Most IC Compiler commands operate within the context of the current design. For example, if you use the `save_mw_cel` command without specifying the name of the design you want to save, IC Compiler saves the current design. When multiple designs are open, you can reset the current design to a different design.

To change the current design,

- In the main window, select a design name in the list on the Cells List toolbar.

Alternatively, you can use the `current_mw_cel` command.

Note:

If you open a design by using the View Settings panel to display it as an overlay on a layout view, it is not set as the current design.

Saving Designs

If you make changes in the design, you can save it for use with IC Compiler or other tools based on the Milkyway database. When you save the design, IC Compiler automatically updates the Milkyway database.

To save designs,

1. Click the  button on the File toolbar, or choose File > Save Design.

The Save Design dialog box appears with a list of designs that have unsaved changes.

2. Do one of the following:

- To save all design changes, click Save All.
- To save or discard changes for individual designs, select “Show advanced options” to display additional options in the list of designs, set options as needed, and click OK or Apply.

For details about setting advanced options, see [“Saving or Discarding Design Changes” on page 2-36](#).

Alternatively, you can use the `save_mw_cel` command with the `-design` option and a list of design names.

```
icc_shell> save_mw_cel -design design_name
```

If you do not use `-design`, IC Compiler saves the current design (when you choose File > Save in the main window) or the primary design for the window (when you choose File > Save in a layout window).

Closing Designs

You can close designs at any time during an IC Compiler session. If you have unsaved changes in one or more designs, you can save or discard the changes before closing the designs.

To save or discard all design changes and close the designs,

1. Choose File > Close Design.

If you have only one design open and have not made any unsaved changes, IC Compiler closes the design automatically. Otherwise, the Close Design dialog box appears with a list of designs that have unsaved changes.

2. If the Close Design dialog box appears, do one of the following:

- To save all design changes and close the designs, click Save All.
- To discard all design changes and close the designs, click Discard All.
- To save or discard changes and close individual designs, select “Show advanced options” to display additional options in the list of designs, set options as needed, and click OK or Apply.

For details about setting advanced options, see [“Saving or Discarding Design Changes” on page 2-36](#).

Alternatively, you can use the `close_mw_cel` command. However, this command automatically discards any unsaved changes in open designs.

If you need to open another Milkyway library, you must first close the one that is already opened. If you have unsaved changes in one or more designs, you can save or discard the changes before closing the library.

To save or discard all design changes and close the library,

1. Choose File > Close Library.

If you have only one design open and have not made any unsaved changes, IC Compiler closes the library automatically. Otherwise, the Close Library dialog box appears with a list of designs that have unsaved changes.

2. If the Close Library dialog box appears, do one of the following:

- To save all design changes and close the designs, click Save All.
- To discard all design changes and close the designs, click Discard All.
- To save or discard changes for individual designs, select “Show advanced options” to display additional options in the list of designs, set options as needed, and click OK or Apply.

For details about setting advanced options, see [“Saving Designs” on page 2-34](#).

Alternatively, you can use the `close_mw_lib` command. However, this command automatically discards any unsaved changes in open designs.

Saving or Discarding Design Changes

When you save designs, close designs, close the design library, or exit IC Compiler, you can set advanced options to save or discard the changes for individual designs. For each open design, you can

- Save a new CEL version of the design
- Save a copy of a design with a new name
- Save the design hierarchy (a modified top-level design together with all its subdesigns and attached files) into the same CEL version
- Close a design after saving or discarding changes

To save or discard changes for individual designs,

1. Do one of the following:

- To save designs, click the  button on the File toolbar, or choose File > Save Design.
The Save Design dialog box appears.
- To close designs, choose File > Close Design.
The Close Design dialog box appears.
- To close the design library, choose File > Close Library.
The Close Library dialog box appears.
- To exit IC Compiler, choose File > Exit.
The Exit IC Compiler dialog box appears.

2. Select “Show advanced options.”

The dialog box displays additional options and additional columns in the list of designs.

3. (Optional) To display only the names of designs with unsaved changes, select the “All modified cells” option.

The “All open cells” option is selected by default.

4. Set options to save or discard design changes as needed.

You can select which designs you want to save and which methods to use to save them by setting options in the design list.

- To save a new version of a design, select the Save option.

By default, this option is selected for designs that have unsaved changes and deselected for designs that do not have unsaved changes. If you do not want to save the changes for a design, deselect its Save option.

- To save a copy of a design, select the Save As option and type a new design name in the Save As Name box.

Do not include a period (.) in the name unless you want to include the CEL view name. In Milkyway, the period is a special character used to separate design names from view names. A period followed by an invalid view name causes an error.

- To save a modified top-level design together with all its subdesigns and attached files in the same CEL version, select the Save Hierarchy option.

The subdesigns must be in the same design library as the top-level design. Subdesigns that are instantiated from the reference libraries are not included.

This option is not available when the Save As option is selected.

- To close a design after saving it, select the Close option.

If you need to close a design and discard the changes, select the Close option and deselect the Save option.

You can select or deselect options for all designs by using the buttons below the list.

- To select the Save options for all designs, click the Select Save All button.
- To deselect the Save options for all designs, click the Discard All Changes button.
- To select the Save Hierarchy options for all designs, click the Select Hierarchy All button.
- To deselect the Save Hierarchy options for all designs, click the Clear Hierarchy All button.

5. Click OK or Apply.

Alternatively, you can use the `save_mw_cel` command with the `-design` option and a list of design names.

```
icc_shell> save_mw_cel -design design_name
```

If you do not use `-design`, IC Compiler saves the current design. To save a copy of the design with a different name, include the `-as` option.

```
icc_shell> save_mw_cel -design design_name -as new_name
```

To save the top-level design together with its subdesigns and attached files in a single CEL version, include the `-hierarchy` option.

```
icc_shell> save_mw_cel -design design_name -hierarchy
```

Archiving Designs

The `archive_design` command copies a specific Milkyway design, logic libraries, and TLUPlus files and creates a `design_setup.tcl` file in the given archive directory. The extracted design and library files are a subset of the original design and are placed in a new Milkyway Design library.

The `archive_design` command provides a `design_setup.tcl` file and archives the following:

- The Milkyway library and design data of a specific design library
- The logic libraries used by link and compile
- The TLUPlus files used for timing correlation

Note:

When running the `archive_design` command, libraries should not be open, or the command aborts.

When archiving a Milkyway design, the `archive_design` command copies all versions, views, and reference cells of the design. The library and all its reference libraries are stored under the `design` subdirectory. The library references maintained in the archived libraries are updated to reflect the new locations of the reference libraries in the archive location.

The logical libraries are specified using the following variables.

- `search_path`
- `target_library`
- `link_library`

When archiving logic libraries, the `archive_design` command checks the `target_library` and the `link_library` variables to identify the logic libraries to archive. These two variables contain filenames which may or may not be absolute paths. If the filenames are not absolute paths, the `archive_design` command checks if the file is

found in the list of paths specified by the `search_path` variable. All files that match the files in the search paths are copied. The hierarchy of the directories is preserved to prevent filename conflicts. If the filename is an absolute path, only that file is copied. The logic library files are copied to the `logic_db` subdirectory.

TLUPlus files are also copied to the new archive location. The minimum, maximum, and map files are set by the `set_tlu_plus_files` command with the `-max_tluplus`, `-min_tluplus`, and `-tech2itf_map` options. If the values are not set, the TLUPlus files are not copied. The TLUPlus files are stored under the `tlu_plus` subdirectory.

The `design_setup.tcl` file contains variable initialization to help you set up the design. It contains the values of application variables from the current session similar to the output of the `write_app_vars` command. The `search_path`, `target_library`, and `link_library` variables may be modified to accommodate the new archive location and to remove the `combo_itf_library`, `site_info_file`, and `motif_files` application variables. The `design_setup.tcl` file is stored in the root directory of the archive location.

The syntax to archive a design is

```
archive_design
[-source library_path]
[-design design_name]
[-archive archive_directory]
```

`-source`

Specifies the path to the library in which the Milkyway design exists.

`-design`

Specifies the Milkyway design to archive.

`-archive`

Specifies the location for the new archive directory.

[Example 2-1](#) shows the output of the `archive_design` command.

Example 2-1 The archive_design Command Output

```
icc_shell> archive_design -source ~/top_design_library -design top_design\  
-archive my_archive  
Copied Library Files  
  From: /remote/home/top_design_library  
  To: /remote/home/my_archive/design/top_design_library  
Copied Milkyway Design: top_design  
  From: /remote/home/top_design_library  
  To: /remote/home/my_archive/design/top_design_library  
Copied Library Files  
  From: /remote/home/std_cell_library  
  To: /remote/home/my_archive/design/std_cell_library  
Copied Milkyway Design: bufbd3  
  From: /remote/home/std_cell_library  
  To: /remote/home/my_archive/design/std_cell_library  
Copied Library Files  
  From: /remote/home/macro_cell_library  
  To: /remote/home/my_archive/design/macro_cell_library  
Copied Milkyway Design: nand_macro  
  From: /remote/home/macro_cell_library  
  To: /remote/home/my_archive/design/macro_cell_library  
Copied Milkyway Design: nd02d0  
  From: /remote/home/std_cell_library  
  To: /remote/home/my_archive/design/std_cell_library  
WARNING : Cannot copy cell "sdnrq1.err;1" from /remote/home/std_cell_library to / remote/  
home/my_archive/std_cell_library. Cell view may be missing! (MWUI-160)  
WARNING : Cannot copy cell "sdnrq1.err;2" from /remote/home/std_cell_library to / remote/  
home/my_archive/std_cell_library. Cell view may be missing! (MWUI-160)  
Copied Milkyway Design: sdnrq1  
  From: /remote/home/std_cell_library  
  To: /remote/home/my_archive/design/std_cell_library  
Copied Logic db File: logic.db  
  From: /remote/home/foo  
  To: /remote/home/my_archive/logic_db  
Copied TLU+ File: tlu_plus_file.max  
  From: /remote/home/foo  
  To: /remote/home/my_archive/tlu_plus  
1
```

Using IC Compiler Command-Line Interfaces

The IC Compiler command language provides capabilities similar to UNIX command shells, including variables, conditional execution of commands, and control flow commands. The syntax for an `icc_shell` command is

```
command_name [-option [argument]] [command_argument]
```

`command_name`

The name of the command.

`-option`

Modifies the command and passes information to the tool. Options specify how the command should run. A hyphen (-) precedes option names. Options that do not take an argument are called switches.

`argument`

The argument to an option. Some options require values, keywords, or other information. Separate multiple arguments with commas or spaces. You can enclose arguments in parentheses.

Arguments are separated by commas or spaces and can be enclosed in parentheses. If you omit a required argument, an error message and a usage statement appear.

`command_argument`

Some IC Compiler commands require command arguments that do not begin with a hyphen (-). These are positional arguments that must be entered in a specific order relative to each other. Nonpositional arguments (that begin with a hyphen) can be entered in any order and can be intermingled with positional arguments.

Command names and variable names are case-sensitive, as are other values, such as file names, design object names, and strings.

You can abbreviate command names and options to the shortest unambiguous (unique) string. For example, you can abbreviate the `get_attribute` command to `get_attr` or the `create_clock` command's `-period` option to `-p`. However, you cannot abbreviate most built-in commands. If you enter an ambiguous command, `icc_shell` attempts to help you find the correct command.

Note:

Command abbreviation is meant as an interactive convenience. Do not use command or option abbreviation in script files because script files are susceptible to command changes in subsequent versions of the tool. Such changes can cause abbreviations to become ambiguous.

If you enter a long command with many options and arguments, you can split it across more than one line by using the continuation character, the backslash (\). There is no limit to the number of characters in an `icc_shell` command line.

if you want to put more than one command on a line, separate the commands with a semicolon. If you want to include a comment in a command, begin the comment with the “#” character. The comment can start anywhere on a line. It ends with the end of the line.

Every `icc_shell` command returns a value, either a status code or design-specific information. The command status codes in `icc_shell` are

- 1 for successful completion
- 0 or { } (null list) for unsuccessful execution

The IC Compiler command language also supports wildcard characters and aliases. In addition, you can rerun commands and redirect or append command output. For details, see the following sections:

- [Using Wildcard Characters](#)
- [Using Aliases](#)
- [Listing and Rerunning Previously Entered Commands](#)
- [Redirecting and Appending Command Output](#)

For additional information about Tcl commands and the IC Compiler command language, see the *Using Tcl With Synopsys Tools* manual and the *Design Compiler Command-Line Interface Guide*.

Using Wildcard Characters

The IC Compiler command language has two wildcard characters:

- The wildcard character “*” matches one or more characters in a name.
For example, `u*` indicates all object names that begin with the letter `u`, and `u*z` indicates all object names that begin with the letter `u` and end in the letter `z`.
- The wildcard character “?” matches a single character in a name.
For example, `u?` indicates all object names exactly two characters in length that begin with the letter `u`.

Using Aliases

You can use aliases to create short forms for the commands you commonly use. When you use aliases, keep the following points in mind:

- The `icc_shell` interface recognizes aliases only when they are the first word of a command.
- An alias definition takes effect immediately but lasts only until you exit the IC Compiler session. To save commonly used alias definitions, store them in the `.synopsys_dc.setup` file.
- You cannot use an existing command name as an alias name; however, aliases can refer to other aliases.

The following example shows how you can use the `alias` command to create a shortcut for the `report_timing` command:

```
icc_shell> alias rt100 {report_timing -max_paths 100}
```

Use the `unalias` command removes alias definitions created with the `alias` command. For example, to remove all aliases beginning with `f*` and the `rt100` alias, enter

```
icc_shell> unalias f* rt100
```

Listing and Rerunning Previously Entered Commands

You can use the `history` command to list the commands used in an `icc_shell` session. It prints an ordered list of previously executed commands. You can control the number of commands printed. The default number printed is 20. The `history` command is complex and can generate various forms of output. For detailed information about this command, see the man page.

You can rerun and recall previously entered commands by using the exclamation point (!) operator. For more information, see the *Design Compiler Command-Line Interface Guide*.

Redirecting and Appending Command Output

If you run `icc_shell` scripts overnight to compile a design, you cannot see warnings or error messages echoed to the command window while your scripts are running. You can direct the output of a command, procedure, or script to a specified file in two ways:

- By using the traditional UNIX redirection operators (`>` and `>>`)
- By using the `redirect` command

You can use the UNIX redirection operators (`>` and `>>`) in the following ways:

- Divert command output to a file by using the redirection operator (`>`).
- Append command output to a file by using the append operator (`>>`).

Note:

Unlike UNIX, IC Compiler treats `>` and `>>` as arguments to a command. Therefore, you must use white space to separate these arguments from the command and the redirected file name. The pipe character (`|`) has no meaning in the `icc_shell` interface.

You cannot use the UNIX style redirection operators with built-in commands because the UNIX redirection operators are not part of Tcl. Always use the `redirect` command when using built-in commands. The Tcl built-in command `puts` does not respond to redirection of any kind. Instead, use the `echo` command, which does respond to redirection.

The `redirect` command performs the same function as the traditional UNIX redirection operators (`>` and `>>`); however, the `redirect` command is more flexible. Also, the UNIX redirection operators are not part of Tcl and cannot be used with built-in commands. You must use the `redirect` command with built-in commands.

- The result of a redirected command that does not generate a Tcl error is an empty string.
- Screen output from a redirected command occurs only when there is an error.
- You can redirect multiple commands or an entire script.

Although the result of a successful `redirect` command is an empty string, you can get and use the result of the command you redirected. You do this by constructing a `set` command in which you set a variable to the result of your command, and then redirecting the `set` command. The variable holds the result of your command. You can then use that variable in a conditional expression.

You can direct command output to the standard output device as well as a redirect target by using the `-tee` option.

Checking the Syntax and Semantics of Your Scripts

The Synopsys Syntax Checker, which is based on the TclPro Checker (procheck), helps you find syntax and semantic errors in your Tcl scripts. Everything that you need for syntax and semantic checking is shipped with IC Compiler. In this environment, the following items are checked:

- Unknown options
- Ambiguous option abbreviation
- Use of exclusive options
- Missing required options
- Validation of literal option values for numerical range (range, <=, >=)
- Validation of one-of-string (keyword) options
- Recursion into constructs that have script arguments, such as the `redirect` and `foreach_in_collection` commands
- Warning of duplicate options that override previous values

The open-source TclPro toolkit also includes the following tools, which are not included in the IC Compiler distribution, but which can be downloaded separately:

- TclPro Compiler (procomp), which is a stand-alone bytecode compiler
- TclPro Debugger (prodebug), which is a Tcl debugger

For more information about the TclPro tools, see the following Web address:

<http://tcl.sourceforge.net>

The following sections discuss how to use these tools in the Synopsys environment:

- [Installation Requirements](#)
- [Running the Synopsys Syntax Checker](#)
- [Limitations of the Synopsys Syntax Checker](#)
- [Bytecode-Compiled Files](#)
- [TclPro Limitations](#)

Installation Requirements

To access only the syntax checker, you do not need to download the TclPro tools. To use any TclPro tools other than the syntax checker, you must install TclPro on your system. After you have installed TclPro on your system, you must define the `SNPS_TCLPRO_HOME` environment variable to define where the TclPro installation exists.

For example, if you installed TclPro version 1.5 at `/u/tclpro1.5`, you must set the `SNPS_TCLPRO_HOME` variable to point to that directory. IC Compiler uses this variable as a base for launching some of the TclPro tools. In addition, other Synopsys applications use this variable to link to the TclPro tools.

Running the Synopsys Syntax Checker

The Synopsys Syntax Checker for IC Compiler is a stand-alone executable called `iccprocheck`. This executable is located at `$root/bin/iccprocheck`

[Example 2-2](#) shows a script that is being run through `iccprocheck`. [Example 2-3](#) shows the resulting output from `iccprocheck`. Each line in the output shows where a syntax or semantic error was detected. The offending command is shown with a caret (^) below the character that begins the offending token.

Example 2-2 Sample Script

```
create_clock [get_port CLK] -p
create_clock [get_ports CLK] -p -12.2

sort_collection
set paths [get_timing_paths -nworst 10 -delay_type mx_fall -r]
my_report -from [sort_collection [sort_collection $a b] {b c d} -x]
foreach_in_collection x $objects {
    query_objects $x
    report_timing -through $x -thr $y -from a -from b -to z > r.out
}
all_fanout -from X -clock_tree
pwd foo
```

Example 2-3 iccprocheck Output

```
Synopsys Tcl Syntax Checker - Version 1.0

Loading snps_tcl.pcx...
Loading icc.pcx...
scanning: /disk/scripts/ex1.tcl
checking: /disk/scripts/ex1.tcl
/disk/scripts/ex1.tcl:1 (warnUndefProc) undefined procedure: get_port
get_port CLK
^
/disk/scripts/ex1.tcl:1 (SnpsE-MisVal) Value not specified for 'create_clock -period'
create_clock [get_port CLK] -p
^
```

```

/disk/scripts/ex1.tcl:2 (SnpsE-BadRange) Value -12.2 for 'create_clock -period' must be
>= 0.000000
create_clock [get_ports CLK] -p -12.2
      ^

/disk/scripts/ex1.tcl:3 (SnpsE-MisReq) Missing required positional options for
sort_collection: collection criteria
sort_collection
^

/disk/scripts/ex1.tcl:4 (badKey) invalid keyword "mx_fall" must be: max min min_max
max_rise max_fall min_rise min_fall
get_timing_paths -nworst 10 -delay_type mx_fall -r
      ^

/disk/scripts/ex1.tcl:4 (SnpsE-AmbOpt) Ambiguous option 'get_timing_paths -r'
get_timing_paths -nworst 10 -delay_type mx_fall -r
      ^

/disk/scripts/ex1.tcl:5 (warnUndefProc) undefined procedure: my_report
my_report -from [sort_collection \
^

/disk/scripts/ex1.tcl:5 (SnpsE-UnkOpt) Unknown option 'sort_collection -x'
sort_collection [sort_collection $a b] {b c d} -x
      ^

/disk/scripts/ex1.tcl:9 (SnpsW-DupOver) Duplicate option
'report_timing -from' overrides previous value
report_timing -through $x -thr $y -from a -from b -to z > r.out
      ^

/disk/scripts/ex1.tcl:11 (SnpsE-Excl) Can only specify one
of
these options for all_fanout: -from -clock_tree
all_fanout -from X -clock_tree
^

/disk/scripts/ex1.tcl:12 (numArgs) wrong # args
pwd foo
^

```

Limitations of the Synopsys Syntax Checker

The following limitations apply to the Synopsys Syntax Checker:

- Command abbreviations are not checked. Use of abbreviated commands will show up as undefined procedures.
- Aliases created with the `alias` command are not expanded and will show up as undefined procedures.
- A few checks done when the application is running might not be checked using the `snps_checker` environment. For example, some cases where one option requires another option are not checked.
- Script size is an issue with `snps_checker` and TclPro 1.3 and 1.5. Scripts of up to a few thousand lines can be reasonably checked, but beyond that, CPU time becomes a factor. Do not try to check extremely large scripts using `snps_checker`.

- IC Compiler allows you to specify Verilog-style bus names on the command line without rigid quotation. An example of this is `A[0]`. This format with indexes from 0 to 255 is checked. The wildcard characters `**` and `%` are also checked. Other forms, including ranges such as `A[15:0]`, will show up as undefined procedures, unless represented as `{A[15:0]}`.
- User-defined procedures enhanced with the `define_proc_attribute` command are not checked. Because such procedures are declared with the `args` argument, no semantic errors are reported.

Bytecode-Compiled Files

Bytecode-compiled files are created using the TclPro Compiler, `procomp`, and have the following advantages over ASCII scripts:

- They are efficient to load.
- They are secure because the content is not readable.
- The body of the compiled Tcl procedures is hidden.

IC Compiler can load bytecode-compiled scripts. To load a bytecode-compiled file, you source it like you do other scripts. You do not need any files other than the application to load bytecode-compiled files.

TclPro Limitations

TclPro is not supported on RS/6000. However, bytecode-compiled files can be loaded into this application on all platforms, including RS/6000, and `snps_checker` is supported on all UNIX platforms.

Working With Licenses

You can view a list of the licenses you are currently using and a list of all licenses that are currently checked out. You can also check out additional licenses, queue licenses that are not currently available, and release licenses you no longer need. To learn how to determine what licenses are in use and know how to obtain and release licenses, see the following sections:

- [Listing the Licenses in Use](#)
- [Getting Licenses](#)
- [Enabling License Queuing](#)

- [Releasing Licenses](#)
- [Getting and Releasing Licenses in the GUI](#)

Listing the Licenses in Use

To view the licenses that you currently have checked out, use the `list_licenses` command. For example,

```
icc_shell> list_licenses
```

```
Licenses in use:  
    Galaxy-Common  
    Galaxy-ICC  
1
```

To determine which licenses are already checked out, use the `license_users` command. For example,

```
icc_shell> license_users  
bill@eng1 Galaxy-Common, Galaxy-FP, Galaxy-ICC  
matt@eng2 Galaxy-Common, Galaxy-ICC, Milkyway-Interface  
2 users listed.  
1
```

Getting Licenses

When you invoke IC Compiler, the Synopsys Common Licensing software automatically checks out the appropriate license. For example, if you open a Milkyway design library, Synopsys Common Licensing checks out the Milkyway-Interface license.

If you know the tools and interfaces you need, you can use the `get_license` command to check out those licenses. This ensures that each license is available when you are ready to use it. For example,

```
icc_shell> get_license Milkyway-Interface
```

Once a license is checked out, it remains checked out until you release it or exit `icc_shell`.

Enabling License Queuing

IC Compiler has a license queuing functionality that allows your application to wait for licenses to become available if all licenses are in use. To enable this functionality, set the `SNPSLMD_QUEUE` environment variable to true. The following message is displayed:

```
Information: License queuing is enabled. (ICCSH-001)
```

When you have enabled the license queuing functionality, you might run into a situation where two processes are waiting indefinitely for a license that the other process owns. To prevent such situations, set the `SNPS_MAX_WAITTIME` environment variable and the `SNPS_MAX_QUEUEETIME` environment variable. You can use these variables only if the `SNPSLMD_QUEUE` environment variable is set to true.

The `SNPS_MAX_WAITTIME` variable specifies the maximum wait time for the first key license that you require, such as the license for starting `icc_shell`. Consider the following scenario:

You have two IC Compiler packages, both of which are in use, and you are attempting to start a third `icc_shell` process. The queuing functionality places this last job in the queue for the specified wait time. The default wait time is 259,200 seconds (or 72 hours). If the license is still not available after the predefined time, you might see a message similar to the following:

```
Information: Timeout while waiting for feature 'Galaxy-ICC' (ICCSH-005)
```

The `SNPS_MAX_QUEUEETIME` variable specifies the maximum wait time for checking out subsequent licenses within the same `icc_shell` process. You use this variable after you have successfully checked out the first license to start `icc_shell`. Consider the following scenario:

You have already started IC Compiler and are running a command that requires a Galaxy-FP license. The queuing functionality attempts to check out the license within the specified wait time. The default is 28,800 seconds (or eight hours). If the license is still not available after the predefined time, you might see a message similar to the following:

```
Information: Timeout while waiting for feature 'Galaxy-FP'. (ICCSH-005)
```

As you take your design through the synthesis flow, the queuing functionality might display other status messages, such as the following:

```
Information: Successfully checked out feature 'Galaxy-FP'. (ICCSH-002)
Information: Started queuing for feature 'Galaxy-FP'. (ICCSH-003)
Information: Still waiting for feature 'Galaxy-FP'. (ICCSH-004)
```

Releasing Licenses

To release a license that is checked out to you, use the `remove_license` command. For example,

```
icc_shell> remove_license Milkyway-Interface
```

Getting and Releasing Licenses in the GUI

You can view a list of the licenses you are currently using and a list of additional licenses, check out additional licenses, or release licenses you no longer need.

To display current license information,

- Choose File > Licenses.

The Application Licenses dialog box appears. The Allocated Licenses list shows the licenses you are currently using. The Available Licenses list shows other licenses you can use.

When you finish viewing the license information, click Close to close the dialog box.

To check out an additional license,

1. Choose File > Licenses.
2. Select a license in the Available Licenses list.
3. Click Allocate.

IC Compiler checks out a copy of the license if one is available or displays an error message if all the licenses are already taken.

To release a license,

1. Choose File > Licenses.
2. Select a license in the Allocated Licenses list.
3. Click Release.

IC Compiler releases the selected license.

3

Preparing the Design

IC Compiler uses a Milkyway design library to store your design and its associated library information. This chapter describes how to set up the libraries, create a Milkyway design library, read your design, and save the design in Milkyway format.

These steps are explained in the following sections:

- [Setting Up the Libraries](#)
- [Reading the Design](#)
- [Annotating the Physical Data](#)
- [Preparing for Timing Analysis and RC Calculation](#)
- [Saving the Design](#)

Setting Up the Libraries

IC Compiler requires both logic libraries and physical libraries. The following sections describe how to set up and validate these libraries:

- [Setting Up the Logic Libraries](#)
- [Setting Up the Physical Libraries](#)
- [Verifying Library Consistency](#)

Setting Up the Logic Libraries

IC Compiler uses logic libraries to provide timing and functionality information for all standard cells. In addition, logic libraries can provide timing information for hard macros, such as RAMs.

IC Compiler supports logic libraries that use nonlinear delay models (NLDMs), Composite Current Source (CCS) models (either compact or noncompact), or both. IC Compiler automatically selects the timing model to use based on the contents of the logic libraries. If the logic libraries contain only CCS models, IC Compiler uses CCS models. If the libraries contain nonlinear delay models (NLDMs) or both NLDM and Composite Current Source (CCS) models, IC Compiler uses the NLDM models. The selected models are used for both preroute and postroute delay calculations.

Note:

When using CCS models, the tool might not use all of the available CCS data in some instances, such as during preroute optimization or on non-Arnoldi nets during postroute delay calculations. This is done to balance runtime against the required accuracy.

This section describes the following tasks:

- [Defining the Logic Library Settings](#)
- [Validating Logic Libraries](#)
- [General Logic Checks](#)
- [Specific Logic Checks](#)
- [Special Checks](#)
- [Reporting on Logic Library Options](#)

Defining the Logic Library Settings

IC Compiler uses variables to define the logic library settings. In each session, you must define the values for the following variables so that IC Compiler can access the libraries:

- `search_path`

Lists the paths where IC Compiler can locate the logic libraries.

- `target_library`

Lists the logic libraries that IC Compiler can use to perform physical optimization.

- `link_library`

Lists the logic libraries that IC Compiler can search to resolve references.

You can set these variables in the `.synopsys_dc.setup` file by restoring the values saved in the Milkyway design library, or by setting the variables interactively in `icc_shell`.

The first library in the `link_library` path is the main library. If you are using an existing Milkyway design library, ensure that the main library you specify is the same one that was used when the Milkyway design was saved.

The units used for time, voltage, current, resistance, capacitance, and power in the main library are used as the default units for the design. If your design or SDC file does not have unit settings, IC Compiler uses these default settings.

If you are performing simultaneous minimum and maximum timing analysis, the logic libraries specified by the `link_library` variable are used for both maximum and minimum timing information, unless you specify separate minimum timing libraries by using the `set_min_library` command. The `set_min_library` command associates minimum timing libraries with the maximum timing libraries specified in the `link_library` variable. For example,

```
icc_shell> set link_library "* maxlib.db"
icc_shell> set_min_library maxlib.db -min_version minlib.db
```

To find out which libraries have been set to be the minimum and maximum libraries, use the `list_libs` command. In the generated report, the letter “m” appears next to the minimum library and the letter “M” appears next to the maximum library.

By default, IC Compiler saves the logic library definitions in the Milkyway design library when you save your design. If you do not want to save these definitions, set the `save_mw_cel_lib_setup` variable to false (the default is true). If you do not save these definitions, they take effect only for the current session.

By default, even if you save these definitions in the Milkyway design library, they are not restored when you open the design. If you save these definitions in the Milkyway design library and want IC Compiler to restore them when opening the design, set the `auto_restore_mw_cel_lib_setup` variable to true (or select “Restore library setup” in the GUI). If you restore the definitions when you open a design, you must ensure that the stored libraries are accessible in the current session.

Important:

In general, variable settings are not saved with the design. Because the ability to restore variable settings, such as those used to define the logic libraries, enables an inconsistent user model, the `save_mw_cel_lib_setup` and `auto_restore_mw_cel_lib_setup` variables will be obsolete in a future release.

Validating Logic Libraries

Use the `check_library` command to check the quality of a logic library. Use this command before or after you setup your design to ensure that the libraries do not have problems. You can perform selective checks by setting the options using `set_check_library_options` command. If you do not set any options using the `set_check_library_options` command, the `check_library` command performs default checking.

To perform logic library checks, you must enable the specific checks you want by running the `set_check_library_options` command. [Table 3-1](#) shows the logic library checks and the `set_check_library_options` command options used to enable them.

Table 3-1 Logic Library Checks

Option	Description
<code>-compare {construct value}</code>	General logic library consistency checking
<code>-mcmm</code>	Logic library consistency checking for multicorn-multimode (MCMM)
<code>-scaling {timing noise power}</code>	Logic library consistency checking for CCS timing, CCS noise, and CCS or NLDM power scaling
<code>-tolerance {type relative_tolerance absolute_tolerance}</code>	Specifies a relative tolerance and an absolute tolerance for characterization value comparison
<code>-upf</code>	Logic library consistency checking for multivoltage and UPF
<code>-validate</code>	Logic library consistency checking for library characterization
<code>-logical</code>	Specify to enable all the logic library checks

Table 3-1 Logic Library Checks (Continued)

Option	Description
-reset	Specify to reset the logic library checks to the default settings
-all	Specify to enable all check for logic and physical libraries

Specify the option for each check that you would like to enable, or if you want to enable all logic library checks, use the `-logical` option.

General Logic Checks. The `check_library` command performs the following logic versus logic library general checks. For the libraries to pass a flow check, the general logic checks should not have any problems.

Library Group

For the library group, the `check_library` command reports the library version and the library type. The library type indicates whether the library is power and ground pin based. The command also checks if the libraries have the same units and the same slew trip points for CCS timing scaling. The command checks if the libraries have the default operating conditions that are defined. The `check_library` command performs these checks by default with the exception of the trip points check. Use the `set_check_library_options -scaling timing` command to perform the trip points check.

Cell Group

The `check_library` command checks for the same number of cells in each library. It also checks that the `area`, `function_id`, `dont_use`, and `dont_touch` attributes for cells with the same name are consistent. The `check_library` command performs these checks by default.

Pin and PG Pin Groups in the Cells

The `check_library` command checks if the libraries have the same number of pins, including the power and ground (PG) pins. The command also checks if the PG pin names, directions, types, functions and `voltage_name` in the two libraries are the same. Apart from these checks, the command also checks if the two libraries have the same

- Input or output `signal_level` for rail-based libraries and the same `related_power_pin` or `related_ground_pin` for power and ground pin based libraries
- Derived `pin_number` attribute values

If there is a mismatch, the `check_library` command checks the order of pins in each cell group and function Boolean expressions, and others that may also affect the mismatch and are unknown at this point.

The `check_library` command performs these checks by default.

Timing Arcs

The `check_library` command checks for timing arcs with matching related pins, timing type, timing sense, and when conditions across the libraries. In any two sets of logic (technology) libraries with the same cell group, if such timing arcs are used in at least one design, they are checked for consistency between these libraries. The `check_library` command performs these checks by default.

Specific Logic Checks. The `check_library` command performs the following checks when you use various options of the `set_check_library_options` command.

Use the `set_check_library_options -compare construct` command to specify that the `check_library` command compares all groups, subgroups, attributes, and characterized values such as timing, power, and current between two libraries.

Use the `set_check_library_options -compare value` command to specify that the `check_library` command compares two libraries for all characterized values.

Special Checks. Some flows and applications require checking in addition to the general checks.

CCS Timing Scaling

Use the `set_check_library_options -scaling timing` command to specify that the `check_library` command performs the following checks for CCS timing scaling:

- The number of CCS timing driver model load indexes must be the same for each timing arc.
- The output load indexes in each CCS timing driver model timing arc must be the same.
- For setup and hold pessimism reduction, all load and slew indexes should be the same across the libraries.
- Receiver models across all scaling libraries exist and they should be of the same type.
- The order of the receiver and noise models, and the conditional `when` and default pin models are the same across the libraries.
- The base curve_x is the same across the libraries at a number of points and for each value for Compact CCS.
- All libraries have either the compressed or original format for driver data.

- Characterization waveform type between two libraries should be the same. The command checks for cell or pin level attributes such as `driver_waveform`, `driver_waveform_rise`, `driver_waveform_fall`, and `driver_waveform_name` in the `normalized_driver_waveform` group.

These CCS timing scaling checks are for PrimeTime releases later than version Z-2006.12.

CCS Noise Scaling Checks

Use the `set_check_library_options -scaling noise` command to specify that the `check_library` command performs the following checks for CCS noise scaling:

- The `input_voltage` and `output_voltage` indexes for CCS noise models in the libraries across the scaling groups are the same percentage of VDD.
- The command checks that the conditional `when` pin models and the receiver models are in the same order across the libraries.
- The number of CCS noise models is the same across the libraries for all pins and arcs.

These CCS noise scaling checks are for PrimeTime releases later than version Z-2006.12.

Power Scaling Checks

Use the `set_check_library_options -scaling power` command to specify that the `check_library` command performs the following checks for CCS or NLDM power scaling to ensure that the libraries have the same:

- Set of `dynamic_current` and `leakage_current` groups for each cell
- Set of `intrinsic_parasitic` and `leakage_power` tables for each cell
- `power_cell_type` attribute for each cell
- Set of `internal_power` tables for each pin or cell

Voltage Scaling Checks

For voltage scaling, the `check_library` command checks that the voltages are no more than 20 percent apart for good accuracy. This check is performed for reporting and does not determine the success or failure of the check.

Multivoltage and UPF Checks

Use the `set_check_library_options -upf` command to specify the following checks for multivoltage or UPF flows:

- Different PVT values for different characterizations
- NLDM or CCS model under different voltage conditions exists

- Level shifter, isolation, retention, and switch cells exist with consistent and complete attributes
- The voltage_map and pg_pin groups exist in a multiple power and ground pin based library with consistent attributes
- The same power_down_function for output or inout pins
- Power data exists for all operating conditions

Multicorner-Multimode Checks

Use the `set_check_library_options -mcm` command to specify the following checks for a multicorner-multimode flow:

- Different PVT values for different characterizations between two libraries for different operation conditions, that is, libraries with same-name cells, which have different nominal PVT values
- Same cell attributes such as `dont_use`, `dont_touch`, and black box
- The same power_down_function for output or inout pins
- Power data exists for all operating conditions

Characterization Value Comparison

Use the `set_check_library_options -tolerance {type rel_tol abs_tol}` command to specify a relative tolerance and an absolute tolerance for characterization value comparison, such as delay values.

The valid values for the `type` argument are `time`, `power`, `current`, and `capacitance`. If the `type` argument is not specified, the values are for output load capacitance. To specify an absolute tolerance, do not include the unit. The unit in the first library is used.

The following example specifies tolerances for time and power.

```
-tolerance {time 0.1 0.2 power 0.3 0.4}
```

If you do not specify tolerances for a type, the default values are used. The following default values are consistent with the settings for PrimeTime and the Library Quality Assurance System:

	Load Index	Time	Power
Relative Tolerance	0.01	0.04	0.04
Absolute Tolerance	0.001pF	0.015ns	5pW

If you want to determine whether two libraries are identical by comparing them by groups and by attributes, it is recommended that you set the tolerances to smaller values.

Library Characterization Checks

Use the `set_check_library_options -validate` command to specify the following checks for library characterization using the Library Quality Assurance System:

- The same output load index for each individual cell between CCS and NLDM models. The same load indexes for CCS driver or receiver.
- The same number of NLDM delay and slew indexes and values.
- Difference of delays between NLDM and CCS timing within an acceptable range for both compact CCS and expanded CCS
- The command checking that two CCS driver model current waveforms, including compact CCS with expanded CCS data, are consistent within an acceptable margin.

Inverter and Buffer Cell Checks

The `check_library` command also reports on the existence of inverter and buffer cells and total numbers among libraries. Use the `set_check_library_options -logic` command to specify that the `check_library` command performs this check.

Reporting on Logic Library Options. To report on the values of logic library consistency checking options set by the `set_check_library_options` command, use the `report_check_library_options` command.

```
icc_shell> report_check_library_options -logic
```

You can use the `-default` option to report on the default options.

Setting Up the Physical Libraries

IC Compiler uses Milkyway reference libraries and technology (.tf) files to provide physical library information. The Milkyway reference libraries contain physical information about the standard cells and macro cells in your technology library. In addition, these reference libraries define the placement unit tile. The technology files provide technology-specific information such as the names and characteristics (physical and electrical) for each metal layer.

The physical library information is stored in the Milkyway design library. For each cell, the Milkyway design library contains several views of the cell, which are used for different physical design tasks.

If you have not already created a Milkyway library for your design (by using another tool that uses Milkyway), you need to create one by using the IC Compiler tool. If you already have a Milkyway design library, you must open it before working on your design.

This section describes the following tasks:

- [Creating a Milkyway Design Library](#)
- [Converting a Milkyway Database Model](#)
- [Validating a Milkyway Design Library](#)
- [Opening a Milkyway Design Library](#)
- [Reporting on a Milkyway Design Library](#)
- [Changing Physical Library Information](#)
- [Saving Physical Library Information](#)

Creating a Milkyway Design Library

To create a Milkyway design library, use the `create_mw_lib` command (or choose File > Create Library in the GUI).

At a minimum, you must specify the technology file and the Milkyway reference libraries that are used for the design, as well as the name of the Milkyway design library. In addition, you can specify the bus naming style and hierarchy separator character used in the design. If you do not specify a hierarchy separator character, a slash (/) is used. If you do not specify the bus naming style, %d is used. [Table 3-2](#) shows the command options (and GUI fields) used to specify these items.

Table 3-2 Options for Creating a Milkyway Design Library

Task	Command Line Option	GUI Field
Specify technology file	<code>-technology file_name</code>	"Technology file"
Specify Milkyway reference libraries	<code>-mw_reference_library file_names</code> or <code>-reference_control_file file_name</code>	"Input reference libraries" or "Reference control file"
Specify Milkyway design library name	<code>library_name</code>	"New library path" and "New library name"

Table 3-2 Options for Creating a Milkyway Design Library (Continued)

Task	Command Line Option	GUI Field
Specify bus naming style	<code>-bus_naming_style style</code>	"Bus naming style"
Specify the hierarchy separator character	<code>-hier_separator char</code>	"Hierarchical separator"

As indicated in [Table 3-2](#), there are two ways to specify the Milkyway reference libraries. You can either list them individually by using the `-mw_reference_library` option or you can specify them in a reference control file by using the `-reference_control_file` option.

The syntax for the reference control file is

```
LIBRARY Milkyway_reference_library_name.mw
REFERENCE /path/Milkyway_reference_library_name
REFERENCE /path/Milkyway_reference_library_name
...
```

Note:

Use absolute paths in the control file mode to document where the data originates.

For example, if your design references the LIB_A library, which, in turn, references LIB_B and LIB_C libraries as child references, and if each of these libraries has child references to a reference library (REF_LIB_B for LIB_B and REF_LIB_C for LIB_C), you could define these references in a reference control file, as shown in [Example 3-1](#).

Example 3-1 Reference Control File

```
LIBRARY /absolute_path/libraries/LIB_A
REFERENCE /absolute_path/libraries/LIB_B
REFERENCE /absolute_path/libraries/LIB_C
LIBRARY /absolute_path/libraries/LIB_B
REFERENCE /absolute_path/ref_libraries/REF_LIB_B
LIBRARY /absolute_path/libraries/LIB_C
REFERENCE /absolute_path/ref_libraries/REF_LIB_C
```

If the LIB_A library contains cell A, which references cells B and C, IC Compiler first searches LIB_B for the cell; then, if the cell is not found, it searches LIB_C. If cell B in LIB_B references another cell, the referenced cell must exist in REF_LIB_B. If REF_LIB_B and REF_LIB_C both contain a cell with the same name, the hierarchy of the control file explicitly states which library the referenced cell should come from.

Converting a Milkyway Database Model

To work with IC Compiler version B-2008.09, existing Milkyway design libraries must be converted to the new database model. Before converting a Milkyway design library, you should save a backup copy of the design library with the previous database model.

Caution!

The new Milkyway database model is not backward compatible. You cannot use designs saved with the new Milkyway database model in Synopsys tools with a version prior to B-2008.09, such as Astro, Jupiter, or older versions of IC Compiler.

There are two ways to convert a Milkyway design to the new database model:

- Manual conversion by using the `convert_mw_lib` command
- Automatic conversion by using the `open_mw_cel` command

Note:

Both the `convert_mw_lib` command and the enhanced `open_mw_cel` command convert the designs in the Milkyway design library but not the designs in the Milkyway reference libraries. Database model conversion does not affect FRAM view cells, because they do not need to be updated.

Interface logic models (ILMs) are not included in the database model conversion. You must use the `create_ilm` command in IC Compiler version B-2008.09 to re-create the ILM view. Also, the ILM view that you create with IC Compiler version B-2008.09 cannot be used in Synopsys tools with a version prior to B-2008.09.

Manual conversion

To minimize the runtime and memory consumption when opening a design, you should manually convert designs to the new database model. To manually convert a design to the new database model, use the `convert_mw_lib` command.

You can either convert all designs in the Milkyway design library (`-all` option) or you can specify a design to convert (`-cell_name` option). When you convert a design, `convert_mw_lib` also converts all of its subdesigns and soft macros that are located in the same design library. It does not convert the subdesigns that are located in a reference library nor does it convert the interface logic models.

For each design being converted, the `convert_mw_lib` command opens the design, converts its database model, and saves it with an incremental version number.

To save time when converting a Milkyway design library to the new database model, use the `remove_mw_cel` command to remove cells in your design library that are no longer required before you run the `convert_mw_lib` command.

The syntax of the `convert_mw_lib` command is

```
convert_mw_lib mw_lib -all | -cell_name mw_cell  
mw_lib
```

Specifies the Milkyway design library containing the cells to be converted.

```
-cell_name mw_cell
```

This option converts the specified cell and saves it.

```
-all
```

This option converts all the cells in the design library.

You must specify either the `-cell_name` or the `-all` option when you run the `convert_mw_lib` command.

For example, to convert all designs in the `my_mw_lib` Milkyway design library to the new database model, enter the following command:

```
icc_shell> convert_mw_lib my_mw_lib -all
```

Automatic conversion

When you use the `open_mw_cel` command to open a Milkyway design, it checks the database model version and automatically updates the database model, if necessary.

The `open_mw_cel` command converts only the design being opened. It does not convert its subdesigns. If the `open_mw_cel` command detects that the design that you are opening has any subdesigns or soft macros that have references to the master cells located in the same design library, the command issues a message indicating that you need to use the `convert_mw_lib` command instead.

The `open_mw_cel` command does not save the converted design. To save the converted design, you must use the `save_mw_cel` command. To keep a copy of an older data model when using the `open_mw_cel` command, use the `save_mw_cel -increase_version` command or the `save_mw_cel -as` command. If you do not save it, the next time you use the `open_mw_cel` command to open the design, it automatically converts it again. The `save_mw_cel` command prints a message after a design is converted, indicating that a Synopsys tools with a version prior to B-2008.09 cannot open it.

If you open a design that uses an older database model in read-only mode, the `open_mw_cel` command issues an error and exits.

For more information about the `open_mw_cel` command, see [“Reading a Design in Milkyway Format” on page 3-20](#).

Validating a Milkyway Design Library

You can use the `check_library` command to check the quality of a Milkyway design library. To perform these physical library checks, you must enable the specific checks you want by running the `set_check_library_options` command (by default, the `check_library` command does not perform physical library checks). [Table 3-3](#) shows the physical library checks and the options used to enable them.

Table 3-3 Physical Library Checks

Use this option	To check for
<code>-antenna</code>	Missing antenna properties for cells and missing antenna rules for routing layers
<code>-drc</code>	Design rule constraint (DRC) violations in the routing (FRAM) view of the cells (requires write permission for the library directory)
<code>-phys_property {types}</code>	Physical properties (you can specify one or more of the following types: place, route, cell, and metal_density)
<code>-physical_only_cell</code>	Physical-only cells, such as filler cells with and without metal, diode cells with antenna properties, and corner cells
<code>-rectilinear_cell</code>	Coordinates of cells with rectangular or rectilinear boundaries
<code>-routeability</code>	The on-track accessibility of the physical pins
<code>-same_name_cell</code>	Cells with identical names in different reference libraries
<code>-signal_em</code>	Missing signal electromigration rules for routing layers
<code>-tech</code>	The quality of the technology data for a reference library
<code>-tech_consistency</code>	Consistency in the technology data between the design library and the reference libraries
<code>-view_comparison</code>	Consistency between the layout (CEL) view and the routing (FRAM) view of the cells

Specify the option for each check that you would like to enable, or to enable all physical library checks, use the `-physical` option.

To reset the library checks to the default settings (no physical library checks), run the `set_check_library_options -reset` command.

To see the enabled physical library checks, run the `report_check_library_options -physical` command.

If `check_library` reports any physical library problems, you should fix these problems before you process your design. For more information about physical libraries, see the *IC Compiler Data Preparation Using Milkyway User Guide*.

Verifying the Electromigration Constraints

To perform a check on the signal electromigration constraints in the library, use the following commands:

```
icc_shell> set_check_library_options -signal_em
1
icc_shell> check_library
#BEGIN_CHECK_LIBRARY
...
#BEGIN_CHECK_SIGNALEM
... (signal EM checking results here) ...
#END_CHECK_SIGNALEM
...
#END_CHECK_LIBRARY
```

If the signal electromigration constraints are not defined in the design library, get an updated incremental .plib file containing the electromigration data from your vendor, and then read it into IC Compiler with the following command:

```
icc_shell> set_mw_technology_file -plib plib_file_name design_lib_name
```

You can also use the Milkyway tool to update the design library, as described in the *IC Compiler Data Preparation Using Milkyway User Guide*.

Opening a Milkyway Design Library

To open an existing Milkyway design library, use the `open_mw_lib` command (or choose File > Open Library in the GUI).

If you have specified the TLUPlus files before you open the Milkyway design library, IC Compiler performs consistency checks between the library files in the Milkyway design library and the TLUPlus files, to ensure that you have a valid library setup; otherwise, you must manually perform this validation by running the `check_tlu_plus_files` command after defining the TLUPlus settings (for more information, see [“Setting Up the TLUPlus Files” on page 3-26](#)).

Reporting on a Milkyway Design Library

To report on the reference libraries attached to the design library, use the `-mw_reference_library` option.

```
icc_shell> report_mw_lib -mw_reference_library design_library_name
```

To report on the units used in the design library, use the `report_units` command.

```
icc_shell> report_units
```

Changing Physical Library Information

You can change the technology file and Milkyway reference libraries associated with your design library, as long as the design library you are changing is not already open.

If the Milkyway design library you want to change is open, close it by using the `close_mw_lib` command (or by choosing File > Close Library in the GUI).

Changing the Technology File

To modify the technology file data associated with a Milkyway design library,

1. Generate an editable version of the technology file associated with the Milkyway design library.

To generate an editable version of the technology file, use the `write_mw__lib_files -technology` command (or choose File > Export > Write Library File in the GUI). You specify the name of the generated technology file by using the `-output` option.

```
icc_shell> write_mw_lib_files -technology -output new_tech.tf \  
my_design.mw
```

2. Modify the generated technology file.
3. Attach the modified technology file to the Milkyway design library

To change the technology file attached to the Milkyway design library, use the `set_mw_technology_file` command (or choose File > Set Technology File in the GUI) to specify the new technology file name and the name of the design library.

For example, to change the technology file associated with the `my_design` design library to `new_tech.tf`, enter the following command:

```
icc_shell> set_mw_technology_file -technology new_tech.tf my_design
```

Note:

If you already have a technology file that contains the updated data, use step 3 to attach the new technology file and ignore steps 1 and 2.

Changing the Milkyway Reference Libraries

To change the Milkyway reference libraries, use the `set_mw_lib_reference` command (or choose File > Set Library Reference in the GUI). In the same way as when you create a Milkyway design library, you can specify the Milkyway reference libraries either individually (by using the `-mw_reference_library` option) or in a reference control file (by using the `-reference_control_file` option). For more information about the reference control file, see [“Creating a Milkyway Design Library” on page 3-10](#).

For example, to change the Milkyway reference library associated with the `my_design` design library to `new_mw_ref`, enter the following command:

```
icc_shell> set_mw_lib_reference \  
-mw_reference_library {./lib/new_mw_ref} my_design
```

Saving Physical Library Information

To save the technology or reference control information in a file for later use, use the `write_mw_lib_files` command (or choose File > Export > Write Library File in the GUI). In a single invocation of the command, you can output only one type of file. To output both a technology file and a reference control file, you must run the command twice. See [Table 3-4](#) for options for saving physical library information.

Table 3-4 Options for Saving Physical Library Information

Task	Command Line Option	GUI Field
Specify the input Milkyway design library	<i>libName</i>	“Library name”
Save the technology file	<code>-technology</code>	“Dump technology file”
Save the reference control file	<code>-reference_control_file</code>	“Dump reference control file”
Specify output file name	<code>-output file_name</code>	“Output file”

For example, to output the reference control file for the `my_design` design library to a file called `ref.out`, enter the following command:

```
icc_shell> write_mw_lib_files -reference_control_file \  
-output ref.out my_design
```

Verifying Library Consistency

Consistency between the logic library and the physical library is critical to achieving good results. Before you process your design, ensure that your libraries are consistent by running the `check_library` command.

```
icc_shell> check_library
```

By default, the `check_library` command performs consistency checks between the logic libraries specified in the `link_library` variable and the physical libraries that are referenced in the current Milkyway design library. You can also explicitly specify logic libraries by using the `-logic_library_name` option or Milkyway reference libraries by using the `-mw_library_name` option. The libraries that you explicitly specify override the default libraries.

The `-logic_library_name` option specifies a list of logic libraries to be checked for consistency. If the `-logic_library_name` option is not explicitly specified but the `set_min_library max_library` command option is specified, IC Compiler checks the minimum and maximum libraries as a pair. If neither the `-logic_library_name` option nor the minimum and maximum libraries are specified but the `link_library` and `search_path` variables are set, the command groups the libraries in the list by cell set, and within each group, it checks the consistencies across all the libraries. Libraries that cannot be paired are not checked.

Use the `-cells` option to specify a list of cell names to check. If the cell names are not specified, all the cells in the library are checked. The `check_library` command returns a status message indicating whether the check was successful or not.

By default, the `check_library` command performs the following consistency checks between the logic library and the physical library and verifies that:

- All cells exist in both the logic library and the physical library, including any physical-only cells.
- The pins on each cell are consistent between the logic library and the physical library.

This validation includes consistency in naming, direction, and type, including power and ground pins.

Note:

If the logic library does not contain `pg_pin` definitions, IC Compiler uses the power and ground pins as defined in the physical library.

You can also perform the following consistency checks, by setting the appropriate options with the `set_check_library_options` command:

- Verify that the area for each cell (except pad cells) is consistent between the logic library and the physical library.

```
icc_shell> set_check_library_options -cell_area
```

- Verify that the cell footprints are consistent between the logic library and the physical library.

```
icc_shell> set_check_library_options -cell_footprint
```

- Verify that the same bus naming style is used in the logic library and the physical library.

```
icc_shell> set_check_library_options -bus_delimiter
```

Specify the option for each check that you would like to enable, or to enable all consistency checks, use the `-logic_vs_physical` option.

To reset the library checks to the default settings (cell name and pin consistency checks), run the `set_check_library_options -reset` command.

To see the enabled library consistency library checks, run the `report_check_library_options -logic_vs_physical` command.

If `check_library` reports any inconsistencies, you must fix these inconsistencies before you process your design. For more information about logic libraries, see the Library Compiler documentation. For more information about physical libraries, see the *IC Compiler Data Preparation Using Milkyway User Guide*.

Reading the Design

IC Compiler can read designs in either Milkyway or ASCII (Verilog, DEF, and SDC) format.

In either case, before reading in the design, you must perform the following tasks:

1. Set up the logic libraries, as described in [“Setting Up the Logic Libraries” on page 3-2](#).

```
icc_shell> set search_path my_search_path
icc_shell> set target_library my_library_list
icc_shell> set link_library "* my_link_library_list"
```

2. Open the Milkyway design library associated with the design, as described in [“Opening a Milkyway Design Library” on page 3-15.](#)

```
icc_shell> open_mw_lib design_library_name.mw
```

Note:

If you do not have a Milkyway design library, see [“Creating a Milkyway Design Library” on page 3-10.](#)

The following sections explain how to open designs:

- [Reading a Design in Milkyway Format](#)
- [Reading a Design in ASCII Format](#)

Reading a Design in Milkyway Format

To open a design in Milkyway format,

1. Verify that you have defined the correct main library.

The unit settings in the Milkyway design must be consistent with the unit settings in the main library (the first library in the `link_library` definition). To see the main library unit settings, use the `report_lib` command. To see the Milkyway design library unit settings, use the `report_units` command.

To prevent problems with opening the Milkyway design, ensure that the main library is the same one that was used when the Milkyway design was saved.

2. Open the design by using the `open_mw_cel` command (or by choosing File > Open Design in the GUI).

```
icc_shell> open_mw_cel design_name
```

Note:

If you get unit inconsistency errors when you try to open the Milkyway design, go back to step one and ensure that the main library is correctly defined.

When you open a Milkyway design, IC Compiler

- Checks the database model version and automatically updates the database model, if necessary.

The `open_mw_cel` command converts only the design being opened. It does not convert its subdesigns. If the `open_mw_cel` command detects that the design that you are opening has any subdesigns or soft macros that have references to the master cells located in the same design library, the command issues a message indicating that you need to use the `convert_mw_lib` command instead.

For more information about converting the database model, see [“Converting a Milkyway Database Model” on page 3-12](#).

- Locks the design, which prevents other copies of it from being used and updated while you are working on it.

If you want to use the same Milkyway design for multiple runs, copy the design (use the `copy_mw_cel` command or choose File > Copy Design in the GUI) to make a uniquely named copy of the design that you can open.

```
icc_shell> open_mw_lib DL.mw
icc_shell> copy_mw_cel -from CEL1 -to CEL1_cp1
icc_shell> open_mw_cel CEL1_cp1
```

Reading a Design in ASCII Format

When you are ready to read a design in ASCII format, you first open the Milkyway design library and then read the Verilog, constraint, and DEF data for the design. After you read in the design, IC Compiler stores the design in Milkyway format for use in future sessions.

To read an ASCII-format design into IC Compiler,

1. Read the Verilog netlist file for the design by using the `read_verilog` command (or by choosing File > Import > Read Verilog in the GUI).

The Verilog netlist file is a structural or gate-level design in one file.

```
icc_shell> read_verilog -dirty_netlist design.v
```

2. Uniquify the design by using the `uniquify_fp_mw_cel` command.

The Milkyway format does not support multiply instantiated designs. Before saving the design in Milkyway format, you must uniquify the design to remove multiple instances.

```
icc_shell> uniquify_fp_mw_cel
```

3. Set the current design to the top level of the design by using the `current_design` command.

```
icc_shell> current_design top_level_design_name
```

4. Link the design by using the `link` command (or by choosing File > Import > Link Design in the GUI).

```
icc_shell> link
```

Note:

The `import_designs` command performs steps 1 to 4 with a single command.

```
icc_shell> import_designs -format verilog \  
-top top_level_design_name -cel top_level_design_name
```

5. Read the timing constraints for the design by using the `read_sdc` command (or by choosing File > Import > Read SDC in the GUI).

For information about setting timing constraints, see [“Setting Timing Constraints” on page 3-28](#).

6. Annotate the physical data on the design.

For information about annotating the physical data, see [“Annotating the Physical Data” on page 3-22](#).

Annotating the Physical Data

IC Compiler provides several methods of annotating physical data on the design:

- Reading the physical data from a DEF file
- Reading the physical data from a floorplan file
- Copying the physical data from another design

If a floorplan does not yet exist for your design, see the *IC Compiler User Guide: Design Planning* for information about design planning.

After annotating (or creating) the physical data, validate the physical data before performing physical synthesis.

Reading DEF Files

To read a DEF file, use the `read_def` command (or choose File > Import > Read DEF in the GUI).

```
icc_shell> read_def design_name.def
```

By default, the `read_def` command is additive and adds the physical data in the DEF file to the existing physical data in the design. To avoid this, deselect Incremental in the GUI or use the `-no_incremental` option on the command line.

To analyze the input DEF files before annotating the objects on the design, enable check-only mode by using the `-check_only` option. The check-only mode provides diagnostic information on the correctness and integrity of the DEF file. The check-only mode does not annotate any DEF information into the Milkyway database.

```
icc_shell> read_def -check_only design_name.def
```

For detailed information about the `read_def` command, see the *IC Compiler Data Preparation Using Milkyway User Guide*.

Reading Floorplan Files

A floorplan file is a file that you previously created by using the `write_floorplan` command (or by choosing Floorplan > Write Floorplan in the GUI). This file is a script that contains information about the floorplan. For more information about creating a floorplan file, see the *IC Compiler User Guide: Design Planning*.

Note:

If the power and ground nets are not explicitly defined in your netlist, you must create the power and ground connections by using the `derive_pg_connection` command before reading the floorplan file. For information about how to do this, see the *IC Compiler User Guide: Design Planning*.

To read a floorplan file, use the `read_floorplan` command.

```
icc_shell> read_floorplan floorplan_file_name
```

For detailed information about the `read_floorplan` command, see the man page.

Copying Physical Data

To copy physical data from the layout (CEL) view of one design in the current Milkyway design library to another, use the `copy_floorplan` command (or choose Floorplan > Copy Floorplan in the GUI).

Note:

It is up to the user to ensure netlist consistency between the designs. No netlist consistency checks are performed during the copy process.

Specify the design from which to copy the physical data by using the `-from` option (or in the “From cell” field in the GUI).

Note:

The GUI has “Library name” and “To cell” fields. These fields are automatically filled in with the current Milkyway design library and current design, respectively. You cannot change these values.

```
icc_shell> copy_floorplan -from design1
```

This command copies the following physical data:

- General floorplan data

This includes information such as the design boundary, the core boundary, the placement rows, the placement sites, and the wire tracks.

- Obstructions

These include information such as placement blockages (soft and hard), keepout margins (soft and hard), all route guides, and move bounds (soft and hard).

- Cell instances

These include fixed, soft-fixed, and placed cells (such as macros, pads, and preplaced standard cells), as well as physical cells (such as corner pads, filler cells, and via cells).

Note:

All cell instances are copied; there is no way to select specific cell types to copy.

- Preroutes

These include power and ground preroutes, clock preroutes, signal preroutes, and shieldings.

- Power net connections
- Plan groups

- Macro cell placement (optional)

To copy the macro cell placement information, use the `-macro` option (or select “Copy macro placement” in the GUI).

- I/O pad placement (optional)

To copy the I/O pad placement information, use the `-pad` option (or select “Copy IO pad placement” in the GUI).

- Filler cell placement (optional)

To copy the filler cell placement information, use the `-filler` option (or select “Copy Filler cells placement” in the GUI).

- Power plan (optional)

To connect the power nets and copy the power planning information, use the `-power_plan` option (or select “Connect power net and copy power planning data” in the GUI).

The command does not copy the following data:

- SCANDEF
- Voltage areas and power domains
- Data from the FILL view, such as floating metal fill or notch and gap

By default, the `copy_floorplan` command overwrites the existing physical data in the design. To avoid this, use the `-incremental` option (or select Incremental in the GUI).

Note:

The `copy_floorplan -incremental` command copies the ROWS from the source cell to the destination cell, even though there are already ROWS in the destination cell.

For detailed information about the `copy_floorplan` command, see the man page.

Validating Physical Data

To validate the physical data in your design, run the `check_physical_design` command. This command verifies that the current design can be linked, that the placement area is defined, and that all ports have a location. In addition, it verifies that the TLUPlus files are consistent with the physical libraries.

Preparing for Timing Analysis and RC Calculation

IC Compiler provides RC calculation technology and timing analysis capabilities for both preroute and postroute data. Before you perform RC calculation and timing analysis, you must complete the following tasks:

- Set up the TLUPlus files
- (Optional) Back-annotate delay or parasitic data
- Set the timing constraints
- Specify the analysis mode
- (Optional) Set the derating factors
- Select the delay calculation algorithm

The following sections describe these tasks.

Setting Up the TLUPlus Files

TLUPlus is a binary table format that stores the RC coefficients. The TLUPlus models enable accurate RC extraction results by including the effects of width, space, density, and temperature on the resistance coefficients. For details about modeling these effects in the Interconnect Technology Format (ITF) file, see the Star-RCXT documentation.

To use TLUPlus models for RC estimation and extraction, you specify the following files:

- The map file, which matches the layer and via names in the Milkyway technology file with the names in the ITF file.
- The maximum TLUPlus model file.
- The minimum TLUPlus model file (optional). Specifying the minimum TLUPlus file is mandatory if the minimum and maximum operating conditions are different **and** the TLUPlus models have derating coefficients. In that case, the minimum file must be specified even if it is the same as the maximum file.

You specify these files by using the `set_tlu_plus_files` command (or by choosing File > Set TLU+ in the GUI). You can specify TLUPlus file names with or without their full paths. If you do not include the paths, IC Compiler uses the search paths defined with the `search_path` variable to find the files.

```
icc_shell> set_tlu_plus_files \
-tech2itf_map ./path/map_file_name.map \
-max_tluplus ./path/worst_settings.tlup \
-min_tluplus ./path/best_settings.tlup
```

After specifying the TLUPlus files, you must validate them by running the `check_tlu_plus_files` command. An invalid TLUPlus setup can cause errors when you process your design. When you validate the TLUPlus files, IC Compiler performs consistency checks for conducting layer names and via layer names across the ITF file, Milkyway file, and mapping files. It also verifies that the minimum width and minimum spacing rules for metal layers are consistent between the ITF file and the Milkyway technology file, but it does not check minimum width and minimum spacing for via layers. IC Compiler uses the technology file as the reference for these checks. To perform a consistency check on TLUPlus files, IC Compiler compares the parameters in the TLUPlus files with the corresponding fields in the technology file.

IC Compiler provides several commands for working with the TLUPlus file settings. [Table 3-5](#) lists these commands and their functions.

Table 3-5 Commands for Working With the TLUPlus File Settings

Task	Command	GUI
Define the TLUPlus file settings	<code>set_tlu_plus_files</code>	File > Set TLU+
Validate the TLUPlus file settings	<code>check_tlu_plus_files</code>	N/A
Report the TLUPlus file settings	<code>report_tlu_plus_files</code>	N/A

If you set the current design or link your design before setting up the TLUPlus files, IC Compiler saves the names and locations of the TLUPlus files in the Milkyway design library when you save your design. If you do not want to save these settings, set the `save_mw_cel_lib_setup` variable to false (the default is true). If you do not save these settings, they take effect only for the current session.

By default, even if you save these settings in the Milkyway design library, they are not restored when you open the design. If you save these settings in the Milkyway design library and want IC Compiler to restore the settings when opening the design, set the `auto_restore_mw_cel_lib_setup` variable to true (or select “Restore library setup” in the GUI).

Important:

The `save_mw_cel_lib_setup` and `auto_restore_mw_cel_lib_setup` variables also apply to the logic library variable definitions. In general, variable settings are not saved with the design. Because the ability to restore variable settings, such as those used to define the logic libraries, enables an inconsistent user model, the `save_mw_cel_lib_setup` and `auto_restore_mw_cel_lib_setup` variables will be obsolete in a future release.

Back-Annotating Delay or Parasitic Data

If you have existing delay or parasitic data, you can back-annotate the design with this data instead of using the extraction capability to calculate the data.

To back-annotate the design with delay information provided in a Standard Delay Format (SDF) file, use the `read_sdf` command (or choose File > Import > Read SDF in the GUI). To back-annotate the design with parasitic capacitance and resistance information, use the `read_parasitics` command. IC Compiler accepts parasitic data in Synopsys Binary Parasitic Format (SBPF) or Standard Parasitic Exchange Format (SPEF).

Note:

If you read the parasitic data in SPEF format, IC Compiler uses this data for analysis only. If you read the data in SBPF format, IC Compiler can use the data for both analysis and on-route optimization.

To remove annotated data from your design, use the `remove_annotations` command.

Setting Timing Constraints

At a minimum, the timing constraints must contain a clock definition for each clock signal, as well as input and output arrival times for each I/O port. This requirement ensures that all signal paths are constrained for timing.

To model the clock tree effects for placement (before running clock tree synthesis), you should also define the following constraints for each clock: `set_clock_uncertainty`, `set_clock_latency`, and `set_clock_transition`.

You can use a file of Synopsys Design Constraints (SDC) commands or use individual SDC commands. For details about the SDC commands, see the *Using Synopsys Design Constraints Format Application Note*.

To read a timing constraints file, use the `read_sdc` command (or choose File > Import > Read SDC in the GUI).

```
icc_shell> read_sdc -version 1.7 design_name.sdc
```

Important:

If the SDC file does not contain unit settings, they are derived from the main library (see [“Setting Up the Logic Libraries” on page 3-2](#)). If the SDC file does contain unit settings, they must be consistent with those in the main library; otherwise, `read_sdc` fails. If this happens, update the units and constraint values in the SDC file and rerun `read_sdc`.

IC Compiler does not optimize paths that are not constrained for timing. Before proceeding, use the `check_timing` command to verify that all paths are constrained. If the `check_timing` command reports unconstrained paths, run the `report_timing_requirements` command to verify that the unconstrained paths are false paths (the `check_timing` command considers false paths unconstrained).

To remove the timing constraints, use the following commands:

- `remove_sdc`

This command removes the timing constraints set by SDC commands.

Note:

If you have run extraction on your design, you can prevent removal of the extracted RC network by using the `-keep_parasitics` option when you run the `remove_sdc` command. Keeping the RC network saves runtime because IC Compiler does not need to redo the extraction.

- `remove_ideal_network -all`

This command removes `ideal_network` attributes, latencies, and transition times.

- `reset_design`

This command removes all attributes from the design, including timing constraints, optimization attributes, and physical information.

Specifying the Analysis Mode

Semiconductor device parameters can vary with conditions such as fabrication process, operating temperature, and power supply voltage. The `set_operating_conditions` command specifies the operating conditions for analysis, so that IC Compiler uses the appropriate set of parameter values in the technology library. When you run the `set_operating_conditions` command, you can specify either a maximum operating condition (`-max` option) only or both a minimum operating condition (`-min` option) and a maximum operating condition (`-max` option).

IC Compiler offers two analysis modes with respect to operating conditions:

- Best-case and worst-case (default)

In the best-case and worst-case mode, IC Compiler simultaneously checks the circuit for the two extreme operating conditions, minimum and maximum. For setup checks, it uses maximum delays for all paths. For hold checks, it uses minimum delays for all paths. This mode lets you check both extremes in a single analysis run, thereby reducing overall runtime for a full analysis.

- On-chip variation (`-analysis_type on_chip_variation`)

In the on-chip variation mode, IC Compiler performs a conservative analysis that allows both minimum and maximum delays to apply to different paths at the same time. For a setup check, it uses maximum delays for the launch clock path and datapath and minimum delays for the capture clock path. For a hold check, it uses minimum delays for the launch clock path and datapath and maximum delays for the capture clock path.

If you are performing simultaneous minimum and maximum timing analysis, the logic libraries specified by the `link_library` variable are used for both maximum and minimum timing information, unless you specify separate minimum timing libraries by using the `set_min_library` command. The `set_min_library` command associates minimum timing libraries with the maximum timing libraries specified in the `link_library` variable. For example,

```
icc_shell> set link_library "* maxlib.db"
icc_shell> set_min_library maxlib.db -min_version minlib.db
```

To find out which libraries have been set to be the minimum and maximum libraries, use the `list_libs` command. In the generated report, the letter “m” appears next to the minimum library and the letter “M” appears next to the maximum library.

Setting the Derating Factors

If your timing library does not include minimum and maximum timing data, you can perform simultaneous minimum and maximum timing analysis by specifying derating factors for your timing library.

Use the `set_timing_derate` command to specify the derating factors. For example, to specify that for the maximum operating condition, all early (shortest-path) delays should be decreased by 10 percent and all late (longest-path) delays should be increased by 20 percent, enter the following commands:

```
icc_shell> set_timing_derate -max -early 0.9
icc_shell> set_timing_derate -max -late 1.2
```

For more information about the `set_timing_derate` command, see the man page.

To report the derating factors, use the `report_timing_derate` command. To reset the derating factors to 1.0, use the `reset_timing_derate` command.

Setting the Scaling Factors

If you are using a multivoltage or multicorner design flow, you need timing information for each operating condition. Rather than having separate libraries for each operating condition, you can take advantage of voltage and temperature scaling to get the required timing information.

IC Compiler uses scaling library groups to implement voltage and temperature scaling. IC Compiler derives the timing data for a specified operating condition by interpolating the data from the libraries in the scaling library group.

To specify the libraries in the scaling library group, use the `define_scaling_lib_group` command. A scaling library group must contain at least two libraries, and the libraries must meet the following conditions:

- Each library contains CCS (or mixed CCS and NLDM) models
- The cell data is consistent among the libraries in terms of
 - Cell names
 - Number and names of pins
 - Number and names of timing arcs

It is not necessary for the cells, pins, or timing arcs to be defined in the same order in each library, as long as the data is consistent.

Note:

The `check_library` command does not check for consistency among the libraries in the scaling library group. You must manually ensure consistency among these libraries.

To set a scaling library group on a set of ports or cells, use the `set_scaling_lib_group` command. You can define either a minimum and a maximum scaling library group, or a single scaling library group to be used for both minimum and maximum scaling.

To remove the scaling library group from a set of ports or cells, use the `remove_scaling_lib_group` command.

Selecting the Delay Calculation Algorithm

By default, IC Compiler uses Elmore delay calculation for both preroute and postroute delay calculations. For postroute delay calculations, you can choose to use Arnoldi delay calculation either for clock nets only or for all nets. Elmore delay calculation is faster, but its results do not always correlate with the PrimeTime and PrimeTime SI results. The Arnoldi calculation is best used for designs with smaller geometries and highly resistive nets, but it requires more runtime and memory.

If you are not sure which calculation method to use, use the `compare_delay_calculation` command to compare the results of sample delay calculations for your design. If the results are similar for both methods, use the faster Elmore method. Otherwise, use the Arnoldi method.

To enable Arnoldi delay calculation for clock nets only, enter the following command:

```
icc_shell> set_delay_calculation -clock_arnoldi
```

Note:

This delay model should be used only for signal path optimization after clock routing is complete. Using this delay model for postroute clock tree optimization can result in reduced clock tree QoR.

To enable Arnoldi delay calculation, enter the following command:

```
icc_shell> set_delay_calculation -arnoldi
```

Note:

Using the Arnoldi delay model removes existing back-annotation stored in the database (unlike using the Elmore delay model). In addition, if you save a design with Arnoldi settings (`set_delay_calculation -clock_arnoldi` or `set_delay_calculation -arnoldi`), the delay information is not saved in the Milkyway design.

By default, IC Compiler does not include crosstalk delta delays in the delay calculations. To extract coupling capacitances and include crosstalk delta delays in the postroute delay calculations, enter the following command:

```
icc_shell> set_si_options -delta_delay true
```

Saving the Design

You can save the design for use with Milkyway design tools, with design tools that do not use Milkyway, and for manufacturing. If you are going to work with the design in IC Compiler or another design tool based on Milkyway, save the design as explained in [“Saving the Design in Milkyway Format” on page 3-33](#). If you are going to use a tool that does not use Milkyway, export the design to ASCII format (which includes Verilog, DEF, SDC, and parasitic files), as explained in [“Saving the Design in ASCII Format” on page 3-33](#). If you have completed the design and are ready for manufacturing, save it in GDSII or Oasis format, as explained in [“Saving the Design for Manufacturing” on page 3-34](#).

Saving the Design in Milkyway Format

To save the design in Milkyway format, use the `save_mw_cel` command (or choose File > Save Design in the GUI).

If your design does not have unit settings, IC Compiler applies the default settings from the main library and saves them when you save the design. For more information about the main library and its default settings, see [“Setting Up the Logic Libraries” on page 3-2](#).

Note:

When you exit the IC Compiler GUI, the Exit IC Compiler dialog box gives you the option of automatically saving your design in Milkyway format. For more information about this dialog box, see [Chapter 2, “Working With IC Compiler.”](#)

Saving the Design in ASCII Format

To export the design to ASCII format,

1. Write Verilog data for the design by using the `write_verilog` command.

```
icc_shell> write_verilog design_name.v
```

2. Write the physical data to DEF files by using the `write_def` command (or by choosing File > Export > Write DEF in the GUI).

```
icc_shell> write_def -out design_name.def
```

3. Write the design constraints to SDC files by using the `write_sdc` command.

```
icc_shell> write_sdc design_name.sdc
```

If the SDC file does not have unit settings, IC Compiler uses the default settings from the main library and saves them when you save the SDC file. For more information about the main library and its default settings, see [“Setting Up the Logic Libraries” on page 3-2](#).

4. Write the RC extraction data to the parasitic netlist by using the `write_parasitics` command (or by choosing File > Export > Write Parasitics in the GUI).

IC Compiler can write parasitic netlists in the SPEF and SBPF formats. Specify the format by using the `-format` option (or the “Net format in file” field in the GUI). You can also compress the output in gzip format by using the `-compress` option (or by selecting “Write in gzip format” in the GUI).

```
icc_shell> write_parasitics -format SPEF -output design_name.spef
```

Saving the Design for Manufacturing

After you have finished the design, you can export the design library in GDSII or Oasis format for manufacturing.

To export the design library for manufacturing,

- Use the `write_stream` command with at least the name of the exported file. You can also specify the following:
 - Name of the library containing the layout cells if not using the currently opened library
 - Cells to be output if not all cells in the design are being output
 - Output format (if not using GDSII)
 - Name of the file listing the cells (if using GDSII format)

For detailed information about the `write_stream` command, see the man page.

4

Placement

This chapter describes the IC Compiler placement and optimization capabilities.

This chapter contains the following sections:

- [Defining Placement Blockages](#)
- [Placement Options](#)
- [Preparing for High-Fanout Net Synthesis](#)
- [Performing Clock Tree Synthesis During Placement](#)
- [Performing Placement and Optimization](#)
- [Using Physical Optimization](#)
- [Preroute RC Estimation](#)
- [Analyzing Placement](#)
- [Refining Placement](#)

Defining Placement Blockages

Placement blockages are areas that leaf cells must avoid during placement and legalization, including overlapping any part of the placement blockage. Placement blockages can be hard or soft.

- A hard blockage prevents cells from being put in the blockage area.
- A soft blockage restricts the coarse placer from putting cells in the blockage area, but optimization and legalization can place cells in a soft blockage area.

If you define both hard and soft placement blockages in your design, the hard placement blockages take priority over the soft placement blockages in places where they overlap.

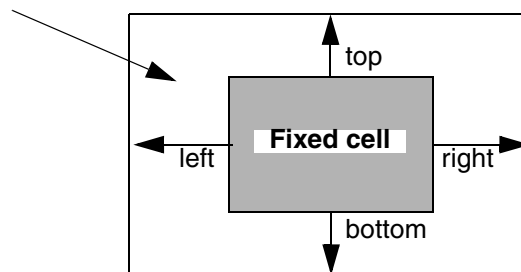
You can define a placement blockage by either specifying a region around fixed macros (keepout margins) or by specifying a rectangular blockage area (area-based placement blockages).

Defining Keepout Margins

A keepout margin is a region (the unshaded portion in [Figure 4-1](#)) around the boundary of fixed macros in your design in which no other cells are placed.

Figure 4-1 Placement Keepout Margin

Keepout margin



Keeping the placement of cells out of such regions avoids congestion and net detouring and produces better QoR.

Keepout margins can be defined as hard or soft. In addition, you can define global keepout margins, which apply to all fixed macros in the design, or cell-specific keepout margins.

The width of the keepout margin on each side of a fixed macro can be the same or different, depending on how you define the keepout margin.

Defining Global Keepout Margins

To define a hard global keepout margin, specify the width, in microns, of the keepout margin by setting the `physopt_hard_keepout_distance` variable. The specified width is used for all sides of each fixed macro in your design.

For example, to specify a hard keepout margin of 10 microns around each fixed macro in your design, enter

```
icc_shell> set physopt_hard_keepout_distance 10
```

To define a soft global keepout margin, specify the width, in microns, of the keepout margin by setting the `placer_soft_keepout_channel_width` variable.

A soft global keepout margin does not apply to every fixed macro in your design or to every side of the fixed macro; it applies only in the following cases:

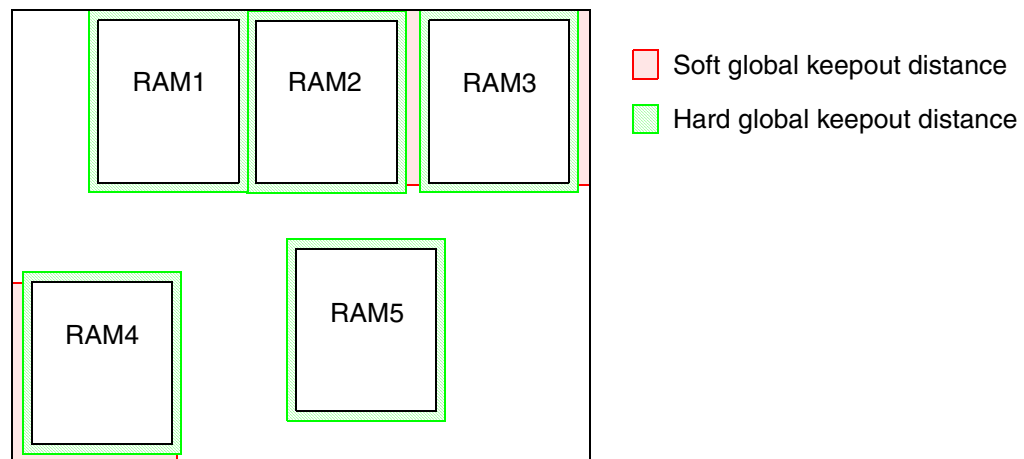
- The distance between the fixed macro boundary and the core area boundary is less than the soft keepout margin.
- The distance between two fixed macros is less than the soft keepout margin, forming a thin channel between the objects.

For example, to specify a soft keepout margin of 25 microns around each fixed macro in your design, enter

```
icc_shell> set placer_soft_keepout_channel_width 25
```

Figure 4-2 shows an example of using both hard and soft global keepout margins in a design containing five RAM cells. The hard keepout margins are applied to all macros, whereas the soft keepout margins are applied only where the distance requirements are met.

Figure 4-2 Global Keepout Distances



Defining Cell-Specific Keepout Margins

To define a keepout margin with different widths on each side or to define a keepout margin for specific cells, use the `set_keepout_margin` command (or choose Placement > Set Keepout Margin in the GUI). This is the full command syntax:

```
set_keepout_margin
  [-type hard | soft]
  [-outer {lx by rx ty}]
  [-tracks_per_macro_pin value]
  [-min_padding_per_macro value]
  [-max_padding_per_macro value]
  [-all_macros] [-macro_masters] [-macro_instances]
  [object_list]
  [-north]
```

For example,

```
icc_shell> set_keepout_margin -type hard -all_macros \
          -outer {10 10 10 10}
```

Use `-type hard` or `-type soft` to specify the type of keepout, either hard or soft. The default is hard. Use `-all_macros` to apply the keepout margins to all macros, `-macro_masters object_list` for all instances of specified macro masters, or `-macro_instances object_list` for specified instances of macros.

You can specify explicit keepout margins by using `-outer {lx by rx ty}`, where the four numbers are the left, bottom, right, and top margins. A value of 0 results in no keepout margin for that side. The `-north` option sets the margins with respect to the north orientation of the cell.

Instead of specifying the keepout margins explicitly, you can have them derived automatically by using `-tracks_per_macro_pin value`. In that case, the keepout margin is calculated from the track width, the number macro pins, and the specified track-to-pin ratio, which is typically set to a value near 0.5. A larger value results in larger keepout margins. The derived keepout margin is always hard; the `-type` setting is ignored. For derived margins, the `-all_macros`, `-macro_masters`, and `-macro_instance` options are not allowed. The derived margins are subject the minimum and maximum values specified by `-min_padding_per_macro` and `-max_padding_per_macro`. Here is an example:

```
icc_shell> set_keepout_margin -tracks_per_macro .6 \
          -min_padding_per_macro .1 -max_padding_per_macro 0.2
```

For more information on setting keepout margins, see the man page for the `set_keepout_margin` command.

To report the cell-specific keepout margins in your design, use the `report_keepout_margin` command. It reports the keepout margin values set by the `set_keepout_margin` command, the values of any pin-count-based parameters defined by the `set_keepout_margin` command, and all derived keepout margins for the specified macro cells and standard cells.

To remove the cell-specific keepout margins from your design, use the `remove_keepout_margin` command.

Inserting Port Protection Diodes

IC Compiler can automatically insert protection diodes on subdesign ports to prevent antenna violations at the top level. You insert the port protection diodes after floorplanning but before starting placement. Use the `insert_port_protection_diodes` command to add diodes to the specified ports to your netlist, one diode per port. The new diodes will be legalized near their corresponding ports. Note the following limitations:

- This command does not add diodes to ports with the `dont_touch` attribute.
- This command is not voltage-area aware. You need to select correct voltage diodes for ports in different voltage areas.

To report the port protection diodes that are inserted in your design, use the `report_port_protection_diodes` command.

Defining Area-Based Placement Blockages

Hard placement blockages can be defined in the DEF as shown in [Example 4-1](#).

Example 4-1 Placement Blockages in DEF

```
BLOCKAGES 2 ;
- PLACEMENT
  RECT ( 0 327600 ) ( 652740 327660 ) ;
- PLACEMENT
  RECT ( 0 327600 ) ( 652740 327660 ) ;
END BLOCKAGES
1
```

You can also create placement blockages in IC Compiler with the `create_placement_blockage` command (or choose Floorplan > Create Placement Blockage in the GUI). This is the command syntax:

```
create_placement_blockage
  -bbox {llx lly urx ury}
  [-type hard | soft | pin | hard_macro | partial]
  [-blocked_layers string]
  [-blocked_percentage integer]
  [-name blockage_name]
```

For example, to create a soft placement blockage in the area enclosed by a rectangle with corners at (10,20) and (100,200), you would use the following command:

```
icc_shell> create_placement_blockage -bbox {10 20 100 200} -type soft
```

The `-type` option specifies the type of blockage: `hard`, `soft`, `pin`, `hard_macro`, or `partial`. Each type of blockage restricts usage of the specified area in the following ways:

- A `hard` blockage prevents the placement of standard cells and hard macros.
- A `soft` blockage prevents the placement of standard cells and hard macros unless the congestion is too high to allow placement elsewhere.
- A `pin` blockage prevents the global router from routing in the area and the pin placer from assigning pins to the area. You can optionally specify which layers are blocked with the `-blocked_layers` option. If you do not specify the layers, all layers are blocked.
- A `hard_macro` blockage prevents the placement of hard macros, but not standard cells.
- A `partial` blockage partially prevents placement of cells in an area. The `-blocked_percentage integer` option must be used to specify the percentage of the area that is blocked.

For example, to set a partial blockage of 40 percent of the area enclosed by the rectangle with corners at (10,20) and (100,200), you would use the following command:

```
icc_shell> create_placement_blockage -bbox {10 20 100 200} \
          -type partial -blocked_percentage 40
```

In this example, the specified area has a blockage of 40 percent, so the maximum allowed cell density is 60 percent. Partial blockage applies only to coarse placement. It has no effect on legalization or optimization.

You can optionally assign a name to a blockage by using the `-name` option. You can then reference that blockage by name when you use the `get_placement_blockage` or `remove_placement_blockage` command.

You can display and select placement blockages in the layout view of the GUI.

To report placement blockages in the design, use the `report_placement_blockages` command.

To return a collection of placement blockages in the current design that match certain criteria, use the `get_placement_blockages` command.

To remove placement blockages from the design, use the `remove_placement_blockage` command.

Placement Options

There are many configuration settings that affect the behavior of placement in IC Compiler, including the following controls:

- [Congestion Options](#)
- [Move Bounds](#)
- [Intercell and Boundary Spacing Rules](#)
- [Buffer Strategy for Optimization](#)
- [Tie Cell Insertion](#)
- [Magnet Placement](#)
- [Placement and Optimization Attributes](#)

Congestion Options

IC Compiler attempts to minimize congestion during placement and optimization. Congestion occurs when the number of wires going through a region exceeds the capacity of that region. This condition is detected by global routing. IC Compiler places and moves cells in such a manner to avoid congestion and to fix congestion problems when they occur.

You can set certain options related to congestion avoidance with the `set_congestion_options` command:

```
set_congestion_options
  [-max_util value]
  [-layer name]
  [-availability value]
  [-coordinate {X1 Y1 X2 Y2}]
```

The `-max_util` option specifies how densely cells can be packed in uncongested regions to relieve congestion in other regions. You should set this variable based on how much area is needed for placement. The default maximum utilization is 0.95.

The `-layer` and `-availability` options specify how much of the routing resource for the given layer is available to be used. For example, in a design whose M5 and M6 layers will be 70 percent occupied by future power and ground routing, you can set the availability of M5 and M6 to 0.30.

Congestion options can be design-wide or regional. Use the `-coordinate` option to specify two corners of the bounding box of the area to be affected by the command. Otherwise, the command affects the whole chip.

You can report and remove congestion option settings with the `report_congestion_option` and `remove_congestion_option` commands. To report congestion conditions, use the `report_congestion` command. You can visualize congestion conditions by using the global route congestion visual mode as described in [“Congestion Map” on page 4-36](#).

Move Bounds

Move bounds are placement constraints that control the placement of groups of leaf cells and hierarchical cells. Move bounds can be either soft, hard, or exclusive.

- Soft move bounds specify placement goals, with no guarantee that the cells will be placed inside the bounds.
- Hard move bounds force placement of the specified cells inside the bounds.
- Exclusive move bounds force the placement of the specified cells inside the bounds. All other cells must be placed outside the bounds.

Defining a move bound allows you to group cells to minimize wire length and place the cells at the most appropriate locations. For example, you might want to define a move bound for clock-gating cells or extremely timing-critical groups of cells that you want to guarantee will not be disrupted for placement by other logic. During placement, IC Compiler ensures that the cells you grouped remain together. However, defining move bounds restricts placement and defining too many can lower QoR. For best results, make the number of cells you place in placement bounds relatively small compared to the total number of cells in the design.

To create a move bound in IC Compiler, use the `create_bounds` command (or choose Floorplan > Create Bound in the GUI). This is the command syntax:

```
create_bounds
  [-name bound_name]
  [-coordinate {llx1 lly1 urx1 ury1 ...}]
  [-dimension {width height}]
  [-effort low | medium | high | ultra]
  [-type soft | hard]
  [-exclusive]
  [-color range_0_to_63]
  [-cycle_color]
  object_list
```

For example,

```
icc_shell> create_bounds -name "b1" -coordinate {100 100 200 200} INST_1
```

You must specify a list of cells, ports, and pins to be included in the bound. If a hierarchical cell is included, all cells in the subdesign belong to the bound. You can also specify the characteristics of the bound, including the name, the coordinates of one or more rectangles for the bound, the dimensions, the effort level to bring the cells closer together, the type (soft, hard, or exclusive; soft is the default). You can also specify the color used to display the bound in the GUI.

To add or remove cells from a move bound, use the `update_bounds` command.

To report the move bounds in your design, use the `report_bounds` command.

To return a collection of move bounds in the current design that match certain criteria, use the `get_bounds` command.

To remove move bounds from your design, use the `remove_bounds` command.

Intercell and Boundary Spacing Rules

IC Compiler places standard cells by using a discrete grid, which is defined by the unit tile. By default, there are no spacing constraints between cells during placement. To enhance yield, you can define the valid spacing between standard cells in the same row.

IC Compiler implements intercell spacing rules by attaching labels, which are similar to attributes, to the left and right sides of library cells, assuming that the cell is in its north orientation, and specifying the invalid spacings between these labels.

To define the intercell spacing rules,

1. Add labels to the right and left sides of library cells that have minimum spacing constraints by using the `set_lib_cell_spacing_label` command.

Use the `-name` option to specify the label names. You can specify one or more labels. The labels are assigned to the right side of the cells specified with the `-right_lib_cells lib_cells` option and to the left side of the cells specified with the `-left_lib_cells lib_cells` option. You can assign the same label to both the right and left sides of a cell by including it in both the `-right_lib_cells` and `-left_lib_cells` options.

For example, to assign a label named X to the right side of the cellA library cell and the left side of the cellB and cellC library cells, enter the following command:

```
icc_shell> set_lib_cell_spacing_label -names {X} \  
-right_lib_cells {cellA} -left_lib_cells {cellB cellC}
```

To assign labels named Y and Z to the right side of the cellB library cell, enter the following command:

```
icc_shell> set_lib_cell_spacing_label -names {Y Z} \  
-right_lib_cells {cellB}
```

2. Define the spacing requirements between the labels by using the `set_spacing_label_rule` command.

Use the `-labels` option to specify which labels are being constrained. You must specify exactly two labels, which can be the same or different labels. Specify the range of invalid spacings, in terms of the unit tile, as the command argument.

For example, to specify that labels X and Y cannot abut (there must be at least one unit tile between them), enter the following command:

```
icc_shell> set_spacing_label_rule -labels {X Y} {0 0}
```

To specify that two X labels cannot have a spacing of two unit tiles, enter the following command:

```
icc_shell> set_spacing_label_rule -labels {X X} {2 2}
```

To specify that labels X and Z must have a spacing of less than two unit tiles or more than four unit tiles, enter the following command:

```
icc_shell> set_spacing_label_rule -labels {X Z} {2 4}
```

To define the boundary spacing rules,

1. IC Compiler assigns a built-in label named SNPS_BOUNDARY to chip boundaries that include chip core boundary (two ends of a row), fixed hard macro boundary, and hard placement blockage boundary. You cannot use the `set_lib_cell_spacing_label` command to assign the built-in label to library cells.
2. Define the spacing requirements between the built-in label and other labels by using the `set_spacing_label_rule` command.

For example, to specify that the label X referenced cells should not be placed within 0 to 1 unit tile from the chip boundaries, enter the following commands:

```
icc_shell> set_lib_cell_spacing_label -names {X} \
           -right_lib_cells {AND*} -left_lib_cells {AND*}
icc_shell> set_spacing_label_rule -labels {X SNPS_BOUNDARY} {0 1}
```

To report both the intercell and boundary spacing rules, use the `report_spacing_rules` command with the `-all` option. To report the intercell spacing rules for a specific collection of library cells, use the `report_spacing_rules` command with the `-of_lib_cell` option.

To remove all spacing rules, use the `remove_all_spacing_rules` command.

Note:

If the power or ground net is defined as a complete blockage, the legalizer and the `check_legality` command ignore the spacing rule violations between library cells and the power or ground net. If the power or ground net is defined as a partial blockage, the legalizer and the `check_legality` command check for spacing rule violations between library cells and the power or ground net.

Buffer Strategy for Optimization

During the optimization step, the `place_opt` command introduces buffers and inverters to fix timing and DRC violations. However, this buffering strategy is local to some critical paths. The buffers and inverters that are inserted become excess later because critical paths change during the course of optimization. You can reduce the excess buffer and inverter counts after `place_opt` by using the `set_buffer_opt_strategy` command, as shown in the following example:

```
icc_shell> set_buffer_opt_strategy -effort low
```

This buffering strategy will not degrade the quality of results (QoR) but will incur a slight increase in runtime. A higher level of effort will result in better reduction in buffer and inverter counts. The default is medium.

Tie Cell Insertion

A tie cell is a special-purpose standard cell whose output is constant high or constant low and is used to hold the input of another cell at the given constant value. To prepare the `place_opt` or `psynopt` commands for automatic insertion and optimization of tie-offs required in the design, execute commands similar to the following:

```
set_auto_disable_drc_nets -constant false
set_physopt_new_fix_constants true
set_attribute [...] max_fanout 12
set_attribute [...] max_capacitance 0.2 -type float
```

Optimization means using a single tie cell to hold as many inputs as possible at a given logic level, while meeting specified maximum fanout and maximum capacitance constraints. The `set_auto_disable_drc_nets` command enables DRC on constant nets. Setting the `physopt_new_fix_constants` variable to true causes IC Compiler to observe the maximum capacitance constraint during tie-off optimization. The maximum capacitance constraint is determined by the `max_capacitance` attribute, which can be set with the `set_max_capacitance` or `set_attribute` command. The `set_attribute` command can be used to specify explicitly both the maximum fanout and maximum capacitance constraints for objects in the design.

You can also insert tie cells manually with the `connect_tie_cells` command. The command inserts tie cells and connects them to specified cell ports, while meeting the maximum fanout and maximum wire length constraints specified in the command. For details, see the man page.

Magnet Placement

To improve congestion for a complex floorplan or to improve timing for the design, you can use magnet placement to specify fixed objects as magnets and have IC Compiler move their connected standard cells close to them. You can specify fixed macro cells, a pin of a fixed macro cell, or an I/O pin as the magnet object.

For the best results, perform magnet placement before the standard cells are placed.

To perform magnet placement, use the `magnet_placement` command with a specification of the magnets and options for any special functions you need to perform.

```
icc_shell> magnet_placement [options] magnet_objects
```

To do this	Use this option
Enable the movement of fixed (or soft fixed) cells and display a warning that lists the fixed cells to be moved	-move_fixed -move_soft_fixed
Fix (or soft fix) cells in their locations after magnet placement	-mark_fixed -mark_soft_fixed
Specify the number of logic levels from the magnet that should be checked for magnet placement	-logical_level <i>number</i>
Prevent movement of buffers and inverters	-exclude_buffers
Prevent cells from being placed over soft blockages	-avoid_soft_blockages
Prevent placement beyond sequential cells	-stop_by_sequential_cells
Prevent placement beyond the specified endpoints	-endpoints

To return a collection of cells that can be moved with magnet placement, use the `get_magnet_cells` command with the options you need.

```
icc_shell> get_magnet_cells [options] magnet_list
```

To do this	Use this option
Get the fixed (or soft fixed) cells that can be moved	-move_fixed -move_soft_fixed
Get the cells in the specified number of logic levels from the magnet	-logical_level <i>number</i>
Get the cells encountered between magnet cells and sequential cells	-stop_by_sequential_cells
Get the cells encountered between the magnet and the specified endpoints	-endpoints
Exclude the buffers and inverters	-exclude_buffers

Placement and Optimization Attributes

Table 4-1 lists the restrictions imposed on the placement and optimization of cells by attributes.

Table 4-1 Restrictions Imposed by Placement and Optimization Attributes

Attribute	Coarse placement	Detail placement	Optimization
<code>is_fixed</code>	Cannot move cells	Cannot move cells	Cannot move, rotate, or resize cells
<code>cts_fixed</code> (imposed on clock buffers by clock tree synthesis)	Cannot move cells	Cannot move cells	Cannot move, rotate, or resize cells
<code>is_soft_fixed</code>	Cannot move cells	No restrictions	No restrictions
<code>size_only</code>	No restrictions	No restrictions	Can only resize cells
<code>in_place_size_only</code>	Cannot move cells	No restrictions	Can resize cells only if there is room
<code>cts_in_place_size_only</code> (imposed on clock sinks by clock tree synthesis)	Cannot move cells	No restrictions	Can resize cells only if there is room
<code>dont_touch</code>	No restrictions	No restrictions	Cannot move, rotate, or resize cells

Preparing for High-Fanout Net Synthesis

During placement and optimization, IC Compiler does not buffer clock nets as defined by the `create_clock` command, but it does, by default, buffer other high-fanout nets, such as resets or scan enables, using a built-in high-fanout synthesis engine.

Before running high-fanout net synthesis during `place_opt`, define the buffering options by using the `set_ahfs_options` command. You can define the fanout threshold with the `-hf_threshold` option and the library reference cells to use for buffering with the `-default_reference` option. If you set the `-remove_effort` option to `medium` or `high`, the high-fanout synthesis engine automatically removes existing buffer trees that do not

meet the high-fanout constraints and replaces them with new buffer trees. For information on these and other high-fanout synthesis options, see the man page for the `set_ahfs_options` command.

The high-fanout synthesis engine does not buffer nets that are set as ideal nets or nets that are disabled with respect to design rule constraints.

As an alternative to performing high-fanout net synthesis during `place_opt`, you can use the `create_buffer_tree` command to run standalone high-fanout net synthesis.

To get information about the buffer trees in your design, use the `report_buffer_tree` command.

To remove buffer trees from your design, use the `remove_buffer_tree` command.

Performing Clock Tree Synthesis During Placement

If your design contains simple clock tree structures and uses the same design rule constraints for placement and clock tree synthesis, you can simplify the design flow by performing clock tree synthesis and optimization during placement.

To perform clock tree synthesis and optimization during placement,

1. Define the clock trees as described in [Chapter 7, “Clock Tree Synthesis”](#).
2. Identify the clock tree synthesis options by using the `set_place_opt_cts_strategy` command.

The following table shows the options that you can specify.

To do this	Use
Specify the operating condition to use for clock tree synthesis and optimization. If you do not specify this option, the maximum operating condition is used.	<code>-operating_condition min max min_max</code>
Prevent routing of the clock tree.	<code>-no_clock_route</code>
Perform interclock delay balancing.	<code>-inter_clock_balance</code>

To do this	Use
Fix hold time violations on the clock tree.	<code>-fix_hold</code>
Update the clock latency values.	<code>-update_clock_latency</code>

3. Use the `-cts` option when you run the `place_opt` command.

Performing Placement and Optimization

After you have finished design planning and power planning, you can perform placement and optimization on your design.

To perform placement and optimization, use the `place_opt` command or choose Placement > Core Placement and Optimization in the GUI.

The `place_opt` command performs coarse placement, high-fanout net synthesis, physical optimization, and legalization.

During area recovery phase, removing cells on noncritical timing paths increases the fanout. The paths sometimes become timing-critical during post-route optimization to correct signal integrity problems. To avoid removing cells from potential timing-critical paths, you can specify a maximum fanout threshold limit for optimization by setting the `psynopt_high_fanout_legality_limit` variable. By default, this variable has a value of 0 and there is no high-fanout limit. If you set this variable to a nonzero value, optimizations that result in a fanout greater than the specified value are not performed.

[Table 4-2](#) describes the available `place_opt` options.

Table 4-2 The place_opt Options

GUI object	Command option	Description
Effort radio button	-effort low medium high	Improve the quality of results or reduce runtime. Higher effort levels use additional runtime in an attempt to improve the quality of results. The default is medium.
"Recover area" check box	-area_recovery	Recover area for cells not on timing-critical paths.
"Power optimization" check box	-power	Optimize leakage power and dynamic power and perform low-power placement. The power optimizations and low-power placement capability must be previously enabled. For more information, see Chapter 6, "Power Optimization."
"Clock compilation, optimization and routing" check box	-cts	Run clock tree synthesis and optimization with placement and optimization. Typically, this is used for the quick flow for designs not requiring extensive clock tree synthesis and optimization work. For more information about clock tree synthesis and optimization, see Chapter 7, "Clock Tree Synthesis."
"Number of CPUs" field	-num_cpus number	Use additional CPUs. By default, IC Compiler uses one CPU.
"Reorder scan during placement" check box	-optimize_dft	Reorder scan chains. For more information, see Chapter 5, "Design for Test."
"Minimize congestion" check box	-congestion	Reduce congestion for improved routability. For best results, use this option only for congested designs.

Using Physical Optimization

You can run incremental placement-based optimization that supports area recovery, design rule fixing, sizing, and route-based optimization by using the `psynopt` command. By default, this command performs timing optimization and design rule fixing based on the maximum capacitance and maximum transition settings, but it can also perform power optimizations if you specify power constraints. The `psynopt` command continues to optimize until no more optimizations can be performed. When done, it completes the optimizations with a legalized placement of the design.

This process can remove dangling cells (cells that do not drive other cells), resulting in a reduced number of cells in the design. If you need to prevent the removal of such cells, do one of the following:

- To prevent the removal of specific cells, use the `set_dont_touch` command to apply a `dont_touch` attribute on each of those cells.
- To retain the unloaded cells globally, set the `physopt_delete_unloaded_cells` variable to false.

To perform placement optimization with the `psynopt` command, enter

```
icc_shell> psynopt [options]
```

Table 4-3 shows the options supported by the `psynopt` command.

Table 4-3 The `psynopt` Options

To do this	Use this option
Enable area recovery within the cluster boundary for noncritical paths. Using this switch can have a severe runtime impact if the design contains many high-fanout nets.	<code>-area_recovery</code>
Open only area recovery (you cannot use this option with <code>-only_design_rule</code> or <code>-only_hold_time</code>).	
Note: To restrict area recovery to high-density areas, set the <code>physopt_density_area_recovery</code> variable to true.	<code>-only_area_recovery</code>
Exit <code>psynopt</code> without fixing design rule violations, allowing you to check the results in a constraint report before fixing the violations.	<code>-no_design_rule</code>
You cannot use <code>-no_design_rule</code> with <code>-only_design_rule</code> .	

Table 4-3 *The psynopt Options (Continued)*

To do this	Use this option
Perform design rule fixing without performing timing optimizations. This option uses <code>psynopt</code> placement to target DRC problems. You cannot use <code>-only_design_rule</code> with <code>-no_design_rule</code> or <code>-only_area_recovery</code> .	<code>-only_design_rule</code>
Fix only hold time violations; ignore other design rules. The <code>set_fix_hold</code> command must be specified for hold time fixing to be performed. You cannot use this option with <code>-no_design_rule</code> or <code>-only_area_recovery</code> .	<code>-only_hold_time</code>
Restrict post-route flow optimization to gate sizing only. You cannot use this option with <code>-in_place_size_only</code> .	<code>-size_only</code>
Restrict post-route flow optimization to in-place gate sizing only. You cannot use this option with <code>-size_only</code> .	<code>-in_place_size_only</code>
Restrict cell sizing to cells having the same footprint and size.	<code>-preserve_footprint</code>
Run the on-route optimization flow. You can use this option with <code>-in_place_size_only</code> . You cannot use this option with <code>-size_only</code> .	<code>-on_route</code>
Reduce congestion to improve routability.	<code>-congestion</code>
Open back-annotation-based optimization.	<code>-use_annotation</code>
Open power optimization. This activates the power optimizations enabled by the <code>set_power_options</code> command. By default, only leakage power optimization is enabled, but you can also enable dynamic power optimization for use with the <code>psynopt</code> command.	<code>-power</code>
Open only power optimizations.	<code>-only_power</code>
For more information, see Chapter 6, "Power Optimization."	

Preroute RC Estimation

IC Compiler automatically performs preroute RC estimation when you run the following commands: `place_opt`, `clock_opt`, `create_placement`, `legalize_placement`, and `psynopt`. In addition, you can explicitly perform preroute RC estimation by running the `extract_rc` command.

Note:

By default, the `extract_rc` command performs RC estimation on any unrouted nets in your design and performs RC extraction on the routed nets in your design. If your design contains a mix of routed and unrouted nets, and you do not want to run RC extraction on the routed nets, use the `extract_rc -estimate` command, which performs only RC estimation.

When performing preroute RC estimation, IC Compiler performs the following tasks:

- Determines the RC coefficients

IC Compiler derives the RC coefficients from the TLUPlus models. (For information about attaching the TLUPlus data to your Milkyway design library, see [“Setting Up the TLUPlus Files” on page 3-26.](#))

The TLUPlus models contain layer-specific RC coefficients. Because layers are not assigned for preroute estimation, IC Compiler calculates the RC coefficients by applying averaging techniques to the layer-specific RC coefficients.

For information about how you can modify the RC coefficients to improve your preroute optimization results, see [“Modifying the RC Coefficients” on page 4-21.](#)

- Estimates the routing topology

IC Compiler uses virtual route technology to estimate the routing topology from the placement information. The virtual route does not contain layer assignments.

- Calculates the RC values

IC Compiler uses virtual routing to estimate the preroute parasitics.

- Performs delay estimation

IC Compiler uses the Elmore delay model to estimate the delay values, based on the estimated routing topology and the estimated RC values.

- Saves the timing data and attributes in the Milkyway design library

IC Compiler saves total capacitance as a CEL attribute. To save the parasitics in a data file, run the `write_parasitics` command.

Modifying the RC Coefficients

When IC Compiler runs preroute RC estimation, it displays the derived coefficients on your screen. You can modify these coefficients by using the following methods:

- Defining net-based layer constraints

At small geometries, the RC coefficients, especially resistance coefficients, can vary significantly between layers. In this case, the coefficients derived by averaging the coefficients for all layers might not provide sufficient correlation with the post-route values. You can restrict the set of routing layers used for a specific net by defining net-based layer constraints.

- Introducing pessimism

If, after analyzing the estimation results calculated from the derived RC coefficients, you feel that the derived RC coefficients are too optimistic, you can consider introducing pessimism to these values. Using pessimistic RC values means that you provide larger values, so that for a given net, IC Compiler computes a larger delay. This, in turn, drives the optimization of your design such that extra buffering or more powerful driving cells are introduced.

Defining Net-Based Layer Constraints

In addition to restricting the layers to which a specific net can be assigned during routing, net-based layer constraints improve RC correlation by reducing the set of values used by the averaging techniques that determine the RC coefficients used for preroute estimation.

To set net-based layer constraints, use the `set_net_routing_layer_constraints` command.

```
icc_shell> set_net_routing_layer_constraints {netA} \  
           -min_layer_name metal1 -max_layer_name metal3
```

The routing layer names you specify must exist in both the physical library and the design, and the maximum routing layer must be the same or on top of the minimum routing layer.

If the net you specify already has routing layer constraints set, a warning appears and the constraints specified by this invocation overwrite the existing routing layer constraints.

To report and remove routing layer constraints, use the `report_net_routing_layer_constraints` and `remove_net_routing_layer_constraints` commands.

Introducing Pessimism

You can introduce pessimism by either scaling the derived coefficients or manually specifying the coefficients. Avoid more than doubling the default numbers, because the gate sizes and runtime increase. The basic rule is to increase values by 10 percent (multiply by 1.1). Usually, change the number by small amounts, such as increments of 0.1.

Scaling the RC Coefficients. To scale the RC coefficients,

1. Determine the scaling factor you want to apply.
2. Use the following `set_delay_estimation_options` command options to set the scaling factors:

```
-max_unit_horizontal_capacitance_scaling_factor  
-max_unit_vertical_capacitance_scaling_factor  
-max_unit_horizontal_resistance_scaling_factor  
-max_unit_vertical_resistance_scaling_factor  
-min_unit_horizontal_capacitance_scaling_factor  
-min_unit_vertical_capacitance_scaling_factor  
-min_unit_horizontal_resistance_scaling_factor  
-min_unit_vertical_resistance_scaling_factor
```

IC Compiler produces new RC coefficients by multiplying the derived RC coefficients by the scaling factors you specify. During RC estimation, IC Compiler uses the scaled coefficients.

Manually Specifying RC Coefficients. To specify the RC coefficients,

1. Run RC estimation to derive the base RC coefficients.
2. Determine the new, pessimistic values you want to apply.
3. Use the following `set_delay_estimation_options` command options to specify the RC coefficients:

```
-max_unit_horizontal_capacitance  
-max_unit_vertical_capacitance  
-max_unit_horizontal_resistance  
-max_unit_vertical_resistance  
-min_unit_horizontal_capacitance  
-min_unit_vertical_capacitance  
-min_unit_horizontal_resistance  
-min_unit_vertical_resistance
```

Note:

At the minimum, you must specify all four `-max_*` options; otherwise, IC Compiler uses the derived RC coefficients.

The values you specify override the derived RC coefficients. During RC estimation, IC Compiler uses the specified coefficients.

If you specify both scaling factors and user-defined RC coefficients, the resulting RC coefficients are the result of multiplying your RC coefficients by your scaling factors.

Enabling Via Resistance Estimation

By default, preroute RC estimation does not perform via resistance calculations. To enable via resistance calculations, set the `physopt_enable_via_res_support` variable to true. When this capability is enabled, IC Compiler determines the via resistance values from the TLUPlus models. As with the net RC coefficients, you can introduce pessimism by either scaling the derived values or manually specifying the values.

Scaling the Via Resistance Values

To scale the via resistance values,

1. Determine the scaling factor you want to apply.
2. Use the following `set_delay_estimation_options` command options to set the scaling factors:

```
-max_via_resistance_scaling_factor  
-min_via_resistance_scaling_factor
```

IC Compiler produces new resistance values by multiplying the derived values by the scaling factors you specify. During delay estimation, IC Compiler uses the scaled values.

Manually Specifying Via Resistance Values

To specify the via resistance values,

1. Run RC estimation to derive the base resistance values.
2. Determine the new, pessimistic values you want to apply.
3. Use the following `set_delay_estimation_options` command options to specify the resistance values:

```
-max_via_resistance  
-min_via_resistance
```

The values you specify override the derived via resistance values. During delay estimation, IC Compiler uses the specified values.

Note:

If you specify both scaling factors and user-defined via resistance values, the resulting via resistance values are the result of multiplying your values by your scaling factors.

Analyzing Placement

After placement, you can view and analyze the results. You can report the area utilization with the `report_placement_utilization` command, the design timing with the `report_timing` command, and the power consumption characteristics with the `report_power` command.

Placement Area Utilization

Placement area utilization, or simply “utilization,” means the percentage of area available for placement that is already occupied by placed cells. For example, a utilization of 80 percent means that 80 percent of the available area is occupied by cells and 20 percent is empty and can still be used for additional cell placement, for movement of cells for legalization and optimization, or as an allowance to prevent excessive routing congestion.

You can report the utilization for part or all of the design with the `report_placement_utilization` command:

```
report_placement_utilization
  [-non_fixed_only]
  [-grid_size float]
  [-verbose]
  [-coordinates {X1 Y1 X2 Y2}]
```

For example, to report the utilization of the rectangular area bounded by the corner points (340, 380) and (400, 450), you would use the following command:

```
icc_shell> report_placement_utilization -coordinates {340 380 400 450}
...
Std cell utilization:      84.02% (2193/(2610-0),
                           (std_cell_area/(region_area-blockage_area))
(Non-fixed + Fixed)
Std cell utilization:      77.66% (1450/(2610-743))
(Non-fixed only)
Total region area:        2610      sites, bbox
                           (340.00 380.00 400.00 450.00) um
Std cell area :           2193      sites, (non-fixed:1450, fixed:743)
Macro cell area :         0         sites
Blkgs exclude fixed cells: 0         sites
Blkgs include fixed cells: 743      sites
Std cell count :          301              (non-fixed:183, fixed:118)
Macro cell count :         0
Pnet area :               0         sites

*****
Sub-Region Utilization
*****
```

```

Number of regions with placement utilization 0 - 0.125 is 0 (0.00%)
Number of regions with placement utilization 0.125 - 0.25 is 0 (0.00%)
Number of regions with placement utilization 0.25 - 0.375 is 0 (0.00%)
Number of regions with placement utilization 0.375 - 0.5 is 0 (0.00%)
Number of regions with placement utilization 0.5 - 0.625 is 0 (0.00%)
Number of regions with placement utilization 0.625 - 0.75 is 0 (0.00%)
Number of regions with placement utilization 0.75 - 0.875 is 0 (0.00%)
Number of regions with placement utilization 0.875 - 1 is 16 (100.00%)
1

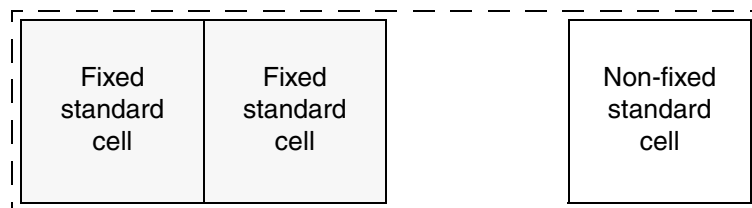
```

If you do not specify the coordinates, the command reports the utilization of the whole chip.

In the reporting of utilization, there are two basic ways to consider the presence of standard cells that have fixed placement (“fixed standard cells”). These cells can be considered the same as unfixed cells, in other words, as cells occupying the available space. On the other hand, the space occupied by fixed cells can be considered unavailable space, in what is called a “non-fixed-only” utilization report.

Consider the example shown in [Figure 4-3](#). This area of four square units is occupied by three cells, each with an area of one square unit, leaving one square unit of empty space.

Figure 4-3 Non-Fixed-Only Utilization Example



Default utilization (fixed and unfixed standard cells): $3/4 = 75$ percent
 Non-fixed-only standard cell utilization: $1/2 = 50$ percent

The default utilization is reported to be 75 percent because $3/4$ of the available area is occupied by standard cells. However, for non-fixed-only utilization reporting, the area occupied by fixed standard cells is considered unavailable, so the available area is two square units, occupied by one square unit of non-fixed standard cells, for a utilization of $1/2$ or 50 percent.

Utilization reporting also considers the presence of placement blockages. For both default and non-fixed-only reporting, any part of the area occupied by a blockage is considered unavailable for placement. Thus, the default utilization is calculated as follows:

$$(\text{non-fixed\_standard\_cell\_area} + \text{fixed\_standard\_cell\_area}) / (\text{total\_area} - \text{blocked\_area})$$

whereas non-fixed-only utilization is calculated as follows:

$$(\text{non-fixed\_standard\_cell\_area}) / (\text{total\_area} - \text{fixed\_standard\_cell\_area} - \text{blocked\_area})$$

The `report_placement_utilization` command reports both the default and non-fixed-only utilization values. It also divides the specified region into subregions and separately reports the utilization within those subregions, to give an idea of the “evenness” of the utilization. By default, the subregion size is five times the placement site height. You can specify a different subregion size, in microns, by using the `-grid_size` option.

The `-verbose` option causes the reporting of the dimensions, locations, and individual utilization values of the subregions. Otherwise, the report shows only a summary listing of the number and percentage of subregions having different ranges of utilization values. The `-non_fixed_only` option specifies whether to use default or non-fixed-only utilization values for the subregion utilization report.

Timing Analysis

After you set the timing constraints such as clocks, input delays, and output delays, it is a good idea to use the `check_timing` command to check for input/output timing setup problems and timing conditions such as incorrectly specified generated clocks and combinational feedback loops. The command checks the timing attributes of the current design and issues warning messages about any unusual conditions found.

This is the full syntax of the `check_timing` command:

```
check_timing
  [-overlap_tolerance minimum_distance]
  [-override_defaults check_list]
  [-include check_list]
  [-exclude check_list]
  [-multiple_clock]
  [-retain]
```

The `-include` and `-exclude` options are new for this release. You can use these options to explicitly specify the types of timing checks to include or exclude during execution of the command. By default, the types of timing checks performed are defined in the `timing_check_defaults` variable, which is also new for the current release.

These are the types of timing checks that can be performed:

```
clock_crossing
data_check_multiple_clock
data_check_no_clock
generated_clocks
generic
loops
```

```

multiple_clock
no_input_delay
retain
unconstrained_endpoints
pulse_clock_cell_type
gated_clock
ideal_timing
clock_no_period

```

The types of timing checks shown in boldface are performed by default. You can include or exclude specific types of timing checks by using the `check_timing` command options or by setting the `timing_check_defaults` variable. For details, see the man pages for the `check_timing` command and `timing_check_defaults` variable.

After placement, you can use the `report_timing` command to report the worst-case timing paths in the design. This is the command syntax:

```

report_timing
  [-to to_list]
  [-from from_list]
  [-through through_list]
  [-path short | full | full_clock | full_clock_expanded | only | end]
  [-delay min | min_rise | min_fall | max | max_rise | max_fall]
  [-nworst paths_per_endpoint]
  [-max_paths max_path_count]
  [-input_pins]
  [-nets]
  [-transition_time]
  [-crosstalk_delta]
  [-capacitance]
  [-attributes]
  [-physical]
  [-slack_greater_than greater_slack_limit]
  [-slack_lesser_than lesser_slack_limit]
  [-lesser_path max_path_delay]
  [-greater_path min_path_delay]
  [-loops]
  [-true [-threshold path_delay]]
  [-justify]
  [-enable_preset_clear_arcs]
  [-significant_digits digits]
  [-nosplit]
  [-sort_by group | slack]
  [-group group_name]
  [-trace_latch_borrow]
  [-derate]
  [-scenario scenario_list]
  [-temperature]
  [-voltage]

```

By default, the command reports the path having the worst maximum (setup) delay in each clock group. Net delays are based on estimated route lengths.

For example, here is a typical `report_timing` command:

```
icc_shell> report_timing
...
Startpoint: I_SDRAM_TOP/I_SDRAM_IF/DQ_out_1_reg_8_
            (falling edge-triggered flip-flop clocked by SDRAM_CLK)
Endpoint: sd_DQ_out[8]
          (output port clocked by SD_DDR_CLK)
Path Group: SD_DDR_CLK
Path Type: max
```

Point	Incr	Path

clock SDRAM_CLK (fall edge)	3.75	3.75
clock network delay (ideal)	0.00	3.75
I_SDRAM_TOP/I_SDRAM_IF/DQ_out_1_reg_8_/CPN (sdcfq1)	0.00	3.75 f
I_SDRAM_TOP/I_SDRAM_IF/DQ_out_1_reg_8_/Q (sdcfq1)	0.36	4.11 r
I_SDRAM_TOP/I_SDRAM_IF/sd_mux_dq_out_8/Z (mx02d4)	0.21 *	4.31 r
I_SDRAM_TOP/I_SDRAM_IF/sd_DQ_out[8] (SDRAM_IF)	0.00	4.31 r
I_SDRAM_TOP/sd_DQ_out[8] (SDRAM_TOP)	0.00	4.31 r
sd_DQ_out[8] (out)	0.00	4.31 r
data arrival time		4.31
clock SD_DDR_CLK (rise edge)	7.50	7.50
clock network delay (ideal)	1.00	8.50
clock uncertainty	-0.05	8.45
output external delay	-2.00	6.45
data required time		6.45

data required time		6.45
data arrival time		-4.31

slack (MET)		2.14
...		

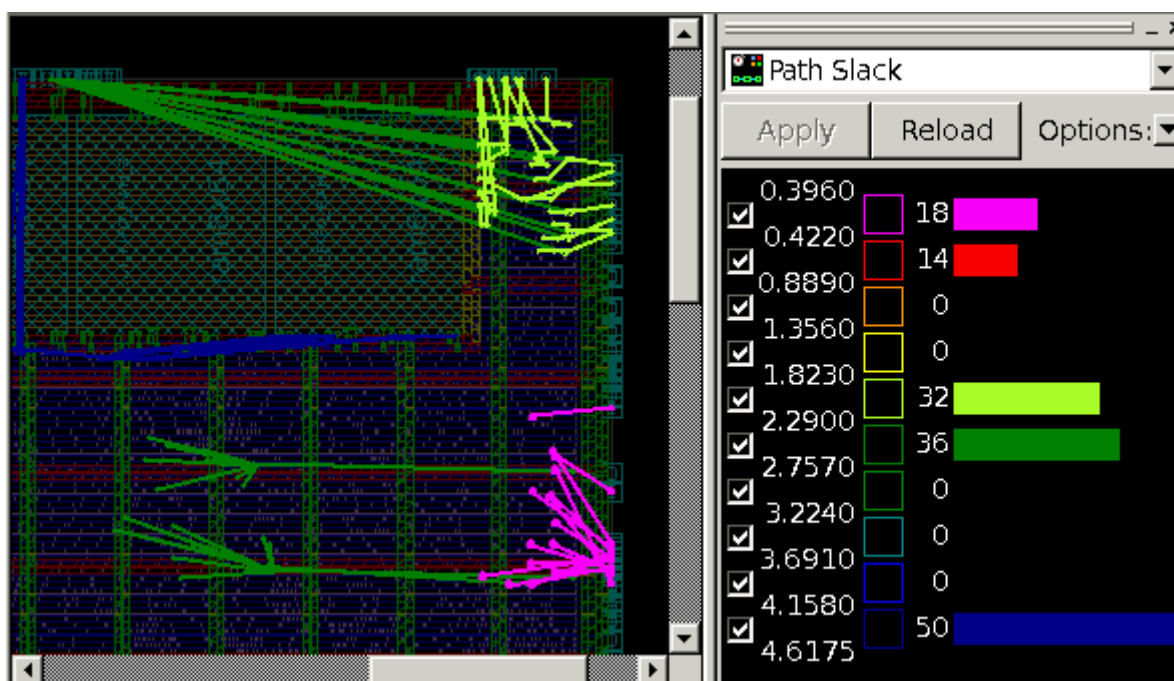
The default report shows the startpoint, endpoint, path group (clock domain), path type (minimum delay, maximum delay, max_rise, min_fall, and so on), the incremental and cumulative time delay values along the data and clock paths, the data required time at the path endpoint, and the timing slack for the path.

The `report_timing` command options let you specify the scope of paths reported (from/to/through specified points in the design), the path types reported, the numbers of worst-case paths reported, and the types of information reported for intermediate points in the path (transition times, capacitance, net delays, and so on).

For more information, see the man page for the `report_timing` command.

Using the GUI, you can generate color-coded diagrams showing the locations of worst-case timing conditions such as path slack, cell slack, net capacitance, clock latency/transition, and crosstalk. From the Visual Mode panel (View > Visual Mode) select the desired type of visual analysis, and then click the Reload button and OK button. An example of a path slack display is shown in [Figure 4-4](#). The layout view shows the worst-case timing paths with the pins and flylines of the paths color-coded according to amount of slack. For more information on generating these views, see the topic “Visual Mode panel” in the IC Compiler online help, or refer to [Appendix A, “Using GUI Tools.”](#)

Figure 4-4 Path Slack Visual Mode



To get a detailed report on the delay calculation at a given point along a timing path, use the `report_delay_calculation` command. This is the command syntax:

```
report_delay_calculation
[-min] [-max]
-from from_pin -to to_pin
[-nosplit]
[-crosstalk]
[-from_rise_transition value]
[-from_fall_transition value]
```

Specify the “from” and “to” pins of the cell or net that you want to report. These two pins can be the input and output pins of a cell to report the calculation of a cell delay, or can be the driver pin and a load pin of a net to report the calculation of a net delay. For example, the following command reports in detail the delay calculation between two pins of a cell:

```
icc_shell> report_delay_calculation \
           -from I_SDRAM_TOP/I_SDRAM_IF/DQ_out_1_reg_8_/CPN \
           -to I_SDRAM_TOP/I_SDRAM_IF/DQ_out_1_reg_8_/Q
...

From pin:          I_SDRAM_TOP/I_SDRAM_IF/DQ_out_1_reg_8_/CPN
To pin:            I_SDRAM_TOP/I_SDRAM_IF/DQ_out_1_reg_8_/Q
Main Library Units: 1ns 1pF 1kOhm

Operating Conditions: cb13fs120_ts_max  Library: cb13fs120_ts_max
Library: 'cb13fs120_ts_max'
Library Units: 1ns 1pF 1kOhm
Library Cell: 'sdcfq1'

arc sense:          falling_edge
arc type:            cell
Clock rise transition: 0
Clock fall transition: 0

Rise Delay

cell delay = 0.35504
Table is indexed by
  (X) input_pin_transition = 0
  (Y) output_net_total_cap = 0.0124603
Relevant portion of lookup table:
      (X) 0.0150      (X) 0.2500
(Y) 0.0070      (Z) 0.3380      (Z) 0.3880
(Y) 0.0140      (Z) 0.3640      (Z) 0.4150

Z = A + B*X + C*Y + D*X*Y
A = 0.3089      B = 0.2085
C = 3.7052      D = 0.6079

Z = 0.35504
scaling result for operating conditions
multiplying by 1 gives 0.35504

...

Fall Delay

...
```

For more information, see the `report_delay_calculation` man page.

Power Analysis

The `report_power` command calculates and reports power for a design. The command uses the user-annotated switching activity to calculate the net switching power, cell internal power, and cell leakage power, and it displays the calculated values in a power report. This is the command syntax:

```
report_power
  [-net]
  [-cell]
  [-only cell_or_net_list]
  [-hier]
  [-hier_level level_value]
  [-verbose]
  [-cumulative]
  [-flat]
  [-exclude_boundary_nets]
  [-include_input_nets]
  [-analysis_effort low | medium | high]
  [-nworst number]
  [-sort_mode mode]
  [-histogram [-exclude_leq le_val | -exclude_geq ge_val]]
  [-nosplit]
  [-scenario scenario_list]
```

For example, here is a typical `report_power` command:

```
icc_shell> report_power
...
Library(s) Used:

    cb13fs120_ts_max (File: /remote/techp5/ref/db/sc_max.db)
    ram8x64_max (File: /remote/techp5/ref/db/ram8x64_max.db)
    ram16x128_max (File: /remote/techp5/ref/db/ram16x128_max.db)

Operating Conditions: cb13fs120_ts_max    Library: cb13fs120_ts_max
Wire Load Model Mode: enclosed
Design              Wire Load Model      Library
-----
ORCA_TOP            ForQA                cb13fs120_ts_max
PCI_FIFO_1_DW01_sub_1 ForQA            cb13fs120_ts_max

Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
...
```

```

Cell Internal Power    = 34.8033 mW    (93%)
Net Switching Power   = 2.7620 mW    (7%)
-----
Total Dynamic Power    = 37.5653 mW    (100%)

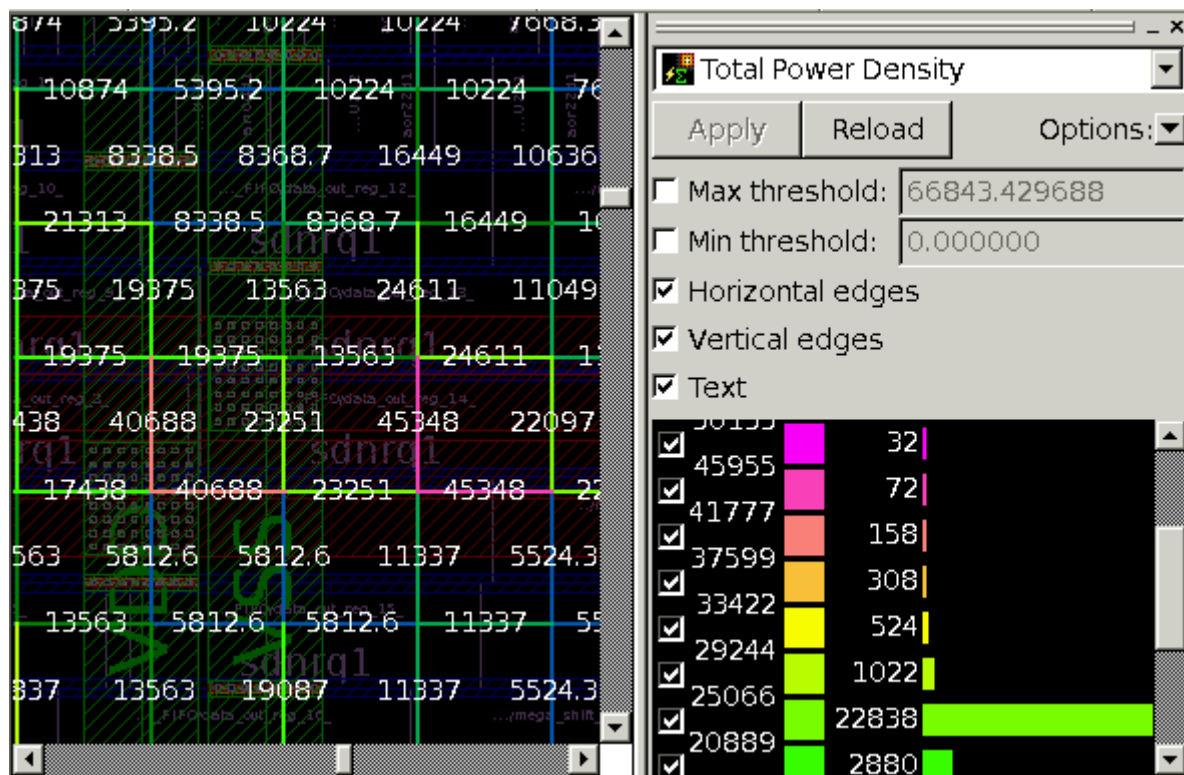
Cell Leakage Power     = 424.5199 uW

```

By default, the `report_power` command shows the total dynamic power and leakage power for the whole chip. The command options let you restrict the scope of the power report to specified instances, to nets or cells alone, or to particular cells or nets obtained by filtering. You can also control the format of the generated report, including the levels of hierarchy reported, the sorting of nets and cells reported, and an optional histogram display of net power. For more information, see the man page for `report_power`.

Using the GUI, you can display color-coded power density maps for the chip. From the Power menu, select the desired type of power density map: leakage, internal, switching, dynamic, or total power density. This opens the Map Mode panel, in which you click the Reload button and then OK to display the power density map. An example of a total power density map is shown in [Figure 4-5](#). For more information, see the topic “power density maps” in the IC Compiler online help, or refer to [Appendix A, “Using GUI Tools.”](#)

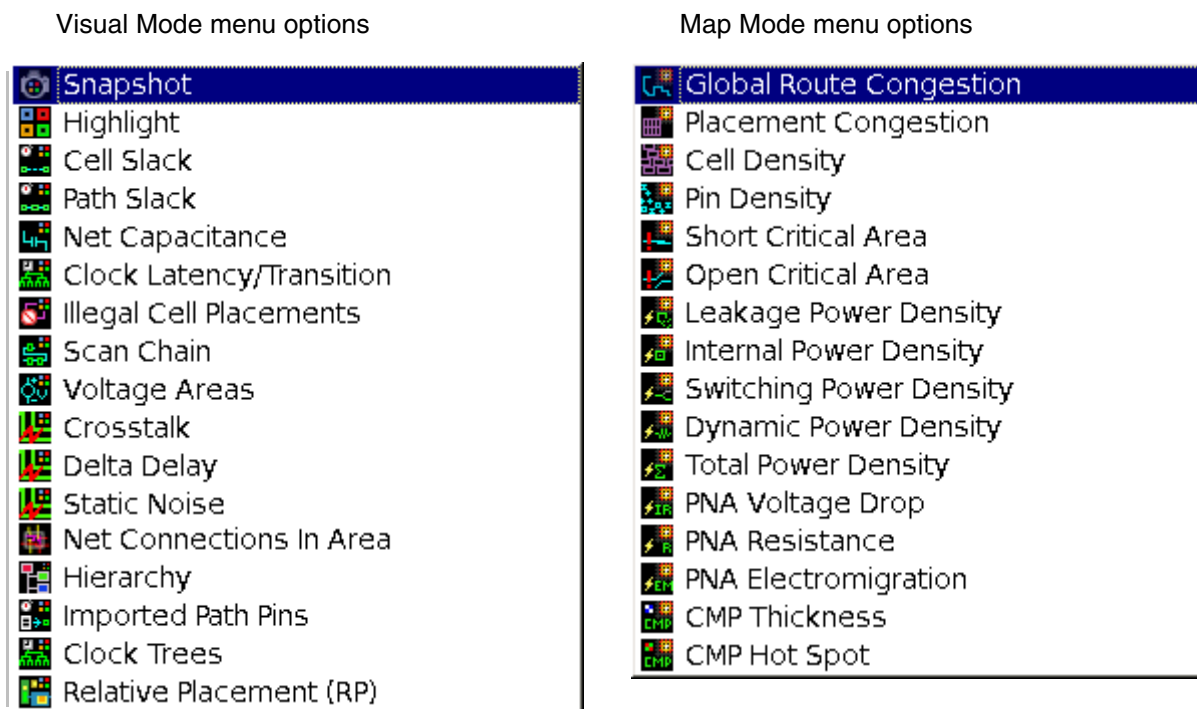
Figure 4-5 Total Power Density Map



Placement Analysis Tools in the GUI

The IC Compiler graphical user interface (GUI) provides a wide range of tools for analyzing and visualizing the quality of results after placement. To generate a layout view showing the desired quality-of-results parameter, open the Visual Mode or Map Mode panel (View > Visual Mode or View > Map Mode), and from the pull-down menu in the panel, select the desired type of report. Figure 4-6 shows the types of visual reports available in the Visual Mode and Map Mode panels.

Figure 4-6 Report Types in the Visual Mode and Map Mode Panels



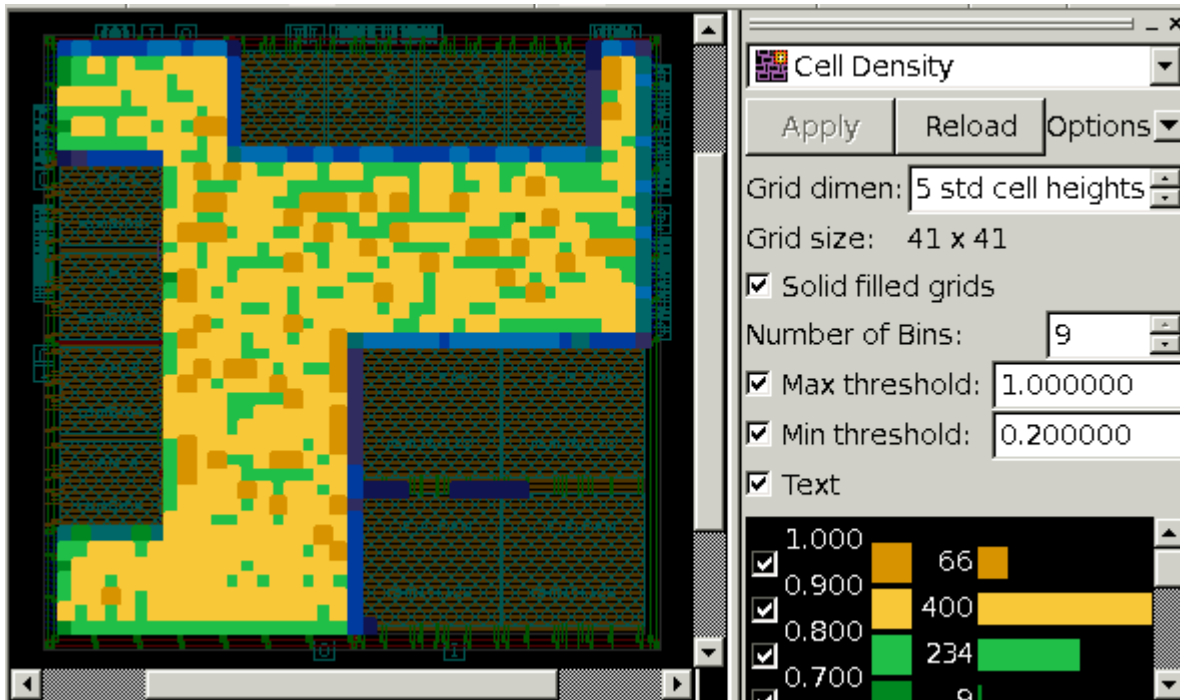
After you select the desired type of visual report, the panel changes to show the options for that type of display. Select the desired options and click the Reload button, which opens a dialog box that typically offers some additional options. In the dialog box, click OK to update the display.

Some of the GUI features commonly used after placement are described in the following sections.

Cell Density Map

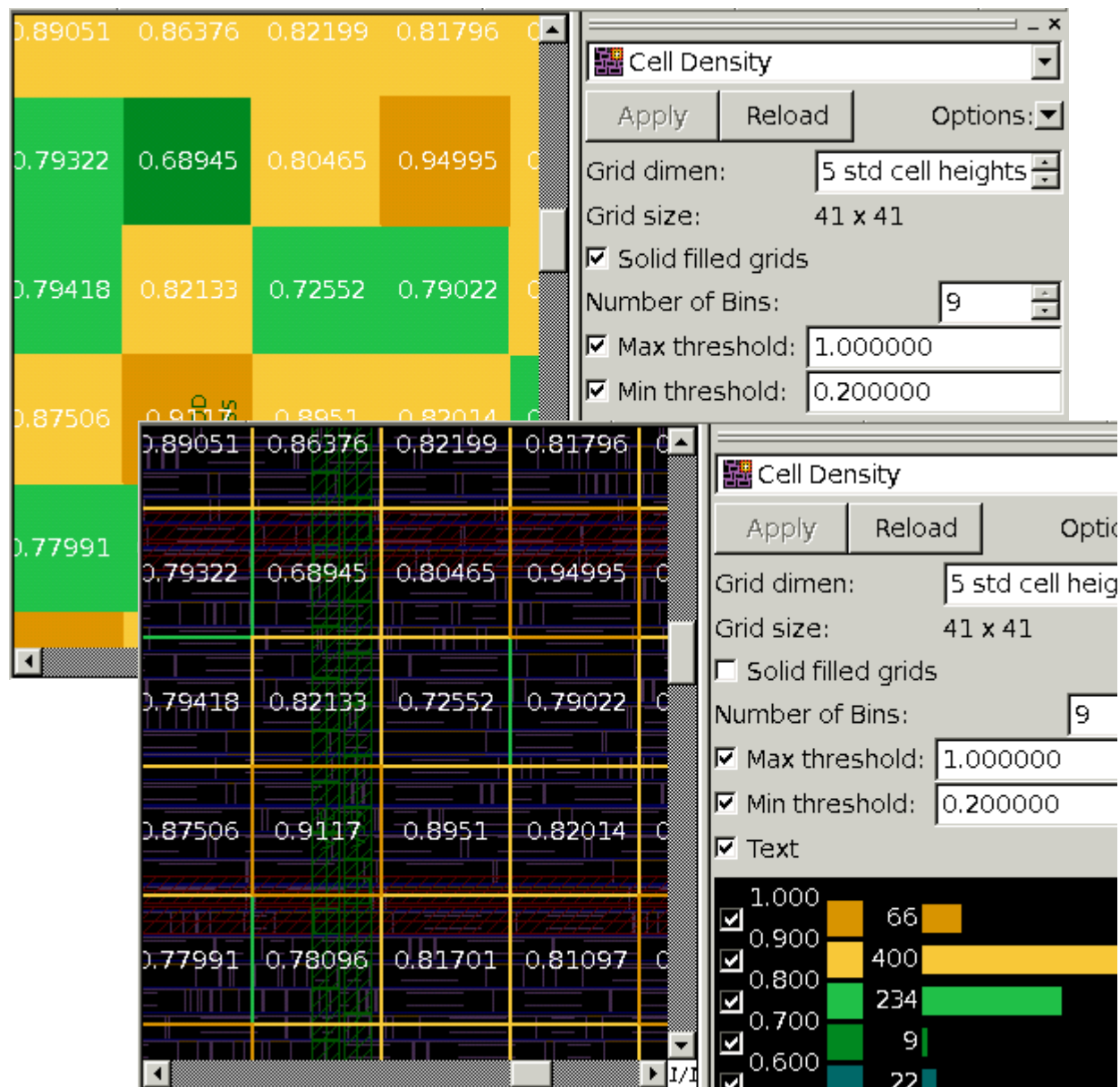
A Cell Density map is a typical GUI report. When you select Cell Density as the Map Mode, the Map Mode panel offers you the option to specify the analysis grid dimensions, solid or unfilled grid units, the number of histogram bins, the bin threshold levels, and the utilization text display. Figure 4-7 shows a typical cell density map.

Figure 4-7 Cell Density Map, Full-Chip View



To generate a chip density map, IC Compiler divides the layout into a grid of squares measuring five standard-cell heights on each side. It finds the area utilization in each grid square and then color-codes each square according to the utilization value. It also displays a histogram showing the number of grid squares in each of the utilization value bins. When the view is zoomed in enough, the utilization values are displayed inside the grid squares, as shown in Figure 4-8. Otherwise, the text values are omitted from the display. Figure 4-8 shows both a solid-filled grid display and an unfilled grid display.

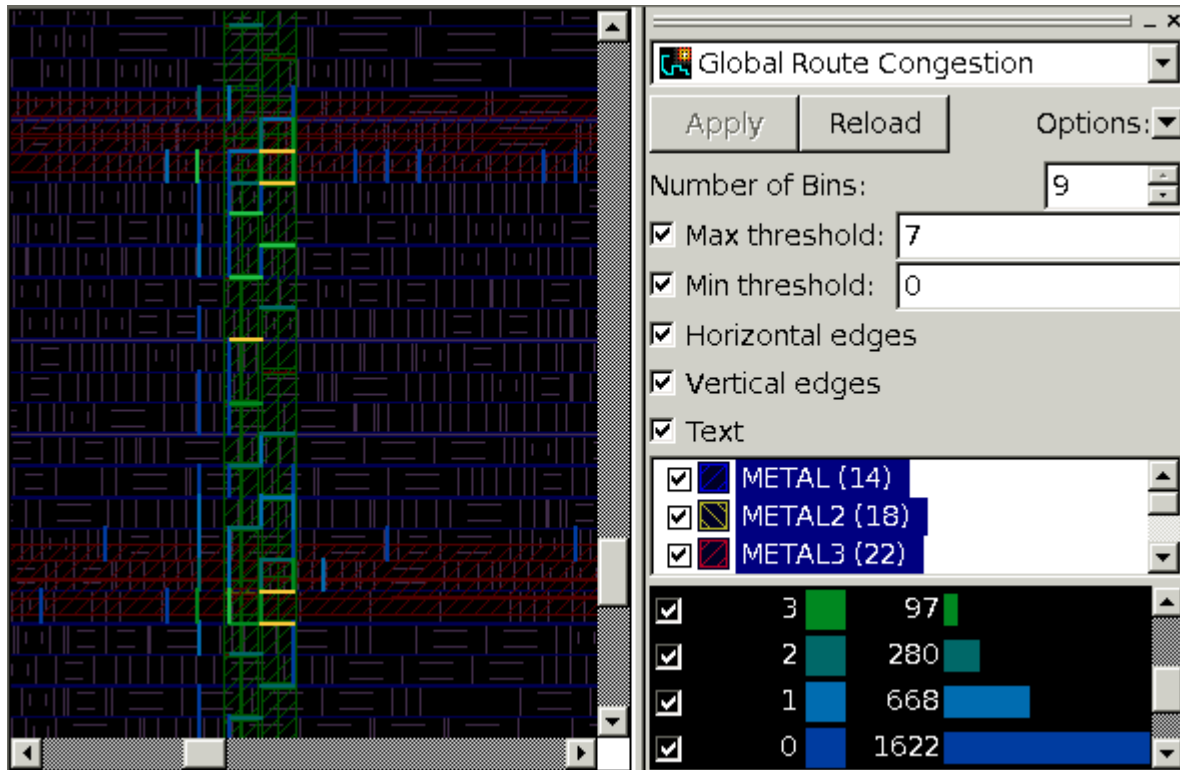
Figure 4-8 Cell Density Maps With and Without Solid Filled Grids



Congestion Map

A congestion map can help you visualize the quality of placement with respect to the avoidance of routing congestion. An example is shown in [Figure 4-9](#).

Figure 4-9 Global Route Congestion Map



To generate a congestion map, IC Compiler divides the chip into square units called global routing cells. It determines the demand for routing tracks, both horizontally and vertically, through each global routing cell, based on the locations of the cell pins and their respective unrouted connections. If the number of routes exceeds the number of available tracks, it is called an overflow. The overflow value is the number of tracks needed minus the number of available tracks in a given direction through a global routing cell. Thus, the higher the number, the worse the overflow. A negative overflow value indicates that all required routes can be accommodated.

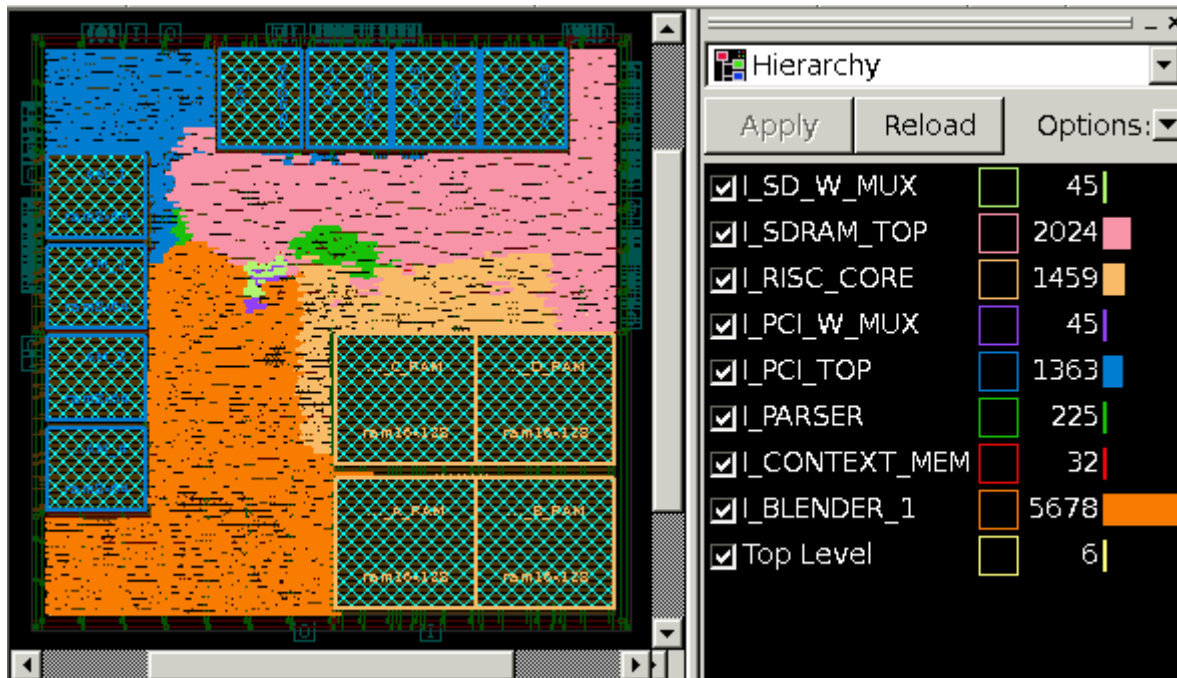
A congestion map shows the borders between global routing cells highlighted with different colors representing different levels of overflow. In addition to the map, a histogram shows the number of global routing cells in different ranges of overflow values.

To generate the congestion map, IC Compiler invokes the `route_global -congestion_map_only` command. For more information on global routing, see [Chapter 8, “Routing.”](#)

Hierarchy Visual Mode

The Hierarchy Visual Mode is a display of the design with cells highlighted and color-coded according to block membership or levels of hierarchy. By default, the layout display shows the cells belonging to each top-level block highlighted in its own color, as shown in [Figure 4-10](#). A histogram-like bar graph shows the number of cells belonging to each top-level block, identified by name.

Figure 4-10 Hierarchy Visual Mode



You can also highlight cells using a different color for each second-level hierarchical block or each third-level hierarchical block, and so on, or you can highlight only the cells belonging to one or more particular hierarchical blocks, identified by name, at any level. You can set the hierarchical coloring options in the dialog box opened by clicking the Reload button.

Clock Tree Visual Mode

The Clock Tree Visual Mode highlights the flylines and cells of a clock tree using a different color for each buffer level of the tree. For example, [Figure 4-11](#) shows the connections in levels 0, 1, and 2 of the sys_clk clock tree, with the level 0 in yellow, level 1 in magenta, and level 2 in blue. This display can help you visualize the extent and approximate route lengths of the clock tree. A histogram-like bar graph shows the number of objects at each level of the clock tree.

Figure 4-11 Clock Tree Visual Mode



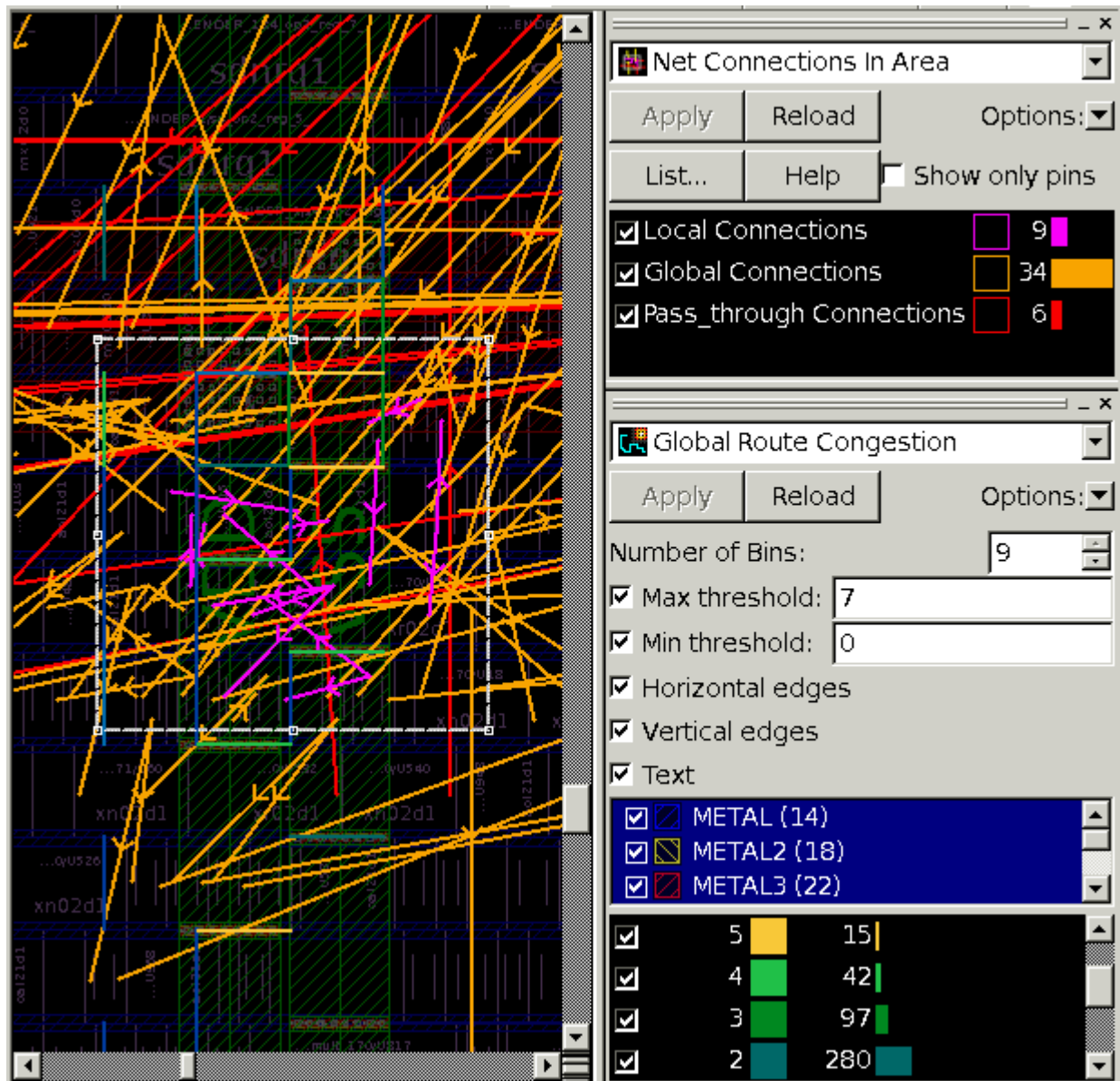
A dialog box displayed by clicking the Reload button lets you choose the clock, the clock levels, and the types of objects to be highlighted: levels, sinks, preexisting cells, inverters, or buffers.

Net Connections in Area Visual Mode

The Net Connections by Area Visual Mode lets you visualize the nets enclosed in or crossing a specified rectangular area of the design and examine the properties of those nets. To display net connections in an area, first select Net Connections in Area in the Visual Mode panel. Click Reload. In the dialog box, you can enter the coordinates of two corners of the desired area, or you can drag a rectangle directly in the layout view. You can also specify the types of nets to be displayed: signal and clock nets by default, and optionally power, ground, tie-high, and tie-low nets. Then click OK to update the display.

Figure 4-12 shows an example of a Net Connections in Area display superimposed on a Global Route Congestion map. The area of interest is the white dotted rectangle, which was drawn to analyze nets in a region of high congestion. There are three types of flyline connections highlighted in the view, called the local, global, and pass-through connections.

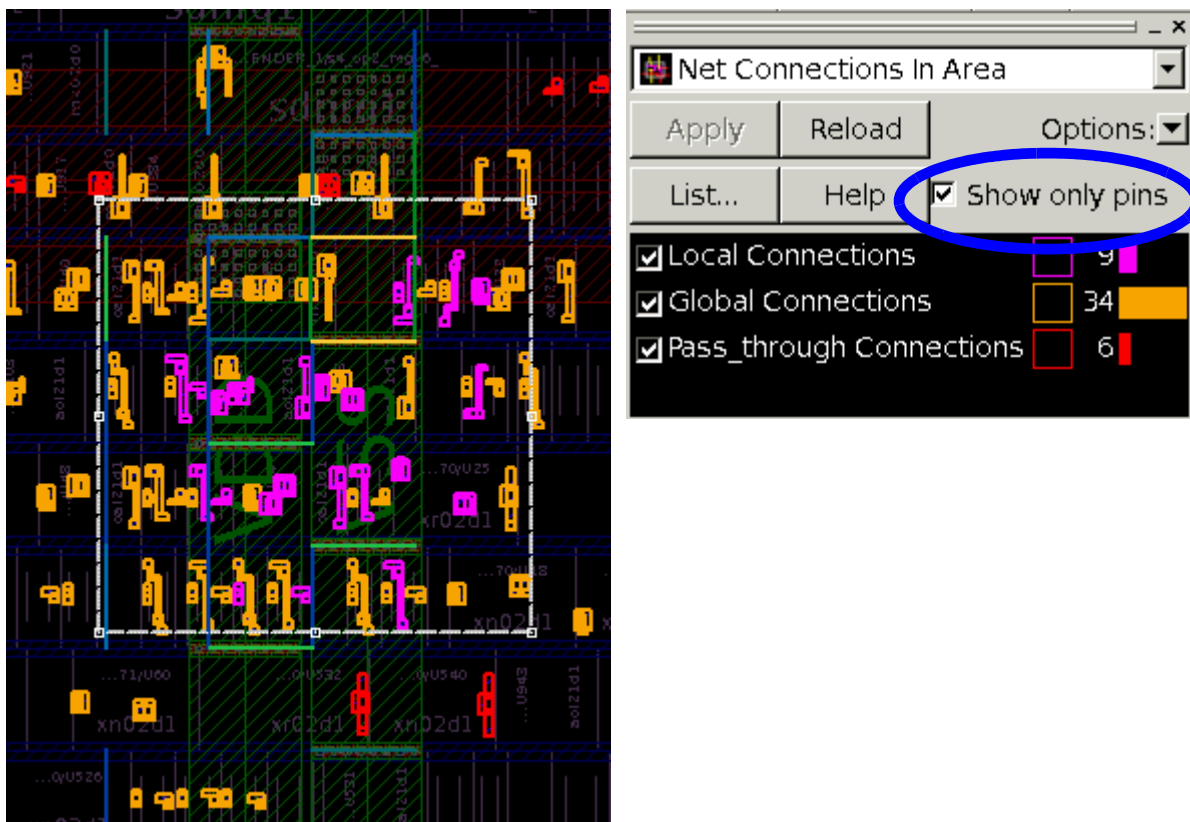
Figure 4-12 Net Connections in Area, Flyline Display



The local connections, shown in magenta, are completely enclosed by the rectangle. The global connections, shown in orange, are the nets that have one or more pins both inside and outside the rectangle. The pass-through connections, shown in red, have pins outside of the rectangle only; the nets are expected to pass through the rectangle without connecting to any pins inside the rectangle.

You can optionally turn off the display of flylines and show the connection pins instead by checking the “Show only pins” option in the Net Connections Area panel. See [Figure 4-13](#). You can also separately enable or disable the display of local, global, or pass-through type pins or net connections by checking or unchecking the respective options in the Visual Mode panel.

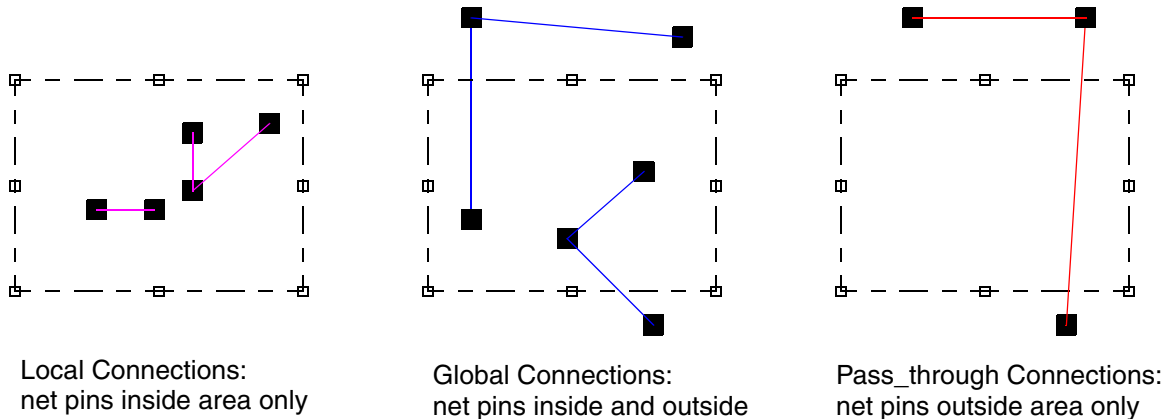
Figure 4-13 Net Connections in Area, Pin-Only Display



The three types of net connections highlighted in the Net Connections in Area view traverse the designated rectangular area in different ways, as demonstrated in [Figure 4-14](#). Connections are local if all the pins of the net are inside the designated area. Connections

are global if one or more pins of the net are both inside and outside the area. Connections are pass-through if all the pins of the net are outside the area and the connections are expected to pass through the area.

Figure 4-14 Local, Global, and Pass-Through Connections



In the Net Connections in Area display, a bar graph shows the numbers of local, global, and pass-through nets. To get more information on these nets in a category, select the category in the bar graph and then click the List button. This opens a table in a new dialog box. The table lists the local and hierarchical net names, net types, number of pins, and other information in the nets. From this table, you can change the types of net information displayed, highlight specific nets in the layout, generate histograms of the net data, and export the data to a spreadsheet.

The Net Connections in Area display capability lets you determine the net topology in a particular region and observe the properties of the nets in that area. This feature works with both routed and unrouted designs. You can analyze the nets that cross a specific area and use the information to change the floorplan or add constraints such as blockages. For example, excessive local nets might imply that utilization changes are needed, or excessive global or pass-through nets might imply that floorplan or placement constraint changes are needed.

Refining Placement

If your design shows large timing or power violations after you run `place_opt`, adjust the `place_opt` options and rerun `place_opt`, as described in [“Performing Placement and Optimization” on page 4-16](#).

If your design shows small timing or power violations after you run `place_opt`, run `psynopt` to fix these violations, as described in [“Using Physical Optimization” on page 4-18](#).

If your design has congestion violations after you run `place_opt`, rerun `place_opt` with high-effort congestion reduction (`-congestion` option). If your design still has congestion violations, you can refine the placement to fix these violations.

To refine the placement, use the `refine_placement` command (or choose Placement > Refine Placement in the GUI). The `refine_placement` command performs incremental placement and legalization.

[Table 4-4](#) describes the available `refine_placement` options.

Table 4-4 refine_placement Options

GUI object	Command option	Description
“Congestion optimization effort” radio button	<code>-congestion_effort low medium high</code>	Improve the quality of results or reduce runtime. Higher effort levels use additional runtime in an attempt to improve the quality of results. The default is medium.
“Perturbation level effort” radio button	<code>-perturbation_level min medium high max</code>	
“Whole chip” radio button	N/A	Perform incremental placement and legalization on the entire chip.
“Specified region” radio button, plus Coordinates list	<code>-coordinate {X1 Y1 X2 Y2}</code>	Perform incremental placement and legalization on the specified region.
“Snap to” list box	N/A	
“Ignore scan-chain connections during placement” check box	<code>-ignore_scan</code>	
“Number of CPUs” list box	<code>-num_cpus number</code>	Use additional CPUs. By default, IC Compiler uses one CPU.

To refine the placement with a minimum impact on QoR, use the medium-effort congestion removal algorithm:

```
icc_shell> refine_placement -congestion_effort medium
```

To achieve the best congestion removal, use high-level perturbation, and then perform physical optimization:

```
icc_shell> refine_placement -perturbation_level high  
icc_shell> psynopt
```


5

Design for Test

This chapter describes the IC Compiler physical design for test (DFT) features.

During logic synthesis, sequential elements are replaced with scan cells that can be configured as serial chain-of-state elements in order to aid in testability. By setting certain test signals, the scan cells can either capture values from the functional combinational logic or the test data.

The order of the scan cells can be changed under some constraints without affecting the testability of the device. However, the scan chain order from logic synthesis is not based on physical proximity of the scan cells. For higher quality of results (QoR) in physical design, you can use IC Compiler to reorder scan cells based on physical data.

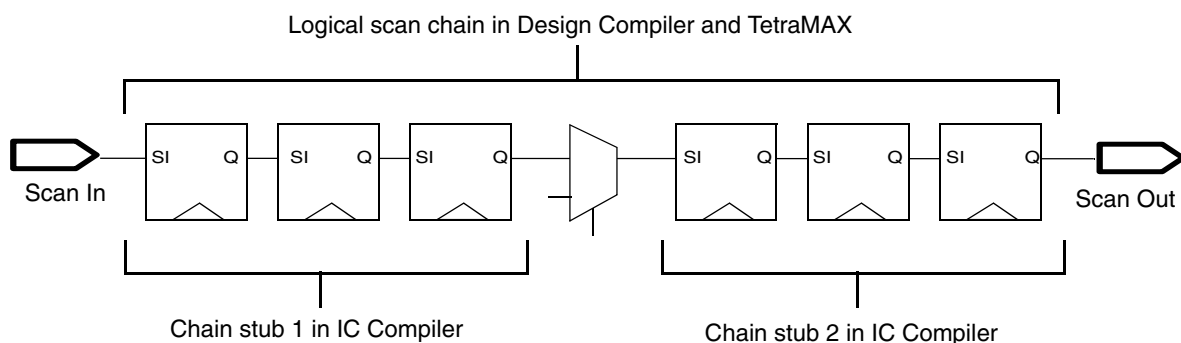
This chapter contains the following sections:

- [Overview of the DFT Flow](#)
- [Preparing for Physical DFT Using the read_def Flow](#)
- [Generating SCANDEF Data Using the trace_scan_chain Flow](#)
- [Scan Chain Consistency Checking](#)
- [Removing SCANDEF Data](#)
- [Optimizing Scan Chains](#)
- [Viewing Scan Chains](#)

Overview of the DFT Flow

Physical DFT relies on the identification of scan chain components. In IC Compiler, this defined scan chain data is referred to as SCANDEF. SCANDEF specifies the scan chain components and their pins used for the scan path, along with constraints on how they can be reordered. The view of scan chains in the SCANDEF data is different from that in DFT Compiler or TetraMAX (see [Figure 5-1](#)). SCANDEF can specify chain stubs that are contiguous subsets of scan cells of a logical scan chain.

Figure 5-1 Comparing Views of Scan Chains



There are two major flows for performing physical DFT in IC Compiler:

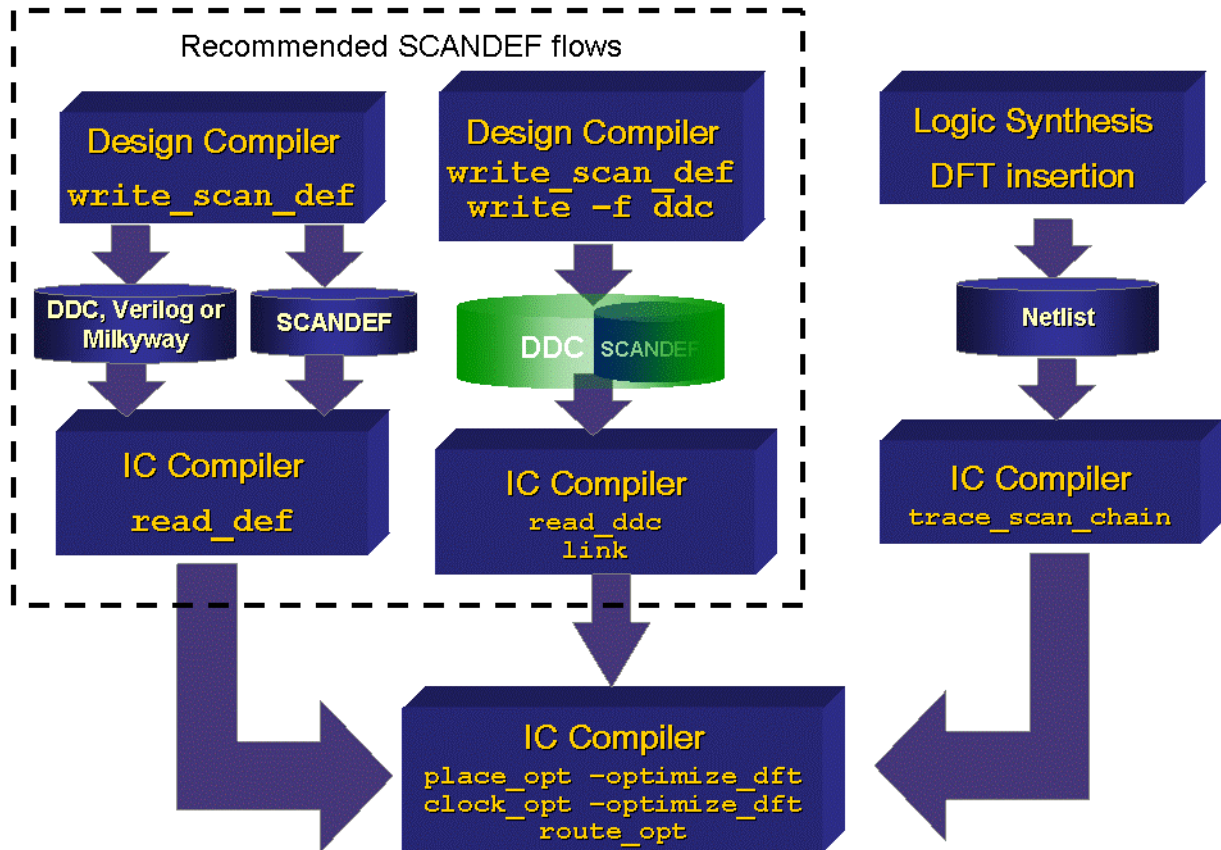
- The SCANDEF flow (recommended) accepts SCANDEF data (a subset of DEF) generated by DFT Compiler. Scan chain repartitioning and reordering are supported.
- The `trace_scan_chain` flow is used when a SCANDEF file is not available; in this case, IC Compiler internally generates SCANDEF data when you identify the scan chain endpoints for tracing. Only scan chain reordering is supported.

Note that you can use SCANDEF data generated from third-party flows. However, this data must conform to similar rules as DFT Compiler. For example, all combinational cell instances present in the SCANDEF must be in ORDERED lists. All multi-input components along a scan path of a specified chain stub must be included in the SCANDEF. All START (STOP) points should be input (output) ports of the current design or output (input) pins of cell instances.

You can check the consistency of the SCANDEF data against the netlist by using the `check_scan_chain` command.

Figure 5-2 shows the overall physical DFT flow in IC Compiler.

Figure 5-2 Overview of Physical DFT Flow in IC Compiler



Preparing for Physical DFT Using the read_def Flow

The SCANDEF flow is recommended when you use DFT Compiler for DFT insertion and use Design Compiler (Topographical) to generate SCANDEF data. This flow carries out the following primary steps:

1. Performs scan synthesis and generates SCANDEF data in Design Compiler. Creates an ASCII SCANDEF file or embeds the SCANDEF data in a .ddc file.
2. Loads in the SCANDEF data in IC Compiler.

Performing Scan Synthesis and Generating a SCANDEF File

In the SCANDEF flow, scan synthesis is performed first in Design Compiler. You then use the `write_scan_def` command to generate the SCANDEF data. This can be transferred to IC Compiler in either an ASCII SCANDEF file or a .ddc file with embedded SCANDEF data.

The following example shows a typical command sequence for scan insertion and for generating SCANDEF data in Design Compiler. Both the ASCII SCANDEF file and a .ddc file with embedded SCANDEF data are generated, but you only need to load the .ddc file in IC Compiler, because it encapsulates both design and SCANDEF data.

```
insert_dft
dft_drc
change_names -hierarchy -rule verilog
write_scan_def -output ./myscan.def
write -format -out ./design_with_scandef.ddc
```

The output SCANDEF file is an ASCII file that specifies a list of stub chains that is reordered by IC Compiler without requiring additional test design rule checking. Each stub chain defines a scan chain segment whose endpoints are an I/O port or pins of a lockup latch, or a multiplexer.

The following example shows a section of a SCANDEF file.

```
DESIGN my_design
SCANCHAINS 2;
-1
+ START PIN test_sil
+ FLOATING A ( IN SI ) ( OUT Q )
  B ( IN SI ) ( OUT Q )
  C ( IN SI ) ( OUT Q )
  D ( IN SI ) ( OUT Q )
+ PARTITION CLK_45_45
+ STOP PIN test_sol
```

The SCANDEF data embedded in the .ddc file can be loaded into IC Compiler.

Matching Hierarchy in Hierarchical Flow

If you are working with a hierarchical design, the logic hierarchy in Design Compiler where DFT was inserted must match the physical hierarchy in IC Compiler. This means that in DFT Compiler, after reading the block-level test models, you must generate SCANDEF data for each block, as well as for the top level. Then, in IC Compiler, you will later need to perform DFT optimization at the block level, and create an interface logic model (ILM). At the top level, you will need to read the top-level SCANDEF and block-level ILMs. Make sure you propagate the placement, keep out, and delay data to the top level blocks, before performing

the `place_opt` and `clock_opt` DFT optimizations. For details on the hierarchical flow, please see [Chapter 11, “Interface Logic Modeling”](#) and the “SCANDEF-Based Reordering Flow” section of the “Exporting to Other Tools” chapter of the *DFT Compiler User Guide: Scan*.

Reading in a SCANDEF File

For the SCANDEF flow, you should load SCANDEF data from either the ASCII SCANDEF file or the .ddc file with embedded SCANDEF data, before performing physical DFT.

The ASCII SCANDEF file can be loaded using the `read_def` command. To generate detailed information about errors and warnings that occur when reading the SCANDEF file, use the `-verbose` option.

```
read_def ./myscan.def
```

The SCANDEF data from the .ddc file can be loaded by using the `read_file -format ddc`, or `read_ddc`, or `import_design` command, followed by `link` as shown in the following example.

```
read_ddc ./design_with_scandef.ddc
link
check_scan_chain
report_scan_chain
place_opt -optimize_dft
```

Loading SCANDEF data from the .ddc file ensures that the netlist and the SCANDEF data are consistent. It also simplifies file management.

After reading in the SCANDEF file, you are ready to run consistency checking. For details on this process, see [“Scan Chain Consistency Checking” on page 5-7](#).

Generating SCANDEF Data Using the `trace_scan_chain` Flow

If you don't have a SCANDEF file and cannot generate one using the `write_scan_def` command in Design Compiler, you can generate internal SCANDEF data by using the `trace_scan_chain` command. You can then perform physical DFT optimizations.

Note that using `trace_scan_chain` instead of a SCANDEF file might result in lower QoR of the physically aware optimizations because the `trace_scan_chain` command cannot infer the PARTITION attribute. Scan cells will only be reordered within individual chain stubs.

The START and STOP points for the chain stub must be carefully selected to ensure reordering any enclosed cell will not disturb test or mission mode functionality. For example, if the STOP point feeds mission mode logic in addition to the scan network, changing the last scan cell may cause mission mode logic to become incorrect (Figure 5-3). Likewise, if lockup latches are traced, causing scan cells of different clock domains or edges to be mixed, the scan chain may fail to shift.

Diagram illustrating the placement of a STOP point in mission-mode logic. The diagram shows a sequence of logic blocks connected in a chain. The first block has inputs D and SI, and outputs Q and SO. The Q output of the first block is connected to the SI input of the second block, labeled "Better STOP point". The Q output of the second block is connected to the SI input of the third block, labeled "Bad STOP point". The Q output of the third block is connected to the SI input of the fourth block, labeled "Scan-out port". Dashed arrows point from the text "Mission-mode logic" to the Q outputs of the second and third blocks.

The document is for study only,if tort to your rights,please inform us,we will delete

edges are not supported in a single `trace_scan_chain` call. You have to specify START and STOP points so that the stubs connected to the lockup latches or edge-boundary scan cells are split into two. The following example shows a typical `trace_scan_chain` flow.

```
# when encountering an instance of mx02d0 from lib sc, trace
# back on pin I1.
set_scan_pin_type -type in -ref_pin \
  [get_physical_lib_pins sc/mx02d0/I1]
set_scan_pin_type -type out -ref_pin \
  [get_physical_lib_pins sc/mx02d0/Z]

# trace backwards from fanin of ucore/U186/Z
trace_scan_chain -from ucore/U23/Q -to ucore/U186/Z

# trace backwards from port out[2]
trace_scan_chain -from in[2] -to out[2]

# sanity check
check_scan_chain
```

Scan Chain Consistency Checking

IC Compiler maintains the SCANDEF data that describes scan chain characteristics and constraints, as well as the scan-stitched netlist. The netlist and SCANDEF data must be consistent with each other. To validate their consistency, you can use the `check_scan_chain` command. By default, a summary of the consistency of all the chains is displayed. It includes the number of scan cells (sequential depth) in each chain. Upon failure, the `-chain` option can be used to debug the inconsistency.

Consistency checking is done on elements between the START and STOP points of the chain stubs. The check starts at the STOP point and proceeds backward. Each scan chain stub is given a status after the checks are complete. The `check_scan_chain` and `report_scan_chain` commands report the following status:

- NOT YET VALIDATED: `check_scan_chain` has not been run
- FAILED (F): SCANDEF and netlist are inconsistent for this chain
- VALIDATED (V): SCANDEF and netlist are consistent for this chain

Only VALIDATED scan chain stubs will be optimized. If there are failures, the SCANDEF or the netlist must be fixed to have them optimized.

The following items are not checked during `check_scan_chain`:

- MAXBITS and scan chain lengths
- PARTITION labels

Further validation can be done using the Formality equivalency checker to ensure the pre-layout netlist is equivalent to the post-layout netlist, with test signals de-asserted or not compared.

Removing SCANDEF Data

If either the .ddc file with embedded SCANDEF or the ASCII SCANDEF file contains a chain with the same name as an existing scan chain loaded previously, the existing chain definition is preserved. If you want to overwrite the existing chain definitions, you must first remove the SCANDEF data. You can use the `remove_scan_def` command to remove all SCANDEF data that was previously loaded from a .ddc file or an ASCII SCANDEF file, or that was created from the `trace_scan_chain` command. Note, however, that physical DFT optimizations cannot be done without SCANDEF data.

Optimizing Scan Chains

After performing consistency checking, there are two, non-exclusive physical DFT methods you can use to optimize scan chains with the validated status:

- [Placement-Aware Scan Chain Reordering](#) — The scan chains are repartitioned and reordered, based on scan cell locations.
- [Clock-Aware Scan Reordering](#) — The scan chains are reordered to minimize the number of clock buffer crossings in the scan chain.

Note that reordering is done on scan cells in SCANDEF data regardless of the presence of any attributes, such as `dont_touch`.

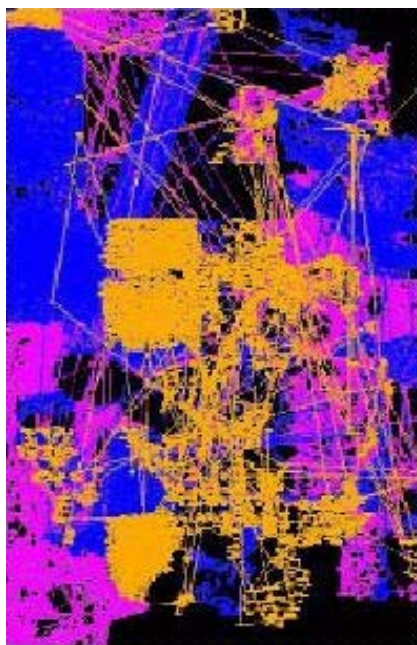
Placement-Aware Scan Chain Reordering

The `place_opt -optimize_dft` command can repartition and reorder the scan chains, based on scan cell locations. Repartitioning involves swapping scan chain components across scan chains so that the scan chains will contain components that are near each other. Using the placement information to optimize the scan chain provides the following benefits:

- Reduces scan chain wire length
- Minimizes congestion and improves routability

[Figure 5-4](#) shows the wiring diagram for a design containing scan chains that were synthesized by Design Compiler. Because these scan chains are not optimally ordered with respect to actual placement and clock trees, they can cause congestion and increased net delays.

Figure 5-4 Scan Chains Before Physically Aware Optimizations

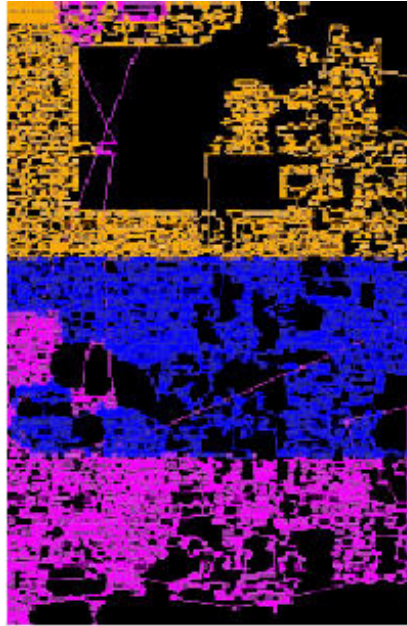


IC Compiler gets the scan chain information from the SCANDEF data and improves the quality of results by ignoring nets connected to the scan-in pin of scan cells during initial placement and by performing physically aware scan chain optimizations during `place_opt` and `clock_opt`. Note that `dont_touch` attributes are not honored by scan chain reordering.

To repartition and reorder the scan chains using placement information, select “Reorder scan during placement” in the Core Placement and Optimization dialog box (Placement > Core Placement and Optimization) or use the `-optimize_dft` option when you run the `place_opt` command. The `optimize_dft` stand-alone command is also available.

By default, IC Compiler partitions scan chains horizontally, as shown in [Figure 5-5](#).

Figure 5-5 Horizontally Partitioned Scan Chains



If partitioning the scan chains horizontally significantly increases congestion, use the `set_optimize_dft_options -partition vertical` command to select vertical partitioning and rerun the `place_opt` command.

Following optimizations, approximate wire lengths of the scan path wires that can be reordered are displayed. (Scan path nets in ORDERED lists cannot be reordered and therefore are not counted.) These lengths are in Milkyway database units and meant only for comparing wire lengths before and after scan chain optimizations, as shown in the following example.

```
Wire Length Before Ordering 1377917
Wire Length After Ordering 604876
```

Clock-Aware Scan Reordering

The `clock_opt` command reorders the scan chains to minimize the number of clock buffer crossings in the scan chain. Minimizing the number of buffer crossings can reduce hold time violations in the scan chain.

To reorder the scan chains using clock information, use the `-optimize_dft` option when you run the `clock_opt` command. The `optimize_dft -clock_buffer` stand-alone function is also available. In the GUI, select the “Reorder scan cells” setting in the “Core CTS and Optimization” dialog box (Clock > Core CTS and Optimization).

Example Optimization Flow

The following example shows a typical physical DFT optimization flow:

```
# following read_def, read_ddc, or trace_scan_chain

check_scan_chain
place_opt -optimize_dft
check_scan_chain
clock_opt -optimize_dft

# check_scan_chain must be run before report_scan_chain

report_scan_chain
route_opt
save_mw_cel
```

Viewing Scan Chains

You can view scan chains either by generating a scan chain report or by viewing the scan chains in the GUI.

To generate a scan chain report, use the `report_scan_chain` command. The scan chain report lists the current scan chain connections. The `report_scan_chain` command displays the chain stub connectivity as it exists in the netlist after a chain has been validated following `check_scan_chain`. All cells in the scan chain are reported, not just scan cells. Note that a SCANDEF file does not necessarily follow the order of connectivity defined in the netlist. Components inside a floating list are considered unordered, and ordered lists are also unordered among the other lists.

To view the scan chains in the GUI,

1. Choose Placement > Color By Scan Chain.

The Visual Mode window opens.

2. Load the scan chain information by clicking Reload in the Visual Mode panel.

The number it displays next to each scan chain is the total number of cells in each scan chain plus two additional cells for the start and stop points.

3. Select the scan chains that should be highlighted in the layout window.

The layout window displays the scan chains.

6

Power Optimization

This chapter describes the IC Compiler power optimization capabilities.

When you activate power optimization, IC Compiler can perform the following types of power optimization:

- [Low-Power Placement](#)
- [Leakage Power Optimization](#)
- [Dynamic Power Optimization](#)

The default setup for power optimization performs only leakage power optimization during placement or routing. To perform other power optimizations, you must enable each of them separately. You can also change the constraints and other settings that are used for these power optimizations. The following sections explain the set up and use of the power optimizations.

Low-Power Placement

Low-power placement reduces capacitance between nets with high switching activity. The reduction in capacitance is achieved by shortening the nets during optimization and placement. To shorten the nets, IC Compiler runs clock tree synthesis to generate a clock tree that helps to determine where best to shorten each net by moving its cells closer together.

To set up for low-power placement,

1. Set the `-low_power_placement` option of the `set_power_options` command to true.

```
icc_shell> set_power_options -low_power_placement true
```

To disable low-power placement, set this option to false. The default value of the `-low_power_placement` option is false.

Note:

To see the current power optimization settings set with the `set_power_options` command, use the `report_power_options` command.

2. Set up clock tree synthesis just as you do after performing placement and optimization.

For more information, see [Chapter 7, “Clock Tree Synthesis.”](#)

3. Annotate switching activity information to the design for the best results or apply the default toggle rate.

For information, see [“Reading Switching Activity Information” on page 6-3.](#)

To run low-power placement, use the `-power` option of the `place_opt` command.

```
icc_shell> place_opt -power
```

Alternatively, in the GUI, choose Placement > Core Placement and Optimization and then select Power Optimization.

Reading Switching Activity Information

For the best results with low-power placement (dynamic and total power optimization), you should annotate the high-quality switching activity information to the design before you perform these power functions.

To read in the switching activity information, use the `read_saif` command with the following syntax:

```
icc_shell> read_saif -input saif_file -instance path
```

Alternatively, you can use the `set_switching_activity` command. If no switching activity has been annotated to the design, IC Compiler applies the default toggle rate to the primary inputs and black box outputs and then propagates it throughout the design.

Leakage Power Optimization

Leakage power optimization is an additional step to timing optimization that minimizes leakage power without degrading performance. By default, IC Compiler uses an adaptive leakage power optimization algorithm that provides one-pass leakage optimization with minimal impact on timing, area, and congestion. IC Compiler can perform leakage optimization during placement (`place_opt`), clock tree synthesis (`clock_opt`), and routing (`route_opt`).

Leakage power optimization is enabled by default. To disable leakage power optimization, set the `-leakage` option to false using the `set_power_options` command. You can set up multiple threshold voltage libraries to reduce the leakage power, though leakage optimization works on single threshold libraries also. For more information, see [“Setting Up the Logic Libraries” on page 3-2](#).

To run leakage power optimization (and any other power optimizations that you have enabled and that work with placement, clock tree synthesis and routing functions), do one of the following:

- To run placement and optimization, use the `place_opt -power` command. Alternatively, in the GUI, choose Placement > Core Placement and Optimization, and then select Power Optimization.
- To run incremental placement, use the `psynopt -power` command.

```
icc_shell> psynopt -power
```

If you want to perform only power optimization during incremental placement, use the `-power_only` option.

```
icc_shell> psynopt -power_only
```

- To run routing and power optimization, use the `route_opt -power` command. Alternatively, in the GUI, choose Route > Core Routing and Optimization, and then select Perform Power Optimization.

Improving Leakage Power Using Multithreshold Voltage Cells

If your target library supports multithreshold voltage cells, you can reduce the leakage power by controlling the percentage of lower threshold voltage cells in your design. Use the `set_max_lvth_percentage` command to specify the maximum percentage of the total design area that can be occupied by lower threshold voltage cells. The optimization commands, `place_opt`, `route_opt` and `clock_opt` when used with the `-only_psyn` option honor the constraint set by the `set_max_lvth_percentage` command. Note that clock tree synthesis does not honor the constraint set by this command.

The syntax of the `set_max_lvth_percentage` command is as follows:

```
set_max_lvth_percentage [max_lvth] [-lvth_groups groups] [-reset]
```

The *max_lvth* specifies the maximum percentage of the total design area that can be occupied by cells belonging to the low voltage threshold group. The value of the *max_lvth* can be between 0 and 100.

The `-lvth_groups` option can be used to specify the list of voltage threshold groups that belong to the low voltage threshold group.

If you have already set the maximum percentage constraint on the low voltage threshold cells, reset the value by using the `-reset` option.

Use the `report_threshold_voltage_group` command to report the percentages of different multivoltage cells in the design.

Dynamic Power Optimization

After performing RTL clock gating or operand isolation, you can use gate-level dynamic power optimization, an additional step to timing optimization, to further reduce the dynamic power during a placement function. This optimization targets a dynamic power constraint, which is set, by default, to 0 for a well-balanced runtime and quality of results. You can run this optimization when performing placement and optimization or placement alone.

To set up for dynamic power optimization,

1. Enable dynamic power optimization.

To enable dynamic power optimization, set the `-dynamic` option to true with the `set_power_options` command. By default, dynamic power optimization is disabled.

```
icc_shell> set_power_options -dynamic true
```

2. (Optional) Annotate the switching activity information to the design unless you want IC Compiler to use the default switching activity.

For more information, see [“Reading Switching Activity Information” on page 6-3](#).

To run dynamic power optimization and any other power optimizations that you have enabled and that work with these placement functions, use the `-power` option with `place_opt` or `psynopt` command. To perform only power optimizations, use `-power_only` option with the `psynopt` command.

If you are using the GUI, choose Placement > Core Placement and Optimization, then select Power Optimization.

7

Clock Tree Synthesis

This chapter provides detailed information about the IC Compiler clock tree synthesis capability.

This chapter contains the following sections:

- [Prerequisites for Clock Tree Synthesis](#)
- [Analyzing the Clock Trees](#)
- [Defining the Clock Trees](#)
- [Specifying Clock Tree Exceptions](#)
- [Specifying the Clock Tree References](#)
- [Specifying Clock Tree Synthesis Options](#)
- [Specifying Clock Tree Optimization Options](#)
- [Inserting User-Specified Clock Trees](#)
- [Handling Specific Design Characteristics](#)
- [Verifying the Clock Trees](#)
- [Implementing the Clock Trees](#)
- [Performing Multicorner Clock Tree Optimization](#)
- [Using Batch Mode to Reduce Runtime](#)

- [Analyzing the Clock Tree Results](#)
- [Fine-Tuning the Clock Tree Synthesis Results](#)
- [Analyzing and Refining the Design](#)

Prerequisites for Clock Tree Synthesis

Before you run clock tree synthesis, ensure that your design and libraries meet the prerequisites described in the following sections.

Design Prerequisites

Before running clock tree synthesis, your design should meet the following requirements:

- The design is placed and optimized.

Use the `check_legality -verbose` command to verify that the placement is legal. Running clock tree synthesis on a design that does not have a legal placement might result in long runtimes and reduced QoR.

The estimated QoR for the design should meet your requirements before you start clock tree synthesis. This includes acceptable results for

- Congestion

If congestion issues are not resolved before clock tree synthesis, the addition of clock trees can increase congestion. If the design is congested, you can rerun `place_opt` with the `-congestion` and `-effort high` options, but the runtime can be long.

- Timing
- Maximum capacitance
- Maximum transition time

To ensure that the clock tree can be routed, verify that the placement is such that the clock sinks are not in narrow channels and that there are no large blockages between the clock root and its sinks. If these conditions occur, fix the placement before running clock tree synthesis.

- The power and ground nets are prerouted.
- High-fanout nets, such as scan enables, are synthesized with buffers.

Library Prerequisites

Before you run clock tree synthesis, your libraries must meet the following requirements:

- Any cell in the logic library that you want to use as a clock tree reference (a buffer or inverter cell that can be used to build a clock tree) or for sizing of gates on the clock network must be usable by clock tree synthesis and optimization.

By default, clock tree synthesis and optimization cannot use buffers and inverters that have the `dont_use` attribute to build the clock tree. To use these cells during clock tree synthesis and optimization, you can either remove the `dont_use` attribute by using the `remove_attribute` command or you can override the `dont_use` attribute by specifying the cell as a clock tree reference by using the `set_clock_tree_references` command, as described in [“Specifying the Clock Tree References” on page 7-18](#).

- The physical library should include
 - All clock tree references (the buffer and inverter cells that can be used to build the clock trees)
 - Routing information, which includes layer information and nondefault routing rules
- TLUPlus models must exist.

Extraction requires these models to estimate the net resistance and capacitance.

Analyzing the Clock Trees

Before running clock tree synthesis, analyze each clock tree in your design to determine its characteristics and its relationship to other clock trees in the design.

For each clock tree, determine

- What the clock root is
- What the desired clock sinks and clock tree exceptions are

IC Compiler supports the following types of clock tree exceptions: exclude pins, stop pins, float pins, don't touch subtrees, don't buffer nets, and don't size cells.

- Whether the clock tree contains preexisting cells, such as clock gating cells

If your design contains existing clock trees, you might want to either identify them or remove them before running clock tree synthesis. For information about handling existing clock trees, see [“Handling Existing Clock Trees” on page 7-45](#).

- Whether the clock tree converges, either with itself (a convergent clock path) or with another clock tree (an overlapping clock path)
- Whether the clock tree has timing relationships with other clock trees in the design, such as interclock skew requirements
- What the logical design rule constraints (maximum fanout, maximum transition time, and maximum capacitance) are
- What the routing constraints (routing rules and metal layers) are

Use this information when you define the clock trees and to validate that IC Compiler has the correct clock tree definitions.

Note:

You can generate clock tree reports to analyze the clock tree structure, even before you perform clock tree synthesis and optimization. For information about generating clock tree reports, see [“Generating Clock Tree Reports” on page 7-79](#).

Defining the Clock Trees

IC Compiler uses the clock sources defined by the `create_clock` command as the clock roots and derives the default set of clock sinks by tracing through all cells in the transitive fanout of the clock roots. You can designate either an input port or internal hierarchical pin as a clock source. To define clock sources on hierarchical pins, set the `cts_enable_clock_at_hierarchical_pin` variable to true before using the `create_clock` command. This variable is false by default.

Note:

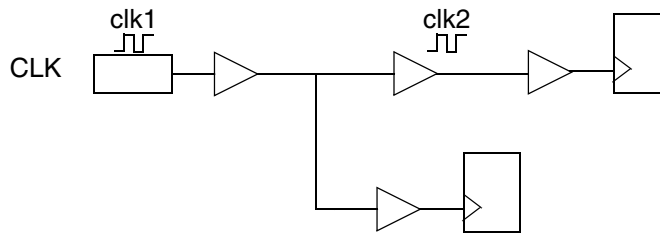
IC Compiler will not synthesize a clock tree if its source is set to a constant value by using `set_case_analysis`.

In addition to simple clock trees, IC Compiler also supports cascaded clock trees (a clock tree that contains another clock tree in its fanout). The nested clock tree can either have its own source (identified by the `create_clock` command) or be a generated clock (identified by the `create_generated_clock` command).

Cascaded Clocks

If a nested clock tree has its own source, as shown in the example that is given in [Figure 7-1](#), IC Compiler considers the source pin of the driven clock (clk2 in this example) to be an implicit exclude pin of the driving clock (clk1 in this example). Sinks of the driven clock are not considered sinks of the driving clock.

Figure 7-1 Cascaded Clock With Two Source Clocks

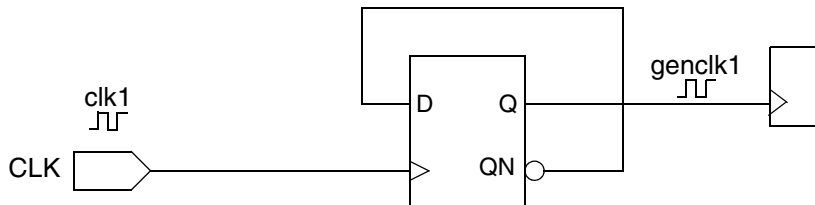


To verify that the clock sources are correctly defined, use the `check_clock_tree` command.

Cascaded Generated Clocks

If a nested clock tree has a generated source, as shown in [Figure 7-2](#), IC Compiler traces back to the master clock source from which the generated clock is derived and considers the sinks of the generated clock to be the sinks of the driving clock tree.

Figure 7-2 Cascaded Clock With Generated Clock



Incorrectly defining the master clock source, as in the following cases, results in poor skew and timing QoR.

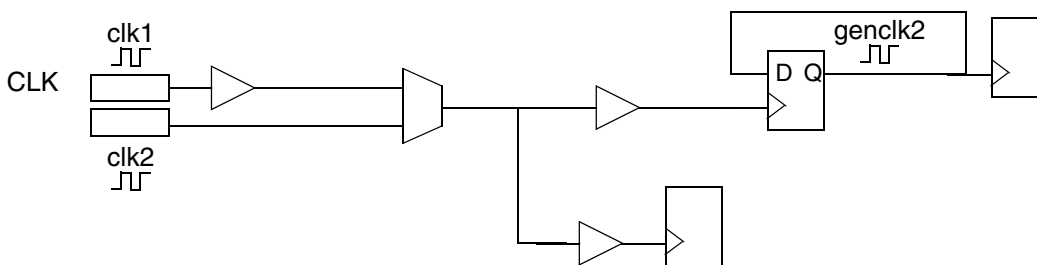
- If IC Compiler cannot trace back to the master clock source, the tool cannot balance the sinks of the generated clock with the sinks of its source.
- If the master clock source is not a clock source defined by `create_clock` or `create_generated_clock`, IC Compiler cannot synthesize a clock tree for the generated clock or its source.

Use the `check_clock_tree` command to verify that your master clock sources are correctly defined.

If a nested clock tree has a generated source that is defined at the intersection of overlapping clocks, IC Compiler traces back to the master clock source from which the generated clock is derived and considers the source pin of the generated clock on a per clock basis during synthesis. By default, IC Compiler clock tree synthesis supports querying phase delay and clock tree exceptions for each clock domain to achieve the best QoR for cascaded generated clocks with overlapping clocks.

For example, assume that a flip-flop inside the `clk1` and `clk2` overlapping domains has a generated clock, `genclk2`, which is defined at its Q output and which uses `clk2` as the master clock. If you perform clock tree synthesis on `clk1`, the clock tree after that flip-flop is synthesized and balanced as a generated clock of `clk1`, even though it is actually a generated clock of `clk2`, as is shown in [Figure 7-3](#).

Figure 7-3 Cascaded Generated Clock with Overlapping Clock Domains



Identifying the Clock Tree Endpoints

Clock paths have two types of endpoints:

- Stop pins

Stop pins are the endpoints of the clock tree that are used for delay balancing. During clock tree synthesis, IC Compiler uses stop pins in calculations and optimizations for both design rule constraints and clock tree timing (skew and insertion delay).

Stop pins are also referred to as sink pins.

- Exclude pins

Exclude pins are clock tree endpoints that are excluded from clock tree timing calculations and optimizations. IC Compiler uses exclude pins only in calculations and optimizations for design rule constraints.

IC Compiler traces the transitive fanout from the clock roots to determine the implicit stop pins (the default sink pins) and the implicit exclude pins.

When IC Compiler finds a pin in the clock's transitive fanout that is defined as a clock pin of either a sequential cell (a latch or flip-flop) or a macro cell, it adds that pin to the default set of clock sinks (unless the fanout of that cell drives a generated clock).

If the fanout of a sequential cell drives a generated clock, IC Compiler considers the clock pin to be an implicit nonstop pin and traces through the sequential cell to locate the clock tree endpoints. In addition, IC Compiler considers the clock input pins of integrated clock gating (ICG) cells to be implicit nonstop pins.

IC Compiler defines the following clock endpoints as implicit exclude pins:

- Source pins of clock trees in the fanout of another clock
- Nonclock input pins of sequential cells
- Multiplexer select pins
- Three-state enable pins
- Output ports
- Incorrectly defined clock pins (for example, the clock pin does not have trigger edge information or does not have a timing arc to the output pin)
- Buffer or inverter input pins that are held constant (by using `set_case_analysis`)
- Input pins of combinational cells or integrated clock gating cells that do not have any fanout or that do not have any enabled timing arcs

Verify that the default sink pins (implicit stop pins), implicit nonstop pins, and implicit exclude pins are accurate by generating a clock tree exceptions report, as described in [“Generating Clock Tree Reports” on page 7-79](#).

If the default sink pins, implicit nonstop pins, and implicit exclude pins are correct, you are done with the clock tree exception definition. Otherwise, first identify any timing settings, such as disabled timing arcs and case analysis settings, that affect the clock tree traversal. To identify disabled timing arcs in your design, use the `report_disable_timing` command. To identify case analysis settings in your design, use the `report_case_analysis` command. Remove any timing settings that cause an incorrect clock tree definition.

You can modify the set of sink (stop) pins, nonstop pins, and exclude pins by setting clock tree exceptions, as described in [“Specifying Clock Tree Exceptions” on page 7-9](#).

Defining the Clock Root Attributes

If the clock root is an input port (without an I/O pad cell), you must accurately specify the driving cell of the input port. A weak driving cell does not affect logic synthesis, because logic synthesis uses ideal clocks. However, during clock tree synthesis, a weak driving cell can cause IC Compiler to insert extra buffers as the tool tries to meet the clock tree design rule constraints, such as maximum transition time and maximum capacitance.

For example, if clock tree CLK1 has input port CLK1 as its root and CLK1 is driven by cell CLKBUF, enter

```
icc_shell> set_driving_cell -lib_cell mylib/CLKBUF [get_ports CLK1]
```

If you do not specify a driving cell (or drive strength), IC Compiler assumes that the port has infinite drive strength.

If the clock root is an input port with an I/O pad cell, you must accurately specify the input transition time of the input port.

For example, if clock tree CLK1 has input port CLK1 as its root and the I/O pad cell has already been inserted, enter

```
icc_shell> set_input_transition -rise 0.3 [get_ports CLK1]
icc_shell> set_input_transition -fall 0.2 [get_ports CLK1]
```

Specifying Clock Tree Exceptions

To define clock tree exceptions, use the `set_clock_tree_exceptions` command (or choose Clock > Set Clock Tree Exceptions in the GUI). You can set clock tree exceptions on pins or hierarchical pins. [Table 7-1](#) summarizes the clock tree exceptions supported by IC Compiler and the command option or GUI object used to specify them.

Table 7-1 Clock Tree Exceptions

Command option	GUI object	Description
Pin Exceptions		
<code>-non_stop_pins</code>	“Non stop pins” field	Nonstop pins are pins through which clock tree tracing continues. For more information, see “Specifying Nonstop Pins” on page 7-11 .
<code>-exclude_pins</code>	“Exclude pins” field	Exclude pins are clock tree endpoints that are not used in clock tree timing calculations and optimizations (they are used for design rule constraint calculations and optimizations). For more information, see “Identifying the Clock Tree Endpoints” on page 7-7 and “Specifying Exclude Pins” on page 7-12 .
<code>-float_pins</code> <code>-float_pin_logic_level</code> <code>-float_pin_max_delay_fall</code> <code>-float_pin_max_delay_rise</code> <code>-float_pin_min_delay_fall</code> <code>-float_pin_min_delay_rise</code>	“Float pins” field “Max delay fall” field “Max delay rise” field “Min delay fall” field “Min delay rise” field “Logic level” field	Float pins are clock sinks that have special insertion delay requirements. For more information, see “Specifying Float Pins” on page 7-13 .
<code>-stop_pins</code>	“Stop pins” field	Stop pins are clock tree endpoints that are used in calculations and optimizations for both design rule constraints and clock tree timing (skew and insertion delay). For more information, see “Identifying the Clock Tree Endpoints” on page 7-7 and “Specifying Stop Pins” on page 7-15 .

Table 7-1 Clock Tree Exceptions (Continued)

Command option	GUI object	Description
<code>-dont_touch_subtrees</code>	“Don’t touch subtree pins” field	Prevents modification of the clock network beyond the specified pins. IC Compiler considers the sinks of the don’t touch subtree when balancing clock delays or computing the clock skew. For more information, see “Specifying Don’t Touch Subtrees” on page 7-16 .
Net Exceptions		
<code>-dont_buffer_nets</code>	“Don’t buffer nets” field	Prevents buffering of the specified nets during clock tree synthesis. IC Compiler still performs global prerouting on these nets. For more information, see “Specifying Don’t Buffer Nets” on page 7-17 .
Cell Exceptions		
<code>-dont_size_cells</code>	“Don’t size cells” field	Identifies cells that should not be sized during clock tree synthesis and optimization. For more information, see “Specifying Don’t Size Cells” on page 7-17 .

If you issue the `set_clock_tree_exceptions` command multiple times for the same pin, the pin keeps the highest-priority exception. IC Compiler prioritizes the clock tree pin exceptions as follows:

1. Nonstop pins
2. Exclude pins
3. Float pins
4. Stop pins

Note:

The don’t touch subtree exception is compatible with the nonstop, exclude, float, or stop pin exception. You can set both exceptions on a pin, and clock tree synthesis honors both exceptions.

To remove clock tree exceptions, use the `remove_clock_tree_exceptions` command (or choose Clock > Remove Clock Tree Exceptions in the GUI).

To see the clock tree exceptions defined for your design, generate a clock tree exceptions report by running the `report_clock_tree -exceptions` command (or by choosing choose Clock > Report Clock Tree in the GUI and selecting Exceptions).

Note:

If your design contains sequential cells with unconnected outputs, the clock pins of these cells are marked as implicit ignore pins. When you run clock tree synthesis, these unloaded sequential cells are deleted from the design. As a result, at the end of clock tree synthesis, you no longer see these implicit ignore pins. To prevent the removal of these sequential cells, set the `physopt_delete_unloaded_sequential_cells` variable to false before running clock tree synthesis.

If your design contains dangling nets, the clock tree exceptions report might show false implicit ignore pins on these nets. If this happens, you can remove the false implicit ignore pins by saving your design and reopening it.

The following sections provide details for each of the clock tree exceptions.

Specifying Nonstop Pins

Nonstop pins are pins that would normally be considered endpoints of the clock tree, but instead IC Compiler traces through them to find the clock tree endpoints. The clock pins of sequential cells driving generated clocks are implicit nonstop pins. In addition, IC Compiler supports user-defined (or explicit) nonstop pins.

To specify a nonstop pin, use the `set_clock_tree_exceptions -non_stop_pins` command.

For example, to specify pin U2/CLK as a nonstop pin, enter

```
icc_shell> set_clock_tree_exceptions -non_stop_pins [get_pins U2/CLK]
```

To remove the nonstop pin definition from a pin, use the `remove_clock_tree_exceptions -non_stop_pins` command.

For example, to remove the nonstop pin definition from pin U2/CLK, enter

```
icc_shell> remove_clock_tree_exceptions -non_stop_pins [get_pins U2/CLK]
```

Specifying Exclude Pins

Exclude pins are clock tree endpoints that are excluded from clock tree timing calculations and optimizations. IC Compiler uses exclude pins only in calculations and optimizations for design rule constraints. In addition to the exclude pins inferred by IC Compiler (the implicit exclude pins), IC Compiler supports user-defined (or explicit) exclude pins. For example, you might define an exclude pin to exclude all branches of the clock tree that fan out from some combinational logic or to exclude an implicit stop pin.

During clock tree synthesis, IC Compiler isolates exclude pins (both implicit and explicit) from the clock tree by inserting a guide buffer before the pin. Beyond the exclude pin, IC Compiler never performs skew or insertion delay optimization, but does perform design rule fixing.

To specify an exclude pin, use the `set_clock_tree_exceptions -exclude_pins` command.

For example, to exclude clock sink U2/CLK, enter

```
icc_shell> set_clock_tree_exceptions -exclude_pins [get_pins U2/CLK]
```

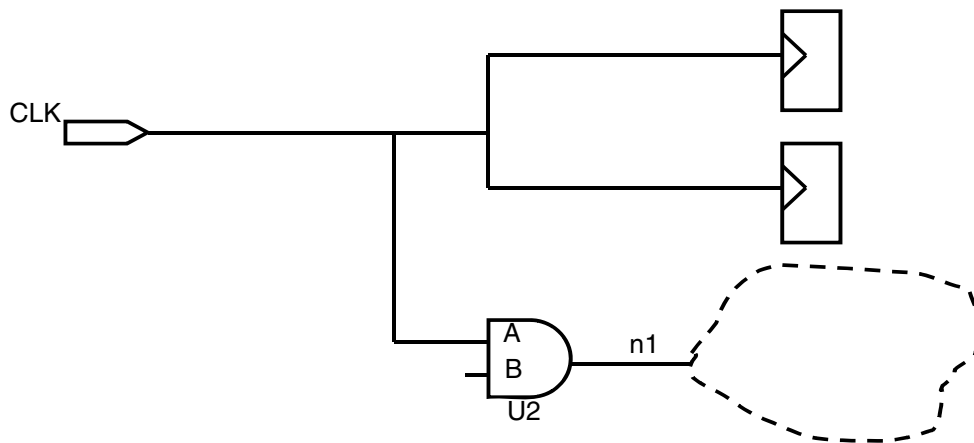
To remove the exclude pin definition from a pin, use the `remove_clock_tree_exceptions -exclude_pins` command.

For example, to remove the exclude pin definition from pin U2/CLK, enter

```
icc_shell> remove_clock_tree_exceptions -exclude_pins [get_pins U2/CLK]
```

For another example, assume that a clock tree also drives combinational logic, as shown in [Figure 7-4](#).

Figure 7-4 Explicit Exclude Pin



To exclude all branches of the clock tree that fan out from this point, enter

```
icc_shell> set_clock_tree_exceptions -exclude_pins [get_pins U2/A]
```

Specifying Float Pins

Float pins are clock pins that have special insertion delay requirements. IC Compiler adds the float pin delay (positive or negative) to the calculated insertion delay up to this pin.

To specify a float pin and its timing characteristics, use the following `set_clock_tree_exceptions` options:

- `-float_pins [get_pins, pin_list]`
- `-float_pin_max_delay_fall max_delay_fall_value`
- `-float_pin_max_delay_rise max_delay_rise_value`
- `-float_pin_min_delay_fall min_delay_fall_value`
- `-float_pin_min_delay_rise min_delay_rise_value`
- `-float_pin_logic_level logic_level_value`

The `-float_pin_logic_level` option specifies the number of logic levels beyond the float pin (this information is used for logic-level balancing). For more information about logic-level balancing, see [“Enabling Logic-Level Balancing” on page 7-34](#).

Note:

If you use the `-float_pins` option, you must specify at least one of the float pin delay options or an error occurs.

IC Compiler assumes the active edge of the float pin based on the specified float delay options. [Table 7-2](#) describes the edge-aware float pin behavior.

Table 7-2 Edge-Aware Float Pins Delays

Options specified	Active edge	Comments
Fall delays only: -float_pin_max_delay_fall -float_pin_min_delay_fall	Falling ¹	If only one of these options is specified, the specified value is used for both the minimum and maximum fall delay.
Rise delays only: -float_pin_max_delay_rise -float_pin_min_delay_rise	Rising ¹	If only one of these options is specified, the specified value is used for both the minimum and maximum rise delay.
Mix of rise and fall delays: -float_pin_max_delay_fall -float_pin_max_delay_rise -float_pin_min_delay_fall -float_pin_min_delay_rise	Both	If one of the fall delay options is not specified, the specified fall delay option is used for both the minimum and maximum fall delays. If one of the rise delay options is not specified, the specified rise delay option is used for both the minimum and maximum rise delays.

1. To define an edge-aware stop pin, set the appropriate float pin delay values to zero.

For example, to define an active-low float pin (so that clock tree synthesis considers only the falling edge), enter

```
icc_shell> set_clock_tree_exceptions -float_pins [get_pins U1/CLK] \
          -float_pin_max_delay_fall 0.10 -float_pin_min_delay_fall 0.08
```

The clock tree exceptions report (report_clock_tree -exceptions) shows the float pin values only for the active edge. For example, the report for the float pin defined above shows

```
Explicit sync pins: 1
(F) U1/CLK ( - 0.100 - 0.080 )
```

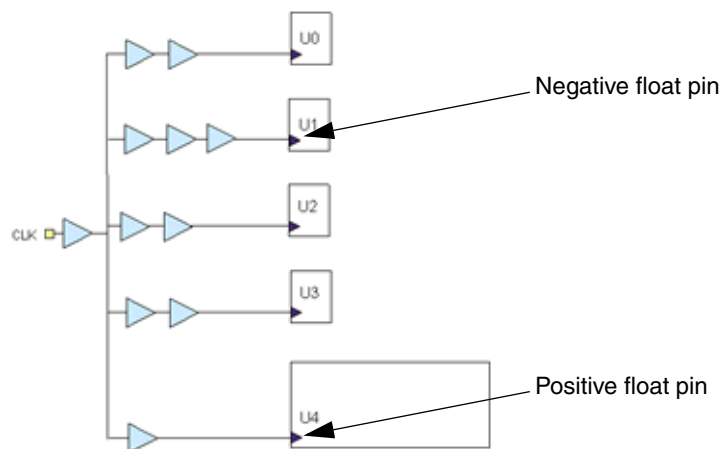
The float pin delay values can be either positive or negative, depending on your timing requirements. To increase the path delay to a pin, specify a negative float pin delay. To decrease the path delay to a pin, specify a positive float pin delay.

For example, the following float pin exceptions result in a clock tree similar to that shown in [Figure 7-5](#):

```
# Specifying a negative float pin
icc_shell> set_clock_tree_exceptions -float_pins U1/CLK \
        -float_pin_max_delay_rise -0.5 -float_pin_max_delay_fall -0.5

# Specifying a positive float pin
icc_shell> set_clock_tree_exceptions -float_pins U4/CLK \
        -float_pin_max_delay_rise 0.5 -float_pin_max_delay_fall 0.5
```

Figure 7-5 Float Pin Timing



For an example of defining float pins, see [“Handling Hard Macro Cells”](#) on page 7-44.

To remove the float pin definition from a pin, use the `remove_clock_tree_exceptions -float_pins` command.

Specifying Stop Pins

Stop pins are the endpoints of the clock tree that are used for delay balancing. During clock tree synthesis, IC Compiler uses stop pins in calculations and optimizations for both design rule constraints and clock tree timing (skew and insertion delay).

The default clock sinks are implicit stop pins. In addition, IC Compiler supports user-defined (or explicit) stop pins. For example, you might define a stop pin to end a branch at an input to a combinational cell or to use an implicit exclude pin as a clock sink.

IC Compiler assigns a phase delay of zero to all stop pins (implicit and explicit) and uses this delay during delay balancing.

To specify a stop pin, use the `set_clock_tree_exceptions -stop_pins` command.

For example, to specify pin U2/A as a stop pin, enter

```
icc_shell> set_clock_tree_exceptions -stop_pins [get_pins U2/A]
```

To remove the stop pin definition from a pin, use the `remove_clock_tree_exceptions -stop_pins` command.

For example, to remove the stop pin definition from pin U2/A, enter

```
icc_shell> remove_clock_tree_exceptions -stop_pins [get_pins U2/A]
```

Specifying Don't Touch Subtrees

In some cases you will want to preserve a portion of an existing clock tree. You need to do this, for example, when two clock networks share part of some clock logic behind a multiplexer. The portion of the clock tree that is preserved is called a don't touch subtree.

To specify a don't touch subtree, specify the root pin of the don't touch subtree by using the `set_clock_tree_exceptions -dont_touch_subtrees` command.

Although IC Compiler does not make any modifications to the don't touch subtree during clock tree synthesis, it does propagate the clock tree attributes and the nondefault routing rules beyond the don't touch subtree. To prevent propagation of the clock attributes and the nondefault routing rules, set the `cts_traverse_dont_touch_subtrees` variable to false.

IC Compiler considers the sinks in the don't touch subtree when balancing clock delays and computing the clock skew.

To remove the don't touch subtree exception from a pin, use the `remove_clock_tree_exceptions -dont_touch_subtrees` command.

Specifying Don't Buffer Nets

In some cases you might be able to improve the results by preventing IC Compiler from buffering certain nets (IC Compiler still performs global prerouting on these nets).

Note:

During clock tree synthesis, the don't buffer nets exception has priority over the clock tree design rule constraints. However, the clock tree specification in the clock tree configuration file has priority over the don't buffer nets exception.

To specify nets that should not be buffered, use the `set_clock_tree_exceptions -dont_buffer_nets` command.

For example, to specify net n1 as a don't buffer net, enter

```
icc_shell> set_clock_tree_exceptions -dont_buffer_nets [get_nets n1]
```

To remove the don't buffer net exception, use the `remove_clock_tree_exceptions -dont_buffer_nets` command.

For example, to remove the don't buffer net exception from net n1, enter

```
icc_shell> remove_clock_tree_exceptions -dont_buffer_nets [get_nets n1]
```

Specifying Don't Size Cells

During clock tree synthesis and optimization, IC Compiler ignores the `dont_touch` attribute on cells and nets. To prevent sizing of cells on the clock path during clock tree synthesis and optimization, you must identify the cells as don't size cells.

To specify cells that should not be sized, use the `set_clock_tree_exceptions -dont_size_cells` command.

For example, to specify cell U1/U3 as a don't size cell, enter

```
icc_shell> set_clock_tree_exceptions -dont_size_cells [get_cells U1/U3]
```

To remove the don't size cell exception, use the `remove_clock_tree_exceptions -dont_size_cells` command.

For example, to remove the don't size cell exception from cell U1/U3, enter

```
icc_shell> remove_clock_tree_exceptions -dont_size_cells [get_cells U1/U3]
```

Specifying the Clock Tree References

IC Compiler uses four clock tree reference lists:

- One for clock tree synthesis
- One for boundary cell insertion
- One for sizing
- One for delay insertion

By default, each clock tree reference list contains all the buffers and inverters in your technology library.

To fine-tune the results, you can restrict the set of buffers and inverters used for one or more of these operations. For example, if your clock tree has too many levels, it could be that the clock tree synthesis references have a low drive strength.

To define a clock tree reference list, use the `set_clock_tree_references` command (or choose Clock > Set Clock Tree References in the GUI). When you define a clock tree reference list, ensure that the buffers and inverters that you specify have a wide range of drive strengths, so that clock tree synthesis can select the appropriate buffer or inverter for each cluster.

Note:

The clock tree synthesis reference list must include at least one inverter, or clock tree synthesis fails. If you are using the default clock tree reference list, you must ensure that your target library contains at least one inverter that does not have a `dont_use` attribute. If you define a clock tree synthesis reference list, you must ensure that it contains at least one inverter.

[Table 7-3](#) shows the options used for generating the reference lists. If you specify a reference list, but do not select any of these options, the specified reference list is used for clock tree synthesis. If you specify a reference list with one or more of these options, the specified clock tree reference lists are created (the clock tree synthesis reference list is not changed).

Table 7-3 Clock Tree Reference List Options

Reference list type	Option
Clock tree synthesis	N/A
Boundary cell insertion	-boundary_cell_only ("Boundary cell only" check box in the GUI)
Sizing	-sizing_only ("Sizing only" check box in the GUI)
Delay insertion	-delay_insertion_only ("Delay insertion only" check box in the GUI)

When you run the `set_clock_tree_references` command, IC Compiler verifies that the cells you specify exist in the target libraries, and it generates a warning message if it cannot find a cell.

Note:

For multicorner-multimode designs, IC Compiler checks only the libraries associated with the clock tree synthesis scenario. You need to ensure that the specified clock references exist in the target library specified for the clock tree synthesis scenario.

When you explicitly include a cell in a clock tree reference list, IC Compiler can use the cell for the task associated with the reference list, even if the cell has a `dont_use` attribute. However, if you set the `dont_use` attribute on a cell after it is included in a clock tree reference list, IC Compiler honors the `dont_use`.

If you issue the `set_clock_tree_references` command multiple times, the new references you specify are added to existing references. References you previously listed but omitted from a later list are not deleted. To delete references, use the `reset_clock_tree_references` command (or choose Clock > Set Clock Tree References in the GUI and click Default).

For example, to create a clock tree synthesis reference list, enter

```
icc_shell> set_clock_tree_references \
  -references {clk1a6 clk1a9 clk1a15 clk1a27}
```

IC Compiler uses this clock tree reference list for all clock trees.

Specifying Clock Tree Synthesis Options

[Table 7-4](#) describes the clock tree synthesis options supported by IC Compiler. Some options can be applied either on all clocks (globally) or on a per-clock basis, while other options can be applied globally only.

To define clock tree synthesis options, use the `set_clock_tree_options` command (or choose Clock > Set Clock Tree Options in the GUI). [Table 7-4](#) summarizes the clock tree synthesis options supported by IC Compiler and the command option or GUI object used to specify them.

Table 7-4 Clock Tree Synthesis Options

Option	Scope	Default	Description
Clock Tree Selection Options			
<code>-clock_trees</code> ("Clock tree" field)	N/A	All clock trees	Specifies the clock trees that the options apply to.
Clock Tree Design Rule Constraints (see "Setting Clock Tree Design Rule Constraints" on page 7-24)			
<code>-max_capacitance</code> ("Max capacitance" field in the "Target and CTO" tab)	global or per-clock	0.6 pF	Specifies the maximum capacitance constraint.
<code>-max_fanout</code> ("Max fanout" field in the "Target and CTO" tab)	global or per-clock	2000	Specifies the maximum fanout constraint.
<code>-max_transition</code> ("Max transition" field in the "Target and CTO" tab)	global or per-clock	0.5 ns	Specifies the maximum transition time constraint.

Table 7-4 Clock Tree Synthesis Options (Continued)

Option	Scope	Default	Description
Clock Tree Timing Goals (see “Setting Clock Tree Timing Goals” on page 7-26)			
-target_skew (“Target skew” field in the “Target and CTO” tab)	global or per-clock	0	Specifies the maximum skew goal.
-target_early_delay (“Target early delay” field in the “Target and CTO” tab)	global or per-clock	0	Specifies the minimum insertion delay goal.
Clock Tree Level Restrictions (see “Setting Level Restrictions” on page 7-26)			
-max_buffer_levels (“Max buffer levels” field in the “Target and CTO” tab)	global or per-clock	20	Specifies the maximum number of clock tree levels.
Clock Tree Routing Options (see “Setting Clock Tree Routing Options” on page 7-27)			
-layer_list (Layers check boxes in the “Routing” tab)	global or per-clock	All routing layers	Specifies the preferred layers for clock tree routing.
-routing_rule (“Routing rule” field in the “Routing” tab)	global or per-clock	Default routing rule	Specifies the nondefault routing rule used for clock tree routing.
-use_default_routing_for_sinks (“Use default routing for sinks at level” check box in the “Routing” tab)	global	Specified routing rule for all nets	Specifies that default routing rules are used for the clock tree levels closest to the clock sink.
Boundary Cell Insertion (see “Inserting Boundary Cells” on page 7-32)			
-insert_boundary_cell (“Insert boundary cell” check box)	global	Disabled	Enables boundary cell insertion.

Table 7-4 Clock Tree Synthesis Options (Continued)

Option	Scope	Default	Description
Clock Tree Clustering Algorithm (see “Selecting the Clock Tree Clustering Algorithm” on page 7-33)			
-ocv_clustering ("OCV clustering" check box)	global	Clustering based on minimum wire length	Selects clustering based on on-chip variation (OCV).
Logic Level Balancing (see “Enabling Logic-Level Balancing” on page 7-34)			
-logic_level_balance ("Logic level balance" check box)	global	Disabled	Enables logic-level balancing.
Embedded Clock Tree Optimizations (see “Specifying Clock Tree Optimization Options” on page 7-36)			
-buffer_relocation ("Buffer relocation" check box)	global or per-clock	Enabled	Enables buffer relocation during optimization.
-buffer_sizing ("Buffer sizing" check box)	global or per-clock	Enabled	Enables buffer sizing during optimization.
-delay_insertion ("Delay insertion" check box)	global or per-clock	Disabled	Enables delay insertion during optimization.
-gate_relocation ("Gate relocation" check box)	global or per-clock	Enabled	Enables gate relocation during optimization.
-gate_sizing ("Gate sizing" check box)	global or per-clock	Disabled	Enables gate sizing during optimization.

Table 7-4 Clock Tree Synthesis Options (Continued)

Option	Scope	Default	Description
Clock Tree Configuration File Options (see “Inserting User-Specified Clock Trees” on page 7-39)			
<code>-config_file_read</code> (“Read file” field in “Configuration files” box)	global	None	Specifies the clock configuration file to read.
<code>-config_file_write</code> (“Write file” field in “Configuration files” box)	global	None	Specifies the name of the clock configuration file that is written after clock tree synthesis.

The `set_clock_tree_options` command is additive; you can run the command multiple times to set different options. If you specify the same option multiple times, the new specification overrides the old specification.

To reset clock tree synthesis options to their default values, use the `reset_clock_tree_options` command (or choose Clock > Set Clock Tree Options in the GUI and click Default).

To see the current settings for the clock tree synthesis options, use the `report_clock_tree -settings` command (or choose Clock > List Clock Tree Options in the GUI).

The following sections provide details for each of the clock tree synthesis options.

Specifying the Clock Tree Synthesis Goals

The optimization goals used for synthesizing the design and the optimization goals used for synthesizing the clock trees might differ. Perform the following steps to ensure that you are using the proper constraints:

- Set the clock tree design rule constraints
- Set the clock tree timing goals

IC Compiler prioritizes the clock tree synthesis optimization goals as follows:

1. Design rule constraints
 - a. Meet maximum capacitance constraint
 - b. Meet maximum transition time constraint
 - c. Meet maximum fanout constraint

2. Clock tree timing goals

- a. Meet maximum skew target
- b. Meet minimum insertion delay target

Setting Clock Tree Design Rule Constraints

IC Compiler supports the following design rule constraints for clock tree synthesis:

- Maximum capacitance (`set_clock_tree_options -max_capacitance`)
If you do not specify this constraint, the clock tree synthesis default is 0.6 pF.
- Maximum transition time (`set_clock_tree_options -max_transition`)
If you do not specify this constraint, the clock tree synthesis default is 0.5 ns.
- Maximum fanout (`set_clock_tree_options -max_fanout`)
If you do not specify this constraint, the clock tree synthesis default is 2000.

You can specify the clock tree design rule constraints for a specific clock (by using the `-clock_trees` option to specify the clock) or for all clocks (by omitting the `-clock_trees` option).

Note:

IC Compiler does not support the specification of per-clock design rule constraints for overlapping clock domains.

In addition to the clock tree design rule constraint values that you specify, IC Compiler also considers the design rule constraint values from the logic library and the design. [Table 7-5](#) summarizes how IC Compiler determines the design rule constraint values used during the design rule fixing stage of clock tree synthesis and optimization. The clock tree synthesis value refers to the value you specified by using the `set_clock_tree_options` command (or the clock tree synthesis default if you did not specify a value).

Table 7-5 Clock Tree Synthesis Design Rule Constraints

Variable Settings			
Design Rule Constraint	Default behavior:	Use library and SDC settings for maximum fanout:	Use only clock tree synthesis settings:
	<code>cts_use_lib_max_fanout = false</code> <code>cts_use_sdc_max_fanout = false</code> <code>cts_force_user_constraints = false</code>	<code>cts_use_lib_max_fanout = true</code> <code>cts_use_sdc_max_fanout = true</code> <code>cts_force_user_constraints = false</code>	<code>cts_force_user_constraints = true</code>
Maximum Capacitance	The minimum value from - The clock tree synthesis value - The logic library - The SDC constraints	The minimum value from - The clock tree synthesis value - The logic library - The SDC constraints	The clock tree synthesis value
Maximum Transition Time	The minimum value from - The clock tree synthesis value - The logic library - The SDC constraints	The minimum value from - The clock tree synthesis value - The logic library - The SDC constraints	The clock tree synthesis value
Maximum Fanout	The clock tree synthesis value	The minimum value from - The logic library - The SDC constraints	The clock tree synthesis value

To see the design rule constraint settings that will be used for clock tree synthesis, generate the clock tree settings reports, as described in [“Generating Clock Tree Reports” on page 7-79](#). Review these values. If any of the values are incorrect for clock tree synthesis, you can override them by setting clock tree design rule constraints.

Note:

The clock tree settings report displays the design rule constraints based on the default behavior; it does not consider the effects of the `cts_force_user_constraints` or `cts_use_lib_max_fanout` variables.

Setting Clock Tree Timing Goals

During clock tree synthesis, IC Compiler considers only the clock tree timing goals. It does not consider the latency (as specified by the `set_clock_latency` command) or uncertainty (as specified by the `set_clock_uncertainty` command).

Note:

IC Compiler can consider the clock latency specification during interclock delay balancing. For more information, see [“Balancing Multiple Clocks” on page 7-52](#).

You can specify the following clock tree timing goals for a clock tree:

- Maximum skew (`set_clock_tree_options -target_skew`)

During optimization, IC Compiler computes the skew value by comparing the arrival times of all clock signals in a clock domain, including those that do not communicate through datapaths (global skew).

If you do not specify a maximum skew value, IC Compiler uses 0 as the maximum skew.

- Minimum insertion delay (`set_clock_tree_options -target_early_delay`)

IC Compiler checks the minimum insertion delay after synthesizing the initial clock tree. If the synthesized clock tree does not meet the specified minimum insertion delay, IC Compiler inserts buffers at the clock root to match the requirement.

If you do not specify a minimum insertion delay value, IC Compiler uses 0 as the minimum insertion delay.

You can specify the clock tree timing goals for a specific clock (by using the `-clock_trees` option to specify the clock) or for all clocks (by omitting the `-clock_trees` option).

Setting Level Restrictions

By default, IC Compiler allows a maximum of 20 levels in each subtree of a clock tree. If you require a different value, use the `set_clock_tree_options -max_buffer_levels` command to specify the maximum number of levels per subtree.

You can specify the maximum level count for a specific clock (by using the `-clock_trees` option to specify the clock) or for all clocks (by omitting the `-clock_trees` option).

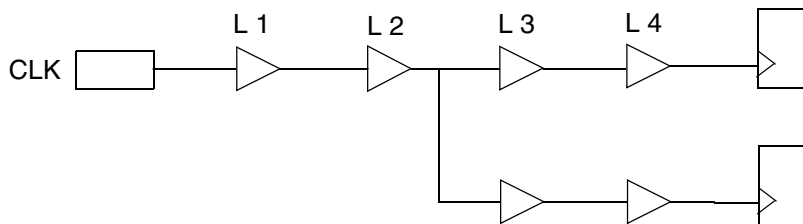
Note:

During clock tree synthesis, the maximum number of levels has priority over the clock tree design rule constraints.

For example, the following command specifies that all subtrees of the clock tree CLK are to have a maximum of four levels as shown in Figure 7-6:

```
icc_shell> set_clock_tree_options -clock_trees CLK -max_buffer_levels 4
```

Figure 7-6 Maximum Clock Buffer Levels



Setting Clock Tree Routing Options

IC Compiler allows you to specify the following options to guide the clock tree routing:

- Which routing rule (type of wire) to use
- Which clock shielding methodology to use
- Which routing layers to use

Specifying Routing Rules

If you do not specify which routing rule to use for clock tree synthesis, IC Compiler uses the default routing rule (default wires) to route the clock trees. To reduce the wire delays in the clock trees, you can use wide wires instead. Wide wires are represented by nondefault routing rules.

Before you can use a nondefault routing rule, the rule must either exist in the Milkyway design library or have been previously defined by using the `define_routing_rule` command. For example, to define the `clk_rule` nondefault routing rule, enter the following command:

```
icc_shell> define_routing_rule clk_rule \  
-widths {M1 0.28 M2 0.28 M3 0.28 M4 0.28 M5 0.28 M6 0.28 M7 0.28 M8 0.84} \  
-spacings {M1 0.28 M2 0.28 M3 0.28 M4 0.28 M5 0.28 M6 0.28 M7 0.28 M8 0.84}
```

To see the current routing rule definitions, run the `report_routing_rules` command.

To set the clock tree routing rule, use the `set_clock_tree_options -routing_rule` command.

You can specify the clock tree routing rule for a specific clock tree by using the `-clock_trees` option to specify the clock or for all clocks by omitting the `-clock_trees` option.

For example, to use the previously defined `clk_rule` nondefault routing rule for routing all clock trees, enter the following command:

```
icc_shell> set_clock_tree_options -routing_rule clk_rule
```

By default, the specified routing rule is used for all nets in the clock tree. However, wide wires are often not required on the nets closest to the clock sinks. To use default wires on the nets connected to the clock sinks and the bottom $n-1$ levels of the clock tree, use the `-use_default_routing_for_sinks` option on the command line.

Note:

If you enable default routing for sinks, it applies to all clock trees. You cannot enable this capability on a per-clock basis. In addition, the default routing applies only to clock sinks connected to flip-flops. Clock sinks connected to macro cells are not affected by this option.

For finer control of the clock tree routing rules, you can specify the routing rules in a clock configuration file. For information about the clock configuration file, see [“Inserting User-Specified Clock Trees” on page 7-39](#). Clock tree routing rules defined in a clock configuration file override those set by using `set_clock_tree_options`.

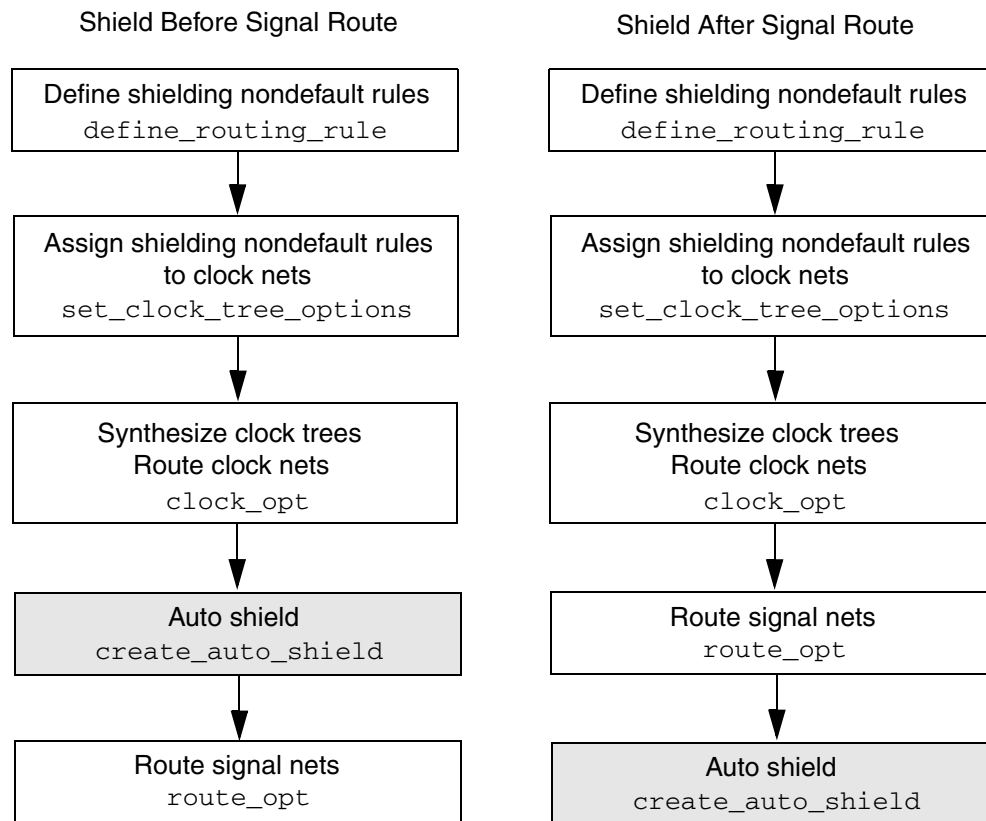
To see the nondefault routing rules defined for the clock trees in your design, run the `report_clock_tree -settings` command.

Shielding Clock Nets

IC Compiler implements clock shielding using nondefault routing rules. You can choose either to shield clock nets before routing signal nets or vice versa. The methodology of shielding clock nets before routing signal nets yields better shielding coverage but can cause more DRC violations during signal net routing compared to the methodology of routing signal nets before shielding clock nets.

[Figure 7-7](#) shows the flow of both clock shielding methodologies.

Figure 7-7 Clock Shielding Methodologies



To define nondefault routing rules for clock shielding, use the `define_routing_rule` command.

The syntax is

```
define_routing_rule rule_name
  [-snap_to_track]
  [-shield_spacings shield_spacings]
  [-shield_widths shield_widths]
```

To assign nondefault routing rules to clock nets, use the `set_clock_tree_options` command.

The syntax is

```
set_clock_tree_options [-clock_tree_name clock_tree_name]
  [-root pin_name]
  [-routing_rule rule_name]
  [-use_default_routing_for_sinks]
```

The nondefault routing rules of clock shielding apply only to nets that are assigned with nondefault routing rules. You can indicate whether to add shielding to leaf pins by using the `-use_default_routing_for_sinks` option.

After assigning nondefault routing rules to clock nets, you can synthesize clock trees and route clock nets using the `clock_opt` command.

IC Compiler router and extractor honor virtual shielding rules. Virtual shielding rules require that the router leaves enough routing resources for shielding to be inserted later and that the extractor considers the shielding effect before shielding metal is inserted. Virtual shielding is supported by virtual routing, global routing, track assignment, and detail routing steps.

To route signal nets, you can use stand-alone commands such as `route_global`, `route_detail`, and `route_auto`, or the `route_opt` command. After routing signal nets, you can add shielding to the routed clock nets using the `create_auto_shield` command. Alternatively, you can choose to add shielding to the routed clock nets before routing signal nets.

The syntax is

```
create_auto_shield [-nets {collection of nets}]
                  [-with_ground]
                  [-ignore_shielding_net_pins]
```

The following sample script shows the methodology for routing signal nets before clock shielding.

```
#Create new nondefault routing rule named SP
define_routing_rule SP \
  -widths {M1 0.14 M2 0.14 M3 0.14 M4 0.14 M5 0.14 M6 0.42} \
  -spacings {M1 0.14 M2 0.42 M3 0.42 M4 0.42 M5 0.42 M6 1.60} \
  -via_cuts {V12 "1x1" V23 "1x1" V34 "1x1" V45 "1x1" V56 "1x1"} \
  -shield_widths {M1 0.14 M2 0.14 M3 0.14 M4 0.14 M5 0.14 M6 0.42} \
  -shield_spacings {M1 0.14 M2 0.42 M3 0.42 M4 0.42 M5 0.42 M6 1.60}

#Assign nondefault routing rule to clock nets
set_clock_tree_options -clock_tree_name CLK \
  -root [get_ports CLK] -routing_rule SP

#Synthesize and route clock trees
clock_opt
set clock_nets [get_nets -of [all_fanout -clock_tree]]

#Route signal nets
route_opt

#Add shielding to clock nets
create_auto_shield -nets $clock_nets
```

Specifying Routing Layers

If you do not specify which routing layers to use for clock tree synthesis, IC Compiler can use any routing layers. For more control of the clock tree routing, you can specify preferred routing layers by using the `set_clock_tree_options -layer_list` command.

You can specify the preferred clock tree routing layers for a specific clock tree by using the `-clock_trees` option to specify the clock or for all clocks by omitting the `-clock_trees` option. For example,

```
icc_shell> set_clock_tree_options -clock_trees CLK1 \  
        -layer_list {metal4 metal5}
```

When you specify the clock tree routing layers by using this command, the specified layers apply to all levels of the clock tree. For finer control of the clock tree routing layers, you can specify the layer constraints in a clock configuration file. For information about the clock configuration file, see [“Inserting User-Specified Clock Trees” on page 7-39](#). Clock tree layer constraints defined in a clock configuration file override those set by using `set_clock_tree_options`.

By default, IC Compiler treats this specification as a soft constraint and can use lower layers for clock tree routing, if necessary. To require clock tree routing on the specified layers, set the following option before running clock tree synthesis:

```
icc_shell> set_droute_options -name hardMinLayerConx -value 1
```

Note:

If you have defined layer constraints on signal nets, you must reset this option to 0 before performing detail routing on the design.

To remove the restrictions on the clock tree routing layers, use the `reset_clock_tree_options -layer_list` command.

Inserting Boundary Cells

When you are working on a block-level design, you might want to preserve the boundary conditions of the block's clock ports (the boundary clock pins). A boundary cell is a fixed buffer that is inserted immediately after the boundary clock pins to preserve the boundary conditions of the clock pin.

To enable boundary cell insertion during clock tree synthesis,

1. Specify the buffers (or inverters) used for boundary cell insertion. (See [“Specifying the Clock Tree References”](#) on page 7-18.)
2. Enable boundary cell insertion by using the `set_clock_tree_options -insert_boundary_cell true` command.

If you enable boundary cell insertion, it applies to all clock trees. You cannot enable boundary cell insertion on a per-clock basis.

Note:

You cannot use boundary cell insertion together with a clock tree configuration file. If you specify both options, IC Compiler disables the boundary cell insertion and generates a warning message.

When boundary cell insertion is enabled, IC Compiler inserts a cell from the buffer insertion clock tree reference list immediately after the boundary clock pins. For multivoltage designs, IC Compiler inserts the boundary cells in the default voltage area.

IC Compiler does not insert a boundary cell when the net is either a don't buffer net or a bidirectional net or when there is a large blockage at the boundary clock pin, which would cause a large distance between the boundary cell and the clock pin.

The boundary cells are fixed for clock tree synthesis; after insertion, IC Compiler does not move or size the boundary cells. In addition, no cells are inserted between a clock pin and its boundary cell.

Selecting the Clock Tree Clustering Algorithm

IC Compiler performs clustering of the clock sinks to minimize wire length. If your design is sensitive to on-chip variation (OCV), IC Compiler can also consider on-chip variation effects during clustering.

If you are using a multicorner design flow, you can reduce skew variation by using RC constraint-based clustering. To use RC constraint-based clustering, you must use a clock configuration file to specify the clock tree structure. For more information about clock configuration files and RC constraint-based clustering, see [“Inserting User-Specified Clock Trees” on page 7-39](#).

Note:

As of version Z-2007.03-SP3, IC Compiler no longer considers the `cts_target_transition` and `cts_target_cap` values during clustering. To control the transition time and maximum capacitance constraints, use `set_clock_tree_options` to define the clock tree synthesis values for these constraints and set the `cts_force_user_constraints` variable to true.

Enabling On-Chip-Variation-Aware Clustering

The optional on-chip-variation-aware clustering algorithm considers the timing constraints between clock sinks to influence clustering. Sinks with timing-critical paths driven by the same gates will be clustered together. To enable the on-chip-variation-aware clustering, use the `set_clock_tree_options -ocv_clustering true` command. When you set this option, it applies to all clock trees in your design.

When you use timing derating (`set_timing_derate` command), on-chip-variation-aware clustering can result in better timing (worst negative slack and total negative slack) with minimal impact on the clock tree skew and insertion. However, using on-chip-variation-aware clustering can increase the runtime and power.

Note:

You cannot use on-chip-variation-aware clustering with logic-level balancing or clock tree configuration files. If you specify `-ocv_clustering` when these options are set, IC Compiler ignores the `-ocv_clustering` option. If you specify logic-level balancing or a clock tree configuration file after setting `-ocv_clustering`, IC Compiler generates an error message and ignores both settings.

Enabling Logic-Level Balancing

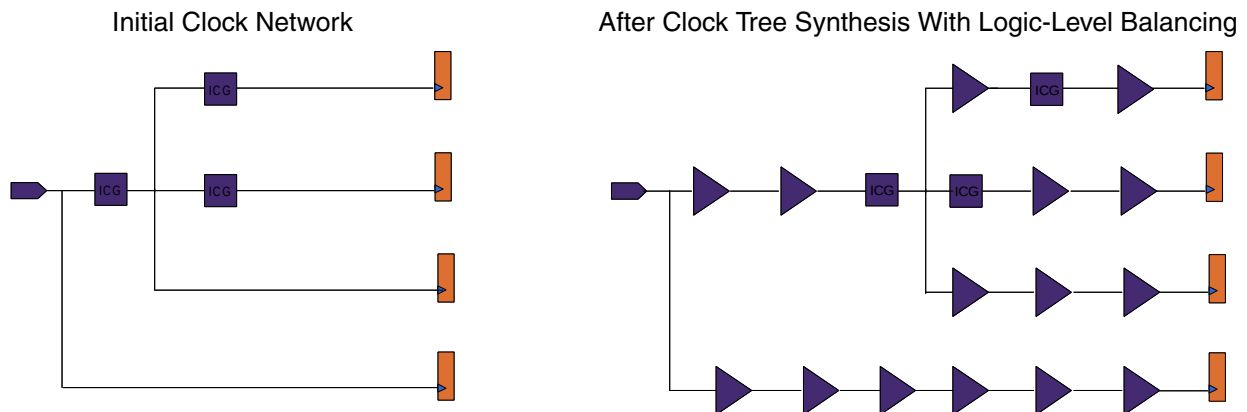
If on-chip variation is an issue for your design, use the logic-level balancing mode.

Note:

If the level count or fanout varies greatly between the branches of the initial clock tree, logic-level balancing might not be able to achieve good clock tree QoR.

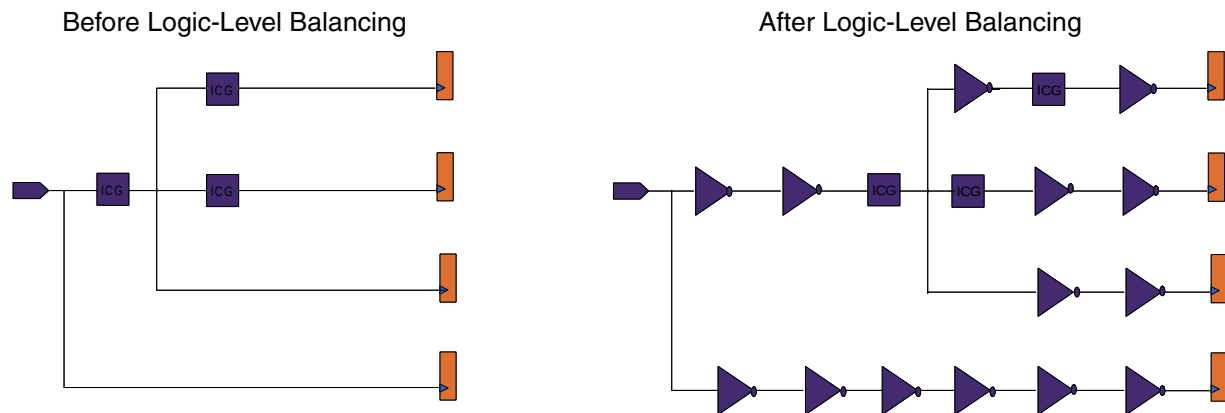
By default, IC Compiler balances the delay in each branch of the clock tree, but it does not consider the number of logic levels in each branch. IC Compiler can take into account both delay and the number of logic levels when balancing the clock tree. This feature is called logic-level balancing. Figure 7-8 shows a clock tree that was synthesized using logic-level balancing.

Figure 7-8 Logic-Level Balancing



Logic-level balancing can use buffers, inverters, or both to balance the logic levels. If you use only inverters for logic-level balancing and the initial clock network does not have balanced logic levels, the generated clock trees might be unbalanced by one level, as shown in Figure 7-9.

Figure 7-9 Logic-Level Balancing Using Inverters

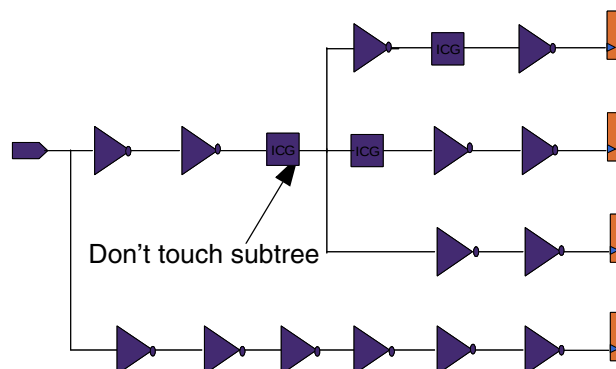


To enable logic-level balancing, use the `set_clock_tree_options -logic_level_balance true` command.

If you enable logic-level balancing, it applies to all clock trees. You cannot enable logic-level balancing on a per-clock basis.

If a clock tree contains a subtree that is not modified during clock tree synthesis (either a don't touch subtree or a subtree within an interface logic model), IC Compiler traces through the subtree to determine the number of logic levels contained in the subtree. IC Compiler considers these logic levels when constructing the clock tree. If the subtree does not have balanced logic levels, IC Compiler generates a warning message and uses the maximum number of levels in the subtree as the number of logic levels for the subtree. For example, for the don't touch subtree shown in Figure 7-10, IC Compiler uses four as the number of logic levels in the don't touch subtree.

Figure 7-10 Logic-Level Balancing With a Don't Touch Subtree



If your design contains hard macros, use the `set_clock_tree_exceptions -float_pin_logic_level` command to specify the number of logic levels within the hard macro. The number of logic levels must be a positive integer. If you do not specify the number of logic levels and IC Compiler cannot derive the number of logic levels, IC Compiler assumes that there are no logic levels within the hard macro.

Note:

When you remove float pin logic level information by using the `remove_clock_tree_exceptions -float_pin_logic_level` command, IC Compiler removes level information for the entire design. It does not remove level information for an individual pin.

After clock tree synthesis finishes, IC Compiler verifies that the clock trees are balanced. If the number of logic levels varies between branches, IC Compiler generates a warning message.

Caution!

If you use logic-level balancing, do not run clock tree optimization with delay insertion. Doing so can unbalance the logic levels. If you use logic-level balancing when running the `clock_opt` command, delay insertion is automatically disabled during the embedded clock tree optimization.

Specifying Clock Tree Optimization Options

IC Compiler optimizes the clock trees by using embedded clock tree optimization during clock tree synthesis (`clock_opt` or `compile_clock_tree` command), preroute clock tree optimization (`clock_opt` or `optimize_clock_tree` command), and postroute clock tree optimization (`optimize_clock_tree` command). During these optimization phases, IC Compiler can perform several optimization tasks, which you can enable or disable by setting the appropriate options.

Note:

IC Compiler uses the clock tree synthesis design rule constraints for all optimization phases, as well as for clock tree synthesis. For information about setting the clock tree synthesis design rule constraints, see [“Setting Clock Tree Design Rule Constraints” on page 7-24](#).

Controlling Embedded Clock Tree Optimization

To set the optimization options for embedded clock tree optimization, use the `set_clock_tree_options` command (or choose Clock > Set Clock Tree Options in the GUI). You can set these options either for a specific clock (by using the `-clock_trees` option) or for all clock trees (by omitting the `-clock_trees` option). [Table 7-6](#) shows the available options and their default values.

Table 7-6 Embedded Clock Tree Optimization Options

Option	Default	Description
<code>-buffer_relocation</code>	true	Optimizes the placement of the buffers and inverters in the synthesized clock trees.
<code>-buffer_sizing</code>	true	Optimizes the sizing of the buffers and inverters in the synthesized clock trees.
<code>-delay_insertion</code>	false	Inserts delays on clock paths to reduce the clock skew while at the same time ensuring that the longest clock path does not change.
<code>-gate_relocation</code>	true	Optimizes the placement of the preexisting gates in the clock trees by moving them closer to the clock sinks. Gates marked as fixed are not moved.
<code>-gate_sizing</code>	false	Optimizes the sizing of the preexisting gates in the clock trees.

Controlling Preroute Clock Tree Optimization

During `clock_opt`, the clock tree optimization phase performs all the optimization tasks listed in [Table 7-7](#). You cannot independently control these tasks for `clock_opt`.

To set the optimization options for stand-alone preroute clock tree optimization, specify the options when you run the `optimize_clock_tree` command (or choose Clock > Optimize Clock Tree in the GUI). [Table 7-7](#) shows the available options and their default values.

Table 7-7 Preroute Clock Tree Optimization Options

Option	Default	Description
<code>-buffer_relocation</code>	true	Optimizes the placement of the buffers and inverters in the synthesized clock trees.
<code>-buffer_sizing</code>	true	Optimizes the sizing of the buffers and inverters in the synthesized clock trees.
<code>-delay_insertion</code>	true	Inserts delays on clock paths to reduce the clock skew, while at the same time ensuring that the longest clock path does not change.
<code>-gate_relocation</code>	true	Optimizes the placement of the preexisting gates in the clock trees by moving them closer to the clock sinks. Gates marked as fixed are not moved.
<code>-gate_sizing</code>	true	Optimizes the sizing of the preexisting gates in the clock trees.

Controlling Postroute Clock Tree Optimization

To set the optimization options for postroute clock tree optimization, specify the options when you run the `optimize_clock_tree` command (or choose Clock > Optimize Clock Tree in the GUI). [Table 7-8](#) shows the available options and their default values.

Table 7-8 Postroute Clock Tree Optimization Options

Option	Default	Description
<code>-buffer_sizing</code>	true	Optimizes the sizing of the buffers and inverters in the synthesized clock trees.
<code>-gate_sizing</code>	true	Optimizes the sizing of the preexisting gates in the clock trees.

Inserting User-Specified Clock Trees

IC Compiler supports the use of a clock configuration file to enable user specification of the clock tree structure. The following sections describe how to

- Read a clock configuration file before clock tree synthesis
- Reduce skew variation by using RC constraint-based clustering
- Save a clock configuration file after clock tree synthesis
- Describe your clock tree structure in a clock configuration file

Reading a Clock Configuration File

If you have a configuration file that describes the desired clock tree structure, run the `set_clock_tree_options -config_file_read config_file` command before running clock tree synthesis. For details about the syntax of the configuration file, see [“Defining the Clock Tree Structure” on page 7-40](#).

Reducing Skew Variation by Using RC Constraint-Based Clustering

If you use a configuration file to specify the clock tree structure for a multicorner design, you can use RC constraint-based clustering to reduce the skew variation across corners. To enable this capability, set the `cts_enable_rc_constraints` variable to true before running clock tree synthesis.

When you enable this capability, IC Compiler uses the RC values to derive maximum delay and maximum skew constraints, which are then used during clustering to reduce the skew variation. Although RC constraint-based clustering reduces skew variation, you should be aware that it can increase the buffer count and runtime. You can reduce the buffer count and runtime impact by relaxing the derived constraints. To relax the constraints, set the `cts_rc_relax_factor` variable to a number greater than 1.0. You can also tighten the constraints (by setting this variable to a number between 0 and 1.0).

Saving a Clock Configuration File

To save the generated clock tree structure in a clock configuration file after clock tree synthesis, run `set_clock_tree_options -config_file_write config_file` command before running clock tree synthesis.

Defining the Clock Tree Structure

Use the following syntax to define the structure for each user-specified clock tree in your design:

```
begin_clock_tree number_of_levels
clock_net net_name [routing_rule rule_name]
[routing_layer_constraints min_layer max_layer]
{buffer_level reference_cell number_of_buffers
 [buffer_level_pin instance/pin]
 [routing_rule rule_name]
 [routing_layer_constraints min_layer max_layer]
 ...}
...
end_clock_tree

begin_clock_tree number_of_levels
```

This statement starts the clock tree structure definition and specifies the number of levels for the clock tree.

You must specify an integer value for this argument. If you specify 0, IC Compiler determines the number of levels (and the number of buffers in each level) for the clock tree.

```
clock_net net_name
```

The *net_name* argument is a string that specifies the name of the clock net. You can specify the clock net name as a regular expression, for example, `clk.*` or `clk[10].*`

```
routing_rule rule_name
```

The *rule_name* argument is a string that specifies the name of the routing rule.

To define a nondefault routing rule that applies to the entire clock tree, insert a `routing_rule` statement after the `clock_net` statement that defines the clock root. If you do not define a nondefault routing rule for the root net, IC Compiler uses the default routing rule.

To define a net-specific nondefault routing rule for a net other than the root net, insert a `routing_rule` statement after the associated `clock_net` statement.

To define a level-specific nondefault routing rule, insert a `routing_rule` statement after the associated `buffer_level` statement.

If you specify a net- or level-specific nondefault routing rule, it overrides the clock tree routing rule (whether it is default or nondefault). If you specify a nondefault clock tree routing rule, you can use the default routing rule on specific nets or levels by specifying `routing_rule default` for that net or level.

If a routing rule has not been explicitly defined for a net or level, IC Compiler determines the routing rule as follows:

- If the clock tree structure is completely specified, IC Compiler uses the clock tree routing rule (whether it is default or nondefault).
- If the clock tree structure is not completely specified because the number of levels is not specified, the routing rule for the previous level is used.
- If the clock tree structure is not completely specified because the number of buffers for a level is not specified, IC Compiler uses the clock tree routing rule (whether it is default or nondefault).

The clock tree routing rules specified in the configuration file override the routing rules specified by using `set_clock_tree_options`.

For more information about nondefault routing rules, see [“Specifying Routing Rules” on page 7-27](#).

`routing_layer_constraints min_layer max_layer`

Specifies the minimum and maximum routing layer to use for clock tree synthesis. You can specify the layers using either the layer names (for example, M2 and M4) or the metal layer numbers (for example, 2 and 4).

To define global layer constraints that apply to the entire clock tree, insert a `routing_layer_constraints` statement after the `clock_net` statement that defines the clock root. If you do not define layer constraints for the root net, IC Compiler can use any routing layer for clock tree routing.

To define net-specific layer constraints for a net other than the root net, insert a `routing_layer_constraints` statement after the associated `clock_net` statement.

To define level-specific layer constraints, insert a `routing_layer_constraints` statement after the associated `buffer_level` statement.

If you specify net- or level-specific layer constraints, they override the clock tree layer constraints. If you define clock tree layer constraints, you can remove these constraints for specific nets or levels by specifying `routing_layer_constraints 0 0` for that net or level. The clock tree layer constraints specified in the configuration file override the routing rules specified by using `set_clock_tree_options`.

For more information about nondefault routing rules, see [“Specifying Routing Layers” on page 7-31](#).

`buffer_level reference_cell number_of_buffers`

Specifies the reference cell and buffer count for each level in the clock tree (one statement per level). The first `buffer_level` statement defines the level closest to the clock source; the last `buffer_level` statement defines the level closest to the clock sink. If you do not define all levels, the specified levels are closest to the clock sink.

The *reference_cell* argument is a string that specifies the buffer or inverter that is used for all clock tree instances at this level.

The *number_of_buffers* argument is an integer value that specifies the total number of buffers or inverters at this level. If you specify 0, IC Compiler determines the number of buffers or inverters at this level.

Note:

If the specification in the clock tree configuration file conflicts with a don't buffer nets exception, the clock tree configuration file has priority.

By default, IC Compiler determines the sink pins connected at each level of the clock tree. For more control, such as when you want a RAM to be placed near the clock root, you can explicitly specify the sink pins by using the `buffer_level_pin` statement.

```
buffer_level_pin instance/pin
```

Specifies the sink pin connections for the associated buffer level.

You define all user-specified clock trees in a single clock configuration file. For any clock tree that is not specified in the clock configuration file, IC Compiler determines the structure for that clock tree.

Complete Specification Without Sink Pins

The following example completely specifies a clock tree that has six levels. Clock tree synthesis does not change the number of levels or the number of buffers in each level.

```
begin_clock_tree 6
clock_net core/clk
buffer_level bufx6 2
buffer_level invx12 4
buffer_level bufx12 7
buffer_level invx12 11
buffer_level bufx12 48
buffer_level bufx12 268
end_clock_tree
```

Complete Specification With Sink Pins

The following example completely specifies a clock tree that has six levels. The sink pins are specified for the first two levels (closest to the clock root) and last level (closest to the clock sinks) of the clock tree. The first level connects to two multiplexers, the second level connects to the RAM, and all other flip-flops are connected at the last level.

```
begin_clock_tree 6
clock_net core/clk
buffer_level bufx6 2
buffer_level_pin top/inst1/inst2mux/u1/A
buffer_level_pin top/inst1/inst3mux/u2/A
buffer_level invx12 4
buffer_level_pin top/inst1/inst_ram/CKB
buffer_level bufx12 7
```

```

buffer_level invx12 11
buffer_level bufx12 48
buffer_level bufx12 268
buffer_level_pin top/inst1/my_reg[5]/CKB
buffer_level_pin top/inst1/my_reg[4]/CKB
...
end_clock_tree

```

Incomplete Specification—Unspecified Levels

The following example defines the bottom four levels of a clock tree but lets IC Compiler determine the structure of the top levels (those closest to the clock root) of the clock tree.

```

begin_clock_tree 0
clock_net core/clk
buffer_level bufx12 7
buffer_level bufx12 11
buffer_level bufx12 48
buffer_level bufx12 268
end_clock_tree

```

Incomplete Specification—Unspecified Buffer Count

The following example defines all six levels of a clock tree but lets IC Compiler determine the number of buffers in the third level of the clock tree.

```

begin_clock_tree 6
clock_net core/clk
buffer_level bufx6 2
buffer_level invx12 4
buffer_level bufx12 7
buffer_level invx12 0
buffer_level bufx12 48
buffer_level bufx12 268
end_clock_tree

```

Handling Specific Design Characteristics

Several design styles might require special considerations during clock tree synthesis. These design styles include

- Hard macro cells
- Preexisting clock trees
- Non-unate gated clocks
- Integrated clock gating (ICG) cells
- Multiple clocks (with balanced skew)

- Hierarchical designs
- Multivoltage designs
- Multimode designs

The following sections describe how to use IC Compiler clock tree synthesis with these design styles.

Handling Hard Macro Cells

The internal delays for a hard macro cell are represented in the cell's timing model. IC Compiler uses the timing model to determine the external clock pins of the hard macro cell and uses these pins as clock sinks. During clock tree synthesis, IC Compiler performs skew balancing and insertion delay minimization up to these external clock pins. No additional specification is required for hard macro cells.

If you do not have a timing model for the hard macro cell or you want to modify the timing characteristics of the hard macro cell, use float pins to specify the timing characteristics of the clock trees internal to the hard macro. You define the timing characteristics by specifying the minimum and maximum insertion delay seen from the float pin to the clock sinks that are internal to the macro. For information about defining float pins, see [“Specifying Float Pins” on page 7-13](#).

For example, assume that timing analysis shows that the delay of the precompiled clock tree from the RAM_block/CLK port to the earliest sink is 0.33 ns, and the delay to the latest endpoint is 0.52 ns.

The following command defines these timing characteristics for IC Compiler clock tree synthesis:

```
icc_shell> set_clock_tree_exceptions -float_pin [get_pins RAM_block/CLK] \
    -float_pin_max_delay_rise 0.52 -float_pin_max_delay_fall 0.52 \
    -float_pin_min_delay_rise 0.33 -float_pin_min_delay_fall 0.33
```

Handling Existing Clock Trees

If your design contains existing clock trees, you must decide how to handle them before you perform clock tree synthesis. By default, IC Compiler keeps an existing clock tree and might modify it. Instead, you can either prevent IC Compiler from modifying the existing clock tree or you can remove the clock tree.

The following sections describe how to

- Identify clock trees synthesized in Astro or a third-party tool
- Preserve all or part of an existing clock tree
- Remove a clock tree

Identifying Existing Clock Trees

You can identify clock trees in your netlist that were created in Astro or a third-party tool and optimize them in IC Compiler. To identify a clock tree, use the `mark_clock_tree -clock_synthesized` command (or choose Clock > Mark Clock Tree in the GUI and select “Clock synthesized”). Specify the startpoint for the clock tree by using the `-clock_trees` option (or the “Clock Tree Pin” field in the GUI). The startpoint can be any port or pin on the clock tree; it does not have to be the clock root. If you do not specify a startpoint, IC Compiler uses the clock roots defined by the `create_clock` and `create_generated_clock` commands.

IC Compiler traverses the clock tree from the specified startpoint and sets the clock tree attributes as if the tree were synthesized. Clock tree traversal continues until it finds an exception pin or a default sink pin.

After running `mark_clock_tree -clock_synthesized`, the imported clock trees are treated just like clock trees synthesized by IC Compiler: the buffers and inverters are fixed; the clock sink cells can only be sized, not moved; the clocks are set to propagated; and the user-defined clock latency values are removed. Similar to using the `cts_fix_clock_tree_sinks` variable to fix clock sink cells after clock tree synthesis, you can use the `-fix_sinks` option when you run `mark_clock_tree` to prevent any modification to the clock sink cells after clock tree synthesis. For more information about the IC Compiler clock tree synthesis process, see [“Implementing the Clock Trees” on page 7-58](#).

Important:

Because the buffers and inverters (and possible clock sink cells) are fixed after running `mark_clock_tree`, you must ensure that the placement is legal before running the `mark_clock_tree` command.

Preserving Portions of an Existing Clock Tree

In some cases you will want to preserve a portion of an existing clock tree. You need to do this, for example, when two clock networks share part of some clock logic behind a multiplexer. The portion of the clock tree that is preserved is called a don't touch subtree. For information about don't touch subtrees, see [“Specifying Don't Touch Subtrees” on page 7-16](#).

Removing Clock Trees

To remove a clock tree, use the `remove_clock_tree` command (or choose Clock > Remove Clock Tree in the GUI).

For example,

```
icc_shell> remove_clock_tree -clock_trees my_clock
```

When IC Compiler removes a clock tree, it traverses from the clock root to each endpoint (stop pin, exclude pin, float pin, or don't touch subtree) and by default, removes all buffers and inverters along those paths, including those with `dont_touch` attributes. If you want to keep buffers and inverters that have a `dont_touch` attribute, specify the `-honor_dont_touch` option when you run `remove_clock_tree` (or select “Honor don't touch” in the GUI). If you want to remove only those buffers and inverters inserted by IC Compiler, specify the `-synopsys_only` option when you run `remove_clock_tree` (or select “Remove only Synopsys CTS added buffers/inverters” in the GUI). In addition, IC Compiler traverses beyond exclude pins, stop pins, and float pins and removes only buffers and inverters inserted by IC Compiler (whether or not you specify `-synopsys_only`). The `dont_touch` attributes on buffers and inverters beyond the exception pins are honored only if you specify `-honor_dont_touch`.

Note:

This command removes only unrouted clock trees. To remove a routed clock tree, you must first remove the clock tree routing.

[Table 7-9](#) shows how clock tree removal is affected by the structure of the clock tree.

Table 7-9 Clock Tree Removal Behavior

Object	Impact on clock tree removal
Boundary cell	Removed.
Cells on don't buffer nets	Preserved.
Don't touch subtree	Preserved.
Fixed cells	Preserved.

Table 7-9 Clock Tree Removal Behavior (Continued)

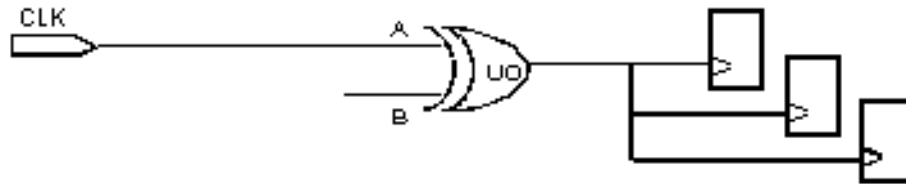
Object	Impact on clock tree removal
Generated clock	Preserved, if generated clock is defined on buffer/inverter pin. Traversal and clock tree removal continue past the generated clock.
Guide buffer	Removed.
Integrated clock gating (ICG) cell	Preserved. Traversal (and clock tree removal) continues past the integrated clock gating cell.
Interface logic model (ILM)	Preserved. Traversal (and clock tree removal) continues past the ILM.
Inverter	Removed in pairs only. If a clock tree contains a single inverter, it is not removed.
Isolation cell	Preserved.
Level shifter	Preserved.
Three-state buffer	Preserved. Traversal (and clock tree removal) stops at the three-state buffer.
Buffer or inverter added beyond exclude pin, stop pin, or float pin	Removed.

Handling Non-Unate Gated Clocks

If the clock-gating logic uses a non-unate cell, such as an XOR or XNOR gate, IC Compiler uses both the positive-unate timing arc and the negative-unate timing arc when tracing the clock path. If this does not correspond to the functional mode of your design, use the `set_case_analysis` command to hold all nonclock inputs of the cell at a constant value. In this way you force the cell into the desired functional mode for timing analysis during clock tree synthesis.

For example, suppose your design has the gating logic shown in [Figure 7-11](#).

Figure 7-11 Non-Unate Gated Clock



To force the XOR gate (U0) into functional mode for timing analysis, enter the following command:

```
icc_shell> set_case_analysis 0 U0/B
```

Handling Integrated Clock Gating Cells

When using clock gating to reduce power, you need to consider the impact not only on power, but also on clock tree QoR. The methodology that you use to insert integrated clock gating cells (ICGs) and optimize the clock tree depend on your power and clock tree QoR goals. For detailed information about inserting integrated clock gating cells, see the clock-gating information in the *Power Compiler User Guide*.

Optimizing for Clock Tree QoR

A balanced clock tree provides the best clock tree timing results. Following these setup recommendations before inserting the integrated clock gating cells results in a more balanced clock tree:

- Use a small maximum clock-gating fanout value.
- Set the `power_cg_all_registers` variable to true.

This variable inserts always-enabled integrated clock gating cells for ungated registers, so that the clock tree is balanced for both gated and ungated registers.

- Set the `power_remove_redundant_clock_gates` variable to false.

This variable prevents Design Compiler from optimizing away the integrated clock gating cells that are used for balancing.

After placing your design, use the `split_clock_net` command to balance the clock tree fanout before running clock tree synthesis.

Note:

You should use `split_clock_net` to balance the clock tree fanout when the clock trees are unbalanced and the design is not meeting your skew targets, or when you want to relax the constraint on the enable pin of the clock gate by delaying the clock arrival time at the clock gate.

The `split_clock_net` command replicates clock-gating cells to balance the clock tree fanout of the clock gates or clock nets specified in the `-objects` option, which can improve the skew and insertion delay for those clock trees.

```
icc_shell> split_clock_net -object clk -gate_sizing -gate_relocation
```

The `split_clock_net` command uses the same algorithm for clustering, which is used during clock tree synthesis. This clustering algorithm improves the skew by using smaller transition time and capacitance constraints, which are based on a technology-dependent computation. This can, however, lead to an increase in clock cell area and power. To force the clustering algorithm to use a more relaxed constraint, use `set_clock_tree_options` to define the setup values for the constraints and set the `cts_force_user_constraints` variable to true. (For information about setting the clock tree design rule constraints, see [“Setting Clock Tree Design Rule Constraints” on page 7-24.](#))

The `split_clock_net` command automatically sizes clock gates replicated to minimize the skew in each cluster.

By default, the `split_clock_net` command

- Does not buffer ungated registers

To buffer ungated registers, specify the `-drive_ungated_registers` option when you run the `split_clock_net` command.

- Replicates only integrated clock gating cells (ICGs)

To replicate any clock gates, specify the `-split_any_cell_type` option when you run the `split_clock_net` command.

- Does not replicate clock gates that have unsynthesized clock subtrees in their fanout

To force replication of these clock gates, specify the `-split_intermediate_level_clock_gates` option when you run the `split_clock_net` command.

The `split_clock_net` command does not replicate the clock-gating cells in the following situations:

- The clock-gating cell has a pin that is defined as a clock (by the `create_clock` command) or as a generated clock (by the `create_generated_clock` command).
- All input pins of the clock-gating cell have a stop pin, exclude pin, or float pin exception.
- The clock-gating cell has a don't buffer net or don't touch subtree exception defined on its input or output pin.
- The clock-gating cell is an integrated clock gating (ICG) cell inside a hierarchical clock-gating wrapper that also contains other cells.
- The clock-gating cell is inside an interface logic model.
- All fanout of the clock-gating cell is inside an interface logic model.

To report the number of clock-gating cells inserted by the `split_clock_net` command, run the `report_clock_tree` command.

Optimizing for Power

When your goal is minimizing power, use a large (or unlimited) maximum clock-gating fanout during insertion of the integrated clock gating cells (ICGs).

After placing your design, perform clock tree power optimization before running clock tree synthesis. To perform clock tree power optimization,

1. Specify the clock tree references (as described in [“Specifying the Clock Tree References” on page 7-18](#)).
2. Specify the clock tree synthesis options (as described in [“Specifying Clock Tree Synthesis Options” on page 7-20](#)) and exceptions (as described in [“Specifying Clock Tree Exceptions” on page 7-9](#)).
3. Enable the power optimization techniques by using the `set_power_options` command:
 - Specify `-clock_gating true` to enable physical optimization of the integrated clock gating cells.
 - Specify `-low_power_placement true` to enable power-aware placement of the integrated clock gating cells and registers.

4. Annotate the switching activity by running the `read_saif` or `set_switching_activity` command.

Important:

If you use a SAIF file to annotate the switching activity, verify that it was generated at the actual clock frequency. If the SAIF file was not generated at the actual clock frequency, the numbers reported for dynamic power might be very small and incorrect.

For more information about annotating the switching activity, see [“Reading Switching Activity Information” on page 6-3](#).

5. Disable automatic group bounding of the gated clocks by setting the `placer_disable_auto_bound_for_gated_clock` variable to true.
6. (Optional) Specify the design constraints used during clock tree power optimization.

You specify the design constraints used during clock tree power optimization in a Tcl script file. You create separate constraint files for clock tree optimization and for post-clock-tree-synthesis optimization.

To specify the name of the constraint file used for clock tree synthesis, use the `-cts_constraint_file` option of the `set_optimize_pre_cts_power_options` command.

To specify the name of the constraint file used for post-clock-tree-synthesis optimization, use the `-psyn_constraint_file` option of the `set_optimize_pre_cts_power_options` command.

7. Perform clock tree power optimization.

To perform stand-alone clock tree power optimization, run the `optimize_pre_cts_power` command (or choose Clock > Optimize Pre-CTS Power in the GUI). To perform clock tree power optimization as part of the `clock_opt` process, use the `-power` option when you run the `clock_opt` command. For more information about the `clock_opt` command, see [“Implementing the Clock Trees” on page 7-58](#).

Note:

When you run the `clock_opt -power` command, IC Compiler also performs leakage optimization and dynamic power optimization, if you have enabled these optimizations by using the `set_power_options` command.

IC Compiler performs the following tasks during clock tree power optimization:

- a. Merges integrated clock gating cells that have the same enable signal.
Only integrated clock gating cells with positive slack are merged; cells with negative slack are not merged.
- b. Performs high-fanout net synthesis to fix design rule violations on the enable signals.
- c. Performs power-aware placement of the integrated clock gating cells and registers.

Note:

If a significant portion of the clock gates are disabled, power-aware placement is not able to effectively optimize the design.

By default, clock tree power optimization ignores `dont_touch` and `size_only` attributes on integrated clock gating cells. To override these defaults, use the `set_optimize_pre_cts_power_options` command (or choose Clock > Set Optimize Pre-CTS Power Options in the GUI).

To honor `dont_touch` attributes, specify the `-honor_dont_touch` option. To honor `dont_size` attributes, specify the `-honor_size_only` option.

To report the current clock tree power optimization options, use the `report_optimize_pre_cts_power_options` command (or choose Clock > Report Pre-CTS Power Optimization Options in the GUI).

Balancing Multiple Clocks

IC Compiler can automatically balance the skew between a group of clocks, either as part of the `clock_opt` process or as a stand-alone process.

Note:

IC Compiler cannot balance skew between a generated clock and other clocks.

This section describes how to set up the interclock delay balancing requirements. For information about performing interclock delay balancing during the `clock_opt` process, see [“Implementing the Clock Trees” on page 7-58](#). For information about running stand-alone interclock delay balancing, see [“Running Interlock Delay Balancing” on page 7-70](#).

To define the interclock delay balancing requirements,

1. Define the buffers used for delay balancing.
2. Define the clock balance groups.
3. Define the interclock delay requirements.

Defining the Delay Balancing Buffers

By default, IC Compiler can use all buffers and inverters in your technology library during interclock delay balancing.

To restrict the set of buffers and inverters used during delay balancing, use the `set_clock_tree_references -delay_insertion_only` command. For more information about specifying clock tree references, see [“Specifying the Clock Tree References” on page 7-18](#).

Defining a Clock Balance Group

By default, IC Compiler balances the delay between all master clocks in the design when you run interclock delay balancing. You can define a clock balance group to restrict the set of clocks considered during delay balancing or to define delay requirements between groups of clocks.

To define a clock balance group, use the `set_inter_clock_delay_options` command (or choose Clock > Set Interlock Delay Options in the GUI). To specify the clock trees in the group, use the `-balance_group` option (or the “Clocks in the group” field in the GUI). You can also assign a name to the balance group by using the `-balance_group_name` option (or the “Group name” field in the GUI). Although assigning a name to the balance group is optional, you must name the clock group to define a relationship between the group and other clock trees in the design.

Defining the Interlock Delay Requirements

By default, IC Compiler has a goal of zero delay offset between clocks. If you have different requirements, you can specify them by using the following methods:

- Specifying a delay offset between clock trees or clock groups
- Using the specified clock latency for each clock

You should not specify multiple interlock delay requirements (for example, specifying both a clock balance group and a delay offset) for a given clock. If a clock has multiple requirements, IC Compiler uses the following precedence when determining the interlock delay requirements:

1. Balancing delays within a clock balance group
2. The delay offset requirements
3. The target insertion delay
4. The SDC clock latency specification, if honored
5. The longest clock network delay

Specifying the Delay Offset Requirements. You specify the delay offset requirement by specifying the following options with the `set_inter_clock_delay_options` command:

- `-delay_offset` (or “Delay offset” field in the GUI)
- `-offset_to` (or “Offset to” field in the GUI)
- `-offset_from` (or “Offset from clock” in the GUI)
- `-offset_from_group` (or “Offset from clock group” field in the GUI)

For example, assume that `clk1` has a delay of 500, `clk2` has a delay of 700, and `clk3` has a delay of 100. If you want to balance the delay between `clk1` and `clk2`, and have `clk3` offset by 200, enter the following commands:

```
icc_shell> set_inter_clock_delay_options \  
-balance_group { clk1 clk2 } -balance_group_name grp1  
icc_shell> set_inter_clock_delay_options -delay_offset -200 \  
-offset_to clk3 -offset_from_group grp1
```

The result is a delay of 700 for `clk1` and `clk2` (the largest of the delays in the balance group) and a delay of 500 (700-200) for `clk3`.

As another example, assume that `clk1` has a delay of 100 and `clk2` has a delay of 200. If you want a delay offset of 200 between `clk1` and `clk2`, enter the following command:

```
icc_shell> set_inter_clock_delay_options -delay_offset 200 \  
-offset_to clk1 -offset_from clk2
```

In this case, the result is a delay of 400 for `clk1` (the `clk2` delay plus the offset delay) and a delay of 200 for `clk2`.

Specifying the Target Insertion Delay. To specify the target insertion delay for a clock, specify the clock by using the `-target_delay_clock` option (or “Target delay clocks” in the GUI) and specify the delay value by using the `-target_delay_value` option (or the “Target delay value” field in the GUI).

When you specify a target insertion delay for a clock, IC Compiler increases the insertion delay of the specified clock if it is less than the specified value. If the insertion delay of the specified clock is greater than the target insertion delay, no delay balancing is performed on that clock.

For example, if you want `clk1` to have a delay of 100, enter the following command:

```
icc_shell> set_inter_clock_delay_options \  
-target_delay_clock clk1 -target_delay_value 100
```

If the delay on clk1 is less than 100, IC Compiler increases the delay to 100. If the delay on clk1 is greater than 100, IC Compiler does not perform delay balancing on clk1.

Using the Clock Latency Specification. The clock latency is specified by using the SDC `set_clock_latency` command. By default, IC Compiler ignores the clock latency during interclock delay balancing. To honor the SDC clock latency specification, use the `-honor_sdc` option (or select “Honor latency defined in SDC” in the GUI).

Hierarchical Designs Using Interface Logic Models

You can use interface logic models (ILMs) to increase the capacity and reduce the runtime for top-level clock tree synthesis. Before creating ILMs for use with top-level clock tree synthesis, you must perform clock tree synthesis on the blocks.

During clock tree synthesis and optimization, IC Compiler

- Identifies any ILMs inside a clock tree

When a clock defined at the top level goes through an ILM, IC Compiler inserts guide buffers before the ILM clock input pin and after the ILM clock output pins. The nets between the input guide buffer and output guide buffers are marked as don't buffer nets.

- Honors clocks or generated clocks defined on an ILM port or a pin internal to the ILM

When a clock is defined on an ILM input port or in an ILM, IC Compiler inserts guide buffers after the ILM clock output pins. Clock nets within the ILM (up to the guide buffers) are marked as don't buffer nets.

- Times the clock subtrees inside the ILM to calculate the phase and transition delays for the ILM

IC Compiler uses the timing information for the clock trees within the ILM to perform skew balancing and insertion delay minimization up to the ILM clock input pins and beyond the ILM clock output pins.

If there are multiple subtrees after an ILM, IC Compiler synthesizes each subtree independently and does not balance the insertion delay between them, which can result in large skew between them. To reduce this skew, run the `optimize_clock_tree` command after performing clock tree synthesis.

- Honors explicit stop pins, exclude pins, and sink pins on an ILM port or inside an ILM

For more information about interface logic models, see [Chapter 11, “Interface Logic Modeling.”](#)

Multivoltage Designs

Clock tree synthesis and optimization are voltage-area-aware. When running clock tree synthesis on multivoltage designs,

- Sink pins are separated and clustered by voltage area so that clock subtrees are built for each voltage area.
- A guide buffer is inserted for the set of sink pins for each voltage area to ensure that any subsequent levels of clustering do not mix pins from different voltage areas.
- Buffers are not inserted between an isolation cell and the shut-down power domain boundary.
- Dual-power always-on clock cells can be inserted or removed as needed on always-on paths in the shut-down or powered-up power domain.

After the clock subtrees are built for each voltage area, clock tree synthesis can proceed in the usual manner, joining the subtrees at the root of the clock net. In addition to the synthesis of the initial clock tree, the preceding behaviors (listed above) are honored by all clock tree optimization techniques, such as buffer relocation, buffer sizing, gate relocation, gate sizing, and delay insertion.

For more information about working with multivoltage designs, see [Chapter 13, “Multivoltage Design Flow.”](#)

Multicorner-Multimode Designs

To perform clock tree synthesis and optimization in a multicorner-multimode design, you must specify which scenario to use for clock tree synthesis and optimization by using the `set_cts_scenario` command. To see the current clock tree synthesis scenario, run the `get_cts_scenario` command.

For more information about multicorner clock tree optimization, see [“Performing Multicorner Clock Tree Optimization” on page 7-71.](#)

For details about working with multicorner-multimode designs, see [Chapter 14, “Using Multicorner-Multimode Technology.”](#)

Verifying the Clock Trees

Before you synthesize the clock trees, use the `check_clock_tree` command to verify that the clock trees are properly defined.

```
icc_shell> check_clock_tree -clocks my_clk
```

If you do not specify the `-clocks` option, IC Compiler checks all clocks in the current design.

The `check_clock_tree` command checks for the following issues:

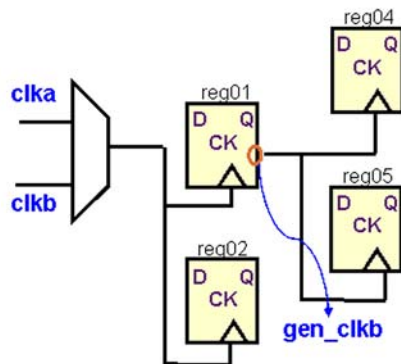
- Hierarchical pin defined as a clock source
- Generated clock without a valid master clock source

A generated clock does not have a valid master clock source in the following situations:

- The master clock specified in `create_generated_clock` does not exist
- The master clock specified in `create_generated_clock` does not drive the source pin of the generated clock
- The source pin of the generated clock is driven by multiple clocks, and some of the master clocks are not specified with `create_generated_clock`.

For example, in [Figure 7-12](#), the `reg01/Q` pin is driven by both `clka` and `clkb`. If only `clkb` is specified as a master clock in a `create_generated_clock` command, `gen_clkb` does not have a valid master clock source.

Figure 7-12 Generated Clock With Invalid Master Clock Source



- Clock (master or generated) with no sinks
- Looping clock
- Cascaded clock with an unsynthesized clock tree in its fanout

- Multiple-clocks-per-register propagation not enabled, but the design contains overlapping clocks
- Ignored clock tree exceptions
- Stop pin or float pin defined on an output pin
- Buffers with multiple timing arcs used in clock tree references
- Situations causing empty buffer list

Each message generated by the `check_clock_tree` command has a detailed man page that describes how to fix the identified issue. Fixing these issues can improve the clock tree and timing QoR.

To see the current clock tree definition, generate the clock tree settings and exceptions reports, as described in [“Generating Clock Tree Reports” on page 7-79](#).

Implementing the Clock Trees

The recommended process for implementing the clock trees in your design is to use the `clock_opt` command, which performs clock tree synthesis, clock tree optimization, and incremental physical optimization. This process results in a timing optimized design with fully implemented clock trees.

Note:

Before implementing the clock trees, save your design. This allows you to refine the clock tree synthesis goals and rerun clock tree synthesis with the same starting point, if necessary.

By default, IC Compiler uses the following naming convention for buffers and inverters inserted during clock tree synthesis

reference_GxByIz

where *reference* is the library reference cell of the buffer or inverter, *x* is the gate level, *y* is the buffer level, and *z* is the instance count. To more easily locate the inserted buffers and inverters in your netlist, you can add a prefix to the instance names by setting the `cts_instance_name_prefix` variable. Similarly, you can add a prefix to any nets inserted during clock tree synthesis by setting the `cts_net_name_prefix` variable.

To perform clock tree synthesis, clock tree optimization, and incremental physical optimization, use the `clock_opt` command (or choose Clock > Core CTS and Optimization in the GUI).

By default, IC Compiler ignores the `dont_touch` attribute on cells and nets during clock tree synthesis and clock tree optimization. To prevent sizing of cells during clock tree synthesis and clock tree optimization, use the `set_clock_tree_exceptions -dont_size_cells` command.

By default, the `clock_opt` command uses virtual routing throughout the clock tree synthesis and optimization process to estimate the wire delay and capacitance. For better correlation with postroute timing, you can use the integrated clock global router to estimate the wire delay and capacitance during the clock tree optimization and interclock delay balancing phases. To enable the integrated clock global router for these phases, use a high effort when you run the `clock_opt` command.

[Table 7-10](#) describes the options you can specify for the `clock_opt` command.

Table 7-10 Clock Tree Synthesis and Optimization Options

Command option	GUI object	Description
N/A	"CTS, clock routing, extraction, optimization and fix hold time violations" radio button	Selects the default <code>clock_opt</code> behavior.
N/A	"Allow cell insertion and deletion" radio button	Selects the default <code>clock_opt</code> clock tree optimization behavior.
<code>-area_recovery</code>	"Recover area" check box	Enables area recovery during placement and timing optimization.
<code>-cts_effort</code>	"CTS effort" check box	Selects clock tree synthesis effort: low (default) or high.
<code>-fix_hold_all_clocks</code>	"Fix hold time violation for all clocks" check box	Fix hold time violations.
<code>-in_place_size_only</code>	"Minimal ECO placement changes" radio button	Restrict placement and timing optimization to gate sizing only when there is enough space. You cannot use this option with <code>-size_only</code> . For more information, see "Using Physical Optimization" on page 4-18.
<code>-inter_clock_balance</code>	"Balance interclock delay" check box	Enables interclock delay balancing.
<code>-no_clock_route</code>	"Route clock nets" check box (uncheck for <code>-no_clock_route</code>)	Do not perform clock routing.

Table 7-10 Clock Tree Synthesis and Optimization Options (Continued)

Command option	GUI object	Description
-only_cts	“CTS only (CTS, CTO, clock routing)” radio button	Do not perform extraction, optimization, or hold time fixing.
-only_hold_time	“Fix hold timing only after CTS” check box	Perform only hold time fixing. (Do not perform clock tree synthesis, clock tree optimization, clock routing, extraction, or optimization.)
-only_psyn	“Incremental optimization only (clock routing, extraction, optimization and fix hold time)” radio button	Do not perform initial gate upsizing, clock tree synthesis, or clock tree optimization.
-operating_condition	“Operating condition” list box	Specifies the operating condition to use for clock tree synthesis and optimization: min, max, or min_max.
-optimize_dft	“Reorder scan cells” check box	Enables scan chain optimization. For more information about scan designs in IC Compiler, see Chapter 5, “Design for Test.”
-power	“Power optimization” check box	Perform the power optimizations enabled by <code>set_power_options</code> . For more information, see “Handling Integrated Clock Gating Cells” on page 7-48 and Chapter 6, “Power Optimization.”
-size_only	“Sizing changes only” radio button	Restrict placement and timing optimization to gate sizing only. You cannot use this option with <code>-in_place_size_only</code> . For more information, see “Using Physical Optimization” on page 4-18 .
-update_clock_latency	“Update real and virtual clock latencies” check box	Enables updating of clock latency values.

The `clock_opt` command does the following:

1. (Optional) Performs clock tree power optimization

To perform clock tree power optimization during the `clock_opt` process, enable physical optimization of the integrated clock gating cells and power-aware placement, as described in [“Optimizing for Power” on page 7-50](#), and use the `-power` option of the `clock_opt` command.

2. Synthesizes the clock trees

Before implementing the clock trees, IC Compiler upsizes, and possible moves, the existing clock gates, which can improve the quality of results (QoR) and reduce the number of clock tree levels.

Note:

To prevent the upsizing of existing clock gates before clustering, set the `cts_prechts_upsize_gates` variable to false. To prevent the moving of existing clock gates before clustering, set the `cts_move_clock_gate` variable to false.

In addition, IC Compiler might move the existing gates, including integrated clock gating (ICG) cells, when this could improve QoR. To prevent IC Compiler from moving existing gates, including integrated clock gating cells, before clustering, set the `cts_move_clock_gate` variable to false.

IC Compiler builds clock trees that meet the clock tree design rule constraints, while balancing the loads and minimizing the clock skew. In addition, IC Compiler optimizes the clock paths beyond exclude pins, stop pins, and float pins to fix any design rule constraint violations.

Note:

Optimization is not performed on don't buffer nets or inside interface logic models (ILMs).

By default, the clock sink cells might be moved or sized during the legalization and optimization steps that occur after clock tree synthesis. To prevent any modification to the clock sink cells after clock tree synthesis, set the `cts_fix_clock_tree_sinks` variable to true. Note that fixing the clock sinks can impact the timing QoR.

You can also run clock tree synthesis as a stand-alone process, using the `compile_clock_tree` command, as described in [“Performing Clock Tree Synthesis” on page 7-65](#).

3. Optimizes the clock trees

During clock tree optimization, IC Compiler uses the optimization techniques, such as buffer relocation, buffer sizing, delay insertion, gate sizing, and gate relocation, to further improve the skew.

Note:

During clock tree optimization, IC Compiler ignores the `dont_touch` attribute on cells and nets. To prevent sizing of cells during clock tree optimization, use the `set_clock_tree_exceptions -dont_size_cells` command.

You can also run clock tree optimization as a stand-alone process, using the `optimize_clock_tree` command, as described in [“Performing Clock Tree Optimization” on page 7-67](#).

4. (Optional) Performs interclock delay balancing

To perform interclock delay balancing during the `clock_opt` process, define the interclock delay balancing requirements, as described in [“Balancing Multiple Clocks” on page 7-52](#), and use the `-inter_clock_balance` option of the `clock_opt` command.

Note:

IC Compiler performs interclock delay balancing by performing delay insertion at the clock root. If the clock root net has a don't buffer net exception, IC Compiler cannot perform interclock delay balancing.

If the clock root is defined as a port of a pad cell, the delay insertion is performed on the net driven by the pad cell.

You can also run interclock delay balancing as a stand-alone process, using the `balance_inter_clock_delay` command, as described in [“Running Inter-clock Delay Balancing” on page 7-70](#).

5. Performs detail routing of the clock nets

You can also perform detail routing of the clock nets as a stand-alone process, using the `route_group -all_clock_nets` command.

To prevent routing of the clock nets, use the `-no_clock_route` option of the `clock_opt` command.

6. Performs RC extraction of the clock nets and computes accurate clock arrival times

7. (Optional) Adjusts the I/O timing

To adjust the input and output delay based on the actual clock arrival times, use the `-update_clock_latency` option of the `clock_opt` command. IC Compiler uses the adjusted input and output delays during placement and timing optimization.

You can also update the I/O timing as a stand-alone process, using the `update_clock_latency` command, as described in [“Adjusting the I/O Timing” on page 7-70](#).

8. (Optional) Optimizes the scan chains

To optimize the scan chains by reordering the chains to minimize the number of buffer crossings in the scan chain, use the `-optimize_dft` option of the `clock_opt` command.

For more information about scan designs in IC Compiler, see [Chapter 5, “Design for Test.”](#)

9. Fixes the placement of the clock tree buffers and inverters

10. Performs placement and timing optimization

If you specify `-update_clock_latency`, IC Compiler uses the adjusted input and output delays during placement and timing optimization. IC Compiler uses propagated arrival times for all clock sinks.

You can customize the placement and timing optimization process by specifying the following options: `-area_recovery`, `-in_place_size_only`, and `-size_only`. You can perform leakage optimization and `-power`. For more information about these options, see [Table 7-10 on page 7-59](#).

You can also run only placement and timing optimization as a stand-alone process, using the `psynopt` command. For more information about the `psynopt` command, see [“Using Physical Optimization” on page 4-18](#).

To prevent placement and timing optimization, use the `-only_cts` option of the `clock_opt` command.

To run only placement and timing optimization (and not clock tree synthesis, clock tree optimization, or clock tree routing), use the `-only_psyn` option of the `clock_opt` command.

11. (Optional) Performs power optimization

You can perform leakage power optimization and dynamic power optimization during the `clock_opt` process by enabling the selected optimizations with the `set_power_options` command and using the `-power` option of the `clock_opt`

command. To enable leakage power optimization, use the `set_power_options -leakage` command. To enable dynamic power optimization, use the `set_power_options -dynamic` command.

12.(Optional) Fixes hold time violations

To fix hold time violations during the `clock_opt` process, use the `-fix_hold_all_clocks` option of the `clock_opt` command.

After running the `clock_opt` command, analyze the results as described in [“Analyzing the Clock Tree Results” on page 7-78](#).

Stand-Alone Clock Tree Synthesis Capabilities

Using the `clock_opt` command is the recommended method for performing clock tree synthesis and optimization with IC Compiler. However, in cases where finer control is required, IC Compiler also provides the following stand-alone clock tree synthesis capabilities:

- Clock tree power optimization
- Clock tree synthesis
- High-fanout net synthesis
- Clock tree optimization
- Inter-clock delay balancing
- I/O timing adjustment

The script in [Example 7-1](#) provides an example of performing clock tree synthesis and optimization by using the stand-alone capabilities. The following sections provide details about these capabilities.

Example 7-1 Clock Tree Synthesis and Optimization Using Stand-Alone Capabilities

```
optimize_pre_cts_power
compile_clock_tree
optimize_clock_tree
balance_inter_clock_delay
route_group -all_clock_nets
update_clock_latency
set_fix_hold [all_clocks]
psynopt -area_recovery -power
```


Performing Clock Tree Power Optimization

For information about performing clock tree power optimization, see [“Optimizing for Power” on page 7-50](#).

Performing Clock Tree Synthesis

Clock tree synthesis is the process of implementing the clock trees based on your requirements. Clock tree synthesis is performed during the `clock_opt` process (see [“Implementing the Clock Trees” on page 7-58](#)) and can also be run as a stand-alone process.

IC Compiler clock tree synthesis is blockage-aware by default. The blockage-aware algorithm avoids routing and placement blockages to reduce DRC violations in designs with complex floorplans. Furthermore, it implements clock trees with minimum clock insertion delay, small clock skew, low buffer count, and small clock cell area to produce the best quality of results (QoR).

During clock tree synthesis, IC Compiler

- Upsizes and moves the existing clock gates, which can improve the QoR and reduce the number of clock tree levels.

Note:

To prevent upsizing of specific cells during this process, use the `set_clock_tree_exceptions -dont_size_cells` command.

- Inserts buffers and inverters to build clock trees that meet the clock tree design rule constraints, while balancing the loads and minimizing the clock skew.
- Fixes DRC violations beyond clock exceptions if the `cts_fix_beyond_clock_exceptions` variable is set to true (the default).
- Builds a blockage map infrastructure per voltage area to identify whether a location is blocked for routing or placement, so the legalizer can move buffers to the nearest unblocked locations toward clock sources
- Locates the shortest blockage-avoiding route path from a start point to an end point with minimum delay to prevent DRC violations.

If your design has logical hierarchy, IC Compiler uses the lowest common parent of a buffer's fanout pins to determine where to insert the buffers.

- If the lowest common parent is not the top level of the design, the buffer is inserted in the lowest common parent.
- If the lowest common parent is the top level of the design, the buffer is inserted in the block that contains the driving pin of the buffer.

IC Compiler adds new ports to the subdesigns where needed. The ports are added such that a minimal number of new ports are added.

To perform stand-alone clock tree synthesis, use the `compile_clock_tree` command (or choose Clock > Compile Clock Tree in the GUI and specify the clock trees in the “Clock tree names” field).

Note:

If you compile one clock at a time, be aware that the order in which you compile the clocks can affect the clock tree QoR. For best results, compile the most critical clock first.

High-Fanout Net Synthesis

You can use the `compile_clock_tree` command to perform high-fanout net synthesis by using the `-high_fanout_net nets_or_driving_pins` option (or by choosing Clock > Compile Clock Tree in the GUI and specifying the high-fanout nets in the “High fanout nets” field).

Note:

In a single `compile_clock_tree` run, you can perform either high-fanout net synthesis (by specifying the `-high_fanout_net` option) or clock tree synthesis (by specifying no clock names or by specifying the clock trees with the `-clock_trees` option); you cannot perform both tasks in a single run.

When you use `compile_clock_tree -high_fanout_net` to perform high-fanout net synthesis, the result is a balanced buffer tree (called a *high-fanout tree*), which is similar to a clock tree. When you use the `create_buffer_tree` command to perform high-fanout net synthesis, the resulting buffer tree might not be balanced.

The `compile_clock_tree` command performs high-fanout net synthesis by balancing the arrival times from the drivers of the nets specified in `-high_fanout_net` to the fanouts of those nets. High-fanout net synthesis does not traverse through preexisting gates on the high fanout net, nor does it support the use of clock tree exceptions.

Important:

If a clock tree exception exists on any fanout pin of a high-fanout net, high-fanout net synthesis generates an error message and fails. You must remove the clock tree exceptions and rerun high-fanout net synthesis.

By default, high-fanout clock tree synthesis can use any of the buffers or inverters in the library. To restrict the set of buffers or inverters used by high-fanout clock tree synthesis, use the `set_clock_tree_references` command, as described in [“Specifying the Clock Tree References” on page 7-18](#).

High-fanout clock tree synthesis determines the clock tree design rules in the same way as standard clock tree synthesis. To define the clock tree design rules, use the `set_clock_tree_options` command, as described in [“Setting Clock Tree Design Rule Constraints” on page 7-24](#).

By default, high-fanout clock tree synthesis uses the rising edge to determine the skew and arrival times. To use the falling edge instead, use the `-sync_phase fall` option when you run `compile_clock_tree`. To use both edges, use the `-sync_phase both` option when you run `compile_clock_tree`.

When you perform high-fanout clock tree synthesis, neither the endpoints nor the inserted buffers are fixed after high-fanout net synthesis. This is to allow `psynopt` to optimize the timing of the high-fanout trees.

To report the skew and path delay of the synthesized high-fanout net, use the `report_clock_tree -high_fanout_net pins_or_nets` command. You can use the following `report_clock_tree` options together with the `-high_fanout_net` option: `-summary`, `-structure`, `-level_info`, `-drc_violators`, `-operating_condition`, and `-nosplit`. All other `report_clock_tree` options are not supported with `-high_fanout_net`.

High-fanout clock tree synthesis has the following limitations:

- High-fanout clock tree synthesis does not support multicorner or multimode designs.
- High-fanout clock tree synthesis does not insert level shifters or isolation buffers in multivoltage designs. You must insert the level shifters and isolation buffers before running high-fanout clock tree synthesis.
- You cannot use `optimize_clock_tree` to optimize the high-fanout trees. Use the `psynopt` command instead.

Performing Clock Tree Optimization

Clock tree optimization improves the clock skew and clock insertion delay by applying additional optimization iterations. Clock tree optimization is performed during the `clock_opt` process (see [“Implementing the Clock Trees” on page 7-58](#)) and can also be run as a stand-alone process before clock routing, after clock tree routing, or after detail routing. Typically, you would perform stand-alone clock tree optimization when timing optimization or incremental placement disturbs the clock skew or clock insertion delay.

By default, clock tree optimization uses virtual routing to estimate the wire delay and capacitance. You specify a low-effort optimization to enable the virtual router by using either of the following commands:

```
icc_shell> clock_opt -cts_effort low
or
icc_shell> optimize_clock_tree -effort low
```

For better correlation with postroute timing, you can enable the integrated clock global router by using a high effort during clock tree optimization. Note that a high effort performs more optimization steps but increases runtime as compared to a low effort. Additionally, the integrated clock global router can achieve better before and after clock routing correlation. To enable the integrated clock global router, you use either of the following commands:

```
icc_shell> clock_opt -cts_effort high
or
icc_shell> optimize_clock_tree -effort high
```

For best QoR, the recommended flow is to specify a high effort during clock tree optimization and to use the `-clock_arnoldi` option whenever possible. However, if runtime is a concern, use a low effort instead.

To run multicorner clock tree optimization, you use the `-enable_multicorner` option to specify at least one corner before running either `clock_opt` or `optimize_clock_tree`. For multicorner optimization, IC Compiler automatically optimizes using a high effort, regardless of what effort level you specify, and it enables the integrated clock global router and sets the `-clock_arnoldi` option, as shown in the following example:

```
icc_shell> set_clock_tree_optimization_options -enable_multicorner all
icc_shell> clock_opt
or
icc_shell> set_clock_tree_optimization_options -enable_multicorner all
icc_shell> optimize_clock_tree
```

To perform stand-alone clock tree optimization, use the `optimize_clock_tree` command (or choose Clock > Optimize Clock Tree in the GUI).

IC Compiler provides the following incremental optimization capabilities: buffer relocation (`-buffer_relocation` option), buffer sizing (`-buffer_sizing` option), delay insertion (`-delay_insertion` option), gate relocation (`-gate_relocation` option), and gate sizing (`-gate_sizing` option).

Note:

During clock tree optimization, IC Compiler ignores the `dont_touch` attribute on cells and nets. To prevent sizing of cells during clock tree optimization, use the `set_clock_tree_exceptions -dont_size_cells` command.

By default, the `optimize_clock_tree` command assumes that the clock trees in your design are not routed. For unrouted clock trees, `optimize_clock_tree` can perform any of the incremental optimization capabilities (default behavior is to perform all incremental optimizations).

If your clock trees are routed, you must explicitly specify the routing stage of the clock trees by using the `-routed_clock_stage` option of the `optimize_clock_tree` command. [Table 7-11](#) shows the supported routing stage keywords.

Table 7-11 Routing Stage Keywords

Value	Description
none	The clock trees are not routed.
global	The clock trees have global routing.
track	The clock trees have track assignment.
detail	The clock trees have detail routing.

For routed clock trees, `optimize_clock_tree` can perform only the sizing optimizations (default behavior is to perform both buffer sizing and gate sizing).

To run a subset of the available optimizations, you must explicitly specify the optimizations that you want. If you specify options that are not compatible with the routing status of your design, IC Compiler generates an error message.

For example, to perform only gate sizing on a routed design, enter the following command:

```
icc_shell> optimize_clock_tree -gate_sizing
```

After optimizing postroute clock trees, the `optimize_clock_tree` command performs engineering change order (ECO) routing and extraction. The type of ECO routing performed depends on the routing stage of the clock trees.

- For global routed clock trees, IC Compiler performs incremental global routing.
- For track assigned clock trees, IC Compiler performs detail routing (utilizing dangling wires).
- For detail routed clock trees, IC Compiler performs detail routing (utilizing dangling wires) and performs two search and repair loops. To change the number of search and repair loops, use the `-search_repair_loop` option of the `optimize_clock_tree` command.

To disable ECO routing during `optimize_clock_tree`, specify the `-no_clock_eco_route` option.

Running Interlock Delay Balancing

Interlock delay balancing balances the skew between a group of clock trees, either as part of the `clock_opt` process (see [“Implementing the Clock Trees” on page 7-58](#)) or as a stand-alone process.

For information about defining the interlock delay balancing requirements, see [“Balancing Multiple Clocks” on page 7-52](#).

By default, interlock delay balancing uses virtual routing to estimate the wire delay and capacitance. For better correlation with postroute timing, you can use the integrated clock global router, instead. To enable the integrated clock global router, use either `clock_opt -cts_effort high` or `optimize_clock_tree -effort high` before running `balance_inter_clock_delay`.

To run stand-alone interlock delay balancing, use the `balance_inter_clock_delay` command (or choose Clock > Balance Interlock Delay in the GUI).

Adjusting the I/O Timing

After implementing the clock trees, IC Compiler can update the input and output delays to reflect the actual clock arrival times. When you adjust the I/O timing, IC Compiler calculates the median insertion delay for each clock tree and applies these values as the clock latency. The Milkyway database and SDC constraints are automatically updated, so you can easily export this data to PrimeTime for detailed timing analysis.

To adjust the I/O timing,

- Run the `update_clock_latency` command.
- or
- Specify the `-update_clock_latency` option when you run the `clock_opt` command.

IC Compiler adjusts the I/O timing to achieve the accuracy of clock latency and to prevent false timing violations on I/O paths after clock tree synthesis in the following ways:

- For synthesized generated clocks, the network latency of a clock object is updated, but the source latency of the clock object is updated when its master clock is synthesized.
- For synthesized clocks, network latency is computed by using the median value of the clock propagation delay, that is, the arrival time relative to the clock root at all boundary registers.

- For virtual clocks defined with the same `create_clock` command, network latency is calculated using the clock propagation delay of the boundary registers clocked by the individual virtual clocks.

To adjust the I/O timing for virtual clocks, you must define the relationships between the virtual clocks and the real clocks before you adjust the I/O timing as follows:

```
icc_shell> set_latency_adjustment_options \
           -to_clock my_virtual_clock -from_clock my_real_clock
icc_shell> update_clock_latency
```

When you save your design in Milkyway format, the relationships defined by the `set_latency_adjustment_options` command are stored in the Milkyway design library.

When adjusting the I/O timing based on virtual clocks, the `update_clock_latency` command defines the clock latency for both the real clock and its associated virtual clocks as the median insertion delay of the real clock.

You can report the virtual clock definitions by using the `report_latency_adjustment_options` command. You can remove the virtual clock definitions by using the `reset_latency_adjustment_options` command.

Performing Multicorner Clock Tree Optimization

IC Compiler can perform multicorner clock tree optimization to achieve the best QoR for your designs by processing various scenario corners simultaneously. Running multicorner clock tree optimization automatically enables the integrated clock global router and the Arnoldi delay analysis (`-clock_arnoldi`) to obtain the best clock network correlation and to ensure accurate insertion delay and skew.

Specifying Corners for Optimization

To run multicorner clock tree optimization, you must first specify scenario corners by either using `set_clock_tree_optimization_options -enable_multicorner` (or choosing Clock > Set Clock Tree Optimization Options to select the Specify Multicorner Optimization check box in the GUI). If you do not specify any scenario corner, IC Compiler assigns the corner that is the maximum corner of the default clock tree synthesis scenario.

IC Compiler determines the default clock tree synthesis scenario as follows:

- Use the clock tree synthesis scenario as the default if the default is defined.
- Use the current scenario as the default if the clock tree synthesis scenario is not defined.
- Use both the maximum and minimum corners of regular synthesis if no other scenario is defined.

Every scenario has a maximum and minimum corner. For example, to specify both the maximum and minimum corners of the default clock tree synthesis scenario, enter the following command:

```
icc_shell> set_clock_tree_optimization_options \  
-enable_multicorner "default:max default:min"
```

If you want to use a scenario other than the default clock tree synthesis one, you can specify the scenarios that you define such as scn1 and scn2 as shown in the following example. For more information about defining a scenario, see [“Using the Clock Tree Synthesis Scenario” on page 14-17](#). Note that IC Compiler first uses the maximum corner of the default clock tree synthesis scenario to optimize a design even if the maximum corner is not specified or the default clock tree synthesis scenario is not active. Therefore, five corners are used for multicorner clock tree optimization in the following example.

```
icc_shell> set_clock_tree_optimization_options \  
-enable_multicorner "scn1:max scn1:min scn2:max scn4:min"
```

To use both the maximum and minimum corners of all active scenarios including the maximum corner of the default clock tree synthesis scenario, enter the following command:

```
icc_shell> set_clock_tree_optimization_options -enable_multicorner all
```

If two corners contain the identical library, process, voltage, temperature, and TLUPlus model specifications, IC Compiler performs only one optimization process.

Note:

Clock definitions and exceptions are taken from either the clock tree synthesis scenario or the current scenario if the clock tree synthesis scenario is not defined (By default, the clock tree synthesis scenario is the scenario you set up to run `optimize_clock_tree`). The library, process, voltage, temperature, and TLUPlus model information are taken from the scenarios specified by the `-enable_multicorner` option.

Specifying Corners Using Multicorner-Multimode Scenarios

You can use multicorner-multimode scenarios to specify clock tree optimization corners. Each multicorner-multimode scenario has two corners: one maximum and one minimum corner. For each corner, you define an operating condition and parasitic information files, using the `set_operating_conditions` command and the `set_tlu_plus_files` command respectively. For each scenario, you associate the cell libraries of both maximum and minimum corners, using the `set_scaling_lib_group` command, not the `set_min_library` command. The `set_min_library` command, which associates a maximum library with a minimum library across all scenarios, is not scenario-specific; therefore, you use the `set_scaling_lib_group` command instead. The following shows an example of one scenario declaration:

```
icc_shell> define_scaling_lib_group -name slow $lib_slow
icc_shell> define_scaling_lib_group -name fast $lib_fast
icc_shell> create_scenario scn1
icc_shell> set_operating_conditions -max $opcond_slow -min $opcond_fast
icc_shell> set_tlu_plus_files -max tluplus $tlu_high_r \
    -min tluplus $tlu_low_r -tech2itf_map $tlu_map
icc_shell> set_scaling_lib_group -max slow -min fast
```

Specifying Constraints for Multicorner Optimization

IC Compiler applies the same design rule constraints that clock tree optimization uses to all corners that you specify during multicorner clock tree optimization except the following:

- Float pin constraints
- Skew target goals

Setting Float Pin Constraints

When you specify a float pin value on the leaf node of a clock tree, you must scale the float pin value to apply to each scenario corner. However, achieving a single float pin phase delay across all corners is impossible because cell delays differ in speed. The float pin values that you specify using the `set_clock_tree_exceptions` command apply only to the corners of the default clock tree synthesis scenario in which the clock tree synthesis constraints are specified.

For other corners, specify float pin values by using one of the following two ways:

- Scale float pin values based on the intrinsic speed of cells in different corners by using IC Compiler's automatic capability. For example, if the cells of a corner have a speed twice as fast as that of the default clock tree synthesis scenario, IC Compiler scales down the float pin values that you specify for the default clock tree synthesis scenario by half for the corner.
- Use the `set_clock_tree_exceptions` command with the `-max_float_pin_scale_factor` and the `-min_float_pin_scale_factor` options to specify float pin value scaling factors. You can set the scaling factors on a per scenario basis as shown in the following example:

```
icc_shell> current_scenario scn1
icc_shell> set_clock_tree_exceptions \
    -max_float_pin_scale_factor 1.0 -min_float_pin_scale_factor 0.6

icc_shell> current_scenario scn2
icc_shell> set_clock_tree_exceptions \
    -max_float_pin_scale_factor 0.9 -min_float_pin_scale_factor 0.5
```

Setting Skew Target Goals

By default, you specify the same skew target goal for all corners using the following command:

```
icc_shell> set_clock_tree_options -target_skew 0.050
```

You can specify various skew target goals for different corners in a multicorner-multimode design as follows:

```
icc_shell> set_clock_tree_options -corner_target_skew \
    "scn1:max=0.100 scn_cts:max=0.050 scn2:min=0.070"
```

Generating Multicorner Optimization Report

To generate a clock tree report for a particular scenario corner, you use the `current_scenario` command to set your current scenario before running the `report_clock_tree` command. For example, to report clock trees of `scn1`, `scn2`, and `scn3` scenarios, enter the following:

```
icc_shell> set_cts_scenario scn1
icc_shell> optimize_clock_tree
icc_shell> current_scenario scn1
icc_shell> report_clock_tree
icc_shell> report_clock_tree -op min
```

```

icc_shell> current_scenario scn2
icc_shell> report_clock_tree
icc_shell> report_clock_tree -op min

icc_shell> current_scenario scn3
icc_shell> report_clock_tree
icc_shell> report_clock_tree -op min

```

Note:

When running the `optimize_clock_tree` command during multicorner clock tree optimization, IC Compiler automatically uses the high effort level, enables the integrated clock global router, and sets the `clock_arnoldi` option.

Because the `report_clock_tree` command requires that you define clocks in the scenario you want to report, you should define clocks in every scenario when using multicorner clock tree optimization.

```

icc_shell> create_scenario scn1
icc_shell> set_operating_conditions -max $opcond1
icc_shell> set_tlu_plus_files -max_tluplus $tlu1 -tech2itf_map $tlu_map
icc_shell> create_clock -name clock -period 4 clock

icc_shell> create_scenario scn2
icc_shell> set_operating_conditions -max $opcond2
icc_shell> set_tlu_plus_files -max_tluplus $tlu2 -tech2itf_map $tlu_map
icc_shell> create_clock -name clock -period 4 clock

```

Performing Clock Routing After Multicorner Optimization

After you finish multicorner clock tree optimization, you can perform clock routing using either the classic router or Zroute.

The classic router supports both the integrated clock global routing and balanced-mode routing. To achieve the best correlation results when using multicorner clock tree optimization, IC Compiler enables the integrated clock global router and saves clock global routing information in the Milkyway database. You use the `route_group -all_clock_nets` command to perform clock routing. The `route_group` command can detect the clock global routing information in the Milkyway database and set the global route incremental model automatically.

For information about using the Zroute to perform clock routing, see the *IC Compiler User Guide: Zroute*.

A Sample Script

The following sample script shows an example of running multicorner clock tree optimization.

```
create_scenario scn_cts
set_operating_conditions \
    -max $opcond_slow -max_library $lib_slow \
    -min $opcond_fast -min_library $lib_fast
set_tlu_plus_files \
    -max_tluplus $tlu_high_r \
    -min_tluplus $tlu_low_r \
    -tech2itf_map $tlu_map

create_scenario scn1
set_operating_conditions \
    -max $opcond_slow -max_library $lib_slow1\
    -min $opcond_fast -min_library $lib_fast1
set_tlu_plus_files \
    -max_tluplus $tlu_high_r \
    -min_tluplus $tlu_slow_r \
    -tech2itf_map $tlu_map

create_scenario scn2
set_operating_conditions \
    -max $opcond_slow -max_library $lib_slow\
    -min $opcond_fast -min_library $lib_fast2
set_tlu_plus_files \
    -max_tluplus $tlu_high_r \
    -min_tluplus $tlu_low_r \
    -tech2itf_map $tlu_map

set_clock_tree_options -corner_target_skew \
    "scn1:max=0.100 scn_cts:max=0.050 scn2:min=0.070"
set_clock_tree_optimization_options -enable_multicorner \
    "scn1:max scn1:min scn2:max scn2:min"

set_cts_scenario scn_cts
optimize_clock_tree

current_scenario scn_ctl
report_clock_tree
report_clock_tree -op min

current_scenario scn1
report_clock_tree
report_clock_tree -op min

current_scenario scn2
report_clock_tree
report_clock_tree -op min

route_group -all_clock_nets
```

Using Batch Mode to Reduce Runtime

When you run multiple `compile_clock_tree` commands on your design, the overhead of repeated clock tree synthesis pre- and post-processing increases runtime. To reduce the `compile_clock_tree` runtime, you can enable the clock tree synthesis batch mode by using the `set_cts_batch_mode` command.

The following script shows an example of how to use the batch mode with `compile_clock_tree`:

```
icc_shell> cts_batch_mode_enable
icc_shell> compile_clock_tree -clock c1
icc_shell> compile_clock_tree -clock c2
icc_shell> cts_batch_mode_status
...
icc_shell> compile_clock_tree -clock c3
...
icc_shell> cts_batch_mode_disable
```

The `reset_cts_batch_mode` command disables the batch mode and invokes the post-processing cleanup of excess clock trees that were inserted. The `report_cts_batch_mode` command displays whether the flow is in the batch mode or normal mode.

Using the batch mode for `compile_clock_tree` saves an average of 15 percent in runtime with a minimum impact on QoR. For best results, avoid using trigger commands that call clock tree synthesis pre- or post-processing between the `set_cts_batch_mode` and `reset_cts_batch_mode` commands. Some examples of trigger commands are `skew_opt`, `get_attribute`, `set_attribute`, `insert_buffer`, `remove_buffer`, `save_mw_cel`, `set_clock_tree_references`, and latency commands.

To use the batch mode with `clock_opt`, set the `cts_clock_opt_batch_mode` variable to true. For example,

```
icc_shell> set cts_clock_opt_batch_mode true
icc_shell> clock_opt
```

When using batch mode with `clock_opt`, the average runtime improvement is about 5 percent.

Analyzing the Clock Tree Results

After synthesizing the clock trees, analyze the results to verify that they meet your requirements. Typically the analysis process consists of the following tasks:

- Analyzing the clock tree reports (as described in [“Generating Clock Tree Reports” on page 7-79](#))
- Analyzing the clock tree timing (as described in [“Analyzing Clock Timing” on page 7-81](#))
- Verifying the placement of the clock instances, using the IC Compiler GUI (as described in [“Analyzing Clock Trees in the GUI” on page 7-81](#))

If the clock trees meet your requirements, you are ready to analyze the entire design for quality of results, as explained in [“Analyzing and Refining the Design” on page 7-96](#).

If the synthesis results do not meet your requirements, IC Compiler can help you debug the results by outputting additional information during clock tree synthesis. Set the `cts_use_debug_mode` variable to true before running clock tree synthesis to output the following additional information:

- User-defined design rule constraints (maximum transition time, maximum capacitance, and maximum fanout)

For more information about user-defined design rule constraints, see [“Setting Clock Tree Design Rule Constraints” on page 7-24](#).

- User-defined clock tree timing constraints (skew and insertion delay)

For more information about user-defined clock tree timing goals, see [“Setting Clock Tree Timing Goals” on page 7-26](#).

- User-defined level restrictions (maximum level count)

For more information about user-defined level restrictions, see [“Setting Level Restrictions” on page 7-26](#).

- Clustering targets set by IC Compiler (maximum transition time and maximum capacitance)

In addition, you can output detailed characterization data for the clock tree references by setting the `cts_do_characterization` variable to true.

Using this information, you can change the clock tree definitions to improve your results.

Generating Clock Tree Reports

To generate clock tree reports, use the `report_clock_tree` command (or choose Clock > Report Clock Tree in the GUI). By default, a report is generated for all clock trees. To limit the report to specific clock trees, use the `-clock_trees`. To generate the report for a specific operating condition, use the `-operating_condition` option. Table 7-12 shows the clock tree reports that are supported by IC Compiler.

Table 7-12 Clock Tree Reports

Report type	Command option GUI object)	Description
Default		Reports the global skew for the specified clock trees.
Clock path from specific roots	<code>-from</code> "From pins" check box	Reports the fanout clock trees from the specified pins.
Clock path to specific sinks	<code>-to</code> "To pins" check box	Reports the clock paths to the specified sink pins.
DRC violators	<code>-drc_violators</code> "DRC violators" check box	Lists the design rule violations that occur in the clock trees up to the endpoints (default sinks or don't touch subtrees), except for violations on don't buffer nets, nets inside interface logic models, nets connected to boundary cells or pad cells.
All DRC violators	<code>-all_drc_violators</code> "Include violators beyond exceptional pins" check box	Lists all design rule violations that occur in the clock trees up to the default sinks (through don't touch subtrees).
Exceptions	<code>-exceptions</code> "Exceptions" check box	Lists the stop pins, exclude pins, float pins, nonstop pins, don't touch subtrees, don't buffer nets, and don't size cells.
Level information	<code>-level_info</code> "Level information" check box	Reports on the structural and timing characteristics for all levels of the clock trees.

Table 7-12 Clock Tree Reports (Continued)

Report type	Command option GUI object)	Description
Settings	-settings “Settings” check box	Lists the following global clock tree settings: clock tree synthesis algorithm, logic-level balancing, clock tree design rule constraints, clock tree configuration files nondefault routing rules, on-chip-variation-aware clustering. Lists the following per-clock tree settings: clock tree timing constraints, clock tree design rule constraints, maximum buffer levels, clock tree optimizations.
Sinks	-show_all_sinks “Show all default sinks in design” check box	Lists the default sink pins.
Structure	-structure “Clock tree structure” radio button -partial_structure_within_exceptions “Clock tree structure within exceptions” radio button	Lists the structure of the clock tree. Lists the structure of the clock tree up to any explicit exclude pins or float pins. The clock structure beyond these exception pins is not reported.
Summary	-summary “Summary” check box	Reports the global clock skew summary.

Analyzing Clock Timing

The timing characteristics of the clock network are important in any high-performance design. To obtain detailed information on the clock networks in a design, use the `report_clock_timing` command. This command reports the clock latency, transition time, and skew characteristics at specified clock pins of sequential elements in the network.

In the `report_clock_timing` command, use the `-type` option to specify the type of report you want (see [Table 7-13](#) for available report types), the scope of the design to analyze, and any desired filtering or ordering options for the report. For analyzing the effect of clock tree synthesis exceptions and clock propagation, include the `-cts_mode` switch. IC Compiler gathers the requested information and reports it in the specified order.

Table 7-13 Clock Timing Report Types

To generate this report	Use this keyword
Single-clock local skew	skew
Transition	transition
Interlock skew	interlock_skew
Latency	latency
Summary (shows the worst instances of transition time, latency, and skew for the specified clocks)	summary

Analyzing Clock Trees in the GUI

IC Compiler provides several GUI windows that you can use to analyze the clock trees:

- Layout window

The clock tree visual modes allow you to view the structure or timing distribution of the clock trees in the layout window.

- Interactive clock tree synthesis window

The interactive clock tree synthesis window provides information about all the clocks defined in your design.

- Clock tree option viewer

The clock tree option viewer displays the current settings of the clock tree synthesis options.

- Clock tree hierarchy browser

The clock tree hierarchy browser provides a hierarchical listing of the clock tree.

- Fanin and fanout schematics
- Clock arrival histogram

The following sections describe these windows.

Viewing Clock Trees in the Layout Window

IC Compiler provides two clock tree visual modes:

- Clock tree structure
- Clock tree timing distribution

These modes are useful for analyzing the clock tree structure after clock tree synthesis. Analyzing the structure enables you to identify possible sources for clock tree timing problems, such as a restrictive floorplan or the use of nonoptimal references.

Viewing the Clock Tree Structure. To view the clock tree structure in visual mode,

1. In the layout window, choose Clock > Color By Clock Trees.

The Visual Mode panel appears.

2. Click Reload.

The Configure Clock Tree Visual Mode dialog box appears.

3. Select the clock tree levels or cell types that you want displayed. (To view the clock tree display choices for a tree, click the plus sign (+) to the left of the clock tree name.)

[Table 7-14](#) describes the clock tree display choices. You must select at least one level or cell type.

Table 7-14 Clock Tree Display Choices

Partition name	Description
AllLevel	Displays the entire clock tree in a single color.
Level 0...Level <i>n</i>	Displays each selected clock tree level in a different color.
buffer	Displays the buffers inserted into the clock tree by IC Compiler.

Table 7-14 Clock Tree Display Choices (Continued)

Partition name	Description
inverter	Displays the inverters inserted into the clock tree by IC Compiler.
preexisting	Displays the preexisting gates in the clock tree.
sink	Displays the clock tree sinks.

To analyze the synthesized clock tree, you would typically select one or more clock tree levels (Level 0...Level n) to be displayed. You can use this display to debug the placement of the clock tree cells.

4. Click OK or Apply.

The GUI displays the selected clock tree levels and cell types in the layout view.

By default, the clock tree nets are displayed. To display the clock tree cells, deselect Cells in the “Exclude objects” section. To disable the display of clock tree nets, select Nets in the “Exclude objects” section.

Viewing the Clock Tree Timing Distribution. To display the clock latency or transition time distribution,

1. In the layout window, choose Clock > Color By Clock Latency/Transition.

The Visual Mode panel appears.

2. Click Reload.

The Clock Latency/Transition dialog box appears.

3. In the Clock Latency/Transition dialog box, do the following:

- Type or select a clock tree in the Clock box.
- Select a Delay Type option (Latency or Transition).
- Set other options as needed.

4. Click OK.

Using the Interactive Clock Tree Synthesis Window

The interactive clock tree synthesis window provides information about all of the clocks defined in your design. You can select clock trees in the window for further examination with other analysis tools.

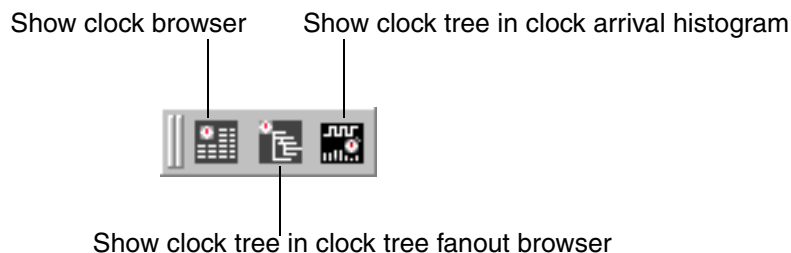
To open an interactive clock tree synthesis window,

- Choose Clock > New Interactive CTS Window.

When you open an interactive clock tree synthesis window, it displays the clock browser view, which includes a table with information about each clock tree. The table columns show the clock names, sources, and other details about each clock tree.

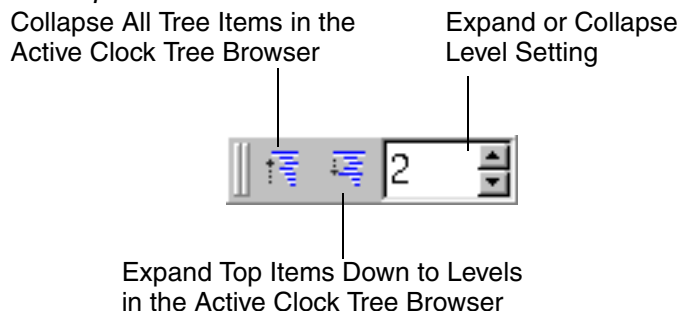
The Browsers toolbar, shown in [Figure 7-13](#), lets you select a clock in the clock browser view and open a clock tree fanout browser or generate a clock arrival histogram. You can also display the clock browser view when it is hidden.

Figure 7-13 Browsers Toolbar



The Expand Browser toolbar, shown in [Figure 7-14](#), lets you expand or collapse fanout levels for the clock tree displayed in the active clock tree fanout browser. You can expand or collapse all levels, or expand a specified number of levels. This toolbar can be found in the interactive clock tree synthesis window.

Figure 7-14 Expand Browser Toolbar



You can open the following clock tree analysis windows from the interactive clock tree synthesis window:

- The clock tree option viewer
- The clock tree hierarchy browser
- The fanout schematic
- The clock tree arrival histogram

You can also select clock tree objects in the clock tree fanout browser for analysis with tools in the layout window.

Using the Clock Tree Option Viewer

IC Compiler provides a clock tree option viewer that you can use to view or set the clock tree options for each clock tree in the design.

To open the clock tree option viewer,

- In the layout window or the interactive clock tree synthesis window, choose Clock > List Clock Tree Options.

The clock tree option viewer displays a list of all the clock trees in the design. The list contains columns with the current values (whether default or previously set) for each clock tree option. You can use the buttons below the list to set or change options for the selected clock tree, display or hide columns in the list, or close the window.

Viewing the Clock Tree Hierarchy

The clock tree hierarchy browser provides a hierarchical listing of the clock tree. This can be useful for analyzing complex clock tree structures before or after clock tree synthesis.

If you are not familiar with the clock trees in your design, you can explore the clock tree structures and gather information about objects (buffers, inverters, and preexisting cells) at each level. You can also select the names of objects you want to examine in graphic views or with other analysis tools.

Before you perform clock tree synthesis, use the clock tree hierarchy browser to understand the existing clock tree structure.

After you perform clock tree synthesis, use this view to analyze the resulting clock tree structures and for troubleshooting if you failed to achieve the expected QoR.

To display the clock tree hierarchy view,

1. In the interactive clock tree synthesis window, select the clock you want to view.
2. Choose View > Clock Tree Hierarchy Browser.

The clock tree hierarchy browser is associated with the interactive clock tree synthesis window in which it was opened. You can open multiple clock tree hierarchy browsers (one for each clock tree).

The view window consists of two panes, with expandable level trees (one for each clock tree) in the left pane and an object information table in the right pane. You can explore the complete structure of a clock tree, observe how many levels are present, and examine the clock tree fanout information for the cells at each level.

If you hold the cursor over a clock tree object, in either the left or right pane of the clock tree hierarchy browser, information about the clock object is displayed.

If you select a clock and right-click, a menu appears. You can open the following clock tree analysis windows from this menu:

- Clock tree visual mode
- The fanout or fanin schematic

In addition, you can view the longest and shortest paths for a clock tree in the clock tree browser. To view the longest path of the selected clock, click the longest path icon () in the interactive clock tree synthesis window. To view the shortest path of the selected clock, click the shortest path icon () in the interactive clock tree synthesis window.

Viewing the Clock Tree Arrival Time Histogram

Clock arrival histograms provide a high-level overview of the timing quality in the fanout network of a selected clock tree. Create a clock arrival histogram to identify clock network pins that failed their constraints.

To display the clock tree arrival time histogram,

- In the interactive clock tree synthesis window, choose Timing > Clock Arrival Histogram.

The clock arrival histogram appears in a new histogram view window. The window is split into two panes, with the histogram bar graph in the left pane and an object table in the right pane. The number at the top of the tallest bin indicates the number of clock tree objects in the bin. Green bins (on the positive side of 0) contain objects in the design that met their constraints. Red bins (on the negative side of 0) contain objects that failed their constraints.

You can select one or more pin names in the table for further examination with other analysis tools. You can

- Select pins and display the clock fanin network in a path schematic
- View object properties for the selected pins

Viewing the Fanout or Fanin Schematic

IC Compiler can provide a schematic showing the fanout or fanin of a clock tree object.

To display a fanout or fanin schematic,

1. Select a clock tree object in the interactive clock tree synthesis window or clock tree hierarchy browser.
2. Right-click to display the pop-up menu.
3. Choose the appropriate menu selection:
 - In the interactive clock tree synthesis window, choose “Clock Fanout Network Schematic From Selected” to display the fanout of the selected clock.
 - In the clock tree hierarchy browser, choose “Clock Fanout Schematic from Selected” to display the fanout from the selected clock object.
 - In the clock tree hierarchy browser, choose “Clock Fanin Schematic from Selected” to display the fanin to the selected clock object.

Fine-Tuning the Clock Tree Synthesis Results

IC Compiler supports the following methods for fine-tuning the clock tree synthesis results:

- Using the useful skew technique
- Resynthesizing the clock trees after modifying the clock tree settings
- Modifying the nondefault routing rule
- Modifying the clock trees in the GUI

The following sections describe these methods.

Using the Useful Skew Technique

The useful skew technique improves timing QoR by adjusting the clock arrival times to take advantage of positive slack in the network. Use the `skew_opt` command to automatically perform useful skew analysis and generate the useful skew constraints.

skew_opt Details

By default, the `skew_opt` command performs the following tasks:

- Analyzes the design to determine which paths can be used for useful skew.

IC Compiler looks for paths with positive slack where the slack could be redistributed (by adjusting the clock latencies) along the input-to-output paths to improve the overall design timing.

By default, IC Compiler evaluates only the setup constraints. To consider both setup and hold constraints, specify the `-hold` option. You can also tighten the setup and hold constraints by specifying the `-setup_margin` and `-hold_margin` options, respectively.

IC Compiler does not adjust the latency on the following types of paths:

- Paths that loop from a register to itself
- Paths that contain level-sensitive latches
- Combinational paths from an input port to an output port
- Paths within interface logic models

Note:

IC Compiler does not adjust the latency on a nonstop pin. For paths that start or end at nonstop pins (either implicit or explicit), IC Compiler will only adjust the latency at the other endpoints or startpoints.

By default, there is no limit to the amount of latency adjustment that can be made. To limit the amount of latency adjustment, specify the `-adjustment_limit` option. To further limit the amount the latency can be decreased, specify the `-decrease_factor` option.

To disable this analysis and use the existing clock latencies, specify the `-no_optimization` option.

- Determines the interclock relationships in the design.

IC Compiler analyzes the interclock relationships in the design and uses this analysis to define the interclock delay balancing groups.

- Generates a script file that is used to adjust the clock latencies and set the interclock delay balancing constraints.

The script contains `set_clock_latency` and `set_clock_tree_exceptions -float` commands to adjust the clock latencies and `set_inter_clock_delay_options` commands to define the interclock delay balancing groups.

By default, a script is generated if any latency adjustments are identified. If you want to generate a script only if the worst negative slack (WNS) improves by a certain amount, specify the `-improvement_threshold` option.

By default, the generated script file is named `skew_opt.tcl`. To use another file name, specify the `-output` option.

- Sources the generated script file to apply the clock latency adjustments and interclock delay balancing constraints.

You can disable sourcing of some of the commands in the generated script, or of the entire generated script.

- To disable sourcing of the `set_clock_latency` commands, set the `skew_opt_skip_ideal_clocks` variable to true before running `skew_opt`.
- To disable sourcing of the `set_clock_tree_exceptions -float` commands, set the `skew_opt_skip_propagated_clocks` variable to true before running `skew_opt`.
- To disable sourcing of the `set_inter_clock_delay_options` commands, set the `skew_opt_skip_clock_balancing` variable to true before running `skew_opt`.
- To disable sourcing of the entire script, specify the `-no_auto_source` option when you run `skew_opt`.

For detailed information about the `skew_opt` command, see the man page.

skew_opt Flow

The flow for using the `skew_opt` command is as follows:

1. Determine the existing clock latencies by doing a quick `clock_opt` run.

Use the `set_interclock_delay_options` command to define the interclock relationships in the design. For example, if there is a relationship between `clk1` and `clk2`, enter

```
icc_shell> set_inter_clock_delay_options -balance_group {clk1 clk2}
```

Use the `set_latency_adjustment_options` command to identify the timing relationships between a source clock and associated generated clocks or virtual clocks. For example, for a generated clock, enter

```
icc_shell> set_latency_adjustment_options \  
-from_clock master_clk -to_clk gen_clock
```

The following script shows an example of how to determine the latencies for your design:

```
save_mw_cel -as prep  
place_opt  
set_inter_clock_delay_options ...  
set_latency_adjustment_options ...  
save_mw_cel -as place_opt  
clock_opt -inter_clock_balance -update_clock_latency  
write_sdc prep.sdc  
sh grep set_clock_latency prep.sdc > prep.scl  
prep.scl sh grep get_clock > ports.sdc
```

2. Run the `skew_opt` command.

```
icc_shell> skew_opt  
icc_shell> close_mw_cel
```

3. Load the updated clock latencies and useful skew solution.

```
icc_shell> open_mw_cel place_opt  
icc_shell> source ports.sdc  
icc_shell> source skew_opt.tcl
```

4. Perform clock tree synthesis and optimization with interclock delay balancing.

```
icc_shell> clock_opt -inter_clock_balance
```

Important:

Because the `skew_opt` command works across all clock domains in your design, you must perform interclock delay balancing when running clock tree synthesis after `skew_opt`.

In addition, do not update the clock latency after clock tree synthesis (`clock_opt -update_clock_latency` option or `update_clock_latency` command). Updating the clock latency after clock tree synthesis invalidates the script file generated by `skew_opt`.

Resynthesizing the Clock Trees

In some cases, to achieve the best results, you must fine-tune the clock tree synthesis settings, then resynthesize the clock trees.

Resynthesizing the clock trees consists of the following steps:

1. Input the postplacement design that you saved before performing clock tree synthesis.

If you did not save the design before running clock tree synthesis (or if you want to keep some synthesized clock trees, but resynthesize others), you must remove the synthesized clock trees before you can resynthesize the clock trees (see [“Removing Clock Trees” on page 7-46](#)). The resulting design might not be identical to your design before running the initial clock tree synthesis, because some gates might have been moved or sized during the clock tree synthesis process.

2. Validate the setup.
3. Fine-tune the clock tree synthesis settings.
4. Resynthesize the clock trees.

The following sections describe these steps.

Validating the Setup

If the synthesis results do not meet your requirements, check the following items before fine-tuning:

- The input design

Ensure that the input design meets the requirements for clock tree synthesis.

- TLUPlus models
- The clock root

Verify that you have correctly defined the clock root, including setting a driving cell if the clock root is a top-level input port. Verify that the attributes of the clock root, such as the input transition time and output capacitance, are correct. Verify that the library model for the root cells is accurate.

- The clock sinks

Run the `report_clock_tree -exceptions` command to verify that the clock tree exceptions (both implicit and explicit) are correctly set.

- The design rule constraints

To verify the design rule constraints and check that they are realistic, use the `report_clock_tree` command to report on the clock tree (or choose Clock > Report Clock Tree in the GUI). If your clock trees have too many cells or levels, this typically indicates that the design rule constraints are too tight.

Resynthesizing the Clock Trees

You can resynthesize the clock trees either by using the `clock_opt` command (as described in [“Implementing the Clock Trees” on page 7-58](#)) or by using stand-alone clock tree synthesis (as described in [“Performing Clock Tree Synthesis” on page 7-65](#)).

Modifying the Nondefault Routing Rule

If your design is congested, you can modify the nondefault routing rule of an existing clock tree. To modify the nondefault routing rule, use the `mark_clock_tree -routing_rule` command (or choose Clock > Mark Clock Tree in the GUI and specify the routing rule in the “Routing rule” field). In the same way that you specify the nondefault rule for unsynthesized clock trees, you can use the default routing rule for the nets connected to the clock sinks by specifying the `-use_default_routing_for_sinks level_count` option (or by selecting “Routing rule” in the GUI and specifying the level count in the “Use default routing for sinks at level” field). In addition, you can specify the layers to use for clock routing by specifying the `-layer_list layers` option (or by selecting the layers in the GUI). For more information about the clock tree routing options, see [“Setting Clock Tree Routing Options” on page 7-27](#).

Note:

If you specify an undefined routing rule or layer, the `mark_clock_tree` command exits with an error message and does not change the existing settings.

When you change the nondefault routing rule, IC Compiler updates the RC data by running the `extract_rc` command to perform extraction using the new routing rule information.

Modifying Clock Trees in the GUI

You can modify clock trees in the GUI by using the interactive clock tree synthesis capability. You can use interactive clock tree synthesis for both preroute and postroute designs.

Note:

Interactive clock tree synthesis considers a design to be postroute only if all clock nets are routed.

Interactive clock tree synthesis allows you to do what-if analysis on the clock trees and then commit the changes, if they meet your requirements.

You can invoke the interactive clock tree synthesis capabilities from the layout menu (by choosing Clock > ECO) or by selecting an object in the interactive clock tree synthesis window and right-clicking to open the pop-up menu. [Table 7-15](#) shows the capabilities supported by interactive clock tree synthesis and the menu option used to invoke this capability.

Table 7-15 Interactive Clock Tree Synthesis Capabilities

Task	Supported phases	Menu option
Insert buffer	preroute	CTS Insert Buffer
Remove buffer	preroute	CTS Remove Buffer
Move buffer or gate	preroute	CTS Move Cell
Move a clock tree branch to another point in the same clock tree.	preroute	CTS Re-parent
Size buffer or gate	preroute or postroute	CTS Size Cell

When you make changes, you can update the timing by clicking Update Timing in the associated dialog box. If the change meets your requirements, click Legalize to legalize the placement and commit the changes. If the change does not meet your requirements, click Undo to remove the changes.

If you are using interactive clock tree synthesis on a postroute design, you can also perform ECO routing (by clicking Route ECO) and extraction (by clicking Extract RC) after committing the sizing changes.

Note:

If new nets are added to the design as a result of the modifications, IC Compiler does not propagate nondefault routing rules to these new nets. You can use the `mark_clock_tree` command to reapply the nondefault routing rule to the clock tree, including the new nets.

Inserting a Buffer

To insert a buffer using interactive clock tree synthesis,

1. Select CTS Insert Buffer.

The CTS Insert Buffer dialog box appears.

2. Specify the following:

- Whether you want to insert a buffer or an inverter pair
- To which clock pin you want to insert the buffer

If you specify an output pin, the new buffer drives all fanouts of the specified pin. If you specify an input pin, the new buffer drives only that pin.

- The library reference cell to use

If you specify a pattern, IC Compiler locates all library reference cells matching the specified pattern.

- (Optional) The names for the new cells and nets
- The physical location of the inserted buffer

Note:

You can also specify the physical location of the inserted buffer by left-clicking in the layout window

- (Optional) the colors to use to highlight the inserted buffer in the layout window

3. Click “Insert Buffer” to insert the buffer.
4. Click “Update Timing” to perform what-if analysis for this change.
5. Click Legalize to commit this change or click Undo to undo this change.

Removing a Buffer

To remove a buffer using interactive clock tree synthesis,

1. Select CTS Remove Buffer.

The CTS Remove Buffer dialog box appears.

2. Specify the buffer to remove.
3. Click “Remove Buffer” to remove the buffer.
4. Click “Update Timing” to perform what-if analysis for this change.
5. Click Legalize to commit this change or click Undo to undo this change.

Moving a Buffer or Gate

To move a buffer or gate using interactive clock tree synthesis,

1. Select CTS Move Cell.

The CTS Move Cell dialog box appears.

2. Click “Move Cell” to enter move mode in the layout window.
3. Move the cell (a clock buffer or clock gate) in the layout window.
4. Click Legalize to commit this change or click Undo to undo this change.

Reparenting a Subtree

To reparent a subtree using interactive clock tree synthesis,

1. Select CTS Re-parent.

The CTS Re-parent dialog box appears.

2. Specify the following:
 - The subtree root pin
 - The new driving pin for the subtree
3. Click Re-parent to perform what-if analysis for this change.
4. Click Legalize to commit this change or click Undo to undo this change.

The new driving pin that you specify for the reparented subtree must meet the following requirements:

- It must be in the same clock tree as the existing driving pin.
- It must be in the same logical hierarchy as the existing driving pin.
- It must not be a don't touch subtree pin.
- It must not drive a don't buffer net.

Sizing a Buffer or Gate

To size a buffer or gate using interactive clock tree synthesis,

1. Select CTS Size Cell.

The CTS Size Cell dialog box appears.

2. Specify the following:

- The buffer or gate to be sized (the cell must be on the clock path)
- The library reference cell that can be used

3. Click "Size Cell" to perform what-if analysis for this change.

4. Click Legalize to commit this change or click Undo to undo this change.

Analyzing and Refining the Design

After you are satisfied with the clock tree synthesis results, analyze the QoR of the entire design by reporting on constraints (use the `report_constraint` command or choose Design > Report Constraints in the GUI) and timing (use the `report_timing` command).

Use the reports to check the following parameters:

- Worst negative slack (WNS)
- Total negative slack (TNS)
- Design rule constraint violations

8

Routing

This chapter describes the routing capabilities of the IC Compiler classic router. For information about Zroute, see the *IC Compiler User Guide: Zroute*.

This chapter contains the following sections:

- [Prerequisites for Routing](#)
- [Checking Routability](#)
- [Setting Up for Routing](#)
- [Routing Critical Nets](#)
- [Routing Signal Nets](#)
- [Shielding Nets](#)
- [Performing ECO Routing](#)
- [Reporting Cell Placement and Routing Statistics](#)
- [Verifying the Routed Design](#)
- [Routing Nets and Buses in the GUI](#)
- [Postroute RC Extraction](#)
- [Reliability Analysis Using PrimeRail](#)

Prerequisites for Routing

Before you can perform routing, your design must meet the following conditions:

- Power and ground nets have been routed after design planning and before placement.

For more information, see the *IC Compiler User Guide: Design Planning*.

- Clock tree synthesis and optimization have been performed.

For more information, see [Chapter 7, “Clock Tree Synthesis.”](#)

- Estimated congestion is acceptable.
- Estimated timing is acceptable (about 0 ns of slack).
- Estimated maximum capacitance and transition have no violations.

To verify that your design meets the last three prerequisites, you can check the routability of its placement as explained in [“Checking Routability”](#) in the following section.

If you are using the IC Compiler package, which includes routing, you can perform routing and postroute optimization with IC Compiler. If you are using the IC Compiler-PC package, you need to export the design (see [“Saving the Design” on page 3-33](#)) so that you can perform routing with another tool.

Checking Routability

After placement is completed, you can have IC Compiler check whether your design is ready for detail routing. The tool checks pin access points, cell instance wire tracks, pins out of boundaries, minimum grid and pin design rules, and blockages to make sure they meet design requirements. It creates an error file named after the top cell in your design (*top_cell_name.err*), with a list of violations you should correct before performing detail routing.

To verify that your design is ready for detail routing, use the `check_routeability` command (or choose Route > Check Routability in the GUI).

After you run the `check_routeability` command, you can use the `report_error_coordinates` command to report the location of each error. You can also use the error browser to examine the errors in the GUI. For more information about the error browser, see [“Examining Routing and Verification Errors” on page A-47](#) and the “Examining Routing and Verification Errors” topic in IC Compiler Help.

Setting Up for Routing

You can specify general routing setups for the classic router to use whenever you perform routing. The following sections describe how to specify these setups:

- [Specifying Route Guides](#)
- [Using Nondefault Routing Rules](#)
- [Specifying the Routing Layers](#)
- [Setting Net Aggressors](#)
- [Setting Routing Types](#)
- [Setting Routing Options](#)
- [Enabling Distributed Routing](#)

Specifying Route Guides

You can create or remove route guides that do the following for a specific area of your design:

- Prevent routing for signal or prerouted nets
- Change the wiring direction
- Control wiring density
- Fix violations

To create a route guide, use the `create_route_guide` command (or choose Floorplan > Create Route Guide in the GUI). [Table 8-1](#) defines the `create_route_guide` options. For more information about these options, see the man page.

Table 8-1 create_route_guide Command Options

Command option	GUI object	Description
<code>-name name</code>	Name box	Specifies the name of the route guide.
<code>-repair_as_single_sbox</code>	“Repair as single SBox” check box	Enables fixing of violations.
<code>-no_signal_layers layers</code>	“No signal route on layers” check box and list	Prevents the routing of signal wires on the specified layers.

Table 8-1 *create_route_guide Command Options (Continued)*

Command option	GUI object	Description
-zero_min_spacing	"Zero min-spacing" check box	Allows the routing of wires within the keepout margin of the route guide.
-no_preroute_layers layers	"No automatic preroutes on layers" check box and list	Prevents automatic preroutes on the specified layers.
-preferred_direction_ only_layers layers	"Layers with preferred direction only" check box and list	Specifies the layers that must be routed in the preferred direction.
-horizontal_track_ utilization percentage	Horizontal box in "Route track utilization" section	Specifies the allowable horizontal track utilization.
-vertical_track_ utilization percentage	Vertical box in "Route track utilization" section	Specifies the allowable vertical track utilization.
-coordinate rectangle	Coordinates box	Specifies the coordinates for the route guide. In the GUI, you can type the coordinates in the dialog box or draw the rectangle in the layout view. You can also specify that the route guide should snap to the minimum grid, placement site, routing track (the default), middle routing track, or user grid. From the command line, the snapping is controlled by the <code>set_object_snap_type</code> command.
-switch_preferred_ direction	"Switch preferred direction" check box	Switches the preferred wiring direction.

To find route guides, use the `get_route_guides` command. For example, to get all the route guides in your design, enter

```
icc_shell> get_route_guides *
```

You can also find route guides by using a filter expression (`-filter expression`) or by using rectangular areas that encompass (`-within {{x1 y1} {x2 y2}}`) or touch (`-touch {{x1 y1} {x2 y2}}`) the guides.

To remove route guides, use the `remove_route_guides` command. You can remove a collection of route guides, such as that returned by the `get_route_guides` command; all route guides (`-all`); or a specific named route guide (`-name`).

For more information about the `get_route_guides` or `remove_route_guide` command, see its man page.

Setting the Preferred Routing Direction

Use the `set_preferred_routing_direction` command to reset and override the default preferred routing direction specified in the library, or the design, for a specific layer. The layer direction set with this command applies to the current design only.

The syntax is

```
set_preferred_routing_direction
    -layers list_of_layers
    -direction horizontal | vertical
```

For example, to set the preferred routing direction to vertical for layer M5 and M7, enter

```
icc_shell> set_preferred_routing_direction -layers "M5 M7" \
    -direction vertical
```

Note:

Settings with the `create_route_guide -switch_preferred_direction` command, which changes the preferred direction within the area that is covered by the route guide, will override the settings made with the `set_preferred_routing_direction` command.

Use the `report_preferred_direction` command to report the preferred routing direction for all the routing layers. The report lists the library and user-defined routing directions, as well as the direction that the tool will use.

Here is a sample report:

```
*****
Report : Layers
Design : core_chip
Version: Y-2006.06
Date   : Thu Mar 23 03:43:06 2006
*****
Layer Name      Library      Design      Tool understands
metal1          Horizontal  Not Set     Horizontal
metal2          Vertical   Not Set     Vertical
metal3          Horizontal  Not Set     Horizontal
metal4          Vertical   Not Set     Vertical
metal5          Horizontal  Vertical    Vertical
metal6          Vertical   Vertical    Vertical
```

Use the `remove_preferred_routing_direction` command to remove the user-defined directions for a specific layer from the design. The syntax is

```
remove_preferred_routing_direction -layers list_of_layers
```

Using Nondefault Routing Rules

IC Compiler supports the use of nondefault routing rules, both for routing and for shielding. Before you can use a nondefault routing rule, you must define it. The following sections describe how to define and apply nondefault routing rules.

Defining Nondefault Routing Rules

You define nondefault or default routing rules for specific nets by using the `define_routing_rule` command (or by choosing Route > Routing Setup > Define Routing Rule in the GUI). These rules define wire width and spacing rules and via types. [Table 8-2](#) defines the `define_routing_rule` options. For more information about these options, see the man page.

Table 8-2 define_routing_rule Command Options

Command option	GUI object	Description
<code>rule_name</code>	“New rule name” box	Specifies the name of the rule. This argument is required.
<code>-reference_rule_name</code>	“Reference rule” section Specified radio button	Specifies the name of the reference rule.
<code>-default_reference_rule</code>	Default radio button	This option is required.

Table 8-2 define_routing_rule Command Options (Continued)

Command option	GUI object	Description
-taper_level	"Taper level" box	Specifies the tapering level. The default tapering level is 1.
-snap_to_track	"Snap to track" check box	Snaps shielding wires to the track when selected. By default, snapping is not enabled.
-multiplier_width	"Multiplier width" box	Specifies the layer width multiplier.
-multiplier_spacing	"Multiplier spacing" box	Specifies the layer spacing multiplier.
-widths	Metal table section Width column	Specifies the width and spacing rules.
-spacings		
-shield_widths		
-shield_spacings		
-via_cuts	"Contact table" section	Specifies the via types.

Applying Nondefault Routing Rules

To apply a net routing rule to one or more nets, use the `set_net_routing_rule` command (or choose Route > Routing Setup > Set Net Routing Rule in the GUI). [Table 8-3](#) defines the `set_net_routing_rule` options. For more information about these options, see the man page.

Table 8-3 set_net_routing_rule Command Options

Command option	GUI object	Description
<code>list_of_nets</code>	Nets box	Specifies the nets to which to apply the rule. This argument is required.
-rule <code>rule_name</code>	"Routing rule" box	Specifies the name of the nondefault rule to apply. This option is required.
-top_layer_probe AnyPort OutPort AllPort	"Top layer probe mode" box	Specifies which ports to use for top-layer probing. The default is AnyPort.

Table 8-3 *set_net_routing_rule* Command Options (Continued)

Command option	GUI object	Description
<code>-reroute normal minorchange freeze</code>	"Net re-routability" box	Specifies when to reroute nets. The default is normal.
<code>-timing_driven_spacing</code>	"Timing driven spacing" check box	When timing-driven spacing is enabled, IC Compiler adds space between critical nets and nets that are parallel to them to reduce the intralayer coupling capacitances. By default, timing-driven spacing is not enabled.

Reporting Nondefault Routing Rules

To report the nets that have nondefault routing rules, use the `report_net_routing_rules` command.

```
icc_shell> report_net_routing_rules [get_nets *]
```

In addition, the name of the nondefault routing rule for a net is stored in the `var_route_rule` net attribute. You can access the value of this attribute by using the `get_attribute` command.

Specifying the Routing Layers

IC Compiler lets you specify which layers can be ignored for routing and which layers are to be ignored for RC and congestion estimation. You can also specify the routing layers to use for specific nets (net layer constraints).

By default, when you set a maximum routing layer, it is a hard constraint. You can change it to a soft constraint or you can allow the use of higher layers only for pin connections by setting the `hardMaxLayerConx` detail route option. This option applies to both ignored layers (`set_ignored_layers` command) and net layer constraints (`set_net_routing_layer_constraints` command).

By default, when you set a minimum routing layer, it is a soft constraint. You can change it to a hard constraint or you can allow the use of lower layers only for pin connections by setting the `hardMinLayerConx` detail route option. This option applies to both to ignored layers (`set_ignored_layers` command) and net layer constraints (`set_net_routing_layer_constraints` command).

Specifying the Ignored Layers

To specify the ignored layers, use the `set_ignored_layers` command (or choose Route > Routing Setup > Set Ignored Layers in the GUI).

Table 8-4 shows the `set_ignored_layers` options. For more information about these options, see the man page.

Table 8-4 *set_ignored_layers Command Options*

Command option	GUI object	Description
<code>-min_routing_layer name</code>	"Minimum routing layer" check box and list	Specifies the lowest routing layer. IC Compiler uses the specified layer and the layers above it for routing.
<code>-max_routing_layer name</code>	"Maximum routing layer" check box and list	Specifies the highest routing layer. IC Compiler uses the specified layer and the layers below it for routing.
<code>-rc_congestion_ignored_layers names</code>	"Select layers to be ignored in RC computation" section	Specifies the layers to ignore for RC and congestion estimation. By default, RC and congestion estimation use the layers you specify for routing.

For example, to define layers M2 through M7 for routing, use the following command:

```
icc_shell> set_ignored_layers -min_routing_layer M2 -max_routing_layer M7
```

To define the same layers for routing and to force the RC and congestion estimation functions to ignore the M1, M2, M8, and M9 layers, use the following command:

```
icc_shell> set_ignored_layers \  
-min_routing_layer M2 -max_routing_layer M7 \  
-rc_congestion_ignored_layers {M1 M2 M8 M9}
```

Specifying Routing Layers for Specific Nets

To specify the routing layers for specific nets, use the `set_net_routing_layer_constraints` command (or choose Route > Routing Setup > Set Net Layer Constraints in the GUI).

Table 8-5 shows the `set_net_routing_layer_constraints` options. For more information about these options, see the man page.

Table 8-5 *set_net_routing_layer_constraints* Command Options

Command option	GUI object	Description
<i>nets</i>	"Specified nets" box	Specifies the nets to which to apply the routing constraints. This argument is required.
<i>-min_layer_name name</i>	"Minimum routing layer" list	Specifies the lowest routing layer. IC Compiler uses the specified layer and the layers above it for routing the specified nets.
<i>-max_layer_name name</i>	"Maximum routing layer" list	Specifies the highest routing layer. IC Compiler uses the specified layer and the layers below it for routing the specified nets.

For example, to define layers M2 through M7 for routing net n1, use the following command:

```
icc_shell> set_net_routing_layer_constraints -min_layer_name M2  
-max_layer_name M7 [get_nets n1]
```

Setting Net Aggressors

To minimize crosstalk-induced noise, you can specify the aggressors for a victim net. Crosstalk prevention during postroute optimization uses that information to minimize the crosstalk-induced noise from the aggressor nets.

To specify aggressors for a victim net, use the `set_net_aggressors` command (or choose Route > Routing Setup > Set Net Aggressors in the GUI). To specify the victim net, use the `-victim_net` option (or the "Victim net" box in the GUI). To specify the aggressor nets, use the `-aggressor_nets` option (or the "Aggressor nets" box in the GUI).

Setting Routing Types

For debugging, you can set or change the routing types of wires, vias, or paths to indicate which classic router routing engine is to route them.

To specify a routing of any type, use the `set_route_type` command (or choose Route > Routing Setup > Set Route Type in the GUI).

- For signal nets, you can specify the following route types: `detail_route` or `user`.

```
icc_shell> set_route_type -signal type objects
```

- For clock nets, you can specify the following route types: `ring`, `strap`, `tie_off`, or `user`.

```
icc_shell> set_route_type -clock type objects
```

- For power and ground nets, you can specify the following route types: `ring`, `strap`, `tie_off`, `user`, `std_cell_pin_conn`, or `macro/IO_pin_conn`.

```
icc_shell> set_route_type -pg type objects
```

To remove a routing type, use the `remove_route_by_type` command (or choose Route > Delete Route in the GUI).

Setting Routing Options

You can specify options that control how the classic router performs global routing, track assignment, and detail routing, as well as miscellaneous routing options.

- To set routing options, use the `set_route_options` command (or choose Route> Routing Setup > Set Route Options in the GUI). To reset the options to their default values, use `set_route_options -default` (or click Default in the GUI).
- To report the settings of all routing options, use the `report_route_options` command.

Note:

You can set additional global route options by using the `set_groute_options` command. You can set additional detail route options by using the `set_droute_options` command. For more information about these commands, see the man pages.

Table 8-6 lists the routing options set by the `set_route_options` command.

Table 8-6 *set_route_options* Command Options

Option	Valid values	Description
Global routing options (Global Routing tab in the GUI)		
<code>-groute_skew_control</code> (“Skew control” check box in the GUI)	<code>true</code> <code>false</code>	Enables (true) or disables (false) skew control during global routing. The default is <code>false</code> .
<code>-groute_skew_weight</code> (“Skew control Weight” box in the GUI)	<code>int</code> (must be between 1 and 10)	Specifies the weight associated with skew control. The default is 5.
<code>-groute_timing_driven</code> (“Timing driven” check box in the GUI)	<code>true</code> <code>false</code>	Enables (true) or disables (false) timing-driven global routing. The default is <code>false</code> .
<code>-groute_timing_driven_weight</code> (“Timing driven Weight” box in the GUI)	<code>int</code> (must be between 1 and 7)	Specifies the weight associated with timing-driven global routing. The default is 4.
<code>-groute_congestion_weight</code> (“Congestion weight” box in the GUI)	<code>int</code> (must be between 1 and 12)	Specifies the weight associated with congestion-driven global routing. The default is 4.
<code>-groute_clock_routing</code> (“Clock routing” radio buttons in the GUI)	<code>normal</code> <code>comb</code> <code>balanced</code>	Specifies the global-routing clock topology. The default is <code>balanced</code> .
<code>-groute_incremental</code> (Incremental check box in the GUI)	<code>true</code> <code>false</code>	Enables (true) or disables (false) incremental global routing. The default is <code>false</code> .
Track assignment options (Track Assign tab in the GUI)		
<code>-track_assign_timing_driven</code> (“Timing driven” check box in the GUI)	<code>true</code> <code>false</code>	Enables (true) or disables (false) timing-driven track assignment. The default is <code>false</code> .
<code>-track_assign_timing_driven_weight</code> (“Timing driven” Weight box in the GUI)	<code>int</code> (must be between 1 and 10)	Specifies the weight associated with timing-driven track assignment. The default is 1.

Table 8-6 *set_route_options Command Options (Continued)*

Option	Valid values	Description
Detail routing options (Detail Routing tab in the GUI)		
-droute_connect_tie_off (“Connect tie off” check box in the GUI)	true false	Enables (true) or disables (false) connection of tie-off nets during detail routing. The default is true.
-droute_connect_open_nets (“Connect open nets” check box in the GUI)	true false	Enables (true) or disables (false) connection of open nets during detail routing. The default is true.
-droute_reroute_user_wires (“Reroute user wires” check box in the GUI)	true false	Specifies whether the router can reroute user-created wires. The default is false.
-droute_CTS_nets (“Change CTS nets” radio buttons in the GUI)	normal minor_change_only	Specifies whether only minor changes can be made to clock nets. The default is minor_change_only.
-droute_single_row_column_via_array (“Single row column via array” radio buttons in the GUI)	center optimize	Specifies how to handle via arrays that consist of a single row and single column. The default is center.
-droute_stack_via_less_than_min_area (“Stack via less than min area” radio buttons in the GUI)	forbid add_metal_stub	Specifies how to handle stacked via arrays that do not meet the minimum area rule. The default is add_metal_stub.
-droute_stack_via_less_than_min_area_cost (“Stack via less than min area Cost” box in the GUI)	int (must be between 1 and 10)	Controls the cost associated with adding metal stubs. The default is 0.

Table 8-6 *set_route_options* Command Options (Continued)

Option	Valid values	Description
Miscellaneous options (Miscellaneous tab in the GUI)		
-poly_pin_access	auto off	Controls poly pin access is handled. The default is auto.
-drc_distance	diagonal manhattan	Specifies how to measure distances for DRC checking. The default is diagonal.
-same_net_notch	ignore check_and_fix	Specifies whether to check the same net notch rule. The default is ignore.
-fat_wire_check	quick merge_then_check	Controls how the fat wire check is handled. The default is merge_then_check.
-fat_blockage_as	thin_wire fat_wire	Specifies how to treat fat blockages. The default is fat_wire.
-wire_contact_eol_rule	ignore check_and_fix	Specifies whether check the end-of-line rule. The default is ignore.

Setting Signal Integrity Options

By default, the classic router does not perform crosstalk reduction. If you enable signal integrity mode, the classic router performs crosstalk reduction during global routing and track assignment. To enable signal integrity mode, enter the following command:

```
icc_shell> set_si_options -route_xtalk_prevention true
```

The default crosstalk prevention threshold is 0.35 volts, which is probably too relaxed. If your design has crosstalk violations, you should use the `set_si_options -route_xtalk_prevention_threshold` command to lower the crosstalk prevention threshold during track assignment to a value between in the range of 0.25 to 0.35 volts. When you set the crosstalk prevention threshold, IC Compiler automatically sets the `threshold_noise_ratio` common route option.

When you enable crosstalk prevention, the classic router avoids putting long, parallel wires on adjacent tracks during track assignment. To minimize noise, track assignment estimates the potential noise with a simplified crosstalk checker and reassigns wires to reduce the potential noise using the noise threshold.

Enabling Distributed Routing

You can use distributed routing to decrease the runtime for the following routing commands:

- `route_opt`
- `route_group`
- `route_auto`
- `route_detail`
- `route_search_repair`
- `route_eco`
- `route_spreadwires`
- `insert_redundant_vias`
- `optimize_wire_via`
- `verify_route`

Note:

You can also use distributed routing for the following power and ground routing commands: `create_power_straps` and `preroute_standard_cells`. These commands are described in the *IC Compiler User Guide: Design Planning*.

To perform distributed routing, you must

1. Have the appropriate licenses

To perform distributed routing using three or four CPUs, you must have the Galaxy-MultiRoute4 license. To perform distributed routing using more than four CPUs, you must have the Galaxy-MultiRoute8 license.

2. Set up the network

To set up the network for distributed routing, run the `set_distributed_route` command (or choose File > Distributed Route Jobs > Set Distributed Route in the GUI). For more information about setting up the network, see the following section, [“Setting Up the Distributed Routing Network.”](#)

3. Select the partitioning algorithm

Distributed routing divides the design into several routing partitions, based on size and congestion. These partitions are then routed on separate CPUs, in parallel, greatly reducing the overall runtime. One CPU is used to reassemble the partitions. For example, with four CPUs available, a typical runtime improvement would be a 3.5x reduction in overall routing time.

By default, the classic router uses a two-pass partitioning algorithm. If your design is very large, you might be able to reduce runtime by using the one-pass algorithm instead. To use the one-pass algorithm, set the `droute_enable_one_pass_partitioning` variable to 1.

4. Invoke distributed routing

To enable distributed routing for all routing commands that support distributed routing, set the `droute_numCPUs` variable to a number greater than one before running the routing commands.

To invoke distributed routing for a specific routing command, use the `-num_cpu` option to specify multiple CPUs when you run the routing command.

Note:

If you specify both the `droute_numCPUs` variable and the `-num_cpu` command option, the value specified by the `-num_cpu` option overrides the value specified by the `droute_numCPUs` variable. In addition, the number of CPUs specified must be less than or equal to the number of CPUs defined by the `set_distributed_route` command.

Setting Up the Distributed Routing Network

The classic router supports distributed routing operations by using either a package called *jp* to set up the communication between jobs on a network or by using resources under the control of the Load Sharing Facility (LSF) network. The following sections describe how to set up the network for distributed routing using each of these methods.

Setting Up Distributed Routing With jp

To set up distributed routing with the *jp* package,

1. If you are using processors on other machines, create a `.rhosts` file in your home directory.

The `.rhosts` file specifies the remote machines that you can access (one entry per line). You might need to specify the complete host name, including the domain name.

To determine the host name to put in the .rhosts file,

- a. Log in the machine you want to use by using the (`rlogin`) shell command.
- b. Determine the host name by using the `finger -l` shell command.

To be safe, you can include both the simple and the complete host name for each remote machine, as shown in the following sample .rhosts file.

```
machineName1
machineName1.domain.company.com
machineName2
machineName2.domain.company.com
```

2. Open the Milkyway design library, as described in [Chapter 3, "Preparing the Design."](#)
3. Run the `set_distributed_route` command to specify the following:

- The machines to use (`-jp_machines` option)

You can specify up to 16 machines, including the current machine.

If you are using only the current machine, you do not need to specify this option, because this is the default.

For example, to use *machineName1*, *machineName2*, and *machineName3* in addition to the current machine, enter

```
icc_shell> set_distributed_route \
    -jp_machines {machineName1 machineName2 machineName3}
```

- The maximum number of CPUs to use on each machine (`-jp_limit` option)

For each machine for which you want to limit the number of CPUs used, specify the machine name and the maximum number of CPUs. The machine name must match a machine specified in the `-jp_machines` option. If you do not specify a limit for a machine, distributed routing will use all CPUs on that machine.

For example, to use a maximum of two CPUs on each machine, enter

```
icc_shell> set_distributed_route \
    -jp_limit {machineName1 2 machineName2 2 machineName3 2}
```

- The location of the executable file for each machine (`-jp_bin` option)

By default, distributed routing uses the location of the current `icc_shell` executable. If the executable file for a remote machine is in a location other than the default location, you must specify the machine name and the executable location for that machine. The machine name must match a machine specified in the `-jp_machines` option.

For example, to specify the executable for *machineName1*, enter

```
icc_shell> set_distributed_route \
    -jp_bin {machineName1 /machineName1/$SYNOPTSYS/bin/icc_shell}
```

- The location of the Milkyway design library (`-jp_lib` option)

By default, distributed routing uses the location of the current Milkyway design library. If the design library for a remote machine is in a location other than the default location, you must specify the machine name and the design library location for that machine. The machine name must match a machine specified in the `-jp_machines` option.

Note:

When using the `jp` package, you can disconnect the network used for distributed routing by using the `set_distributed_route -jp_disconnect` option. This command disconnects all previously defined remote machines. You can also use the `close_distributed_route` command to close all the sockets and shut down the daemons.

Setting Up Distributed Routing With LSF

To set up distributed routing with LSF, run the `set_distributed_route` command to

- Select LSF as the distributed routing method (`-lsf` option)
- Specify the LSF `bsub` command options to use when submitting jobs (`-lsf_advanced` option)

By default, distributed routing with LSF does not use any `bsub` options. For more control over the job submissions, you can specify the `bsub` options to use. Common `bsub` options include `-q` (the queue to which to submit the jobs), `-m` (the machines to which to submit the jobs), and `-M` (the minimum memory required). For more information about the `bsub` command options, see the LSF documentation.

Reporting Distributed Route Jobs

To get information about a specific distributed route job, use the `report_distributed_route` command. You must use the `-job` option to identify the job to report. To get the job identification number, use a system command such as the UNIX `top` or `ps` command.

Cancelling Distributed Route Jobs

To cancel an active distributed route job, use the `remove_distributed_route` command. You must use the `-job` option to identify the job to cancel. To get the job identification number, use a system command such as the UNIX `top` or `ps` command.

Routing Critical Nets

You can preroute a group of nets, such as clock nets, before routing the rest of the nets in the design. Taking this step can help you find timing problems. For example, you can route clock and bus pins to prerouted clock and bus wires and then perform timing analysis on these nets. If the timing results are satisfactory, you can route the remaining nets.

To preroute a group of nets, use the `route_group` command (or choose Route > Net Group Route in the GUI). To route all clock nets, use the `-all_clock_nets` option (or select the “All clock nets” radio button in the GUI); otherwise, you can specify a collection of nets to route by using the `-nets` option (or the “Specified nets” box in the GUI).

For example, to route all clock nets, enter

```
icc_shell> route_group -all_clock_nets
```

Table 8-7 lists the options for the `route_group` command.

Table 8-7 route_group Command Options

Option	Valid values	Description
<code>-no_global</code> (“Global route” check box in the GUI)	N/A	Skips the global routing phase. By default, <code>route_group</code> performs global routing, track assignment, and detail routing.
<code>-no_track</code> (“Track assignment” check box in the GUI)	N/A	Skips the track assignment phase. By default, <code>route_group</code> performs global routing, track assignment, and detail routing.
<code>-no_detail</code> (“Detail route” check box in the GUI)	N/A	Skips the detail routing phase. By default, <code>route_group</code> performs global routing, track assignment, and detail routing.
<code>-nets</code> (“Specified nets” radio button and text box in GUI)	<i>collection_of_nets</i>	Specifies the nets to route. Either this option or <code>-all_clock_nets</code> is required.
<code>-all_clock_nets</code> (“All clock nets” radio button in the GUI)	N/A	Routes all clock nets. Either this option or <code>-nets</code> is required.

Table 8-7 *route_group Command Options (Continued)*

Option	Valid values	Description
-dont_optimize_routing_pattern (“Optimize routing pattern” check box in the GUI)	N/A	Disables optimization of the routing pattern. By default, the routing pattern is optimized.
-utilize_dangling_wires (“Utilize dangling wires” check box in the GUI)	N/A	Enables reuse of existing dangling routes to fix open nets. By default, the router does not reuse existing dangling routes.
-trim_antenna_of_user_wires (“Trim wire segments of prerouted wires” check box in the GUI)	N/A	Enables trimming for wire segments of prerouted wires. By default, the router does not trim wire segments of prerouted wires.
-num_cpus (“Clock routing” radio buttons in the GUI)	<i>int</i> (must be between 1 and 63)	Specifies the number of CPUs to use for distributed routing. For information about setting up for distributed routing, see “Enabling Distributed Routing” on page 8-15 . The default is 1.

Shielding Clock Nets

The classic router uses nondefault routing rules to define the shielding width and spacing. For information about nondefault routing rules, see [“Using Nondefault Routing Rules” on page 8-6](#).

Note:

If you do not define the shielding spacing for clock nets, the classic router adds default spacing as shield spacing on clock nets.

After defining the nondefault routing rules and routing the clock nets, shield the clock nets by using the `create_auto_shield` command (or choosing Route > Create Auto Shield in the GUI). For information about shielding nets, see [“Shielding Nets” on page 8-33](#).

Routing Signal Nets

Before you route the signal nets, all clock nets must be routed without violations.

You can route the signal nets by using one of the following methods:

- Use the `route_opt` core command.

The `route_opt` command performs global routing, track assignment, detail routing, search and repair, and postroute optimization.

- Use automatic routing (the `route_auto` basic command).

The `route_auto` basic command performs global routing, track assignment, and detail routing.

When you run `route_auto`, the classic router reads the design database before starting routing and updates the database when all routing steps are done. If you stop auto-routing before it performs detail routing, the classic router checks the input data when you restart routing with this command.

- Use the basic commands to perform the stand-alone routing tasks.

To perform global routing, use the `route_global` command. To perform track assignment, use the `route_track` command. To perform detail routing, use the `route_detail` command. To perform search and repair, use the `route_search_repair` command.

When you run a stand-alone routing command, such as `route_global` or `route_detail`, the classic router reads in the design database at the beginning of each routing command and updates the database at the end of each command. The router does not check the input data. For example, if the track assignment step is skipped and you run detail routing directly, the classic router might generate bad routing results.

If you need to customize your routing flow or you need to run a large design step-by-step, you might want to use the stand-alone routing commands instead of `route_opt` or automatic routing.

Routing Signal Nets by Using route_opt

Before you use the `route_opt` command to route the signal nets, you must define the routing options, as described in [“Setting Routing Options” on page 8-11](#) and set the `route_opt` routing and optimization strategy, as described in the following section.

Setting the Routing and Optimization Strategy

IC Compiler provides strategy controls that you set to guide how routing and postroute optimizations are run.

To set the routing and optimization strategy, use the `set_route_opt_strategy` command (or choose Route > Set Core Routing and Optimization Strategy in the GUI).

[Table 8-8](#) describes the `set_route_opt_strategy` options. For more information, see the man page.

Table 8-8 set_route_opt_strategy Options

Command option	Valid values	Description
<code>-fix_hold_mode</code> ("Fix hold optimization stage" radio buttons in the GUI)	<code>all</code> <code>route_base</code>	The stages for which hold optimization is performed. You can select prerouting, global routing, and detail routing (<code>all</code>) or global routing and detail routing (<code>route_base</code>). The default is <code>route_base</code> .
<code>-xtalk_reduction_loops</code> ("Maximum crosstalk reduction cycles" box in the GUI)	<code>int</code> (must be between -1 and 10)	The maximum number of crosstalk reduction optimization cycles. The default is 2.
<code>-search_repair_loops</code> ("Maximum initial route Search and Repair cycles" box in the GUI)	<code>int</code> (must be between 0 and 500)	The maximum number of initial routing search and repair iterations. The classic router default is 15.
<code>-eco_route_search_repair_loops</code> ("Maximum ECO Search and Repair cycles" box in the GUI)	<code>int</code> (must be between 0 and 500)	The maximum number of ECO search and repair iterations. The classic router default is 5.
<code>-optimize_wire_via_search_repair_loops</code> ("Maximum ECO Search and Repair cycles" box in the GUI)	<code>int</code> (must be between 0 and 500)	The maximum number of search and repair iterations for optimizing wires and vias. The classic router default is 5.

Table 8-8 *set_route_opt_strategy Options (Continued)*

Command option	Valid values	Description
-route_drc_threshold (“Route violations threshold to trigger the reduction Search and Repair loop” box in the GUI)	<i>int</i>	The threshold number of routing violations before limiting search-and-repair to one iteration. The default is 3000.
-route_run_time_limit (“Run time limit” box in the GUI)	<i>int</i> (must be between -1 and 500)	The maximum CPU time, in minutes, allowed for optimizing wires and vias. The default is -1, which means there is no limit.

To report your settings, use the `report_route_opt_strategy` command.

Running route_opt

To run routing and postroute optimization, use the `route_opt` command (or choose Route > Core Routing and Optimization in the GUI).

By default, the `route_opt` command performs the following tasks:

- Global routing

By default, global routing is not timing-driven. To enable timing-driven global routing, use the `set_route_options -groute_timing_driven true` command (or choose Route > Routing Setup > Set Route Options in the GUI and select “Timing driven” in the Global Routing tab).

- Track assignment

By default, track assignment is not timing-driven and does not do crosstalk prevention.

To enable timing-driven track assignment, use the `set_route_options -track_assign_timing_driven true` command (or choose Route > Routing Setup > Set Route Options in the GUI and select “Timing driven” in the Track Assign tab).

To enable crosstalk-prevention mode, set the signal integrity options as described in [“Setting Signal Integrity Options” on page 8-14](#) and use the `-xtalk_reduction` or `-only_xtalk_reduction` option when you run `route_opt`.

- Detail routing

By default, detail routing is not timing-driven. To enable timing-driven detail routing, use the `set_route_options -droute_timing_driven true` command (or choose Route > Routing Setup > Set Route Options in the GUI and select “Timing driven” in the Detail Routing tab).

- Search and repair
- Postroute optimization

In addition to performing routing optimization, you can perform the following optimizations: signal integrity, leakage power, wire length and via count reduction, and area recovery.

To enable signal integrity optimization, set the signal integrity options as described in [“Setting Signal Integrity Options” on page 8-14](#) and use the `-xtalk_reduction` option when you run `route_opt`.

To enable leakage power optimization, use the `-power` option when you run `route_opt`. For information about leakage power optimization, see [“Leakage Power Optimization” on page 6-3](#).

To enable wire and via optimization, use the `-optimize_wire_via` option when you run `route_opt`.

To enable area recovery, use the `-area_recovery` option when you run `route_opt`.

If you want to analyze the routing results before performing postroute optimization, first run `route_opt` without postroute optimization (`-initial_route_only` option), and then run `route_opt` with postroute optimization only (`-skip_initial_routing` option).

```
icc_shell> route_opt -initial_route_only

# analyze routing results

icc_shell> route_opt -skip_initial_routing
```


Saving Intermediate Results

To save intermediate `route_opt` results, enable checkpointing by setting the `routeopt_checkpoint` variable to true. When checkpointing is enabled, `route_opt` saves the design after each major stage. [Table 8-9](#) shows the designs saved by `route_opt` checkpointing.

Table 8-9 Designs Saved By route_opt Checkpointing

Design name	Saved after
<i>design_name_RT</i>	Initial routing
<i>design_name_ORPRT</i>	Wire and via optimization
<i>design_name_OR_n</i>	Each postroute optimization pass
<i>design_name_ECORT_n</i>	Each ECO routing pass
<i>design_name_RCRED</i>	Crosstalk reduction
<i>design_name_POR</i>	Power optimization
<i>design_name_PECORT</i>	Power optimization ECO routing
<i>design_name_HOLD_PRE</i>	Preroute-based hold optimization
<i>design_name_HOLD</i>	Global-route-based hold optimization
<i>design_name_INCOR</i>	Optimization after incremental <code>route_opt</code>
<i>design_name_HOLD</i>	ECO routing after incremental <code>route_opt</code>

Summary of route_opt Options

Table 8-10 describes all the `route_opt` options. For more information, see the man page.

Table 8-10 *route_opt Command Options*

Command option	GUI object	Description
<code>-effort low medium high</code>	Effort radio buttons	Specifies the optimization effort level. Higher effort levels provide more aggressive optimization at a runtime cost. The default is <code>medium</code> .
<code>-stage global track detail</code>	"Run optimization after" radio buttons	Specifies the last routing stage performed by <code>route_opt</code> before performing optimization. By default, <code>route_opt</code> performs optimization after detail routing.
<code>-xtalk_reduction</code>	"Crosstalk reduction and SI optimization" radio button	Performs crosstalk reduction and signal integrity optimization. By default, <code>route_opt</code> does not perform crosstalk reduction or signal integrity optimization.
<code>-only_xtalk_reduction</code>	"Crosstalk reduction optimization only" radio button	Performs only crosstalk reduction (not signal integrity optimization). By default, <code>route_opt</code> does not perform crosstalk reduction or signal integrity optimization.
<code>-power</code>	"Perform power optimization" check box	Performs leakage power optimization. By default, <code>route_opt</code> does not perform leakage power optimization.
<code>-skip_initial_route</code>	"Skip initial routing" check box	Performs postroute optimization without routing.
<code>-initial_route_only</code>	"Initial routing only" check box	Performs only initial routing without postroute optimization.
<code>-size_only</code>	"Sizing only optimization" check box	Resolves setup time violations by sizing only.
<code>-optimize_wire_via</code>	"Optimize wire and via routing" check box	Performs wire and via optimization. By default, <code>route_opt</code> does not perform wire and via optimization.

Table 8-10 *route_opt* Command Options (Continued)

Command option	GUI object	Description
-area_recovery	"Recover area for cells that are not on critical timing paths" check box	Recovers area for cells not on critical paths. To restrict area recovery to high-density areas, set the <code>physopt_density_area_recovery</code> variable to true.
-wire_size	"Use wire sizing to fix setup time violations" check box	Determines whether wire sizing is used to fix setup time violations.
-incremental	"Incremental mode" check box	Performs incremental postroute optimization.
-incremental -only_wire_size	"Timing optimization only with wire size" radio button	Performs only wire sizing to resolve setup time violations during incremental mode.
-incremental -only_hold_time	"Hold timing optimization only" radio button	Resolves only hold time violations during incremental mode. Use the <code>set_fix_hold</code> command to enable hold time fixing.
-incremental -only_area_recovery	"Area recovery only" radio button	Performs only area recovery during incremental mode. To restrict area recovery to high-density areas, set the <code>physopt_density_area_recovery</code> variable to true.
-incremental -only_design_rule	"Fix design rules only" radio button	Performs only logical design rule fixing during incremental mode.
-incremental -only_power_recovery	"Power recovery only" radio button	Performs only power recovery during incremental mode.
-num_cpus (<i>"Number of CPUs"</i> box in the GUI)	<i>int</i> (must be between 1 and 63)	Specifies the number of CPUs to use for distributed routing. For information about setting up for distributed routing, see "Enabling Distributed Routing" on page 8-15 . The default is 1.

Routing Signal Nets by Using Automatic Routing

Before you use automatic routing to route the signal nets, you must define the routing options as described in [“Setting Routing Options” on page 8-11](#).

To run automatic routing, use the `route_auto` command (or choose Route > Auto Route in the GUI). By default, the `route_auto` command sequentially invokes global routing, track assignment, and detail routing.

[Table 8-11](#) describes the `route_auto` command options. For more information, see the `man` page.

Table 8-11 route_auto Command Options

Command option	Valid values	Description
<code>-no_global</code> ("Global route" check box in Run section in the GUI)	N/A	Skips global routing.
<code>-no_track</code> ("Track assignment" check box in Run section in the GUI)	N/A	Skips track assignment.
<code>-no_detail</code> ("Detail route" check box in Run section in the GUI)	N/A	Skips detail routing.
<code>-save_after_global_route</code> ("Global route" check box in Run section in the GUI)	N/A	Controls whether the design is saved after global routing. When specified, the tool saves the design in a CEL view named <code>auto_GR</code> .
<code>-save_after_track</code> ("Track assignment" check box in Run section in the GUI)	N/A	Controls whether the design is saved after track assignment. When specified, the tool saves the design in a CEL view named <code>auto_TA</code> .
<code>-save_after_detail_route</code> ("Detail route" check box in Run section in the GUI)	N/A	Controls whether the design is saved after detail routing. When specified, the tool saves the design in a CEL view named <code>auto_DR</code> .

Table 8-11 *route_auto* Command Options (Continued)

Command option	Valid values	Description
-search_repair_loop (“Maximum Search & Repair cycles” box in the GUI)	<i>int</i> (must be between 0 and 500)	Specifies the maximum number of search and repair loops. The default is 0.
-effort (Effort radio buttons in the GUI)	minimum low medium high	Specifies the effort level for auto-routing. The default is <i>medium</i> .
-num_cpus (“Number of CPUs” box in the GUI)	<i>int</i> (must be between 0 and 500)	Specifies the number of CPUs to use for distributed routing. For information about setting up for distributed routing, see “Enabling Distributed Routing” on page 8-15 . The default is 1.

Running Individual Routing Steps

After you have specified your routing options, you can debug your design or monitor each routing step by performing each routing step individually. Individual routing steps are explained in the following sections:

- [Global Routing](#)
- [Track Assignment](#)
- [Detail Routing](#)
- [Search and Repair](#)
- [Wire and Via Optimization](#)

Global Routing

Global routing maps general pathways through the design for each unrouted net (signal nets and clock nets). The global router uses a two-dimensional array of global routing cells to model the demand and capacity of global routing. The classic router uses the average height of the standard cells to create the height and width of each global routing cell.

During global routing, the classic router assigns nets to the global routing cells through which they pass. For each global routing cell, the routing capacity is calculated according to the blockages, pins, and routing tracks inside the cell. Although the nets are not assigned to the actual wire tracks during global routing, the number of nets assigned to each global routing

cell is noted. The classic router calculates the demand for wire tracks in each global routing cell and reports the overflows, which are the number of wire tracks still needed after the classic router assigns nets to the available wire tracks in a global routing cell.

The classic router might reduce overflows by detouring nets around congested areas and increasing the wire length. You can examine the global routing report that appears in the command window and display congestion maps to help you decide whether your design can be routed.

The global router considers spacing and wide-wire variable routing rules, as well as shielding variable routing rules, when calculating congestion.

To perform global routing, use the `route_global` command (or choose Route > Global Route in the GUI). By default, the classic router performs medium-effort global routing. To change the effort level, use the `-effort` option.

Viewing the Layer-Based Congestion Map

The classic router generates a layer-based congestion map. You can simultaneously see the congestion for one or more layers. To view a layer-based congestion map,

1. Choose Route > Global Route Congestion Map.

The layout display changes to a congestion map of your design, and the Map Mode dialog box appears.

2. By default, the congestion map is displayed for all metal layers. To display the congestion map for a subset of layers, select (or deselect) the layers in the Map Mode dialog box.
3. Hide or display buckets of congestion by doing one of the following:
 - To display a bucket of congestion, select it in the Map Mode dialog box.
 - To hide a bucket of congestion, deselect it in the Map Mode dialog box.
4. When you have finished using the congestion map, hide it by choosing Route > Global Route Congestion Map.

If you need to display a congestion map only, use the `-congestion_map_only` option.

Track Assignment

Before you perform detail routing, run track assignment to specify which tracks within each global routing cell are to be used for each net. Track assignment operates on the entire design at once; it can make long routes straight and reduce the number of vias, whereas the detail routing routes a small area at a time.

After track assignment finishes, all nets are routed, but not very carefully. There are many violations, particularly where the routing connects to pins. Detail routing works to correct those violations.

To perform track assignment, use the `route_track` command (or choose Route > Track Assignment in the GUI).

Detail Routing

Detail routing uses the general pathways suggested by global routing and track assignment to route the nets (paths and contacts).

To perform detail routing, use the `route_detail` command (or choose Route > Detail Route in the GUI).

By default, stand-alone detail routing

- Performs track assignment before detail routing

To skip track assignment, use the `-track_assign skip` option.

- Does not have a runtime limit

If you need to limit the runtime, specify the maximum number of minutes by using the `-run_time_limit` option.

- Uses one CPU

If you are performing distributed routing, specify the number of CPUs by using the `-num_cpus` option. For information about setting up for distributed routing, see [“Enabling Distributed Routing” on page 8-15](#).

- Does not perform any search and repair passes

To change the maximum number of search and repair passes, use the `-search_repair_loop` option.

Search and Repair

During search-and-repair routing passes, the classic router searches for DRC violations and reroutes wires in an effort to avoid violations. The classic router does not add metal stubs on frozen nets to fix DRC violations, even when the nets have DRC violations.

To run stand-alone search-and-repair routing, use the `route_search_repair` command (or choose Route > Search-and-Repair Route in the GUI). By default, the `route_search_repair` command performs 50 search-and-repair passes. To change the maximum number of passes, use the `-loop` option.

If you need to limit the search-and-repair runtime, specify the maximum number of minutes by using the `-run_time_limit` option. You can reduce the turnaround time for running search-and-repair routing by using distributed routing. To use distributed routing, specify the number of CPUs by using the `-num_cpus` option. For information about setting up for distributed routing, see [“Enabling Distributed Routing” on page 8-15](#).

For more information about the `route_search_repair` command, see the man page.

Wire and Via Optimization

To optimize wires and vias, use the `optimize_wire_via` command (or choose Route > Optimize Wire Via in the GUI). By default, the `optimize_wire_via` command performs 20 search-and-repair passes. To change the maximum number of search-and-repair passes, use the `-search_repair_loop` option.

If you need to limit the wire and via optimization runtime, specify the maximum number of minutes by using the `-run_time_limit` option. You can reduce the turnaround time for running wire and via optimization by using distributed routing. To use distributed routing, specify the number of CPUs by using the `-num_cpus` option. For information about setting up for distributed routing, see [“Enabling Distributed Routing” on page 8-15](#).

For more information about the `optimize_wire_via` command, see the man page.

Shielding Nets

The router shields routed nets by generating shielding wires that are based on the shielding widths and spacing defined in the shielding rules.

Before you shield nets, you must define the shielding rules with the `define_routing_rule` command and assign these rules to the nets to be shielded with the `set_net_routing_rule` command. (See [“Using Nondefault Routing Rules” on page 8-6.](#))

To shield nets with defined rules, use the `create_auto_shield` command (or choose Route > Create Auto Shield in the GUI).

By default, the `create_auto_shield` command performs same-layer shielding on all nets with predefined shielding rules. The shielding wires are tied to ground, standard cell power and ground pins, and standard cell rails.

To explicitly specify the nets on which to perform shielding, use the `-nets` option. To perform coaxial shielding, use the `-coaxial_above` and `-coaxial_below` options. To tie the shielding wires to a named power or ground net, use the `-with_ground` option. To prevent connections to the standard cell power and ground pins, use the `-ignore_shielding_net_pins` option. To prevent connections to the standard cell rails, use the `-ignore_shielding_net_rails` option.

Performing ECO Routing

Whenever you modify the nets in your design, you need to run engineering change order (ECO) routing to reconnect the routing.

To run ECO routing, use the `route_eco` command (or choose Route > ECO Route in the GUI).

By default, the ECO router connects open nets and then fixes design rule violations in the design. All the open nets are run sequentially through global routing, track assignment, and detail routing. The ECO router can reroute existing wires to fix DRC violations. You can modify the default behavior by using the `route_eco` command options.

Table 8-12 describes the `route_eco` command options. For more information, see the man page.

Table 8-12 *route_eco* Command Options

Command option	Valid values	Description
<code>-no_global</code> ("Global route" check box in Run section in the GUI)	N/A	Skips global routing.
<code>-no_track</code> ("Track assignment" check box in Run section in the GUI)	N/A	Skips track assignment.
<code>-no_detail</code> ("Detail route" check box in Run section in the GUI)	N/A	Skips detail routing.
<code>-auto</code> (Auto check box in the Run section in the GUI)	N/A	Enables automatic region-based ECO routing. You must use this option with the <code>-region_based</code> option.
<code>-region_based</code> ("Region based nets filename" box in the GUI)	<i>string</i>	Specifies the name of the file that contains the list of nets on which to perform region-based ECO routing. You must use this option with the <code>-auto</code> option.
<code>-search_repair_loop</code> ("Maximum Search & Repair cycles" box in the GUI)	<i>int</i> (must be between 0 and 500)	Specifies the maximum number of search and repair loops. The default is 20.
<code>-reroute</code> ("Reroute nets" radio buttons in the GUI)	<i>any_nets</i> <i>modified_nets_only</i> <i>modified_nets_first</i> <i>then_others</i>	Controls which nets can be rerouted to fix DRC violations. The default is <i>any_nets</i> .
<code>-utilize_dangling_wires</code> ("Utilize dangling wires" check box in the GUI)		Controls whether the router tries to reuse existing dangling routes to fix open nets. The default is true.
<code>-freeze_routing_on_layer</code> ("Freeze routing on layers" list in the GUI)	<i>list_of_layers</i>	Prevents routing changes on the specified layers.

Table 8-12 *route_eco* Command Options (Continued)

Command option	Valid values	Description
<code>-freeze_vias_on_frozen_metal</code> ("Freeze vias on frozen metal" check box in the GUI)	N/A	Freezes vias on frozen metal layers.
<code>-num_cpus</code> ("Number of CPUs" box in the GUI)	<i>int</i> (must be between 1 and 500)	Specifies the number of CPUs to use for distributed routing. For information about setting up for distributed routing, see "Enabling Distributed Routing" on page 8-15 . The default is 1.

Reporting Cell Placement and Routing Statistics

You can report on cell placement and routing statistics to help you determine whether to perform further optimizations to improve timing. For example, if the statistics show that an area has high congestion, an additional optimization might not have room to add or size up standard cells.

To view the place and route summary report, run the `report_design -physical` command (or choose Design > Report Design in the GUI, and then select "Show physical").

Verifying the Routed Design

After you have completed routing with the classic router, you can check the design for routing design rule violations.

Note:

The classic router uses center lines to determine connectivity. If your design contains wires with zero wire extension, the `verify_route` command might report open and spacing violations due to these zero extension wires. To fix this problem, use the `convert_wire_ends` command to convert all wires with zero extension into wires with half-width extensions. If the `convert_wire_ends` command cannot convert a wire, it issues a warning message and removes the wire from the design cell. Because the `convert_wire_ends` command modifies the wire end information and, in some cases, removes wires, write commands such as `write_def` generate output files that are different from the original input files. For more information about the `convert_wire_ends` commands, see the man page.

IC Compiler provides the following methods for verifying the routed design:

- Use the Hercules tool to check the routing design rules defined in the foundry runset.

To use this method, run the `signoff_drc` command or choose Verification > Signoff DRC in the GUI.

Note:

A Hercules license is required to run the `signoff_drc` command.

- Use the Hercules tool to check the routing design rules defined in the Milkyway technology file.

To use this method, run the `verify_drc` command or choose Verification > DRC in the GUI.

Note:

A Hercules license is required to run the `verify_drc` command.

- Use the IC Compiler DRC checking engine to check the routing design rules defined in the Milkyway technology file.

To use this method, run the `verify_route` command or choose Route > Verify Route in the GUI.

- Import the DRC checking results from the Calibre tool.

The following sections describe these methods.

Using the `signoff_drc` Command

The `signoff_drc` command performs routing DRC checking by using the Hercules tool with the foundry runset and reports all the errors in the top-level design by expanding the lower-level errors without merging them. A Hercules license is required to run the `signoff_drc` command. For more information about Hercules, see the Hercules documentation, which is available on SolvNet.

To use `signoff_drc` to perform DRC checking,

- Set up the Hercules environment.
- Set up the physical signoff options.
- Run the `signoff_drc` command.

The following sections describe these tasks.

Setting Up the Hercules Environment

The `signoff_drc` command runs the Hercules tool. In order to do this, you must have a Hercules license, and you must specify the location of the Hercules executable by setting the `HERCULES_HOME_DIR` environment variable. You can set this variable in your `.cshrc` file. For example,

```
setenv HERCULES_HOME_DIR /root_dir/hercules_2006.12_SP2-4
set path = ($path $HERCULES_HOME_DIR/bin/AMD.64)
```

You must ensure that the version of the Hercules executable that you specify is compatible with the version of IC Compiler that you are using.

Setting Up The Physical Signoff Options

To set up the physical signoff options, use the `set_physical_signoff_options` command (or choose Verification > Set Physical Signoff Options in the GUI).

[Table 8-13](#) shows the physical signoff options that apply to the `signoff_drc` command.

Table 8-13 Physical Signoff Options for `signoff_drc`

Command option	Description
<code>-drc_runset filename</code> (DRC box in GUI “Foundry runset” section)	Specifies the foundry runset to use for DRC checking (required).
<code>-mapfile filename</code> (“Layer mapping file” box in GUI)	Specifies the layer mapping file (optional). For more information, see “Defining the Layer Mapping Between Milkyway and the Hercules Runset.”
<code>-dp_hosts {host_list}</code> (Hosts box in GUI)	Defines the distributed processing options (optional). For more information, see “Performing Distributed DRC Checking.”
<code>-num_cpus int</code> (“Number of CPU’s” box in GUI)	

To report the current settings for the physical signoff options, use the `report_physical_signoff_options` command.

Defining the Layer Mapping Between Milkyway and the Hercules Runset

In general, the Milkyway technology file and the Hercules runset use the same layer numbers. If they do not, you must supply a layer mapping file to map the Milkyway layers to the Hercules layers.

The `signoff_drc` command accepts a layer mapping file in either Hercules or Milkyway format.

The syntax of the mapping information in a Hercules mapping file is:

```
Milkyway_layer_list > Hercules_layer_list
```

The rules for specifying the layer lists are

- Each layer in a layer list must be separated by a space. Or, if specified in a range, the range must consist of a starting layer and an ending layer joined by a dash (-).
- Generally, parentheses should surround the layer list. However, you can omit parentheses if you specify only one layer.

You can include comments in a layer mapping file by preceding the comment with a pound sign (#). All text that follows a pound sign on a given line is a comment.

The syntax of the mapping information in a Milkyway mapping file is:

```
Milkyway_layer Hercules_layer
```

Performing Distributed DRC Checking

By default, `signoff_drc` uses a single process to perform DRC checking. To reduce the runtime used for DRC checking, you can use distributed processing. To enable distributed processing, you must specify the number of CPUs to use (`-num_cpus` option) and identify the CPUs to be used (`-dp_hosts` option). The number of CPUs that you specify must be less than or equal to the number of hosts that you specify. To use multiple CPUs on a given host, you must include the host name multiple times in the host list.

Running the `signoff_drc` Command

By default, the `signoff_drc` command uses the FRAM view to check all routing layers of the entire chip for all rules specified in the foundry Hercules runset. To use the CEL view instead of the FRAM view, use the `-read_cell_view` option or select the “Cell view” radio button in the GUI.

Note:

The `signoff_drc` command uses the on-disk design information, not the design information in memory. You must save the current state of the design before running the `signoff_drc` command.

You can restrict the DRC checking to specific areas of the chip or to specific design rules.

- To restrict the checking to specific areas of the chip,
 - If you are using the command line, use the `-bounding_box` option to specify the coordinates of each area. You can specify multiple areas by specifying the coordinates for each area.
 - If you are using the GUI, select the “Areas” check box and specify the coordinates for each area in the “Bounding Boxes” list. To specify an area, either draw the bounding box or enter the x- and y-coordinates of the box. For each area, you can also specify that the bounding box should snap to the minimum grid (the default), placement site, routing track, middle routing track, or user grid.
- To restrict the checking to specific rules,
 - To check only the specified rules, specify the rules in the `-select_rule` option (or in the Pattern box in the “To use” section of the Rules area in the GUI).
 - To check all rules except the specified rules, specify the rules in the `-unselect_rule` option (or in the Pattern box in the Excluded section of the Rules area in the GUI).

In either case, you specify the rules by specifying a matching pattern for the rule names. The rule names are specified in the COMMENT section in the Hercules runset file.

After you complete DRC checks on the routing layers, you can perform a quick DRC check on the Milkyway database by rerunning the `signoff_drc` command with the `-check_all_layers` option (“Check DRC violations on all layers” check box in the GUI). When you use this option, the design information comes from the CEL view and Hercules checks all layers, including the nonrouting layers.

The `signoff_drc` command creates a Milkyway error cell file that you can use to report or display the DRC violations. By default, the error cell file generated by `signoff_drc` is called *design_sdrc.err*. You can control this name by using the `-error_cell` option (or the “Error cell name” box in the GUI). For information about using the error cell file to debug DRC violations, see [“Analyzing DRC Violations” on page 8-44](#)

In addition to the error cell file, the `signoff_drc` command generates the following files, which you can use for debugging, and places them in a `signoff_drc_run` directory under the current working directory:

- *design.errsum* or *design.LAYOUT_ERRORS*

These files contain an error summary report.

- *sdrc.ev*

This file contains the Hercules runset file, including any options added by the `signoff_drc` command.

- `sdrc.log`

This file contains any errors that occurred during the `signoff_drc` run.

- `./run_details/design.sum`

This file contains the detailed Hercules log file.

Using the `verify_drc` Command

If you do not have a foundry runset, you can use the `verify_drc` command to perform DRC checking with a runset based on the design rules defined in the Milkyway technology file.

Note:

The `verify_drc` command does not check design rules for technologies at the 45-nm process node and below. To check design rules for these technologies, you must use the `signoff_drc` command.

For information about the routing design rules and the technology file attributes that define them, see the *IC Compiler Reference Manual: Technology File and Routing Rules*.

The `verify_drc` command generates a Hercules DRC runset based on the design rules defined in the Milkyway technology file, and then runs Hercules to check for DRC violations. A Hercules license is required to run the `verify_drc` command. For more information about Hercules, see the Hercules documentation, which is available on SolvNet.

To use `verify_drc` to perform DRC checking,

- Set up the Hercules environment

For information about setting up the Hercules environment, see [“Setting Up the Hercules Environment” on page 8-37](#).

- Run the `verify_drc` command.

By default, the `verify_drc` command checks and reports the DRC violations for the entire design. To perform these checks only in a specific region, specify the coordinates of the region by using the `-selected_area` option (or by specifying the coordinates in the “Selected area” box in the “DRC Area” tab in the GUI). To perform these checks for the entire design, except for a specific region, specify the coordinates of the excluded region by using the `-excluded_area` option (or by specifying the coordinates in the “Whole chip except area” box in the “DRC Area” tab in the GUI).

Table 8-14 lists the design rules checked by the `verify_drc` command.

Table 8-14 Design Rules Checked by `verify_drc`

Rule	Checked by default	Command option to enable or disable/ Check box in the “DRC Rules” tab in GUI
Minimum and maximum width	Yes	<code>-ignore_width</code> “Minimum and maximum width rule” check box
Metal spacing (includes minimum spacing, same-net spacing, fat metal spacing, the fat metal parallel length rule, and the fat metal extension range rule)	Yes	<code>-ignore_spacing</code> “All spacing related rules” check box
Minimum area	Yes	<code>-ignore_area</code> “Minimum area” check box
Metal density	Yes	<code>-ignore_density</code> “Metal density rules” check box
Minimum enclosed area (default and fat metal)	Yes	<code>-ignore_enclosed_area</code> “Basic minimum enclosed area rule and fat table minimum enclosed area rule” check box
Minimum number of consecutive short edges	Yes	<code>-ignore_min_edge_length</code> “Minimum number of consecutive short edges rule”
B lockages on metal layers	No	<code>-check_blockage</code> “Blockages on metal layers” check box
Via size rule	No	<code>-check_via_size</code> “Via size rule” check box
Via spacing (includes minimum spacing, the fat contact rule, the fat contact spacing rule, and the fat contact extension range rule)	Yes	<code>-ignore_via_spacing</code> “Via spacing rules” check box
Maximum number of via stack layers	Yes	<code>-ignore_stack_level</code> “Maximum number of stack layer rule” check box

Table 8-14 Design Rules Checked by `verify_drc` (Continued)

Rule	Checked by default	Command option to enable or disable/ Check box in the “DRC Rules” tab in GUI
Stack via rule	Yes	<code>-ignore_stackable</code> “Stack via rule” check box
Maximum number of adjacent vias and enclosed via rules	Yes	<code>-ignore_adjacent_via</code> “Maximum number of adjacent via and enclosed via rules” check box
Minimum number of fat vias	Yes	<code>-ignore_min_via_number</code> Minimum number of vias defined in the fat contact table (including extension range” check box
Via array size and spacing between via arrays	No	<code>-check_via_farm</code> “Via array size and spacing between two via arrays” check box
Via enclosure rule	No	<code>-check_enclosure</code> “Via Enclosure rule” check box
End-of-line rule	No	<code>-check_end_of_line</code> “End of line rule” check box
Fat poly contact rule	No	<code>-check_fat_poly_contact</code> “Fat poly contact rule” check box

The Hercules runset file generated by `verify_drc` is saved in `adrc/adrc.ev`. You can change the directory to which the runset file is saved by using the `-dir` option (or the “Hercules runset directory” box in the GUI).

The `verify_drc` command creates a Milkyway error cell file that you can use to report or display the DRC violations. By default, the `verify_drc` command generates an error cell file called `design_adrc.err`. You can control this name by using the `-error_cell` option (or the “Error cell name” box in the GUI). For information about using the error cell file to debug DRC violations, see See [“Analyzing DRC Violations” on page 8-44](#).

Using the `verify_route` Command

The `verify_route` command checks for routing DRC violations by running an internal DRC checker that checks the design rules defined in the Milkyway technology file. The `verify_route` command also checks for open nets.

Note:

Unlike the `signoff_drc` and `verify_drc` commands, which check the design rules for all objects, the `verify_route` command checks only objects with a signal routing type. For more information about routing types, see [“Setting Routing Types” on page 8-11](#).

For information about the routing design rules and the technology file attributes that define them, see the *IC Compiler Reference Manual: Technology File and Routing Rules*.

By default, the `verify_route` command checks and reports the DRC violations and open nets for the entire design. To perform these checks only in a specific region, specify the coordinates of the region by using the `-bounding_box` option (or by specifying the coordinates in the “DRC Area” area in the GUI). To prevent checking of the DRC violations, use the `-no_drc` option when you run `route_verify`. To prevent checking for open nets, use the `-no_opens` option when you run `route_verify`.

You can reduce the `verify_route` runtime by using distributed processing. For information about how to use distributed processing, see [“Enabling Distributed Routing” on page 8-15](#).

You can also use the `route_verify` command to check for antenna violations. For more information about checking for and fixing antenna violations, see [“Preventing Antenna Problems” on page 9-6](#).

The `verify_route` command can create a Milkyway error cell file that you can use to report or display the DRC violations. To output a Milkyway error cell file, use the `-output filename` option when you run the `verify_route` command. For information about using the error cell file to debug DRC violations, see [“Analyzing DRC Violations” on page 8-44](#).

Using the Calibre Interface

You can perform DRC checking with the Calibre tool, convert the Calibre DRC error file to a Milkyway error cell file, and then report or view the errors in IC Compiler. To convert the Calibre DRC error file to a Milkyway error cell file, use the `read_drc_error_file` command (or choose Verification > Read Third-party DRC Error File in the GUI).

For example,

```
icc_shell> read_drc_error_file Calibre_error_file
```

By default, the command generates a Milkyway error cell file called *cell_name.err*, where *cell_name* is derived from the Calibre error report. You can specify a name for the Milkyway error cell file by using the `-error_cell` option.

Note:

The `read_drc_error_file` command supports only flat Calibre DRC error files; it does not support Calibre hierarchy DRC error files.

You can load the Milkyway error cell file created by the `read_drc_error_file` command into the error browser to report or display the DRC violations. For information about using the error browser, see [“Examining Routing and Verification Errors” on page A-47](#).

Analyzing DRC Violations

After you have generated an error cell file, either by running one of the Synopsys DRC checking commands or by converting the Calibre results to an error cell file, you can analyze the DRC violations by

- Generating a report that lists the location of each error.

To report the location of each violation, use the `report_error_coordinates` command.

- Viewing the errors in the error browser.

You select the errors you need to examine by error type or layer, view error information in the error browser, and view error locations in the layout view. You can filter the error list, mark errors as fixed, highlight errors in the layout view, and select errors interactively with the Error Selection tool. You can also save the errors in a file.

For detailed information about using the error browser, see [“Examining Routing and Verification Errors” on page A-47](#) and the “Examining Routing and Verification Errors” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

Routing Nets and Buses in the GUI

The layout editor in the IC Compiler GUI provides the following routing capabilities:

- [Routing Single Nets](#)
- [Creating Physical Buses](#)
- [Modifying Buses](#)
- [Routing Buses](#)
- [Creating Custom Wires](#)

The following sections describe these capabilities.

Routing Single Nets

You can route single nets interactively in the active layout view by drawing the routes with the Route tool. A route can be composed of one or more horizontal or vertical segments.

To create routes interactively in the active layout view,

1. (Optional) Select a net shape, pin, port, terminal, user shape, or via to identify the net you need to route. If you perform this step, the tool operates in preselection mode.

2. Click the Route tool icon () on the Edit toolbar or choose Edit > Create > Route Tool.

The Create Route Tool dialog box appears. If you preselected the net, the GUI automatically sets the net name, layer, and width options.

3. (Optional) Set options as needed in the Create Route Tool dialog box.

You can set options to

- Draw either one or two route segments between two points in the layout view.

By default, the tool draws a single set of orthogonal segments between two points. You can select an angle mode that permits the tool to draw two sets of segments and controls which segments (horizontal, vertical, or longest) are drawn first.

- Enable or disable grid snapping and select the snap grid.

By default, grid snapping is enabled and the route segments snap to routing track edges.

- Select an end cap style.

The default end cap style is half width.

- Select the route type.

The default route type is signal.

- Select whether to merge connected collinear net shapes, display port guides, display display flylines, and automatically align the pin connections.

If you need to reset the options to their default values, click Default.

4. Click in the layout view where you want to begin the route.

If you are operating the tool in postselection mode, you must identify the net that you need to route by clicking a net shape, pin, port, terminal, user shape, or via at the starting point.

The object you click changes to the edit color (red by default). When you move the pointer, a ghost image appears between the initial point and the current pointer location. The color of the ghost image indicates the layer on which you are drawing the route.

5. (Optional) Click a location if you need to change the routing direction or the layer.

Repeat this step as needed.

- To change the layer, press F1 repeatedly until the ghost shape color matches the layer color.
- To remove the last point, right-click and choose Remove Last Point.
- To remove the entire route and start over, right-click and choose Cancel.

6. Finish the route by doing one of the following:

- Right-click and choose Apply.
- Double-click the endpoint object for the route.
- Click Apply in the Create Route Tool dialog box.

7. (Optional) To draw additional routes, repeat steps 3 through 6.

8. To deactivate the Route tool, press the Escape key or click OK in the Create Route Tool dialog box.

Alternatively, you can create routes by using the `create_net_shape` command with the `-type wire` option.

For more information about using the Route tool, see the “Routing Single Nets” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

Creating Physical Buses

The layout editor can create physical buses in the following ways:

- Manual bus creation

To manually create a physical bus, you specify and sort the nets that you want to include in the bus.

- Automatic bus creation

To have the layout editor automatically create a physical bus, you define a net name pattern, and then the tool constructs the bus from nets with matching names.

Manual Bus Creation

You can manually create a physical bus by specifying and sorting the nets that you want to include in the bus. You can specify the nets by selecting them, searching for them by name, or loading their names from a file. You can also set net order and precedence options and select the index delimiter.

To manually create a physical bus,

1. Choose Edit > Advanced > Physical Buses.

The Physical Buses dialog box appears. If your design already contains physical buses, the bus names appear in the “Bus name” list.

2. Click the Create button.

The Create Physical Bus dialog box appears.

3. Type a unique bus name in the “Bus name” box.

4. Set the net insertion and sorting options as needed.

5. Specify the nets that you want to include in the bus.

You can search for the nets by name, insert selected nets, or load the nets from a file.

The names of the specified nets appear in the “Net name” list.

6. (Optional) Sort or remove nets in the “Net name: list as needed.

7. Click OK or Apply.

The tool displays the bus name in the Physical Buses dialog box.

Alternatively, you can use the `create_physical_bus` command.

For more information about manually creating a physical bus, see the “Creating Buses” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

Automatic Bus Creation

You can automatically create a physical bus by defining a net name pattern and letting the tool construct the bus from nets with matching names. You define the net name pattern by specifying a name prefix and selecting one or more index delimiters. You can also set the index order, ascending or descending, and bus size constraint options.

To automatically create a physical bus,

1. Choose Edit > Advanced > Physical Buses.

The Physical Buses dialog box appears. If your design already contains physical buses, the bus names appear in the “Bus name” list.

2. Click the Auto Define button.

The Auto Define Buses dialog box appears.

3. Type a prefix in the “Net name prefix” box.

4. Select a net order option.

The choices are “ascending” and “descending.” The default is “ascending.”

5. Select or deselect bus index delimiter options under “Accept net name patterns” as needed to match the names of the nets that you want to include in the bus.

By default, all of the bus delimiter options are selected. To exclude nets with names that include specific delimiters, deselect the options for those delimiters. The available delimiters are braces ({i}), square brackets ([i]), angle brackets (<i>), underlines (_i_), colons (:i:), and parentheses ((i)).

6. Set the bus size constraint options as needed.

- To set the minimum number of nets in the bus, select or type a value in the Min box. The default value is 8.
- To set the maximum number of nets in the bus, select or type a value in the Max box. The default value is 128.
- To set the number of signals permitted in the bus, select or type a value in the “Multiple of” box. The default value is 1.

7. Click OK or Apply.

Alternatively, you can use the `create_physical_buses_from_patterns` command.

For more information about automatically creating a physical bus, see the “Defining Buses Automatically” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

Modifying Buses

You can modify a physical bus by specifying and sorting the nets that you want to include in the bus. You can specify the nets by selecting them, searching for them by name, or loading their names from a file. You can also set net order and precedence options and select the index delimiter.

To modify a physical bus,

1. Choose Edit > Advanced > Physical Buses.

The Physical Buses dialog box appears. The names of the existing buses appear in the “Bus name” list, and the names of their constituent nets appear in the “Net name” list.

2. Select the bus that you want to modify from the “Bus name” list.
3. Click the Modify button.

The Modify Physical Bus dialog box appears. The name of the selected bus appears in the “Bus name” box, and the names of its constituent nets appear in the “Net name” list.

4. Set net insertion and sorting options as needed.

- If you want the tool to sort the nets automatically when you insert them, select the “Before selection” option.

By default, the “After selection” option is selected, which means the tool does not automatically sort the nets when you insert them.

- To set the net order, select a “Sort mode” option.

The choices are Ascending, Descending, and None. The default is Ascending.

- To set the net precedence, select a Precedence option.

The choices are Left and Right. The default is Left.

- To set the index delimiter character for the bus, select a “Bus delimiter” option.

The available delimiters are square brackets ([i]), angle brackets (<i>), braces ({i}), parentheses ((i)), underlines (_i_), and colons (:i:). The default is square brackets.

5. (Optional) Add nets to the bus as needed.

You can search for the nets by name, insert selected, or load the nets from a file.

- To search for the nets by name, click the Search button to open the Object Chooser dialog box, set the search parameters as needed, click the Search button, edit the "Search results" list as needed, and click OK.

- To insert selected nets, select the nets and click the Get Selection button.

Alternatively, you can control the insertion order by clicking the arrow button beside the Get Selection button and choosing a command on the menu that appears.

- To load the nets from a file and preserve the net ordering specified in the file, click the "Load from file" button to open the "Read nets from file" dialog box, navigate to the directory where the file is located, select the file or type the file name, and click Open.

The names of the nets appear in the "Net name" list.

6. (Optional) Sort or remove nets in the "Net name" list as needed.

- To sort the nets, set "Sort mode" and precedence options as needed, and click the Sort button.
- To remove individual nets, select the nets and click the Remove button.
- To remove all of the nets, click the Remove All button.

7. Click OK or Apply.

The tool displays the bus name in the Physical Buses dialog box.

Alternatively, you can use the `create_physical_bus` command.

For more information about modifying a physical bus, see the "Modifying Buses" topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

Routing Buses

You can route a physical bus or multiple nets interactively in the active layout view by drawing the routes with the Advanced Route tool. A route can be composed of one or more horizontal or vertical segments.

To create routes interactively in the active layout view,

1. (Optional) Select the buses or nets you need to route. If you perform this step, the tool operates in preselection mode.
2. Click the Advanced Route tool icon () on the Edit toolbar or choose Edit > Advanced > Advanced Route Tool.

The tool dims the visible objects in the active layout view and opens the Advanced Route Tool dialog box.

If you preselected one or more buses or nets that are members of a bus, the bus names appear in the Bus box. If you preselected one or more unassociated nets, the net names appear in the Net box.

3. (Optional) You can select a “Specify nets” option and specify them in the Bus or Net box if you did not preselect the buses or nets that you need to route.
 - To route buses, select the “by Bus name(s)” option, and select or browse for the buses or type the bus name in the Bus box.
 - To route nets, select the “by Net name(s)” option, and select or browse for the nets or type the net name in the Net box.
4. (Optional) Set options as needed in the Advanced Route Tool dialog box.

You can set options to

- Draw either one or two route segments between two points in the layout view.

By default, the tool draws a single set of orthogonal segments between two points. You can select an angle mode that permits the tool to draw two sets of segments and controls which segments (horizontal, vertical, or longest) are drawn first.

- Enable or disable grid snapping to routing tracks.

By default, grid snapping is enabled.

- Assign a route type attribute.

The default route type is signal.

- Avoid blockages and set the maximum number of jump attempts

- Control whether the low end segment or the high end segment is used for the first section of the bus.

- Control how the routes change directions.

You can switch the reference wire from one end to the other, set the skip count, reverse the net order, or force subsequent segments to follow the pitch of the reference wire.

- Enable automatic welding and control welding operations

You can control whether the tool ignores pin access directions, honors the same net contact to contact spacing rule, or connects to multiple layer pins on a layer that you select.

- Extend routes from the start point or end point to touch target pins or wires with the same net ID within the specified search distance (5.0 microns by default).

- Enable automatic alignment for pin connections.

When this option is enabled, the tool automatically shifts bus route positions to match the positions of the target pins during extend first and extend last operations, aligns bus nets to macro pins, and adjusts bus net widths to match pin widths.

Automatic alignment for extend last operations works only when the last two bus segments form an L shape.

- Adjust automatic welding and alignment parameters

You can change the search distance, prohibit the tool from inserting vias on pins, or permit the tool to adjust bus wire end widths to match pin widths.

- Display or hide flylines for connected nets or guides for target port and pin locations.

By default, both flylines and port guides are displayed.

- Enable or disable interactive DRC checking.

By default, interactive DRC checking is enabled. You can set DRC options by clicking the Options button.

If you need to reset the Advanced Route Tool dialog box options to their default values, click Default.

For details about setting options in the Advanced Route Tool dialog box, see the “Setting Advanced Route Tool Options” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

5. In the layout view, click the start point for the routes.

By default, the tool displays flylines and target port and pin locations in the preferred layer color for the specified buses or nets.

If the automatic first-click Bus or Net, Layer, or Width options are enabled and the object you click is a net shape, user shape, pin, port, terminal, or via, the tool automatically identifies the net connected to the object and can perform the following actions:

- Select the bus or net, select the appropriate “Specify by” option in the Advanced Route Tool dialog box, and update the Bus or Net box.

Buses take priority over nets. If the object you click is connected to a net that is a member of a physical bus, the tool selects the bus and selects the “by Bus name(s)” option in the Advanced Route Tool dialog box.

- Identify the preferred routing layer for the net and update the Layer list in the Advanced Route Tool dialog box.
- Identify the default wire width for the net and update the Width box in the Advanced Route Tool dialog box.

By default, the Bus or Net option and the Layer option are enabled and the Width option is disabled.

6. Draw the first wire segments.

As you move the pointer, a ghost image appears between the initial point and the current pointer position. The color of the ghost image indicates the layer on which you are drawing the routes.

By default, the tool draws single horizontal or vertical segments, depending on the direction you move the pointer. If you want to draw two segments at the same time, select an option in the “Angle mode” list in the Advanced Route Tool dialog box.

The default reference wire is the low end segment, the bottom segment for horizontal routes or the leftmost segment for vertical routes in the first section of the bus. If you want to use the high end segment instead, select the “High end” option in the Advanced Route Tool dialog box.

7. Click a point or press Return when you need to finish the route segments and change the routing direction or the layer.

- To change the layer, press F1 or Shift-F1 repeatedly until the ghost shape color matches the color of the layer you want to use. To move up one layer, press F1. To move down one layer, press Shift-F1. When you change the layer, the tool automatically inserts the appropriate vias.
- To remove the last point, press Backspace or right-click and choose Remove Last Point. You can remove multiple points sequentially.
- To remove the entire route and start over, press Escape or right-click and choose Cancel.

If necessary, you can adjust options in the Advanced Route Tool dialog box to control how the routes change direction.

- To switch the reference wire from one end to the other, select the “Switch ends” option.
 - To change the number of wires that each wire crosses, select or type a value in the “Skip count” box.
 - To force subsequent segments to follow the pitch of the reference wire, select the “Follow first segment pitch” option.
8. Repeat steps 6 and 7 as needed.
 9. Finish the routes and instantiate the routed bus or nets by doing one of the following:
 - Double-click a target pin or port.
 - Right-click and choose Apply.
 - Click Apply in the Advanced Route Tool dialog box.
 - 10.(Optional) Do one of the following:
 - To route another bus or group of nets, repeat steps 1 through 9.
 - To deactivate the Advanced Route Tool, press the Escape key or click OK in the Advanced Route Tool dialog box.

For more information about using the Advanced Route tool, see the “Routing Buses or Multiple Nets” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

Creating Custom Wires

You can route one or more nets between specific points by using the Custom Wire tool. You specify the points by clicking in the active layout view and define the routes by setting options in the Create Custom Wires dialog box. A route can be composed of one or more horizontal or vertical segments. Each point you click after the first point specifies a route segment.

To route nets with the Custom Wire tool,

1. (Optional) Select a net shape, pin, port, terminal, user shape, or via to identify the net you need to route. If you perform this step, the tool operates in preselection mode.

2. Click the Custom Wire tool icon () in the Edit toolbar or choose Edit > Create > Custom Wires.

The Create Custom Wires dialog box appears. If you preselected the net, the GUI automatically sets the net name, layer, and width options.

3. (Optional) Set options as needed in the Create Custom Wires dialog box.

If you did not select the nets that you need to route before activating the Custom Wire tool, you can select or browse for the nets or type the net names in the Net box.

If you need to reset the Create Custom Wires dialog box options to their default values, click Default. For details about setting options in the Create Custom Wires dialog box, see the “Setting Route Tool Options” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

4. In the active layout view, click the start point for the routes.

If you selected the automatic first-click Bus or Net, Layer, or Width options and the object you click is a net shape, user shape, pin, port, terminal, or via, the tool automatically identifies the net connected to the object and can perform the following actions:

- Select the net and update the Net box in the Create Custom Wires dialog box.
- Identify the preferred routing layer for the net and update the Layer list in the Create Custom Wires dialog box.
- Identify the default wire width for the net and update the Width box in the Create Custom Wires dialog box.

The Net, Layer, and Width options are disabled by default.

5. Draw the first wire segments.

As you move the pointer, a ghost image appears between the initial point and the current pointer position. The color of the ghost image indicates the layer on which you are drawing the routes.

If you are routing multiple nets, the default reference wire is the low end segment, the bottom segment for horizontal routes or the leftmost segment for vertical routes, in the first section of the routes. If you want to use the high end segment instead, select the “High end” option in the Create Custom Wires dialog box.

6. Click a point or press Return when you need to finish the route segments and change the routing direction or the layer.

- To change the layer, press F1 or Shift-F1 repeatedly until the ghost shape color matches the color of the layer you want to use. To move up one layer, press F1. To move down one layer, press Shift-F1. When you change the layer, the tool automatically inserts the appropriate vias.
- To remove the last point, press Backspace or right-click and choose Remove Last Point. You can remove multiple points sequentially.
- To remove the entire route and start over, press Escape or right-click and choose Cancel.

If necessary, you can adjust options in the Create Custom Wires dialog box to control how the routes change direction.

- To switch the reference wire from one end to the other, select the “Switch ends” option.
- To change the number wires that each wire crosses, select or type a value in the “Skip count” box.

7. Repeat steps 5 and 6 as needed.

8. Finish the routes and instantiate the routed nets by doing one of the following:

- Double-click a target pin or port.
- Right-click and choose Apply.
- Click Apply in the Advanced Route Tool dialog box.

9. (Optional) Do one of the following:

- To route additional nets, repeat steps 3 through 9.
- To deactivate the Custom Wires Tool, press the Escape key or click OK in the Create Custom Wires dialog box.

For more information about creating custom wires, see the “Creating Custom Wires” topic in IC Compiler Help. To view IC Compiler Help, choose Help > IC Compiler Online Help in the GUI.

Postroute RC Extraction

IC Compiler automatically performs postroute RC estimation when you run the `route_opt` command and when you run any of the following timing analysis commands on a routed but not extracted design: `report_timing`, `report_qor`, `report_constraint`, `report_delay_calculation`, `report_net`, `report_clock`, `report_clock_timing`, or `report_clock_tree`. In addition, you can explicitly perform postroute RC extraction by running the `extract_rc` command.

Note:

If the extraction is already done by a timing or extraction command, it will not occur again for the same routed database when you use the reporting commands.

By default, the `extract_rc` command performs RC estimation on any unrouted nets in your design and performs RC extraction on the routed nets in your design. If your design contains a mix of routed and unrouted nets, and you do not want to run RC estimation on the unrouted nets, use the `extract_rc -routed_nets_only` command, which performs only RC extraction.

During postroute RC extraction, IC Compiler performs the following tasks:

- Determines the RC coefficients

IC Compiler derives the RC coefficients from the TLUPlus models (for information about attaching the TLUPlus data to your Milkyway design library, see [“Setting Up the TLUPlus Files” on page 3-26](#)).

- Calculates the RC values

If your design contains global routing, IC Compiler uses a global route extraction model to calculate the parasitics. This model uses resistance values derived from the global routing and estimated capacitance values.

If your design contains track assignments or detail routing, IC Compiler uses a detail route extraction model to calculate the parasitics. This model uses resistance and capacitance values derived from the detail postroute geometries.

Autoextraction does not extract the coupling capacitance values unless you have set the delta delay signal integrity option to true (`set_si_options -delta_delay true`). To force extraction of these values, run the `extract_rc -coupling_cap` command.

- Saves the timing data and attributes in the Milkyway design library

The parasitics are saved in the Milkyway design library, but a parasitic data file is not saved. To save the parasitics in a data file, run the `write_parasitics` command.

To fine-tune the postroute RC extraction, you specify the scaling factors by using the `set_extraction_options` command and its options, as shown in [Table 8-15](#).

Table 8-15 *set_extraction_options Command Options*

To do this	Use this
Scale the capacitance coefficients.	<code>-max_cap_scale</code> <code>-min_cap_scale</code>
Scale the resistance coefficients.	<code>-max_res_scale</code> <code>-min_res_scale</code>
Scale the coupling capacitance coefficients.	<code>-max_ccap_scale</code> <code>-min_ccap_scale</code>
Specify the coupling capacitance net-to-net filtering threshold (default is 0.003 picrofarad).	<code>-max_net_ccap_thres</code> <code>-min_net_ccap_thres</code>
Specify the coupling capacitance net-to-net filtering ratio (default is 0.03 of total capacitance).	<code>-max_net_ccap_ratio</code> <code>-min_net_ccap_ratio</code>
Specify the coupling capacitance average net-to-net filtering ratio (default is 0.25 of average coupling capacitance).	<code>-max_net_ccap_avg_ratio</code> <code>-min_net_ccap_avg_ratio</code>
Specify the process scaling factors (default is 1.0).	<code>-max_process_scale</code> <code>-min_process_scale</code>
Ignore routing obstructions during extraction.	<code>-no_obstruction</code>
Disable the breaking of long wire segments during extraction.	<code>-no_break_segments</code>
Specify the maximum length of wire segments (default is 50 microns). If you use <code>-no_break_segments</code> , this option has no effect.	<code>-max_segment_length</code>
Reset extraction options.	<code>-default</code>

You can use process scaling factors to adjust RC estimation and extraction for a process shrink, such as shrinking from 90 nm to 65 nm. The lengths and widths of the interconnect wires are scaled by the specified factors, without affecting interconnect wire depths or metal geometries inside standard cells and macros. You specify the scaling factors with the `-max_process_scale` and `-min_process_scale` options of the `set_extraction_options` command. These factors affect the RC values generated by the `extract_rc` command for maximum-delay and minimum-delay analysis, respectively, for both preroute RC estimation and postroute RC extraction.

Note:

The process scaling factors of the `set_extraction_options` command are the only settings that apply to both preroute RC estimation and postroute RC extraction. All other options of the `set_extraction_options` command apply only to postroute RC extraction.

Reliability Analysis Using PrimeRail

After performing global routing, detail routing, or postroute chip finishing, you can perform reliability analysis by using the PrimeRail tool. To prepare the data required by PrimeRail, use the `create_rail_setup` command. By default, this command creates a directory called `synopsys_rail_setup` in the current working directory and outputs the files necessary to run rail analysis to this directory. You can change the name of the output directory by using the `-directory` option.

When you run the `create_rail_setup` command on a block (the default), the following files are output:

- The PrimeRail setup file (`synopsys_pr_setup.e`)

This binary file defines the search path, link path, TLUPlus files, operating conditions, Milkyway design library, Milkyway cell, parasitic corner, and power and ground nets.

- The Verilog netlist (`design.v`)

By default, IC Compiler generates a Verilog netlist for the current Milkyway design. To use an existing Verilog netlist, specify the file name by using the `-verilog` option.

- The SDC constraints for the design (`design.sdc`)

By default, IC Compiler generates an SDC file for the current Milkyway design. To use an existing SDC file, specify the file name by using the `-sdc` option.

- The signal parasitics (`design.spef.max` or `design.spec.min`)

By default, IC Compiler performs RC extraction and generates an SPEF file for the maximum corner of the current Milkyway design. If RC extraction has already been done, use the `-no_rc_extract` option to skip RC extraction. To generate an SPEF file for the minimum corner instead of the maximum corner, use the `-parasitic_corner min` option. To use an existing SPEF file, specify the file name by using the `-spef` option.

To generate files for the top level in a hierarchical flow, use the `-hierarchy` option when you run the `create_rail_setup` command. When you use the `-hierarchy` option, only the PrimeRail setup file and the Verilog netlist are generated. You must provide the signal parasitics generated by Star-RCXT and the SDC file.

By default, the `create_rail_setup` command saves the design in a cell called *design_pr*. If you do not want to save the design, use the `-no_save` option.

For information about using PrimeRail to perform reliability analysis, see the *PrimeRail User Guide*.

9

Chip Finishing and Design for Manufacturing

IC Compiler provides chip finishing and design for manufacturing and yield capabilities that you can apply throughout the various stages of the design flow to address process design issues encountered during chip manufacturing.

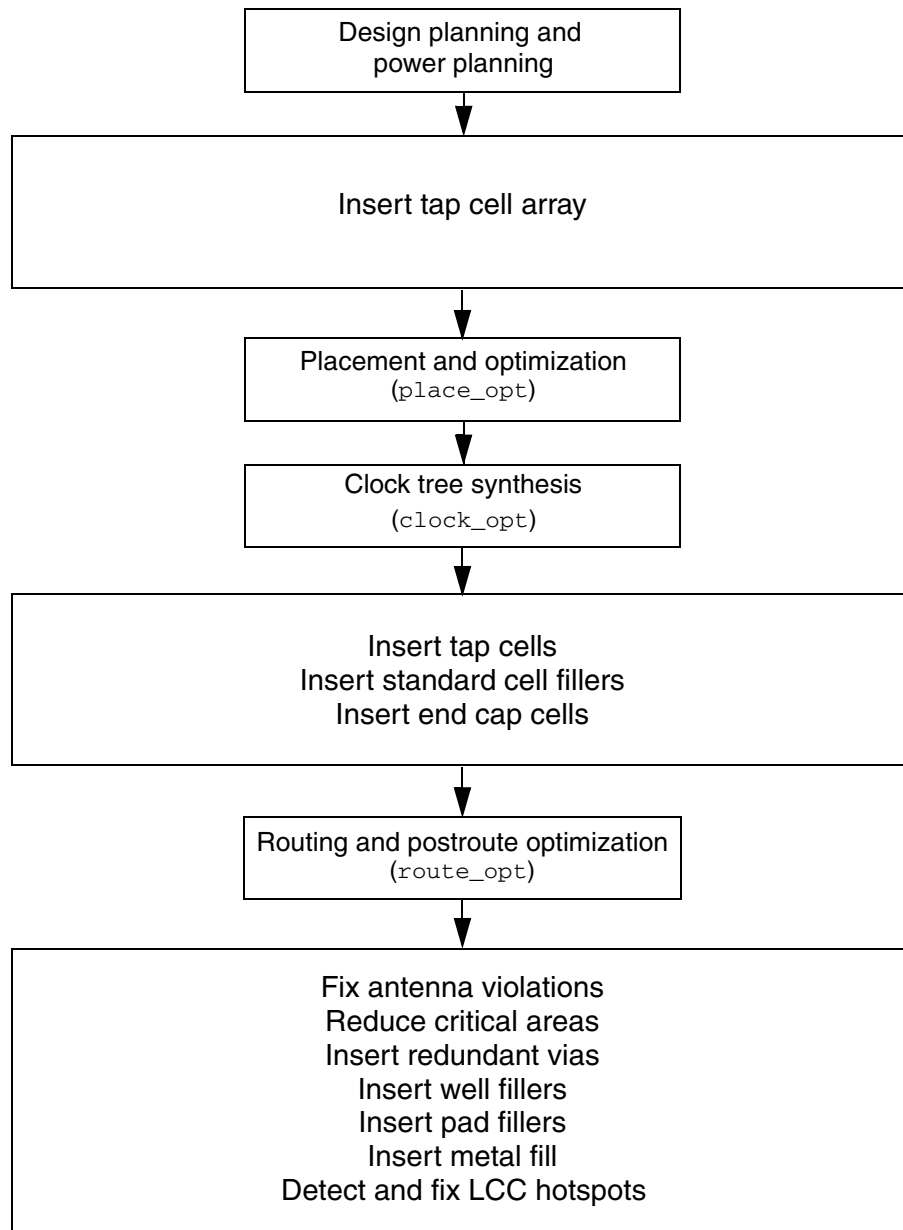
This chapter contains the following sections:

- [Overview](#)
- [Inserting Tap Cells](#)
- [Preventing Antenna Problems](#)
- [Reducing Critical Areas](#)
- [Inserting Redundant Vias](#)
- [Inserting Filler Cells](#)
- [Performing Metal Density Filling](#)
- [Performing Notch and Gap Filling](#)
- [Analyzing CMP Thickness Variation Effects](#)
- [Lithography Compliance Checking](#)

Overview

Figure 9-1 shows the design for manufacturing and chip finishing tasks supported by IC Compiler and how these tasks fit into the overall IC Compiler design flow. The following sections describe how to perform these tasks.

Figure 9-1 Design for Manufacturing and Chip Finishing Tasks in the Design Flow



Inserting Tap Cells

Tap cells are a special nonlogic cell with well and substrate ties. These cells are typically used when most or all of the standard cells in the library contain no substrate or well taps. Generally, the design rules specify the maximum distance allowed between every transistor in a standard cell and a well or the substrate ties.

You can insert tap cells in your design before or after placement:

- You can insert tap cell arrays before placement to ensure that the placement complies with the maximum diffusion-to-tap limit.
- You can insert tap cells after placement to fix maximum diffusion-to-tap violations.

The following sections describe these methods.

Adding Tap Cell Arrays

Before global placement (during the floorplanning stage), you can add tap cells to the design that form a two-dimensional array structure to ensure that all standard cells placed subsequently will comply with the maximum diffusion-to-tap distance limit.

You need to specify the tap distance and offset, based on your specific design rule distance limit. The command has no knowledge of the design rule distance limit. After you run the command, it is recommended that you visually check to ensure that all standard cell placeable areas are properly protected by tap cells.

To add a tap cell array, use the `add_tap_cell_array` command (or choose Finishing > Add Tap Cell Array in the GUI). This is the command syntax:

```
add_tap_cell_array
  -master_cell_name name
  -distance tap_cell_distance
  [-voltage_area voltage_area_collection]
  [-pattern normal | every_other_row | stagger_every_other_row]
  [-offset distance]
  [-tap_cell_identifier tap_cell_prefix]
  [-tap_cell_separator tap_cell_separator]
  [-no_tap_cell_under_layers layer_list]
  [-well_port_name port_name]
  [-well_net_name net_name]
  [-substrate_port_name port_name]
  [-substrate_net_name substrate_tie_net_name]
  [-connect_power_name power_net_name]
  [-connect_ground_name ground_net_name]
  [-fill_boundary_row true | false]
  [-fill_macro_blockage_row true | false]
```

```

[-boundary_row_double_density true | false]
[-macro_blockage_row_double_density true | false]
[-left_macro_blockage_extra_tap by_rule | must_insert | no_insert]
[-right_macro_blockage_extra_tap by_rule | must_insert | no_insert]
[-left_boundary_extra_tap by_rule | must_insert | no_insert]
[-right_boundary_extra_tap by_rule | must_insert | no_insert]
[-ignore_soft_blockage true | false]
[-at_distance_only true | false]
[-skip_fixed_cells true | false]
[-respect_keepout]

```

For example,

```

icc_shell> add_tap_cell_array -master_cell_name Cell11 -distance 30 \
           -pattern normal -voltage_area [get_voltage_areas "V*"] \
           -no_tap_cell_under_layer {M1, M2}

```

You must specify the name of the reference cell to be used for tap insertion using `-master_cell_name name` and the required distance, in microns, between taps using `-distance distance`. The specified distance should be approximately twice the maximum diffusion-to-tap value specified in the technology design rule. Use the `-voltage_area voltage_area_collection` option to restrict tap insertion to specific voltage areas.

You can specify the insertion pattern using the `-pattern` option: `normal` (the default), `every_other_row`, or `stagger_every_other_row`. “Normal” insertion adds tap cells to every row, using the specified distance limit. “Every other row” insertion adds tap cells in the odd-numbered rows only, which reduces the added tap cells by one-half. “Stagger every other row” insertion adds tap cells in every row with tap cells in even rows offset by half the specified `-offset` setting relative to the odd rows, producing a checkerboard-like pattern.

The command also has options to specify the tap cell naming conventions, well port and net names, substrate port and net names, ground net connections, prohibited placement under specified metal layer preroutes, and tap density adjustments at boundary rows and macro blockage rows. For more information about these options and their default settings, see the man page for the `add_tap_cell_array` command.

Fixing Tap Spacing Violations

You can add tap cells to comply with the diffusion-to-substrate or -well-contact maximum spacing design rule. After global placement (typically), you can insert tap cells “by rules” so that all existing standard cells comply with the maximum diffusion-to-tap distance limit.

To insert tap cells to satisfy the diffusion-to-tap design rules, use the `insert_tap_cells_by_rules` command (or choose Finishing > Insert Tap Cells By Rules in the GUI). This is the full command syntax:

```
insert_tap_cells_by_rules
-drc_spacing_check | -tap_cell_insertion
-tap_distance_based | -drc_spacing_based
-move | -freeze
-tap_master physical_lib_cell
[-tap_layer layer_name]
[-tap_distance_limit distance]
[-n_well_layer layer_name]
[-p_well_layer layer_name]
[-contact_layer layer_name]
[-p_diffusion_layer layer_name]
[-n_diffusion_layer layer_name]
[-p_implant_layer layer_name]
[-n_implant_layer layer_name]
[-tap_spacing_design_rule distance]
[-tap_filler_name_identifier prefix]
[-ignore_hard_blockage]
[-ignore_soft_blockage]
[-ignore_double_back_sharing]
[-connect_to_power_net power_net]
[-connect_to_ground_net ground_net]
[-no_tap_cells_under_metal_layer metal_layers_list]
[-voltage_area voltage_area_list]
[-respect_keepout]
```

For example,

```
icc_shell> insert_tap_cells_by_rules -tap_master "MY_TAP" \
-tap_cell_insertion -tap_distance_based -move \
-tap_distance_limit 30.0 \
-no_tap_cells_under_metal_layer {metal1}
```

This example inserts a tap cell named MY_TOP into a placed design to satisfy the condition that the distance from a standard cell to a tap cell must be no more than 30 microns, with the restriction that tap cells cannot be inserted under layer metal1.

In the command, you must specify the name of the tap cell using `-tap_master cell_name`. In addition, you must use either `-drc_spacing_check` or `-tap_cell_insertion` to specify whether to perform DRC tap distance checking only or to both perform checking and fixing of violations by inserting tap cells.

You must use either `-tap_distance_based` or `-drc_spacing_based` to specify the method of calculating distances between standard cells and tap cells. The “Tap distance based” method uses a simple measurement from the standard cell to the tap, irrespective of

the actual metal and contact layers within the cells. This method is used when the actual layout of the standard cells or tap cell is not available in the cell library. You must specify the tap distance limit using `-tap_distance_limit distance`.

The “DRC spacing based” method uses the actual layout of the standard cell and tap cell, with consideration of the diffusion, well, and contact layers in the cells. This method results in the insertion of fewer tap cells because existing taps in standard cells are recognized. You must specify the applicable layer names and design rule distance using command options such as `-n_well_layer` and `-tap_spacing_design_rule`.

You must use either `-move` or `-freeze` to specify whether to allow standard cells to be moved to make room for tap cells. Allowing standard cells to move can help prevent design rule violations and reduce the number of tap cells that must be inserted, but can affect the design timing.

The command also has options to specify the tap layer name, tap cell naming conventions, respect for hard and soft blockages, allowance of tap sharing, power and ground net connections, prohibited placement under specified metal layer preroutes, voltage areas in which to perform tap cell insertion, and respect for hard macro keepout margins. For more information about these options and their default settings, see the man page for the `insert_tap_cells_by_rules` command.

Removing Tap Cells

To remove tap cells, use the `remove_stdcell_filler -tap` command or choose Finishing > Remove Fillers in the GUI and select Tap as the filler type to remove. You can optionally specify a bounding box from which to remove the tap cells.

Preventing Antenna Problems

In chip manufacturing, gate oxide can be easily damaged by electrostatic discharge. The static charge that is collected on wires during the multilevel metallization process can damage the device or lead to a total chip failure. The phenomena of an electrostatic charge being discharged into the device is referred to as either antenna or charge-collecting antenna problems.

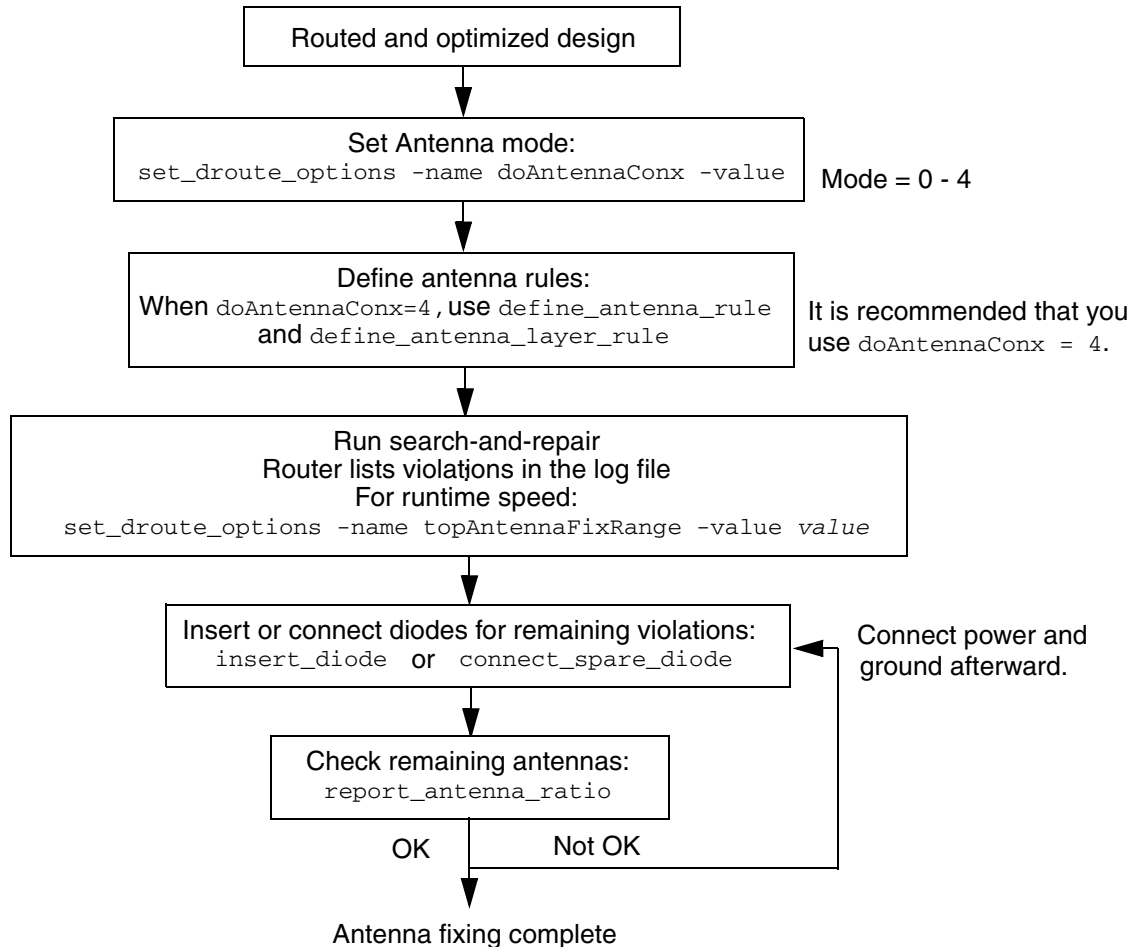
To prevent antenna problems, IC Compiler verifies that the metal antenna area divided by the input pin gate area is less than the maximum antenna ratio given by the foundry:

$$(\text{antenna-area}) / (\text{gate-area}) < (\text{max-antenna-ratio})$$

The antenna rules are categorized into basic and advanced rules. For process nodes above 0.18μm, use the basic antenna rules.

Figure 9-2 shows how to use IC Compiler to perform antenna checking and fixing.

Figure 9-2 IC Compiler Antenna Methodology



Setting the Antenna Mode

IC Compiler supports single-layer, accumulative ratio, accumulative area, and advanced antenna modes.

Using the basic antenna rules, you can select one of the first three antenna modes and decide the area, which can be either surface (polygon) or sidewall, to use for antenna area calculation. Set the `doAntennaConx` detail route option to select an antenna mode and the `useSideWallForAntenna` (0 for surface area) detail route option to choose an area:

- Single-layer mode (Set `doAntennaConx` to 1)
- Accumulative ratio mode (Set `doAntennaConx` to 2)
- Accumulative area mode (Set `doAntennaConx` to 3)
- Advanced mode (Set `doAntennaConx` to 4)

This section discusses only the first three modes using the surface area. For the advanced antenna mode, use the `-mode` option to decide a mode and follow the advanced antenna rules. For more information, see [“Setting the Antenna Mode for the Advanced Antenna Rules” on page 9-13](#)

[Figure 9-3](#) shows a layout example with a lateral view, which is used to explain antenna modes in [Table 9-1](#): single-layer mode, accumulative ratio mode, and accumulative area mode.

Figure 9-3 Layout Example

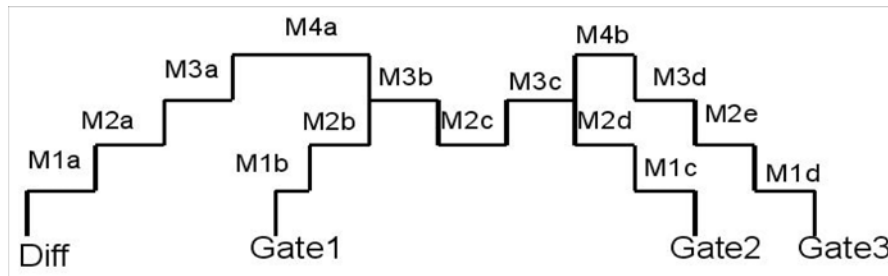


Table 9-1 Antenna Modes and Ratios

Antenna Mode	Antenna Ratio
Single-layer mode (ignores all lower-layer segments; allows best routability): <pre>set_droute_options \ -name doAntennaConx \ -value 1</pre>	antenna_ratio = connected metal area of the layer / total gate area m1 ratios - Gate1: M1b / Gate1 - Gate2: M1c / Gate2 - Gate3: M1d / Gate3 m2 ratios - Gate1: M2b / Gate1 - Gate2: M2d / Gate2 - Gate3: M2e / Gate3 m3 ratios - Gate1, 2: (M3b + M3c) / (Gate1 + Gate2) - Gate3: M3d / Gate3 m4 ratios - Gate1, 2, 3: (M4a + M4b) / (Gate1 + Gate2 + Gate3)
Accumulative ratio mode (includes lower-layer segments to the input pins): <pre>set_droute_options \ -name doAntennaConx \ -value 2</pre>	antenna_ratio = accumulation of ratios for the layer and layers below m1 ratios - Gate1: M1b / Gate1 - Gate2: M1c / Gate2 - Gate3: M1d / Gate3 m2 ratios - Gate1: M1b / Gate1 + M2b / Gate1 - Gate2: M1c / Gate2 + M2d / Gate2 - Gate3: M1d / Gate3 + M2e / Gate3

Table 9-1 Antenna Modes and Ratios (Continued)

Antenna Mode	Antenna Ratio
	m3 ratios - Gate1: $M1b / \text{Gate1} + M2b / \text{Gate1} + (M3b + M3c) / (\text{Gate1} + \text{Gate2})$ - Gate2: $M1c / \text{Gate2} + M2d / \text{Gate2} + (M3b + M3c) / (\text{Gate1} + \text{Gate2})$ - Gate3: $M1d / \text{Gate3} + M2e / \text{Gate3} + M3d / \text{Gate3}$ m4 ratios - Gate1: $M1b / \text{Gate1} + M2b / \text{Gate1} + (M3b + M3c) / (\text{Gate1} + \text{Gate2}) + (M4a + M4b) / (\text{Gate1} + \text{Gate2} + \text{Gate3})$ - Gate2: $M1c / \text{Gate2} + M2d / \text{Gate2} + (M3b + M3c) / (\text{Gate1} + \text{Gate2}) + (M4a + M4b) / (\text{Gate1} + \text{Gate2} + \text{Gate3})$ - Gate3: $M1d / \text{Gate3} + M2e / \text{Gate3} + M3d / \text{Gate3} + (M4a + M4b) / (\text{Gate1} + \text{Gate2} + \text{Gate3})$
Accumulative area mode (includes all lower-layer segments; allows least routability): <pre>set_droute_options \ -name doAntennaConx \ -value 3</pre>	$\text{antenna\_ratio} = \text{all connected metal areas} / \text{total gate area}$ m1 ratios - Gate1: $M1b / \text{Gate1}$ - Gate2: $M1c / \text{Gate2}$ - Gate3: $M1d / \text{Gate3}$ m2 ratios - Gate1: $(M1b + M2b) / \text{Gate1}$ - Gate2: $(M1c + M2d) / \text{Gate2}$ - Gate3: $(M1d + M2e) / \text{Gate3}$ m3 ratios - Gate1, 2: $(M1b + M1c + M2b + M2c + M2d + M3b + M3c) / (\text{Gate1} + \text{Gate2})$ - Gate3: $(M1d + M2e + M3d) / \text{Gate3}$ m4 ratios - Gate1, 2, 3: $\text{all metal areas} / (\text{Gate1} + \text{Gate2} + \text{Gate3})$

Defining Antenna Rules

IC Compiler reads the antenna data that is prepared by Milkyway to fix antenna violations. If a sidewall is used for antenna calculation, you need to define the metal thickness by specifying the `unitMinThickness`, `unitNomThickness`, and `unitMaxThickness` attributes in each layer section of the technology file.

The following three factors influence the computation of antenna rules:

- The antenna mode to identify which metal piece to count
 - Single-layer mode
 - Accumulative ratio mode
 - Accumulative area mode
 - Advanced mode
- The antenna area calculation
 - Surface area = $W \times L$
 - Sidewall area = $(W + L) \times 2 \times \text{thickness}$
- Protection from diodes and pins

The Basic Antenna Rules

The basic antenna rules define the maximum antenna ratio (antenna area to gate area) for metal layers and cut layers.

To use the basic antenna rules, set the `doAntennaConx` detail route option to 1, 2, or 3, depending on which metal pieces you want to count. For example, to set the antenna mode to single-layer mode, enter,

```
icc_shell> set_droute_options -name doAntennaConx -value 1
```

You must also specify whether you want to use surface area or sidewall area. To select the area, set the `useSideWallForAntenna` detail route option to 0 (the default) for surface area. Set the option to 1 for sidewall area. Note that this option is ignored when `doAntennaConx` is set to 4 for advanced antenna mode.

After setting the antenna mode and the area, use the following detail route options to define the maximum antenna ratio:

- `maxAntennaRatio`

This option defines the maximum antenna ratio for metal layers. For example,

```
icc_shell> set_droute_options -name maxAntennaRatio -value 300
```

- `maxCutAntennaRatio`

This option defines the maximum antenna ratio for cut layers. For example,

```
icc_shell> set_droute_options -name maxCutAntennaRatio -value 20
```

To get a report on the options that have been set with the `set_droute_options` command, use the `report_droute_options` command as shown in the following example:

```
icc_shell> report_droute_options
```

The part of the report showing the antenna settings looks like this:

```
Integer Option for droute doAntennaConx 1
#       range [0,4], default=0, stored in cell;
;;      0: no charge-collecting antenna checking;
;;      [1-4] check and avoid charge-collecting antenna;
;;      1: ignore all lower-layer segments, (best routability);
;;      2: include lower-layer segments to the input pins;
;;      3: include all lower-layer segments (worst routability).
;;      4: (advanced mode) use antenna rules defined in libraries
;;      for a combination of mode 1 - 3 and limited diode protection
...
Integer Option for droute maxAntennaRatio 300
...
```


The Advanced Antenna Rules

The advanced antenna rules give you more control over defining antenna rules. You can define the rules per layer and specify the diode ratios.

Setting the Antenna Mode for the Advanced Antenna Rules. To use the advanced antenna rules, set the `doAntennaConx` detail route option to 4 and define the antenna rules by using the `define_antenna_rule` and `define_antenna_layer_rule` commands. You specify which metal pieces to count during antenna calculation by using the `-mode` option. [Table 9-2](#) shows the description for each mode.

Table 9-2 Antenna Modes Using the -mode Option

Antenna Mode	Description
Mode 1: Single-layer	Uses surface area and ignores all lower-layer segments.
Mode 2: Accumulative ratio	Uses surface area which includes all lower-layer segments to the input pins.
Mode 3: Accumulative area	Uses surface area which includes all lower-layer segments.
Mode 4: Single-layer	Uses sidewall area and ignores all lower-layer segments.
Mode 5: Accumulative ratio	Uses sidewall area which includes all lower-layer segments to the input pins.
Mode 6: Accumulative area	Uses sidewall area which includes all lower-layer segments.

For example, to set the advanced antenna mode and to use the advanced antenna rules, enter the following commands:

```
icc_shell> set_droute_options -name doAntennexConx -value 4

icc_shell> define_antenna_rule mw_lib -mode mode \
    -diode_mode diode_mode \
    -metal_ratio metal_ratio \
    -cut_ratio cut_ratio \
    [-protected_metal_scale metal_scale] \
    [-protected_cut_scale cut_scale]
```

```
icc_shell> define_antenna_layer_rule mw_lib -mode mode\
          -layer layer_name\
          -ratio ratio\
          [-pratio pratio]\
          [-nratio nratio]\
          -diode_ratio {v0 v1 v2 v3 [v4]}\
          [-scale_factor scale_factor]
```

Setting the Diode Mode. Antenna ratio calculation with diode protection is based on the vector {v0 v1 v2 v3 [v4]}{s0 s1 s2 s3 s4 s5} that you specify by using the -diode_ratio option of the define_antenna_layer_rule command. The vector defines the maximum allowable antenna ratio (max-antenna-ratio) of the antenna area to the gate area if the antenna is protected by diodes. The antenna ratio of each metal layer must be less than the allowable max-antenna-ratio (antenna-area / gate-area < max-antenna-ratio).

The default value of the vector {v0 v1 v2 v3 [v4]} is {0 0 1 0 0} and can be written as {v0 v1 v2 v3} = {0 0 1 0} without the upper limit diode protection v4. Both vectors, {0 0 1 0 0} and {0 0 1 0}, indicate that the upper limit of the diode protection is 0.

The value of diode protection, d_p , is recorded on each output pin of a cell. If the value of the diode protection of an output pin is d_p , the antenna ratio is calculated as follows:

- For diode modes 0 to 4, use vector {v0 v1 v2 v3} to calculate the allowable max-antenna-ratio.
- For diode modes 5 to 8, use vector {v0 v1 v2 v3} to calculate the antenna ratio.
- For diode modes 9 and 10, use vector {v0 v1 v2 v3 v4} to calculate the allowable max-antenna-ratio.
- For diode modes 11 and 12, use both vectors {v0 v1 v2 v3 v4} and {s0 s1 s2 s3 s4 s5} to calculate the allowable max-antenna ratio.

Note that vector {s0 s1 s2 s3 s4 s5} is used for diode modes 11 and 12 only.

Table 9-3 shows how to calculate the antenna ratio for each diode mode.

Table 9-3 Antenna Ratio Calculation Based on Diode Mode

Diode Mode	Allowable Ratio Calculation
Diode modes 2 to 4	allowable max-antenna-ratio - dp > 0, if v4 <> 0, the allowable max-antenna-ratio = min (((dp + v1) * v2 + v3), v4) - dp > 0, if v4 == 0, the allowable max-antenna-ratio = (dp + v1) * v2 + v3 - When dp <= v0, the allowable max-antenna-ratio = layerMaxRatio
Diode mode 5	antenna ratio = metal_area / (gate_area + equi_gate_area) - if max_diode_protection <= v0, equi_gate_area = 0 - else if (v4 <> 0), equi_gate_area = min (((max_diode_protection + v1) * v2 + v3), v4) - else equi_gate_area = (max_diode_protection + v1) * v2 + v3
Diode mode 6	antenna ratio = metal_area / (gate_area + equi_gate_area) - if total_diode_protection <= v0, equi_gate_area = 0 - else if (v4 <> 0), equi_gate_area = min (((total_diode_protection + v1) * v2 + v3), v4) - else equi_gate_area = (total_diode_protection + v1) * v2 + v3
Diode mode 7	antenna ratio = (metal_area - equi_metal_area) / gate_area - if max_diode_protection <= v0, equi_metal_area = 0 - else if (v4 <> 0), equi_metal_area = min (((max_diode_protection + v1) * v2 + v3), v4) - else equi_metal_area = (max_diode_protection + v1) * v2 + v3 - if equi_metal_area > metal_area, equi_metal_area = metal_area
Diode mode 8	antenna ratio = (metal_area - equi_metal_area) / gate_area - if total_diode_protection <= v0, equi_metal_area = 0 - else if (v4 <> 0), equi_metal_area = min (((total_diode_protection + v1) * v2 + v3), v4) - else equi_metal_area = (total_diode_protection + v1) * v2 + v3 - if equi_metal_area > metal_area, equi_metal_area = metal_area
Diode mode 9	antenna ratio = scale * meta_area / gate_area - if max_diode_protection <= v0, scale = 1.0 - else scale = max (1 / ((max_diode_protection + v1) * v2 + v3), v4)

Table 9-3 Antenna Ratio Calculation Based on Diode Mode (Continued)

Diode Mode	Allowable Ratio Calculation
Diode mode 10	$\text{antenna ratio} = \text{scale} * \text{metal\_area} / \text{gate\_area}$ - If $\text{total\_diode\_protection} \leq v0$, $\text{scale} = 1.0$ - else $\text{scale} = \max(1 / ((\text{total\_diode\_protection} + v1) * v2 + v3), v4)$
Diode mode 11	$\text{antenna ratio} = \text{scale} * \text{metal\_area} / \text{gate\_area}$ - If $\text{max\_diode\_protection} < v0$, $\text{scale} = s0$ - If $\text{max\_diode\_protection} < v1$, $\text{scale} = s1$ - If $\text{max\_diode\_protection} < v2$, $\text{scale} = s2$ - If $\text{max\_diode\_protection} < v3$, $\text{scale} = s3$ - If $\text{max\_diode\_protection} < v4$, $\text{scale} = s4$ - If $\text{max\_diode\_protection} \geq v4$, $\text{scale} = s5$
Diode mode 12	$\text{antenna ratio} = \text{scale} * \text{metal\_area} / \text{gate\_area}$ - If $\text{total\_diode\_protection} < v0$, $\text{scale} = s0$ - If $\text{total\_diode\_protection} < v1$, $\text{scale} = s1$ - If $\text{total\_diode\_protection} < v2$, $\text{scale} = s2$ - If $\text{total\_diode\_protection} < v3$, $\text{scale} = s3$ - If $\text{total\_diode\_protection} < v4$, $\text{scale} = s4$ - If $\text{total\_diode\_protection} \geq v4$, $\text{scale} = s5$

Example for Diode Modes 2, 3 and 4

The vector {v0 v1 v2 v3} is {0.7 0.0 200 2000}, three diodes are connected to a single net, and the value layerMaxRatio is 400.

Diode A: diode protection=0.5, allowable ratio is 400 if protected
 Diode B: diode protection=1.0, allowable ratio is 2200 if protected
 Diode C: diode protection=1.5, allowable ratio is 2200 if protected

The antenna ratio calculation of a route net:
 Diode mode 2: allowable ratio = 2300 (maximum value)
 Diode mode 3: allowable ratio = $400 + 2200 + 2300 = 4900$
 Diode mode 4: allowable ratio = $((0.5 + 1.0 + 1.5) * 200) + 2000 = 2600$

Example for Diode Modes 5 and 6

The vector {v0 v1 v2 v3} is {0.0 0 1 0}, three diodes are connected to a single net, and the value of layerMaxRatio is 400.

Gate area of input pin = 0.6

Diode A: diode protection = 0.5

Diode B: diode protection = 1.0

Diode C: diode protection = 1.5

Total diode protection = 0.5+1.0+1.5 = 3.0

The antenna ratio calculation of a route net:

Diode mode 5: (Maximum diode protection value dp = 1.5)

Antenna ratio = metal_area/(gate_area + equi_gate_area)

$$= \text{metal\_area} / (0.6 + ((1.5+0)*1+0))$$

$$= \text{metal\_area} / 2.1$$

Diode mode 6: (Total diode protection value dp = 3.0)

Antenna ratio = metal_area/(gate_area + equi_gate_area)

$$= \text{metal\_area} / (0.6 + ((3.0+0)*1))$$

$$= \text{metal\_area} / 3.6$$

Example for Diode Modes 7 and 8

The vector {v0 v1 v2 v3} is {0.7 0.0 150 800}, three diodes are connected to a single net, and the value of layerMaxRatio is 400.

Diode A: diode protection = 0.5

Diode B: diode protection = 1.0

Diode C: diode protection = 1.5

Total diode protection = 0.5+1.0+1.5 = 3.0

The antenna ratio calculation of a route net:

Diode mode 7: (Use maximum dp to compute equi_metal_area)

Antenna ratio = (metal_area - equi_metal_area)/gate_area

$$= (\text{metal\_area} - ((1.5+0)*150+800))/\text{gate\_area}$$

$$= (\text{metal\_area} - 1025)/\text{gate\_area}$$

Diode mode 8: (Use total dp to compute equi_metal_area)

Antenna ratio = (metal_area - equi_metal_area)/gate_area

$$= (\text{metal\_area} - ((3.0+0)*150+800))/\text{gate\_area}$$

$$= (\text{metal\_area} - 1250)/\text{gate\_area}$$

The following shows an example of the advanced antenna rules:

```
set lib [current_mw_lib]
remove_antenna_rules $lib
define_antenna_rule $lib -mode 1 -diode_mode 4 -metal_ratio 300 -cut_ratio 20
define_antenna_layer_rule $lib -mode 1 -layer "M1" -ratio 300 \
    -diode_ratio {0.09 0 123 16880}
define_antenna_layer_rule $lib -mode 1 -layer "M2" -ratio 300 \
    -diode_ratio {0.09 0 123 16880}
define_antenna_layer_rule $lib -mode 1 -layer "M3" -ratio 300 \
    -diode_ratio {0.09 0 123 16880}
define_antenna_layer_rule $lib -mode 1 -layer "M4" -ratio 300 \
    -diode_ratio {0.09 0 123 16880}
define_antenna_layer_rule $lib -mode 1 -layer "M5" -ratio 400 \
    -diode_ratio {0.09 0 123 20000}
define_antenna_layer_rule $lib -mode 1 -layer "VIA1" -ratio 20 \
    -diode_ratio {0.09 0 110 500}
define_antenna_layer_rule $lib -mode 1 -layer "VIA2" -ratio 20 \
    -diode_ratio {0.09 0 110 500}
define_antenna_layer_rule $lib -mode 1 -layer "VIA3" -ratio 20 \
    -diode_ratio {0.09 0 110 500}
define_antenna_layer_rule $lib -mode 1 -layer "VIA4" -ratio 20 \
    -diode_ratio {0.09 0 110 500}
```

Reporting Antenna Rules

You can get a report of antenna rules by using the `report_antenna_rules` command. For example,

```
icc_shell> report_antenna_rules -output ./antenna.rule Design
```

Removing Antenna Rules

You can remove the antenna rules you set by using the `remove_antenna_rules` command. For example,

```
icc_shell> remove_antenna_rules Design
```

Checking Antenna Rules

After the antenna modes and rules are defined, you can proceed with antenna rules checking. The design rule checker of the detail router checks antenna rules automatically. Every routing command checks and reports the number of antenna violations as follows:

```
@@@ Total nets not meeting constraints = 116
```

where the number represents the number of nets with antenna violations.

To get a detailed report of antenna violations, you can use `verify_route -antenna` or the `report_antenna_ratio` command. Alternatively, you can choose Route > Verify Route or Route > Report Antenna Ratio in the GUI to get an antenna violation report.

Note:

You can set the `topAntennaFixRange` and `maxAntennaPinCount` detail route options to speed up the runtime of antenna checking and fixing.

Fixing Antenna Violations

You can fix antenna violations either before or after running the `route_opt` command. If both antenna fixing and routing optimization are needed, running `route_opt` first can shorten the overall turnaround time. However, running `route_opt` after antenna fixing can generate a good layout initially. Note that you need to keep the design as DRC-clean as possible before the antenna fixing step. When DRC violations have been eliminated, you can fix the antenna violations by performing search and repair step and diode insertion. Before performing these tasks, ensure that the antenna mode is set and that the antenna rules are defined as described in the previous sections.

Running Search and Repair

After antenna checking and fixing, run search and repair to fix antenna violations. For example,

```
icc_shell> route_search_repair -loop 20 -rerun_drc
```

Search and repair performs two types of metal-jog operations for antenna fixing. Both approaches decrease the antenna ratio by splitting a large metal polygon into several upper-level polygons.

- Break the antenna with a higher-level metal segment.

Use this technique to fix most antenna violations. For antenna violations happening at metal-N, inserting a small segment of metal-(N+1) close to the gate reduces the ratio between the remaining metal-N, making the ratio much lower. This approach is not suitable for fixing top-metal layer antennas when the output pin can provide only limited protection because there is no way for the router to break antenna violations at the topmost metal layer.

- Move down to a lower-level metal.

Use this technique to fix only topmost layer antenna violations when output pins provide only limited protection. For antenna violations happening at metal-N, replace part of the metal-N with metal-(N-1 or lower) to reduce the ratio. However, splitting the metal layer into many pieces might have a negative impact on RC and timing delay.

For more information about search and repair, see [“Search and Repair” on page 8-32](#)

Inserting Diodes

One way to protect gates from antenna effects is to provide a discharge path for the accumulated charge to leave the net. However, the discharge path should not allow current to flow during normal chip operation. Discharging can be accomplished by inserting a reverse-biased diode on the net close to the gate that is being protected.

Based on the results of the antenna checking, IC Compiler inserts a diode cell or multiple diodes when the antenna ratio requires the protection of more than one diode for each violation.

The diode port of the diode cell must be defined as such in the library. In the Milkyway environment, you can use the `write_def` command to define diode pins. For details, see the *Milkyway Environment Data Preparation User Guide*. In addition, the diode protection value must be defined in the physical library. For details, see the description of the `antenna_diffusion_area` attribute in the *Library Compiler User Guide: Physical Libraries* and the *Library Compiler Reference Manual: Physical Libraries*.

To minimize the impact on the existing placement and routing, you can choose to freeze the existing cell placement and the routing so that the diode cells can be placed in existing empty spaces only. You can also choose to complete the routing of the diode cells during the insertion process when the cell placement is already frozen. Whenever possible, IC Compiler ties the diode cells to the existing wires without ripping up and rerouting the entire net.

IC Compiler considers voltage areas when inserting diode cells and also observes the logic hierarchy assignments for diode cells.

To fix antenna violations, use the `insert_diode` command or choose **Finishing > Insert Diode** in the GUI. For example,

```
icc_shell> insert_diode -signal_route_options advanced
```

By default, the `insert_diode` command triggers the internal antenna checker to identify the antenna violations and then inserts one or multiple diode cells for each violation. For diode modes 2, 5, and 7, the `insert_diode` command inserts only one diode cell to fix an antenna violation. For diode modes 3, 4, 6, and 8, the `insert_diode` command inserts multiple diodes as needed to fix an antenna violation.

This is the full command syntax:

```
insert_diode
  [-nets collection_of_nets]
  [-antenna_check_engine internal | hercules]
  [-internal_check_option all | top_layer_only]
  [-no_auto_cell_selection]
  [-diode_cells collection_of_diode_cells]
  [-prefix prefix_name]
  [-use_hierarchical_diode_instance_name]
  [-same_row]
  [[-dont_freeze_existing_placement] |
   [-routing skip | when_all_violations_are_gone | always]]
  [-signal_route_options ignore_lower_layers | include_lower_layers |
   include_all_lower_layers | advanced]
  [-max_ratio max_ratio_number]
```

By default, all nets are checked for antenna violations. To restrict checking to a list of nets, use the `-nets` option. By default, IC Compiler uses its own internal antenna checker. To use the external Hercules check instead, specify `-antenna_check_engine hercules`. To report only top-level antenna violations rather than all violations, specify `-internal_check_option top_layer_only`.

By default, any diode cells can be used for fixing. To specify which diode cells to use, specify `-no_auto_cell_selection` together with `-diode_cells cell_collection`. To attempt diode insertion in the same row as the violation, use the `-same_row` option; by default, there is no preferred row for insertion.

By default, diode insertion does not disturb existing standard cells. To allow existing cells to be moved to make room for inserted diodes, use the `-dont_freeze_existing_placement` option. The `-routing` option lets you specify whether to perform routing: `always`, `when_all_violations_are_gone`, or `skip`.

Use `-signal_route_options` to specify the rules used for antenna checking: `ignore_lower_layers`, `include_lower_layers`, `include_all_lower_layers`, or `advanced`. The `-max_ratio` option specifies the maximum allowable ratio of wiring area to gate area to be used for antenna checking, overriding the default antenna checking rules.

For more information on the command options, see the man page for the `insert_diode` command.

To remove diodes, use the `remove_diode` command or choose **Finishing > Remove Diodes** in the GUI. The `remove_diode` command first disconnects all the diode ports on all the nets specified; then it removes the diode cells that are fully disconnected. For example,

```
icc_shell> remove_diode -nets net_name
icc_shell> remove_diode -nets [get_nets -hierarchical net_name]
```

Connecting Spare Diodes

Instead of inserting new diodes to fix antenna violations, you can insert spare diodes at the same time as standard cell placement and then connect the spare diodes as needed when antenna violations are found. The unused spare diodes are left unconnected. This technique lets you fix antenna violations without disturbing the placement of existing standard cells. However, you must allocate some placement area to accommodate the spare diodes. Adding more spare diodes uses more area resources but makes antenna fixing easier because a nearby spare diode is more likely to be available where needed.

The diode port of the diode cell must be defined as such in the library. In the Milkyway environment, you can use the `write_def` command to define diode pins. For details, see the *Milkyway Environment Data Preparation User Guide*. In addition, the diode protection value must be defined in the physical library. For details, see the description of the `antenna_diffusion_area` attribute in the *Library Compiler User Guide: Physical Libraries* and the *Library Compiler Reference Manual: Physical Libraries*.

To take advantage of spare diodes for antenna violation fixing, you need to add the spare diodes either before or after standard cell placement and before routing the areas where antenna violations might occur. For details, see [“Inserting Spare Cells” on page 15-3](#).

To fix antenna violations by connecting spare diodes, use the `connect_spare_diode` command or choose Finishing > Connect Spare Diode in the GUI. For example,

```
icc_shell> connect_spare_diode
```

The `connect_spare_diode` command checks for antenna violations and fixes those violations by connecting the problem nets to nearby spare diodes. This is the full command syntax:

```
connect_spare_diode
  [-antenna_check_engine internal | hercules]
  [-internal_check_option all | top_layer_only]
  [-routing skip | route]
  [-distance distance_number]
  [-signal_route_options ignore_lower_layers |
    include_lower_layers | include_all_lower_layers | advanced]
  [-max_ratio max_ratio_number]
```

The `-antenna_check_engine` option specifies which antenna checking engine to use, either the IC Compiler internal antenna checker (`internal`, the default) or the external Hercules antenna checker (`hercules`). If the internal checker is being used, the `-internal_check_option` option specifies whether to report all antenna violations (`all`) or only the top-layer violations (`top_layer_only`).

The `-distance` option specifies the maximum distance from the problem net that is searched for a spare diode. If no spare diode can be found within this distance, the command reports an error. You specify the distance as an integer multiple of the global routing cell size. The default is 10, which means that spare diode cells should be made available within 10 global routing cell lengths of anywhere an antenna violation might occur. For example, an array of spare diode cells with a spacing of 14 global routing cell lengths might be sufficient.

The `connect_spare_diode` command finds each antenna violation, identifies one or more nearby spare diodes required for fixing, and sets the net ID of the diode pins to the name of the problem net that must be connected. The number of diodes required to fix a particular antenna violation depends on the antenna checking method. By default, the command also invokes the detailed router in ECO mode to perform the required routing between the problem net and the diode. However, if you use `-routing skip`, the routing step is skipped. In that case, you can perform the necessary routing separately at a later time. Detailed routing should be complete before you attempt to connect the spare diodes.

By default, the `connect_spare_diode` command uses the antenna-checking mode specified by `doAntennaConx` option detail route option, as explained in [“Defining Antenna Rules” on page 9-11](#). You can override the `set_droute_options` setting by using `-signal_route_options` in the `connect_spare_diode` command. The four possible override settings are `ignore_lower_layers`, `include_lower_layers`, `include_all_lower_layers`, and `advanced`. These settings correspond to the `doAntennaConx` antenna mode settings of 1, 2, 3, and 4, respectively. The `doAntennaConx` antenna mode must be set to 1, 2, 3, or 4 (not 0) or `-signal_route_options` must be used in the `connect_spare_diode` command, even if the external Hercules antenna checker is being used.

By default, for antenna mode settings of 1, 2, or 3, the `connect_spare_diode` command checks for an antenna-to-gate area ratio that exceeds the `maxAntennaRatio` parameter of the `set_droute_options` command, as explained in [“Defining Antenna Rules” on page 9-11](#). You can override the `set_droute_options` setting by using the `-max_ratio` option in the `connect_spare_diode` command.

You can also perform the functions of the `connect_spare_diode` command in the GUI by choosing **Finishing > Connect Spare Diode**.

Setting Detail Route Options to Control Antenna Fixing

You can set the following detail route options listed in [Table 9-4](#) to fix antenna violations by using the `set_droute_options` command. The option settings are stored with the cell.

Table 9-4 set_droute_options Command Options

Option	Range	Default	Description
<code>useSideWallForAntenna</code>	[0, 1]	0	0: Uses the wire area to compute antenna ratio. 1: Uses the wire sidewall area to compute antenna ratio. You need to specify the thickness in the technology file. This option is ignored when <code>doAntennaConx</code> is set to 4.
<code>maxCutAntennaRatio</code>	[0, 1000000]	0	0: Does not compute cut area. N: The ratio of the total area of the charge-collecting antenna cuts to the total gate size should not exceed the value N. This option is ignored when <code>doAntennaConx</code> is set to 4.
<code>maxAntennaRatio</code>	[0, 1000000]	1000000	N: The ratio of the total area of the charge-collecting antenna to the total gate size should not exceed the value N. This option is ignored when <code>doAntennaConx</code> is set to 4.
<code>maxAntennaPinCount</code>	[-1, 1000000]	-1	-1: Checks antenna on all nets. N: Skips checking antenna on nets with more than N pins.
<code>minAntennaRatioScale</code>	[0.000, 1.000]	0.000	N: Shows the minimal antenna ratio scale for diode modes 11 and 12.
<code>maxAntennaRatioScale</code>	[0.000, 1.000]	1.000	N: Shows the maximal antenna ratio scale for diode modes 11 and 12.

Table 9-4 *set_droute_options* Command Options (Continued)

Option	Range	Default	Description
topAntennaFixRange	[-1, 10]	-1	-1: Fixes all top-layer antenna violations by pushing wires down. N: Fixes only top-layer antenna violations if the ratio is less than $[\text{safeRatio} \times (1 + 0.1 \times N)]$ to avoid pushing too many wires to lower layers. This option is valid only when <code>doAntennaConx</code> is set to 4 (Note: When <code>topAntennaFixRange</code> is set, the log file also shows the number of nets that are not fixed by the router).
dontMergeGateForAntenna	[0, 1]	0	When 2 or more input gates are connected, 0: The router adds their areas together to compute antenna ratio. 1: The router does not add the area of connected input gates.
outputPinDischarge	[0, 1]	1	0: Assumes that no output pin can discharge static electricity. 1: Assumes that output pins discharge static electricity. This option is ignored when <code>doAntennaConx</code> is set to 4.
defaultTopPinExtAntennaArea	[0.000, 1000000.000]	0.000	N: Sets the default <code>extAntennaArea</code> value at all metal layers for top-cell pins without specifying the <code>defineExtAntennaArea</code> value in the TDF. This option replaces <code>breakAntennaToTopPin</code> .
defaultTopPinExtGateSize	[0.000, 1000000.000]	0.000	N: Sets the default <code>extGateSize</code> value for top-cell pins without specifying the <code>defineExtAntennaGateSize</code> value in the TDF.
breakAntennaToTopPin	[0, 2]	0	0: Treats top-level pins as floating. 1: Treats top-level pins as they are connected to a huge antenna and breaks the antenna (Requires a jumper at top metal layer). 2: Connects all input pins to top-level pins only through the highest routing layer of the net.

Table 9-4 *set_droute_options* Command Options (Continued)

Option	Range	Default	Description
checkAntennaOnPG	[0, 1]	0	0: Does not check antenna on power and ground nets. 1: Checks antenna on power and ground nets.

Reducing Critical Areas

A critical area is a region of the design where, if the center of a random particle defect falls there, the defect will cause circuit failure (a yield loss). A conductive defect causes a short fault, and a nonconductive defect causes an open fault.

The following sections describe how to

- Analyze critical areas
- Reduce critical area short faults by performing wire spreading

Analyzing Critical Areas

After routing is complete, you can report layout-critical areas that are susceptible to random particle defects (which cause shorts and opens) during the fabrication process.

The results from the critical area analysis report are output to an `output_heatmap` text file. You can see the critical area results graphically by displaying critical area heat maps.

Reporting Critical Areas

To report critical areas, use the `report_critical_area` command (or choose Finishing > Report Critical Area Map in the GUI). This is the command syntax:

```
report_critical_area
  [-particle_distr_func_file filename]
  [-input_layers layerlist]
  [-fault_type short | open]
  [-tsmc_encr_particle_distr_file]
  [-layer_alias_DSD_format layeraliaslist]
  [-suppress_zeros_in_report]
  [-multiparticle_report_format]
```

For example,

```
icc_shell> report_critical_area -fault_type {open} -input_layers {m2 m3}
```

In the command, specify the fault type (short or open), the input layers to be analyzed, and the name of the input file containing the particle distribution function to be used for analysis. Use `-multiparticle_report_format` to generate a report containing the critical area analysis result of each individual particle size; use this option when you need the critical area analysis result of each particle size instead of normalized results.

Often the particle probability function is considered proprietary data. When security for a sensitive particle distribution file from a foundry is important, you can use the `process_particle_probability_file` command to encrypt and decrypt the particle probability function. Critical area analysis can work with such an encrypted information.

A secret key is used to encrypt the particle probability function. The encrypted file cannot be decrypted without the key.

To encrypt and decrypt a particle probability function file,

- Use the `process_particle_probability_file` command with its options.

To do this	Use this option
Specify the name of the key used to encrypt and decrypt the particle probability function. The string can be a combination of letters and numbers. The key specified for the encryption of a file must be identical to that specified for decryption. The command fails if they do not match. The key must be a minimum of eight characters long.	<code>-key string</code>
Specify the name of the input file. If it is a text file, the output will be an encrypted file. If it is an encrypted file, the output will be a text file.	<code>-input_file file_name</code>
Specify the name of the output file. If none is specified, an output file name will be created, based on the name of the input file (<code>input_file_name_out</code>).	<code>-output_file file_name</code>

Critical area analysis takes the encrypted file as its input and processes it without the key. The output will be the heat map based on the encrypted particle probability function; it will be in text format. The text format of the particle probability function is still accepted as input to critical area analysis. If an encrypted file is given as input, the file is internally decrypted and used.

Displaying Critical Area Maps

Critical area maps provide an indication of places where a chip might fail due to particle defects, causing shorts and opens.

To display a critical area for the current design,

1. Specify the type of defect to display by choosing one of the following:

- Finishing > Open Critical Area Map
- Finishing > Short Critical Area Map

The Map Mode panel appears.

2. Select a metal layer for which critical area shorts or opens are to be displayed.
3. Modify the critical heat ranges by changing (or removing) the maximum or minimum threshold values.
4. Adjust other display parameters. You can make text visible in the map.
5. Click Apply.

To update the critical area map data after you perform wire spreading, click Reload. This action opens the (Re)Calculate Open Critical Area Map Data dialog box, from which you can run the `report_critical_area` command.

For more information about using critical area maps, see the “Examining Critical Area Maps” topic in IC Compiler Help.

Performing Wire Spreading

After you have finished routing, you can perform wire spreading to increase the average spacing between wire, which reduces the critical area short faults and therefore improves yield.

The `route_spreadwires` command can make the following changes to the layout:

- Widen the spaces between wires (move wires on the same layer)
- Move wires to the upper or lower metal layers as needed, to resolve DRC violations caused by widening spaces

Because any change to the routing pattern can affect the timing, IC Compiler provides timing-driven wire spreading to minimize the timing impact. When you enable timing-driven wire spreading, the router calls the timer and marks all the timing-critical nets. The wire spreading process does not touch these nets, so as to preserve the timing.

In most cases, timing is improved, due to the wider net spacing produced by wire spreading. However, having too many timing-violated nets in the design can have a negative impact on the final critical area improvement. Often, when too many nets have violations, the cause is that only a limited number of segments can be moved during spreading. To allow more nets to be spread, set the setup slack threshold option to a negative value. For example, if this option is set to -0.5 , the router spreads all nets that have slack greater than -0.5 . The default is 0.0 .

To perform wire spreading, use the `route_spreadwires` command (or choose Finishing > Route Spread Wires in the GUI). This is the command syntax:

```
route_spreadwires
  [-timing_driven]
  [-search_repair_loop number_of_loops]
  [-setup_slack_threshold setup_time]
  [-min_jog_length min_ratio]
  [-num_cpus number_of_cpus]
```

For example,

```
icc_shell> route_spreadwires -widen_wires -search_repair_loop 6
           -timing_driven -setup_slack_threshold 0.15
```

Use the `-timing_driven` and `-setup_slack_threshold` options to consider timing during wire spreading. In that case, the command finds the timing-critical nets (those having a slack less than the specified threshold value) and prevents spreading from being performed on them. The `-min_jog_length` option sets the minimum ratio of jog length to layer pitch, thereby establishing a minimum jog size. The `-search_repair_loop` and `-num_cpus` options control the computing resources used for wire spreading.

Inserting Redundant Vias

After routing and postrouting optimization, you can replace single-cut vias with multiple-cut via arrays (a process sometimes called via doubling) or with a different via with the same layers. The via is replaced only if the via array or the new via does not introduce DRC violations.

To insert redundant vias, use the `insert_redundant_vias` command (or choose Finishing > Insert Redundant Vias). This is the command syntax:

```
insert_redundant_vias
  -from_via list_of_from_vias
  -to_via list_of_to_vias
  [-to_via_x_size list_of_x-sizes]
  [-to_via_y_size list_of_y-sizes]
  [-via_array_no_swap]
  [-optimize_level level_of_optimization]
  [-num_cpus number_of_cpus]
  [-auto_mode preview | insert]
```

There are two redundant via insertion methods: manual and automatic.

The manual method uses the `-from_via`, `-to_via`, `-to_via_x_size`, and `-to_via_y_size` options to customize the insertion process by specifying which kinds of redundant vias are inserted and the allowed array dimensions. For example, the following command replaces all VIA23 single-cut vias with 1x3 VIA23 arrays and all VIA34 single-cut vias with 5x1 VIA34 arrays:

```
icc_shell> insert_redundant_vias \
  -from_via {VIA23 VIA34} -to_via {VIA23 VIA34} \
  -to_via_x_size {1 5} -to_via_y_size {3 1}
```

The automatic method uses the `-auto_mode` option. If you use `-auto_mode insert`, the command inserts the default vias defined in the technology file as redundant vias for all layers. For example,

```
icc_shell> insert_redundant_vias -auto_mode insert
```

If you use `-auto_mode preview`, the command lists all of the vias on all the layers without referencing the technology file. You can then cut and paste part or all of the listed vias into a new manual-method command that specifies the `-to_via` and `-from_via` options.

The Insert Redundant Vias dialog box in the GUI (Finishing > Insert Redundant Vias) is very helpful in setting the options for an `insert_redundant_vias` command.

Inserting Filler Cells

Filler cells fill gaps in the design to ensure that all power nets are connected and the spacing requirements are met.

- Before routing, you can
 - Insert standard cell fillers
 - Insert end cap cells
- After routing, you can
 - Insert well fillers
 - Insert pad fillers

The following sections describe how to insert these cells.

Inserting Standard Cell Fillers

After placement and clock tree synthesis (if applicable) have been completed, you can fill empty spaces in the standard cell rows with instances of reference filler cells to make sure all power nets are connected. Multiheight filler cells are allowed.

One method of improving the stability of the power supply is to add decoupling capacitors as filler cells. However, these filler cells often contain metal internally, and they can cause a short or a spacing rule violation with existing metal routes in the design. During insertion, filler cells with metal are retained only when they do not cause DRC violations; otherwise, a regular filler cell without metal is inserted.

To insert standard cell fillers,

1. (Optional) Define the standard cell filler rules by using the `set_left_right_filler_rule` command to define the left and right filler rules. These rules specify the filler cell to insert immediately to the left and right, respectively, of specific standard cells.
2. Use the `insert_stdcell_filler` command (or choose Finishing > Insert Standard Cell Filler in the GUI) to insert the standard cell fillers.

The following sections provide more information about these steps.

Defining the Filler Rules

The left and right filler rules specify the filler cell to insert to the immediate left and immediate right, respectively, of specific standard cells. You use the `set_left_right_filler_rule` command to define these rules.

If there is only a single site between two standard cells, the rule used to fill that site depends on whether the references for the standard cells are the same or different. If the references for the two standard cells are the same, IC Compiler uses the rules for the cell on the left and the rules for the cell on the right are ignored. If the references are different, IC Compiler uses the rule that was defined first.

To report the right and left filler rules defined for your design, run the `report_left_right_filler_rule` command. The report shows the rules and the order in which they were defined.

By default, the left and right filler cells have a north (N) orientation or flipped-south (FS) orientation when the rows are flipped. You can choose to have the left and right filler cells follow the orientation of the standard cell by using the `set_left_right_filler_rule -follow_stdcell_orientation` option.

Inserting the Filler Cells

To insert standard cell fillers, use the `insert_stdcell_filler` command (or choose Finishing > Insert Standard Cell Filler in the GUI). This is the command syntax:

```
insert_stdcell_filler
  [-cell_without_metal lib_cell_without_list]
  [-cell_with_metal lib_cell_with_list]
  [-bounding_box rectangle]
  [-dont_respect_hard_placement_blockage]
  [-dont_respect_soft_placement_blockage]
  [-between_std_cells_only]
  [-randomize]
  [-respect_overlap]
  [-cell_without_metal_prefix cell_without_prefix]
  [-cell_with_metal_prefix cell_with_prefix]
  [-avoid_layers layers_list]
  [-connect_to_power Power_net_name]
  [-connect_to_ground Ground_net_name]
  [-voltage_area voltage_area_list]
  [-vt_filler lib_cell_list]
  [-check_only]
  [-restore_filler_snapshot]
  [-vt_filler_prefix vt_filler_prefix]
  [-respect_keepout]
```

For example,

```
icc_shell> insert_stdcell_filler \  
-cell_without_metal {FILL_4X FILL_2X FILL_1X} \  
-cell_with_metal {FILL_4XM FILL_2XM FILL_1XM} \  
-bounding_box {{10.0 10.0} {5000.0 5000.0}} \  
-between_std_cells_only -avoid_filler_under {M3 M4}
```

This example fills empty spaces between standard cells with filler cells FILL_4X, FILL_2X, and FILL_1X, in that order of preference, in the bounding box with corners at (10,10) and (5000,5000). By specifying the filler cells from largest to smallest, the total number of filler cells is minimized by adding the larger filler cells first. Cells with metal where they do not cause DRC violations, and cells without metal are used otherwise. No filler cells are inserted under metal 3 or metal 4 preroutes.

The `insert_stdcell_filler` also has options to specify the following:

- Whether to respect hard placement blockages, soft placement blockages, keepout margins, and standard-cell overlap check points
- Whether to randomize the selection of filler cells (useful when the filler cells are the same size)
- Filler cell instance naming conventions
- Power and ground connections (useful in multivoltage designs)
- Voltage areas in which to insert filler cells
- Threshold voltage cells to be used as Vt fillers
- Voltage areas in which to insert filler cells
- Whether to perform filler cell insertion or only report the empty spaces

Removing Filler Cells

To remove standard cell fillers, use the `remove_stdcell_filler -stdcell` command (or choose Finishing > Remove Fillers and select “Standard Cell” in the “Filler type” area in the GUI). By default, the filler cells are removed for the whole chip. You can optionally specify a bounding box from which to remove filler cells.

When filler cells are added after signal routing, you can remove all the filler cells that have routing design rule violations. To remove standard cell fillers with violations, use the `remove_filler_with_violation` command (or choose Finishing > Remove Fillers With Violation in the GUI). You can restrict the removal to specific instances by using the `-name` option.

Inserting End Caps

After placing standard cells and before routing, you can add end cap cells at both ends of a cell row. Typically, an end cap cell is a nonlogic cell that serves a certain purpose for the row such as providing a decoupling capacitor for the power rail. Because IC Compiler accepts any standard cell as an end cap, be sure to specify a suitable end cap cell.

To insert end caps, use the `add_end_cap` command (or choose Finishing > Insert End Cap in the GUI). This is the command syntax:

```
add_end_cap
  [-mode both | bottom_left | upper_right]
  [-mirror]
  [-respect_padding]
  [-respect_blockage]
  -lib_cell lib_cell_names [-respect_keepout]
```

For example,

```
icc_shell> add_end_cap -lib_cell MY_END_CAP
```

You must specify the name or names of the end cap cells to be used. The command also has options to specify which ends of rows are to receive end caps, whether to flip the end cap cell, and whether to respect macro padding and placement blockages.

Inserting Well Fillers

After routing is complete, you can fill small gaps that violate the spacing rule for the well layer with well filler cells. You can fill gaps between cells in the same row or between rows.

To insert well fillers, use the `insert_well_filler` command (or choose Finishing > Insert Well Filler in the GUI). This is the command syntax:

```
insert_well_filler
  -layer layer_name_or_number
  [-ignore_PRboundary]
  [-fill_gaps_smaller_than gap_size]
  [-higher_edge min | max]
  [-lower_edge min | max]
  [-gap_type {tt | bb | tb | bt}]
  [-respect_blockages]
  [-row_overlap {row_overlap_value}]
  [-min_gap {min_gap_distance}]
  [-max_gap {max_gap_distance}]
  [-enclosure_only width]
```

For example,

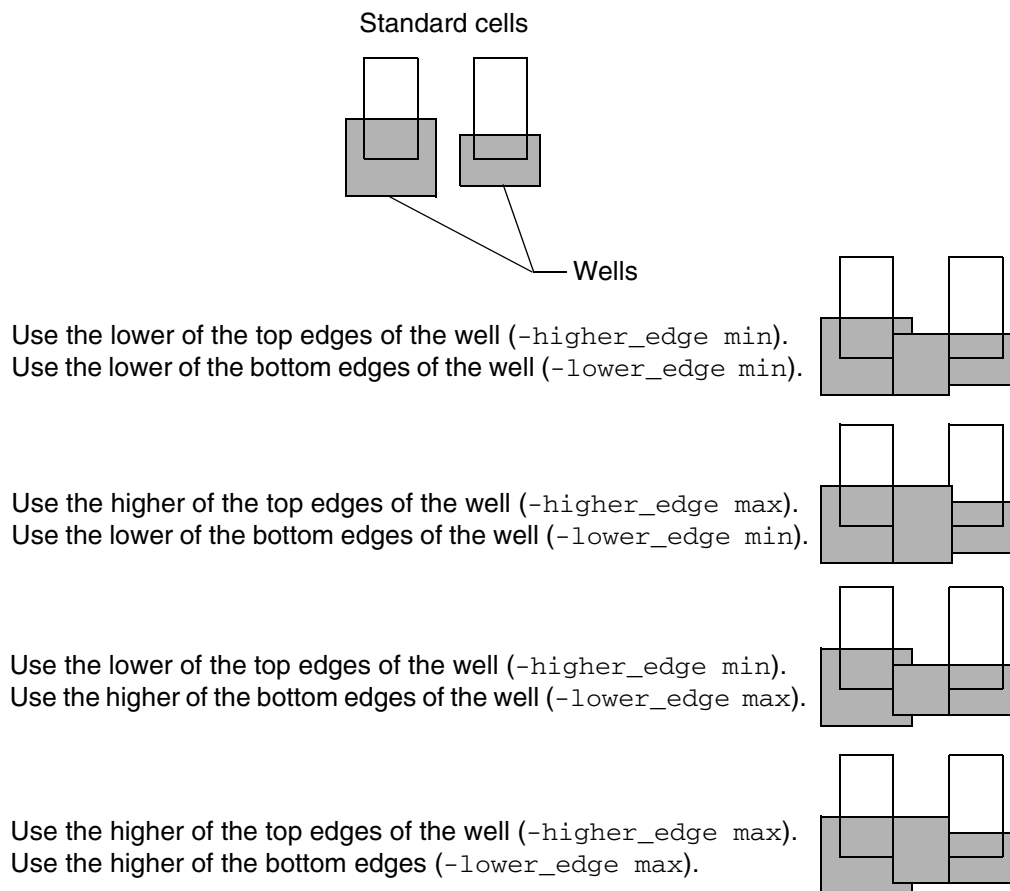
```
icc_shell> insert_well_filler -layer 10 \  
          -fill_gaps_smaller_than 3.0 \  
          -higher_edge "max" -lower_edge "min"
```

This example adds well filler on layer 10 for gaps smaller than 3.0 microns. If the wells from the two standard cells do not line up, the command creates the largest fill box by using the larger top edge and the smaller bottom edge.

By default, only the gaps between standard cells within a row are filled. To fill gaps between rows, use the `-gap_type` option and set it to `tt`, `bb`, `tb`, or `bt`.

The `-higher_edge` and `-lower_edge` options specify how to align the well filler with the wells in the two standard cells when the wells in the cells do not line up. Figure 9-4 shows the alignments and fill boxes that occur as a result of the various settings.

Figure 9-4 Filler Alignment



The `insert_well_filler` also has options to specify the following:

- Whether to ignore the PR boundary layer (layer 207) that extends outside a cell
- Whether to respect placement blockages when the `-gap_type` option is used
- The amount of row overlap between the row and the gap when the `-gap_type` option is used
- The minimum and maximum gap sizes that are filled when the `-gap_type` option is used
- Whether to insert a well enclosure, instead of well fill, and if so, the enclosure width

To remove well fillers, use the `remove_well_filler` command (or choose Finishing > Remove Well Fillers in the GUI).

Inserting Pad Fillers

After routing is complete, you can fill gaps in the pad ring with instances of pad filler cells. These are dummy pad cells that you can use to control the pad spacing and complement the n-well taps in pads. It is recommended that you complete the routing process before you add pad filler cells.

To insert pad fillers, use the `insert_pad_filler` command (or choose Finishing > Insert Pad Filler in the GUI). This is the command syntax:

```
insert_pad_filler
-cell lib_cells
[-overlap_cell overlap_lib_cells]
[-voltage_area voltage_area_list]
[-bounding_box rectangle]
[-prefix prefix]
[-no_left]
[-no_right]
[-no_bottom]
[-no_top]
```

For example,

```
icc_shell> insert_pad_filler -cell "PFILL_2X PFILL_1X" \
            -overlap_cell "PFILL_1X" -voltage_area {V1}
```

This example inserts pad filler cells PFILL_2X and PFILL_1X on the pad ring, with preference for PFILL_2X because it is listed first, inside voltage area V1 only. The PFILL_1X cell is allowed to overlap other pad filler cells to fill gaps that are too small for any pad filler cell.

The command also has options to restrict pad filler insertion to a specified rectangular area, to specify pad filler instance naming conventions, and to exclude pad filler insertion from left, right, top, or bottom boundaries.

To remove pad fillers, use the `remove_stdcell_filler -pad` command (or choose Finishing > Remove Fillers in the GUI and select “Pad” in the “Filler type” area). By default, pad fillers are removed for the whole chip. You can optionally specify a bounding box from which to remove pad fillers.

Performing Metal Density Filling

After routing, you can fill the empty tracks in the design with metal wires to meet the metal density rules required by most fabrication processes. This should be done before layout versus schematic (LVS) connectivity verification, design rule checking (DRC), and layout parasitic extraction.

When you define minimum and maximum metal density rules in the technology file, IC Compiler tries to create fill within the specified ranges.

The router does not recognize dummy metal that the GDS reader creates in the FILL view because the dummy metal areas are created as polygons, not wires. However, if they are converted to rectangles and marked with the Fill Track route type attribute, the router will recognize them correctly. Use the `convert_fill_polygons` command to convert the existing FILL view polygons to rectangles and to mark the rectangles with the Fill Track route type. This command should be called as part of data preparation so that the router does not have a problem reading the fill wires. The syntax is

```
convert_fill_polygons -library library_name -from cell_name -to cell_name
```

where *library_name* is the name of the Milkyway library containing the design cell to modify and *cell_name* is the name of the design cell to modify.

The `convert_fill_polygons` command converts the input cell’s FILL view polygons to rectangles and then stores the cell in the FILL view of the specified output cell name. The input cell is not actually modified unless the output cell name is the same as the input cell name.

IC Compiler supports both real and emulation metal fill extraction.

- To perform emulation metal fill extraction, you must specify the emulation TLUPlus files by using the `-max_emulation_tluplus` and `-min_emulation_tluplus` options of the `set_tlu_plus_files` command.
- To perform real metal fill extraction, you must specify the `-real_metalfill_extraction` option of the `set_extraction_options` command. IC Compiler can treat the metal fill polygons as either floating or grounded. You can either specify how to treat the metal fill polygons (use the `floating` or `grounded` keyword) or let IC Compiler determine the treatment, based on the fill wire track property (use the `auto` keyword). When you use the `none` keyword (the default), fill is not considered during extraction.

In addition to specifying the `-real_metalfill_extraction` option, you must specify TLUPlus files without any emulation information, using the `set_tlu_plus_files` command.

Note:

IC Compiler uses non-emulation TLUPlus files when `set_extraction_options -real_metalfill_extraction` is set to `auto`, `floating`, or `grounded`.

Therefore, you should select real metal fill for extraction only after the metal fill objects have been added.

There are two ways to perform metal fill: by using the internal IC Compiler metal fill capability (`insert_metal_filler` command or Finishing > Insert Metal Filler in the GUI) or by invoking the external Hercules metal fill capability (`signoff_metal_fill` or Finishing > Signoff Metal Fill in the GUI). Using Hercules metal fill ensures DRC conformance and produces better quality of results, but requires a Hercules license.

IC Compiler Metal Fill

To perform metal density filling in IC Compiler, use the `insert_metal_filler` command (or choose Finishing > Insert Metal Filler in the GUI). This is the command syntax:

```
insert_metal_filler
  [-out self | cellview_name]
  [-purge]
  [-bounding_box {{llx lly} {urx ury}}]
  [-dont_overwrite]
  [-timing_driven]
  [-insert_as_instance instance_name]
  [-tie_to_net none | ground | net_name]
  [-create_floating_vias]
  [-floating_via_ftr_spacing]
  [-routing_space route_space]
  [-from_metal from_metal_number]
  [-to_metal to_metal_number]
  [-width {layer width}]
  [-space {layer space_between_fills}]
  [-min_length {layer min_length}]
  [-max_length {layer max_length}]
  [-space_to_route {layer keep_from_metal_space}]
  [-space_to_pg {layer keep_from_pg_nets_space}]
  [-stagger {layer}]
  [-x_offset {layer x_distance}]
  [-y_offset {layer y_distance}]
  [-dont_snap_fill_to_track]
  [-fill_poly]
  [-distance_to_boundary poly_to_cell_distance]
```

You can use the `-width`, `-space`, `-min_length`, `-max_length`, `-space_to_route`, `-space_to_pg`, `-x_offset`, and `-y_offset` options to specify the fill characteristics of one or more layers. You specify each characteristic using one or more pairs of values. Each pair consists of a layer name such as `poly`, `m1`, or `m2`; followed by the value. For example,

```
icc_shell> insert_metal_filler \
           -width {m1 0.1} \
           -space {poly 0.1} \
           -min_length {m1 2 m2 2 m3 2} \
           -max_length {m1 10 m2 10 m3 10}
```

This example fills tracks using width 0.1 for metal 1 and spacing 0.1 for poly, with lengths ranging from 2 to 10 for metal 1, metal 2, and metal 3. The default values are used for the remaining metal fill characteristics not specified explicitly in the command.

These are the fill characteristics you can set:

- `-width`: the width of fill tracks
- `-space`: the spacing between fill tracks on the same layer
- `-min_length`: the minimum length of fill tracks
- `-max_length`: the maximum length of fill tracks
- `-space_to_route`: the space to keep away from existing metal
- `-space_to_pg`: the space to keep away from power and ground nets
- `-x_offset`: the x-offset for staggered fill tracks
- `-y_offset`: the y-offset for staggered fill tracks

Fill tracks are staggered if you use the `-x_offset`, `-y_offset`, or `-stagger` option. In that case, if you do not want the fill snapped to wire tracks, use the `-dont_snap_fill_to_track` option.

Use the `-purge` option to remove, rather than insert, metal fill. You can remove fill-track objects from a specified range of layers using `-from_metal` and `-to_metal`. Otherwise, the `-purge` option removes fill-track objects from all layers.

The `insert_metal_filler` command also has options to specify the following:

- Whether to modify the current cell (the default) or write the output to a new cell
- A bounding box in which to insert metal fill
- The net to which the metal fill must be connected
- The space between normal routing wires and the fill metal, expressed as a multiplier of the minimum spacing for the layer (default 1.0)
- The range of layer numbers to fill
- Whether to fill poly layers as well as metal layers (metal layers only by default)
- Whether to create floating vias, and if so, the required spacing
- Whether to check for timing effects during metal fill
- The spacing of poly to the cell boundary

Hercules Metal Fill

Using the external Hercules metal fill capability ensures DRC conformance and produces the best quality of results. However, it requires a separate Hercules license. Without leaving IC Compiler, you can invoke Hercules to create, view, and change metal fill by using the `signoff_metal_fill` command or Finishing > Signoff Metal Fill in the GUI.

If you plan to use Hercules metal fill from the IC Compiler environment, perform any required setup in the Hercules environment before you invoke IC Compiler. For setup information, see the *General Information on Using Hercules* guide, available on SolvNet.

In IC Compiler, the design must be fully routed and DRC-clean. You must save the most recent revision of the design in a CEL view. Hercules reads and operates on the design data in CEL, FRAM, and FILL views stored on disk, not on the current design in IC Compiler memory.

To prepare for Hercules metal fill, use the `set_physical_signoff_options` command to specify the name of the Hercules runset file for performing metal fill, as shown in the following example:

```
icc_shell> set_physical_signoff_options \  
          -fill_runset my_fill_runset_file
```

You can report the option settings with the `report_physical_signoff_options` command. For more information, see the man page for the command.

To perform metal fill using Hercules, use the `signoff_metal_fill` command (or choose Finishing > Signoff Metal Fill in the GUI). This is the command syntax:

```
signoff_metal_fill  
  [-output_view fillview_name]  
  [-purge]  
  [-eco]  
  [-select_layers {layer_names}]
```

For example,

```
icc_shell> signoff_metal_fill -eco -select_layers {m1 m3}
```

This example removes all metal fill on layers m1 and m3 and then refills those two layers. The other layers are not affected.

The `-output_view` option, if used, specifies the name of the new FILL view created by the command. The default is `cell_name.FILL`.

The `-purge` option removes metal fill in the default FILL view of the current cell. The fill is removed from the layers specified with the `-select_layers` option, or the whole FILL view is removed if `-select_layers` is not used.

The `-eco` and `-select_layers` options can be used to perform post-ECO, layer-based fill removal and refilling. IC Compiler removes existing metal fill from the specified layers and then calls Hercules to refill those layers. The other layers are not affected.

In a layout view in the GUI, you can display metal fill that has been added. In the View Settings panel (View > Toolbars > View Settings), set the View Level to 1, and under the Layers tab, toggle on the Datatypes option. Toggle on the view of the desired fill layer.

After performing metal fill, you can do extraction for timing analysis using the real metal fill, as shown in the following example:

```
icc_shell> set_extraction_options -real_metalfill_extraction floating
...
icc_shell> extract_rc -coupling_cap
...
icc_shell> report_timing
...
icc_shell> report_qor
...
```

Performing Notch and Gap Filling

After routing is complete, you can fill notches and gaps that are smaller than the minimum distance limit between objects of the same net on the same layer. The generated notch-and-gap-filling information is stored in the FILL view cell and can be used when you translate your design data to GDSII format.

To perform notch and gap filling, use the `insert_ng_filler` command (or choose Finishing > Insert Notch Gap Filler in the GUI). This is the command syntax:

```
insert_ng_filler
[-include_existing_notch_gap_fill_cell]
[-skip_filling_child_cell]
[-dont_apply_fat_wire_rule]
[-dont_fill_corner_to_corner]
[-out outname]
[-notch_layer_data_type notch_datatype]
[-gap_layer_data_type gap_datatype]
[-layers layer_list]
```

For example,

```
icc_shell> insert_ng_filler -include_exisitng_notch_gap_fill_cell \  
-dont_apply_fat_wire_fule -out test.err
```

The command has options to specify the following:

- Whether to use existing notch and gap cells existing in the netlist and having the .FILL extension
- Whether to skip reading the child cell's port data and applying fill on those ports.
- Whether to ignore the fat wire rule when filling notches and gaps
- Whether to ignore corner-to-corner notch errors
- Whether to save the notch and gap errors into a file
- The notch layer data type and gap layer data type numbers
- The list of layers for which to fill notches and gaps

You can also perform notch and gap filling during the search-and-repair process. Doing so enables the router to correct DRC violations as it fills notches and gaps, including violations of 90-nm rules. Set the relevant routing option using the command `set_route_options -same_net_notch check_and_fix` (or choose Route > Routing Setup > Set Route Options in the GUI and select the corresponding option under the Miscellaneous tab). This setting instructs the router to fix same-net notch violations during the search-and-repair process. For more information, see [“Search and Repair” on page 8-32](#)

Analyzing CMP Thickness Variation Effects

You can load a parasitic netlist with thickness variation effects generated by the Star-RCXT tool and view chemical-mechanical polishing (CMP) maps to examine thickness variations or hot spots in your design.

Loading the Thickness Data

Before you can display a chemical-mechanical polishing map during a session, you must load the thickness data from a file. The thickness data comes from the Star-RCXT tool.

To load the thickness variation file,

- Use the `read_cmp_thickness -filename string` command.

The information in the file is saved to the database and used for the display of the chemical-mechanical polishing thickness variation and hot spot maps.

The named file can be in single-layer or multilayer format. It contains tile wise information for the entire chip and provides information for each metal layer.

- For single-layer format, the entries of each tile on each line are as follows:

```
"X Coordinate"  
"Y Coordinate"  
"Thickness Variation"
```

- For multilayer format, the entries of each tile on each line are as follows:

```
"X Coordinate"  
"Y Coordinate"  
"Lower Thickness Variation"  
"Upper Thickness Variation"  
"Density Percent"  
"Cumulative Topography"  
"Oxide Thickness"  
"Metal Thickness"
```

Displaying Chemical-Mechanical Polishing Maps

You can display either a thickness variation or a hot-spot type of chemical-mechanical polishing map.

To display a thickness variation map for the current design,

1. Choose Finishing > CMP Thickness.

The Map Mode panel appears.

2. Select a metal layer for which thickness variations are to be displayed.
3. Modify the thickness value ranges by changing (or removing) the maximum or minimum threshold values.
4. Click Apply.

To display a hot spot map for the current design,

1. Choose Finishing > Hot Spot.

The Map Mode panel appears.

2. Select a metal layer for which hot spots are to be displayed.
3. Set or adjust thickness variation, topography variation, average topography variation, or other rules. You can set or adjust the following rules. Variances shown in red on the map are hot spots based on the rule setting.

- Rule 1: Thickness variation from average

Computes the average chemical-mechanical polishing variation for all the tiles in the design. If any tile has a chemical-mechanical polishing variation more than the specified percentage from the average for that layer, it is shown in red.

- Rule 2: Thickness variation from neighbors

Computes the average thickness variation for all the neighbors of each tile. If the thickness variation of the current tile varies by more than the specified percentage from the average value of its neighbors, it is shown in red.

- Rule 3: Topography variation from average

Computes the average cumulative topography of all the tiles of the layer. If the cumulative topography of a tile is off by more than the specified percentage from the average topography of that layer, it is shown in red.

- Rule 4: Topography variation from neighbors

Computes the average cumulative topography for all the neighbors of each tile. If the thickness variation of the current tile varies by more than the specified percentage from the average value of its neighbors, it is shown in red.

- Rule 5: Oxide thickness gradient

For each tile in the design, calculates the maximum value of absolute difference between its oxide thickness and that of its neighbors on the same layer. If this absolute difference is greater than the specified value, this tile is shown in red.

- Rule 6: Row topography

For each row on the specified layer, computes the maximum and minimum values of cumulative topography. If the difference between the maximum and minimum values is greater than the specified value, the entire row is shown in red.

- Rule 7: Column topography

For each column on the specified layer, computes the maximum and minimum values of cumulative topography. If the difference between the maximum and minimum values is greater than the specified threshold, the entire column is shown in red.

- Rule 8: Dishing/Antidishing

For each tile on the specified layer, computes the absolute value of difference between metal thickness and oxide thickness. If this value is greater than the specified threshold, that tile is shown in red.

4. Click Apply.

Before you can display a chemical-mechanical polishing map during a session, you must load the thickness data from a file. The thickness data comes from the Star-RCXT tool.

To load the thickness data for the current design,

1. Click the Reload button on the Map Mode panel.
2. In the file browser dialog box that appears, navigate to the directory where the file is located and select the file name.

Alternatively, you can type the name in the “File name” box.

3. Click Open.

For more information about using chemical-mechanical polishing maps, see the “Examining Chemical Mechanical Polishing Maps” topic in IC Compiler Help.

Lithography Compliance Checking

Lithography compliance checking (LCC) is a method of increasing yields by analyzing the chip layout in detail for conditions that can lead to lithography defects and that cannot be fixed by ordinary optical proximity correction methods. PrimeYield LCC is a Synopsys product designed to find these conditions and generate reports on LCC hotspots (locations of potential defects). It simulates the full resolution-enhancement technology tape-out flow using the same production-baseline technology and manufacturing models as those used by foundries and integrated device manufacturers. It reports the location and type of each occurrence of lithographic sensitivity to potential defects such as line-end narrowing and line-end bridging. It ranks these problems by severity and presents them for review.

In IC Compiler, you can detect potential defect locations with the `detect_lcc_hotspot` command, which invokes PrimeYield LCC in the background to find the hotspots. You can view the reported hotspots in the IC Compiler GUI and fix them with the `fix_lcc_hotspot`

command. The fixing command repairs the reported conditions by moving, filling, and widening routes and by adding, deleting, and moving vias; or in cases where these local fixing methods cannot be used effectively, it rips up and reroutes the problem net.

PrimeYield LCC licensing is required to use these commands. The PrimeYield LCC and Hercules version used by IC Compiler must be the same as the version used to generate the LCC data files. For information on setting up and using PrimeYield LCC, see the *PrimeYield LCC User Guide* provided with the PrimeYield product.

LCC detection should be performed only on a DRC-clean design after via optimization and critical area reduction have been completed.

Detecting LCC Hotspots

The `detect_lcc_hotspot` command invokes PrimeYield LCC to detect potential lithography-related defects. This is the full command syntax:

```
detect_lcc_hotspot
  -lcc_file_path lcc_path_name
  -layers list_of_layers
  [-dp_hosts list_of_dp_hosts]
```

For example,

```
icc_shell> detect_lcc_hotspot \
            -lcc_file_path /LCC/full-chip \
            -layers {M2 M3 M4} \
            -dp_hosts {lin1 lin2 lin3 lin4 lin5 lin6 lin7 lin8}
```

This command performs full-chip LCC detection and produces two fixing guidance files in the working directory, `out00` and `fout00`. These two files contains hotspot information and guidance for local fixing and reroute fixing. They are required to run the `fix_lcc_hotspot` command.

The hotspot detection process also produces an error cell named `top_cell_name_lcc.err`. You can open this error cell in the GUI using Verification > Error Browser and view the hotspots in the current design. Each hotspot is highlighted as a DRC error.

In the `detect_lcc_hotspot` command, you must specify the full path name of the directory containing the PrimeYield LCC runset files necessary for full-chip LCC detection. For information on preparing runset files, see the *PrimeYield LCC User Guide* provided with the PrimeYield product; or see the article “Using PrimeYield LCC,” available as SolvNet article 023080. You must also specify the layers to be checked.

The `detect_lcc_hotspot` command requires distributed processing because of the long runtime for full-chip LCC detection. You can specify the machines to be used with the `-dp_hosts` option, the `.dprc` file under the working directory, or the `.dprc` file specified under the `lcc_path_name` directory. Running hotspot detection on a single CPU is not supported.

The distributed processing engine used by `detect_lcc_hotspot` is different from that used by the `fix_lcc_hotspot` command and many other routing commands. You do not need to run the `set_distributed_route` command before `detect_lcc_hotspot`. If you run `set_distributed_route`, you should run `remove_distributed_route` before `detect_lcc_hotspot` to release the reserved network resources. Otherwise, the communication port between machines might become occupied and block distributed processing functionality.

Fixing LCC Hotspots

The `fix_lcc_hotspot` command attempts to fix hotspots previously detected with the `detect_lcc_hotspot` command. The design must be DRC-clean to obtain maximum fixing quality and to ensure iteration convergence. The two fixing guidance files generated by the `detect_lcc_hotspot` command, `out00` and `fout00`, must be present in the working directory.

This is the full command syntax of the LCC hotspot-fixing command:

```
fix_lcc_hotspot
-lcc_file_path lcc_path_name
[-types list_of_types]
[-level level]
[-num_loops number_of_loops]
[-num_cpus number_of_cpus]
```

For example,

```
icc_shell> set_distributed_route \
           -jp_machines {linux1 linux2 linux3 linux4 linux5}

icc_shell> fix_lcc_hotspot \
           -lcc_file_path /root/LCC/Local-LCC \
           -types {line space lineend slotend} \
           -level 1 -num_cpus 10
```

Distributed processing is recommended because runtimes can be long. First run the `set_distributed_route` command to set up the machines to be used. Then, in the `fix_lcc_hotspot` command, set the `-num_cpus` option to a value greater than 1. For more information on distributed routing, see the [“Enabling Distributed Routing” on page 8-15](#).

In the `fix_lcc_hotspot` command, you must specify the full path name of the same LCC directory used by the `detect_lcc_hotspot` command. This is the directory containing the PrimeYield LCC runset files necessary for full-chip LCC detection. You can optionally specify the types of hotspots to fix, the hotspot severity levels to fix, the maximum number of fixing loops to attempt, and the number of CPUs to use.

The `-types` option specifies the types of hotspots to fix, which can be any combination of `line`, `lineend`, `space`, `slotend`, and `voc` (via overlay check). The default behavior is to fix all types of hotspots.

The `-levels` option specifies the severity levels to fix. If you use this option, the command attempts to fix only the hotspots having a severity number equal to or less than the specified number. Lower numbers represent the more severe hotspots.

There are two severity level systems, LCC-t and LCC-g. IC Compiler gets the severity level system, either LCC-t or LCC-g, from the LCC runsets. You need to know which level system is being used so you can enter an appropriate level number. The LCC-g system has the possible level values 1, 2, 3, 4, and 5. If you specify `-levels 3` in this system, the command fixes only the hotspots having a severity level of 1, 2, or 3. The default maximum severity value for this system is 2.

The `fix_lcc_hotspot` command attempts to fix LCC hotspots using multiple iterations of fixing and rechecking. In each iteration, it first tries to fix hotspots by local-fixing and rerouting. New DRC violations or new LCC hotspots might occur as a result of fixing, so it performs internal DRC checking and launches LCC checking jobs to check the changed patterns. This process is repeated until there are no more LCC hotspots or DRC violations, or until the number of iterations reaches the maximum value specified by the `-num_loops` option. Most designs that can converge do so in less than 10 iterations.

Part II: Advanced IC Compiler Features

10

Signal Integrity

Signal integrity is the ability of an electrical signal to carry information reliably and to resist the effects of high-frequency electromagnetic interference from nearby signals. This chapter describes how to use IC Compiler to analyze and correct signal integrity problems.

The following conditions can impact signal integrity:

- Crosstalk

Crosstalk is the undesirable electrical interaction between two or more physically adjacent nets due to capacitive coupling. Crosstalk can lead to crosstalk-induced delay changes or static noise.

- Signal electromigration

Electromigration is the permanent physical movement of metal in thin wire connections resulting from the displacement of metal ions by flowing electrons. Electromigration can lead to shorts and opens in wire connections, causing functional failure of the IC device. The problem is more severe in modern technologies due to smaller wire widths and increased current densities.

IC Compiler supports signal integrity analysis and optimization in either a flat flow or a hierarchical flow (using interface logic models). When creating an interface logic model for use in a hierarchical signal integrity flow, you must use the `-include_xtalk` option when you run the `create_ilm` command.

IC Compiler signal integrity analysis and optimization supports multivoltage and multimode-multicorner designs.

This chapter contains the following sections:

- [Analyzing and Reducing Crosstalk](#)
- [Analyzing and Reducing Signal Electromigration](#)

Analyzing and Reducing Crosstalk

Crosstalk is the undesirable electrical interaction between two or more physically adjacent nets due to capacitive coupling. The two major effects of crosstalk are crosstalk-induced delay and static noise.

- Crosstalk can affect signal delays by changing the times at which signal transitions occur. That is, it can either speed up or slow down the transition time of the victim net. These changes occur when both the victim and the aggressor nets are switching at the same time. When both are switching in the same direction, the transition time of the victim net decreases and delay is reduced. When the victim and aggressor nets are switching in opposite directions, the transition time of the victim net increases and delay is increased. Therefore crosstalk can affect both setup and hold checking. The noise on the victim net is referred to as switching noise.
- Static noise occurs when the victim net is in a steady state of either a high or low, and the aggressor net is switching. This causes a glitch on the victim net. If a glitch is of sufficient height and width, and if it occurs at the input of a latching structure during a latching operation (possibly as the result of an earlier glitch propagated through combinational logic), then a functional failure will result.

IC Compiler uses crosstalk prevention techniques during track assignment. After you perform detail routing, IC Compiler performs crosstalk-induced noise and delay analysis to identify any remaining violations, and it fixes these violations during the post-routing optimization phases.

- During timing-driven and crosstalk-aware track assignment, IC Compiler minimizes the crosstalk effects by assigning long, parallel nets to nonadjacent tracks. It runs a simplified noise analysis to make sure the noise level from aggressor nets is minimized.
- After detail routing is complete, IC Compiler repairs the remaining problems, first by analyzing the coupling capacitance effects of the circuit and then by crosstalk removal with post-route optimization.

In crosstalk analysis, for each net, IC Compiler extracts parasitic capacitance to the ground as well as any coupling capacitance with neighboring wires. IC Compiler also invokes static timing analysis, using the extracted parasitics to obtain the transition times as well as the arrival time windows of every signal net in the design.

This section contains the following information about the crosstalk capabilities supported by IC Compiler:

- [Setting the Signal Integrity Options](#)
- [Preventing Crosstalk During Placement](#)
- [Preventing Crosstalk During Clock Tree Synthesis](#)
- [Preventing and Fixing Crosstalk During Routing](#)
- [Analyzing Crosstalk](#)
- [Preventing Crosstalk During Sign-Off Optimization](#)
- [Sample Scripts](#)

Setting the Signal Integrity Options

The signal integrity options affect the analysis of crosstalk delay and static noise by the `report_timing` and `report_noise` commands and the optimization of crosstalk delay and static noise in the `route_opt`, `psynopt -on_route`, and `signoff_opt` commands.

You set the signal integrity options by using the `set_si_options` command (or by choosing Timing > Set SI Options in the GUI). [Table 10-1](#) describes the GUI objects and command-line options used to set the options for various signal integrity tasks.

Table 10-1 Signal Integrity Options

GUI object	Command option	Default
Crosstalk prevention options		
“Enable xtalk prevention” check box	-route_xtalk_prevention	false
“Prevention threshold” field	-route_xtalk_prevention_threshold	0.45
Crosstalk optimization options		
“Delta delay” check box	-delta_delay	false
“Min delta delay” check box	-min_delta_delay	false
“Static noise” check box	-static_noise	false
“Static noise threshold (voltage)” fields	-static_noise_threshold_above_low -static_noise_threshold_below_high	0.35 0.35
N/A	-max_transition_mode	normal_slew
Crosstalk analysis options		
“Delta delay” check box	-delta_delay	false
“Min delta delay” check box	-min_delta_delay	false
“Static noise” check box	-static_noise	false
“Static noise threshold (voltage)” fields	-static_noise_threshold_above_low -static_noise_threshold_below_high	0.35 0.35
N/A	-max_transition_mode	normal_slew
“Timing window” check box	-timing_window	false
“Analysis effort” list box	-analysis_effort	low

To report the current signal integrity option settings, run the `report_si_options` command.

Preventing Crosstalk During Placement

To prevent crosstalk at the placement phase, do the following:

- Minimize congestion.

Reducing congestion improves the routability of the design and adds more resources for extra spacing requirements during crosstalk-driven global route and track assignment. Fixing crosstalk on a congested design is very difficult.

- Have good timing closure before crosstalk fixing. If necessary, add more margin before crosstalk fixing.
- Use area recovery on noncritical paths during `place_opt` to reduce congestion and improve routability. Set the `physopt_area_critical_range` variable to a value that is about 15 percent of the clock period to avoid downsizing paths close to zero slack.
- Help prevent crosstalk by controlling the maximum transition constraint defined on the design. The maximum transition constraint is technology and library dependent. You need to find the best tradeoff between a low maximum transition constraint and congestion. The maximum transition constraint can be relaxed during postroute optimization.
- Use the maximum net length constraint in IC Compiler to minimize the crosstalk effect by preventing very long wires. You must determine the best tradeoff between a low `max_net_length` value and congestion. For a design without macros or only small macros, the `max_net_length` value might be 1000.0; for a design with large macros or high utilization, the `max_net_length` value might be 2000.0 or 3000.0.

Preventing Crosstalk During Clock Tree Synthesis

Because clock nets are typically high-frequency nets, they are often strong aggressor nets. You can prevent crosstalk by shielding clock nets with ground wires.

IC Compiler handles clock shielding with nondefault routing rules. For information about the clock shielding flow, see [“Shielding Clock Nets” on page 7-28](#).

Preventing and Fixing Crosstalk During Routing

During the `route_opt` command, IC Compiler can perform the following signal integrity tasks:

- Crosstalk prevention (during global routing and track assignment)
- Crosstalk fixing (during post-route optimization)

By default, these capabilities are disabled. To enable these capabilities, set the appropriate signal integrity options and then run the `route_opt` command with the `-xtalk_reduction` option.

IC Compiler has two crosstalk reduction engines:

- A fast crosstalk reduction that is fast with good QoR (the default)
Enable this crosstalk reduction engine by specifying `route_opt -effort medium`.
- A high-performance crosstalk reduction engine that is slower than the default crosstalk engine, but that provides slightly better QoR.
Enable this crosstalk reduction engine by specifying `route_opt -effort high`.

Preventing Crosstalk

The `route_opt` command can prevent crosstalk in the following ways:

- Crosstalk prevention during track assignment
Enable crosstalk prevention by running `set_si_options -route_xtalk_prevention true` and using the `-xtalk_reduction` option when you run `route_opt`.

You can set the crosstalk prevention threshold voltage by using the `-route_xtalk_prevention_threshold` option of the `set_si_options` command. The lower the threshold voltage, the more that the router tries to prevent crosstalk. The default threshold is 0.45 volts, which could be too relaxed. If your design has crosstalk violations, you should use a lower crosstalk prevention threshold during track assignment, in the range of 0.25 to 0.35 volts.

When you enable crosstalk prevention, the `route_opt` command avoids putting long, parallel wires on adjacent tracks during track assignment. To minimize noise, track assignment estimates the potential noise with a simplified crosstalk checker and reassigns wires to reduce potential noise exceeding the noise threshold.

- Timing-driven global routing

Enable timing-driven global routing by using the `set_route_options -groute_timing_driven true` command (or by choosing Route > Routing Setup > Set Route Options in the GUI and selecting “Timing driven” in the Global Routing tab).

- Timing-driven track assignment

Enable timing-driven track assignment by using the `set_route_options -track_assign_timing_driven true` command (or by choosing Route > Routing Setup > Set Route Options in the GUI and selecting “Timing driven” in the Track Assign tab).

Fixing Crosstalk Violations

For best results, use the crosstalk prevention process in conjunction with crosstalk fixing. When crosstalk is prevented in the placement stage and then in the track assignment stage, the routed design has a smaller number of crosstalk violations. Consequently crosstalk fixing in the post-route optimization stage will be more effective and have a shorter runtime.

Some crosstalk violations might still remain unfixed after you run the noise avoidance processes that occur in the track assignment phase. IC Compiler fixes the remaining crosstalk violations in the post-route optimization stage.

During post-route optimization, the `route_opt` command performs the following crosstalk optimizations when you specify the `-xtalk_reduction` option:

- Optimization with crosstalk delta delay

To optimize timing with crosstalk delta delay, run `set_si_options -delta_delay true`.

- Hold time optimization with crosstalk delta delay

To perform hold fixing in addition to setup fixing with crosstalk delta delay, use the `-min_delta_delay true` option together with `-delta_delay true` in the `set_si_options` command.

- Total slew optimization

To consider crosstalk-induced slew effects during maximum transition time design rule fixing, use the `-max_transition_mode total_slew` option together with `-delta_delay true` in the `set_si_options` command.

- Static noise

To enable static noise reduction, use the `-static_noise true` option together with `-delta_delay true` in the `set_si_options` command. The default threshold (both high and low) for static noise is 0.35 volts.

You can override these threshold values with the `-static_noise_threshold_above_low` and `-static_noise_threshold_below_high` options of the `set_si_options` command. Specify a threshold value as a fraction of the supply voltage, not as an absolute voltage.

For example, if you specify a threshold value of 0.3, as shown in the following example, this means 30 percent of the voltage.

```
icc_shell> set_si_options -static_noise true \
               -static_noise_threshold_above_low 0.3 \
               -static_noise_threshold_below_high 0.3
```

Check the calculated threshold values by using the `report_si_options` command.

```
icc_shell> report_si_options
Static Noise Thresholds:
0.3 (0.49V) above low
0.3 (0.49V) below high
```

Analyzing Crosstalk

After running `route_opt`, you can perform crosstalk analysis on your design. The following sections describe how to prepare for and run crosstalk analysis.

Preparing for Crosstalk Analysis

Before you run crosstalk analysis or timing analysis to report static noise and crosstalk-induced delay, you must specify the following:

- The logic libraries, as described in [“Setting Up the Logic Libraries” on page 3-2](#), to ensure that you are using the correct timing and noise models
- The Arnoldi delay calculation algorithm, as described in [“Selecting the Delay Calculation Algorithm” on page 3-32](#)
- The types of crosstalk analysis to perform

You can select delta delay (`set_si_options -delta_delay true`), minimum delta delay (`-min_delta_delay true`), total slew (`-max_transition_mode total_slew`) and static noise (`-static_noise true`).

- Whether to use fully or partially grounded coupling capacitance during minimum delta delay analysis.

By default, the coupling capacitance is grounded during minimum delta delay analysis. You can improve the correlation for minimum delta delay by using partially grounded coupling capacitance (instead of fully grounded) by setting the `si_use_partial_grounding_for_min_analysis` variable to true (default is false).

- The electrical filtering parameters

Electrical filtering eliminates aggressor nets from analysis based on the size of the voltage bump induced on the victim net by the aggressor net. These filters are used by crosstalk analysis to reduce runtime and for consistency with PrimeTime SI.

If the default electrical filtering parameters do not meet your requirements, set the following variables:

- `si_filter_accum_aggr_noise_peak_ratio` (default is 0.03)
- `si_filter_per_aggr_noise_peak_ratio` (default is 0.01)

- The global noise thresholds

A crosstalk violation occurs when the crosstalk-induced noise voltage exceeds the specified noise threshold voltage (at or above which a false transition is likely to be triggered). Crosstalk analysis uses the noise threshold to report static noise violations. By default, both the above-low and below-high thresholds are set to 0.35 volts. You can override these thresholds by running the `set_si_options` command with the `-static_noise_threshold_above_low` and `-static_noise_threshold_below_high` options. Specify a threshold value as a fraction of the supply voltage, not as an absolute voltage.

- Whether to include timing windows

The effect of crosstalk between two nets depends largely on the overlap of timing windows between the two nets. For example, if the aggressor switches when the victim is in steady state, it will induce a noise bump on the victim. If the aggressor switches when the victim is also switching, it can cause the victim to switch faster or slower. If more than one aggressor switches in the same timing window, the effect of the two will be the accumulated effect of each on the victim net.

The edges of a timing window at a particular pin are calculated as the earliest and latest possible arrival times of the signal at that point, considering multiple paths to that point. If there is only one path to that endpoint, the timing window is very narrow.

To include timing windows, use the `-timing_window true` option together with `-delta_delay` or `-static_noise` in the `set_si_options` command. Using timing windows increases the accuracy of crosstalk analysis at the cost of more runtime.

- Set the analysis effort (low or medium effort)

By default, IC Compiler uses low effort for better runtime by using an adaptive signal integrity calculation. You can better accuracy during crosstalk analysis at the cost of higher runtime by specifying medium effort (`set_si_options -analysis_effort medium`).

Running Crosstalk Analysis

You can perform crosstalk analysis from the command line or in the GUI.

When you perform crosstalk analysis, ensure that

- The input transition time for all the cells in the design is reasonable (less than 1 ns). This is to avoid pessimism in the calculation of peak noise and crosstalk-induced delta delay.
- You use the `set_driving_cell` command to specify the driving cell instead of specifying the input transition for each input pin. The boundary conditions you specify for the input pins determine the driving resistance calculated for the nets assigned to the pins. Using the `set_driving_cell` approach enables IC Compiler to estimate the driving resistance more accurately and to correlate better with PrimeTime SI.

Analyzing Crosstalk-Induced Delay Shift. Crosstalk affects the timing of the design when any of the aggressors switch before the transition of the victim signal has finished. When the aggressor and the victim are switching in the same direction, the victim driver can expend less effort on charging the coupling capacitance and more on increasing the switching speed. This can lead to hold time problems. On the other hand, when the aggressor and victim switch in opposite directions, the transition time of the victim is longer because of a larger effective capacitance, and the result is likely to be a setup time problem. Therefore, to accurately analyze the timing behavior of a circuit, it is necessary to take into consideration the effects of crosstalk.

IC Compiler analyzes the impact on interconnect timing in terms of propagation delay and transition time. The delay shift of a victim net depends on the arrival time window of its aggressors and also affects the net timing windows. During crosstalk-induced delay, IC Compiler takes into account the timing windows of the victim and aggressor nets. For an accurate idea of the true delay shift, you must do iterations between the timing propagation in the static timing analyzer and the net delay calculation, with crosstalk considered.

To display the crosstalk delta delay, use one of the following methods:

- Use the `report_timing -crosstalk_delta` command.
- Use the delta delay visual mode by following these steps:
 1. Choose Timing > Delta Delay Visual Mode.
The Visual Mode panel appears.
 2. Click Reload.
The Delta Delay Visual Mode dialog box appears.
 3. Specify the analysis options.
 4. Click OK.

To get detailed information about the crosstalk calculations for a particular victim net, use the `report_delay_calculation -crosstalk` command.

Analyzing Static Noise. For static noise, crosstalk analysis reports noise violations (above-low and below-high) that are caused by aggressor net transitions. Above-low noise violations occur when the victim net is at the steady state of logic 0 and the aggressor net is switching from logic 0 to logic 1. Similarly, below-high noise violations occur when the victim net is at a steady state of logic 1 and the aggressor is switching from logic 1 to logic 0. When the above-low (rise) and below-high (fall) noise violations exceed the logic thresholds of the technology, they can cause logic failures.

To display static noise violations, use one of the following methods:

- Use the `report_noise` command.
- Use the `get_si_xtalk_bumps` command.
- Use the static noise visual mode by following these steps:
 - Choose Timing > Static Noise Visual Mode.
The Visual Mode panel appears.
 - Click Reload.
The Static Noise Visual Mode dialog box appears.
 - Specify the analysis options.
 - Click OK.

Display the Static Noise Analysis window by following these steps:

- Choose Timing > Static Noise Analysis.
The Static Noise Analysis dialog box appears.
- Specify the analysis options.
- Click OK.

The Static Noise Analysis window provides detailed information about the victim and aggressor nets in your design. It also provides the ability to cross-highlight the aggressor and victim nets in the layout window.

To get detailed information about the noise calculations, use the `report_noise_calculation` command.

Preventing Crosstalk During Sign-Off Optimization

During sign-off optimization, the `signoff_opt` command performs the following crosstalk optimizations when you specify the `-xtalk_reduction` option:

- Optimization with crosstalk delta delay

To optimize timing with crosstalk delta delay, run `set_si_options -delta_delay true`.

- Hold time optimization with crosstalk delta delay

To perform hold fixing with crosstalk delta delay, run `set_si_options -min_delta_delay true`.

- Static noise

To enable static noise reduction, run `set_si_options -static_noise true`. The default threshold (both high and low) for static noise is 0.35 volts.

You can override these threshold values with the `-static_noise_threshold_above_low` and `-static_noise_threshold_below_high` options of the `set_si_options` command. Specify a threshold value as a fraction of the supply voltage, not as an absolute voltage.

Sample Scripts

[Example 10-1](#) shows a sample script for running the flat signal integrity flow (or the top-level hierarchical signal integrity flow).

Example 10-1 Flat Signal Integrity Flow

```
open_mw_lib design
open_mw_cel block
place_opt
clock_opt
set_si_options -delta_delay true -static_noise true
route_opt -xtalk_reduction
signoff_opt -xtalk_reduction
save_mw_cel
```

[Example 10-2](#) shows a sample script for running the block-level hierarchical signal integrity flow.

Example 10-2 Block-Level Hierarchical Signal Integrity Flow

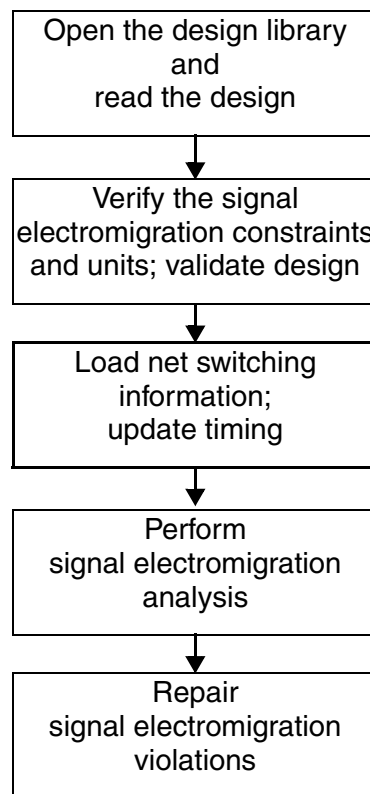
```
open_mw_lib design
open_mw_cel block
place_opt
clock_opt
set_si_options -delta_delay true -static_noise true
route_opt -xtalk_reduction
save_mw_cel
create_ilm -include_xtalk
```

Analyzing and Reducing Signal Electromigration

Signal electromigration problems result from an increase in current density caused by the use of smaller line widths and higher operational speeds in IC designs. Electromigration can lead to shorts or opens due to metal ion displacement caused by the flow of electrons. You can use IC Compiler to analyze and correct electromigration in your design.

[Figure 10-1](#) shows the signal electromigration flow is described in the following sections.

Figure 10-1 Signal Electromigration Flow



Preparing the Design for Signal Electromigration Analysis

Before you perform signal electromigration analysis, you must perform the following tasks:

- Open the design library and read the design, as described in [Chapter 3, “Preparing the Design”](#)
- Check the signal electromigration constraints in the Milkyway design library
- Verify the current unit in the Milkyway design library
- Verify that the design is ready for timing analysis
- Load the net switching information
- Update the timing

Verifying the Electromigration Constraints

To perform a check on the signal electromigration constraints in the library, use the following commands:

```
icc_shell> set_check_library_options -signal_em
1
icc_shell> check_library

#BEGIN_CHECK_LIBRARY
...
#BEGIN_CHECK_SIGNALEM

... (signal EM checking results here) ...

#END_CHECK_SIGNALEM
...
#END_CHECK_LIBRARY
```

If the signal electromigration constraints are not defined in the design library, get an updated incremental .plib file containing the electromigration data from your vendor, and then read it into IC Compiler with the following command:

```
icc_shell> set_mw_technology_file -plib plib_file_name design_lib_name
```

You can also use the Milkyway tool to update the design library, as described in the *IC Compiler Data Preparation Using Milkyway User Guide*.

Verifying the Current Unit

To verify that the current unit is defined in the design library, use the `report_units` command.

```
icc_shell> report_units
```

If the current units are not defined, add the following lines to the header section of the technology file, and then update the design library, as described in [“Changing Physical Library Information” on page 3-16](#):

```
unitCurrentName = "mA"  
currentPrecision = 10000
```

Validating the Design

To perform signal electromigration analysis, the design must have timing constraints and parasitic information. If your design does not yet have timing constraints, see [“Setting Timing Constraints” on page 3-28](#). If your design does not yet have parasitic information, either back-annotate the data, as described in [“Back-Annotating Delay or Parasitic Data” on page 3-28](#); or extract the data, as described in [“Postroute RC Extraction” on page 8-57](#).

Loading Net Switching Information

You must define net switching information to be able to perform accurate electromigration analysis. (The more frequently a net switches, the more susceptible it is to electromigration.) To load the net switching activity information, either read in a SAIF file or annotate the switching activity information on the design nets. If you do both, the annotated switching activity takes priority.

Important:

You must reload the net switching information every time the netlist changes. If you do load the switching activity, IC Compiler generates a warning message and uses $1/time\_unit$ as the net switching frequency, where *time_unit* is the main library time unit.

Reading a SAIF File. To read in a SAIF file, use the `read_saif` command.

```
icc_shell> read_saif -input saif_file
```

Example 10-3 shows a sample SAIF file. For more information about the SAIF file, see the *Power Compiler User Guide*.

Example 10-3 Sample SAIF File

```
(SAIFFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DATE "Thu Dec 12 15:44:56 2002")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "Power Compiler PLI")
(VERSION "3.4")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 22500.00)
(INSTANCE tb
(INSTANCE u_A7S
(NET
(U_TAP_DBG\U_DBG\address_d_25_
(T0 22490) (T1 0) (TX 10)
(TC 0) (IG 0)
)
```

Annotating the Switching Activity. To annotate the switching activity on design nets, use the `set_switching_activity` command with the `-static_probability`, `-toggle_rate`, and `-period` options. After annotating the switching activity, use the `propagate_switching_activity` command to propagate the static probability and toggle rate information to unannotated design objects.

A simple switching activity definition consists of the static probability and the toggle rate. The static probability is the probability that the value of the design object has logic value 1. The toggle rate is the rate at which the design object switches between logic values 0 and 1.

The following example shows how to specify that the value of the net1 net is logic 1 for 20 percent of the time, and that it transitions between logic values 0 and 1 an average of 10 times in 1,000 time units.

```
icc_shell> set_switching_activity [get_net net1] \
          -static_probability 0.2 -toggle_rate 10 -period 1000
icc_shell> propagate_switching_activity
```

The time unit used for the toggle rate is the main library time unit. The `-period` option is optional, and a default value of 1 is used when it is not specified.

Updating the Timing

To update the design timing, use the `update_timing` command:

```
icc_shell> update_timing
```

Performing Signal Electromigration Analysis

After you prepare the design library and load the net switching activity information, you can perform signal electromigration analysis. You use signal electromigration analysis at the post-routing stage to repair electromigration.

Use the `report_signal_em` command to perform signal electromigration analysis for each net by calculating the current on every edge and comparing this data with the library-defined current-density thresholds.

```
icc_shell> report_signal_em -verbose
```

The `report_signal_em` command generates

- A report that lists the signal electromigration violations
For information about the contents of the violations report, see the next section, [“Understanding the Violation Report.”](#)
- A repair file that you can use during the search-and-repair process to fix the violations
For information about fixing the violations, see [“Repairing Electromigration Violations” on page 10-21.](#)

Understanding the Violation Report

When you perform signal electromigration analysis, the tool computes three current density values:

- Average

The average current density is calculated by using the following equation:

$$I_{avg} = \frac{\alpha}{2} \int_0^{T/2} I_p(t) dt - \gamma \frac{\alpha}{2} \int_0^{T/2} I_n(t) dt$$

where alpha (α) is the number of rise or fall transitions that occur in a half-clock cycle time, T is the clock period, and gamma (γ) is the healing factor (as specified by the `-healing_factor` option).

- Root mean square

The root mean square current density is calculated by using the following equation:

$$I_{\text{rms}} = \sqrt{\frac{\alpha}{2} \left[\int_0^{T/2} I_p^2(t) dt + \int_0^{T/2} I_n^2(t) dt \right]}$$

where alpha (α) is the number of rise or fall transitions that occur in a half-clock cycle time and T is the clock period.

- Peak

The peak current density is calculated by using the following equation:

$$I_{\text{peak}} = \text{MAX}(\text{MAX}(I_p) \cdot \sqrt{D_p}, \text{MAX}(I_n) \cdot \sqrt{D_n})$$

where D is the duty cycle, defined as 50 percent of the peak width multiplied by half of the switching activity, as shown in the following equation:

$$D_p = \text{Width}(I_p) \cdot (\alpha / 2)$$

$$D_n = \text{Width}(I_n) \cdot (\alpha / 2)$$

where alpha (α) is the number of rise or fall transitions that occur in a half-clock cycle time. The subscripts p and n denote the rise and fall transitions, respectively.

The generated report displays these values for all violating nets. [Example 10-4](#) shows a sample report.

Example 10-4 Sample Signal Electromigration Report

Report : signal EM

-verbose

Design : test

Version : A-2007.12

Date : Thu Oct 11 06:45:20 2007

net: 'net1'

switching activity: 0.67/ns

Met/Via	Layer	Width	Mode	Eff_Avg	Eff_Rms	Eff_Peak	Rel_Avg	Rel_RMS	Rel_Peak
Via	51 V1	1	min	0.0000e+00	1.1194e-02	8.4890e-03	0.0000e+00	3.4341e-02	1.1459e-02
Via	51 V1	1	max	0.0000e+00	7.8558e-03	5.8093e-03	0.0000e+00	2.4100e-02	7.8416e-03
Metal	35 M5	0.14	min	0.0000e+00	7.7898e-03	5.9100e-03	0.0000e+00	1.0341e-02	2.7787e-03
Metal	35 M5	0.14	max	0.0000e+00	5.4680e-03	4.0444e-03	0.0000e+00	7.2590e-03	1.9016e-03

The report fields are

- Met/Via and Layer denote the metal or via edge and the corresponding layer number.

Note:

Because no actual geometries exist in the virtual routing stage, the Met/Via and Layer fields are not present in prerouting analysis.

- Width lists the width of the net segment.
- Mode is min or max, which designates the corner on which the calculation is based.
- Eff_Avg, Eff_Rms, and Eff_Peak show the calculated average, root mean square, and peak currents, respectively. These values are in terms of the current units defined in the Milkyway design library.
- Rel_Avg, Eff_RMS, and Rel_Peak show the ratio of the calculated value to the constraint value for the average, root mean square, and peak currents, respectively. These values are in terms of the current units defined in the Milkyway design library.

Repairing Electromigration Violations

If `report_signal_em` finds violations, it generates a repair file. The name of the repair file is specified by the required `-repair_file` option. The repair file defines variable routing rules, such as width and contact constraints, for those nets that violate the current constraints. The router follows these rules to route or reroute these nets. For example, there might be an increased wire width constraint for a specific metal layer to help fix a signal electromigration problem.

[Example 10-5](#) shows a sample repair file.

Example 10-5 Sample Repair File

```
define_routing_rule em_rule_3666 \  
  -widths { METAL1 0.96 METAL2 1.12 METAL3 1.12 METAL4 1.12 } \  
  -via_cuts { via1 3x3 via2 3x3 via3 3x3 } \  
  -taper_level 4  
set_net_routing_rule {n5514} -rule em_rule_3666
```

To load the variable routing rules, use the `source` command.

```
icc_shell> source repair_file
```

After you load the repair file, run `route_opt -incremental` to reroute violating nets.

11

Interface Logic Modeling

An interface logic model (ILM) is a structural model of a circuit that is modeled as a smaller circuit representing the interface logic of the block. ILMs are created from mapped designs only. The model contains the cells whose timing is affected by or affects the external environment of a block. Using ILMs in place of the one or more blocks of a design enhances capacity and reduces runtime for top-level optimization.

The concepts and tasks related to using ILMs in IC Compiler are presented in these sections:

- [Overview of Interface Logic Models \(ILMs\)](#)
- [Creating ILMs](#)
- [Reporting Information About ILMs](#)
- [Propagation of ILM Information to the Top-Level Design](#)
- [Instantiating and Using ILMs in the Top-Level Design](#)
- [Multicorner-Multimode Scenarios and ILMs](#)
- [Summary of Commands for ILMs](#)

Overview of Interface Logic Models (ILMs)

The following sections introduce the basic properties of interface logic models (ILMs):

- Description of an ILM
- Benefits of Using ILMs
- General Considerations When Using ILMs

Description of an ILM

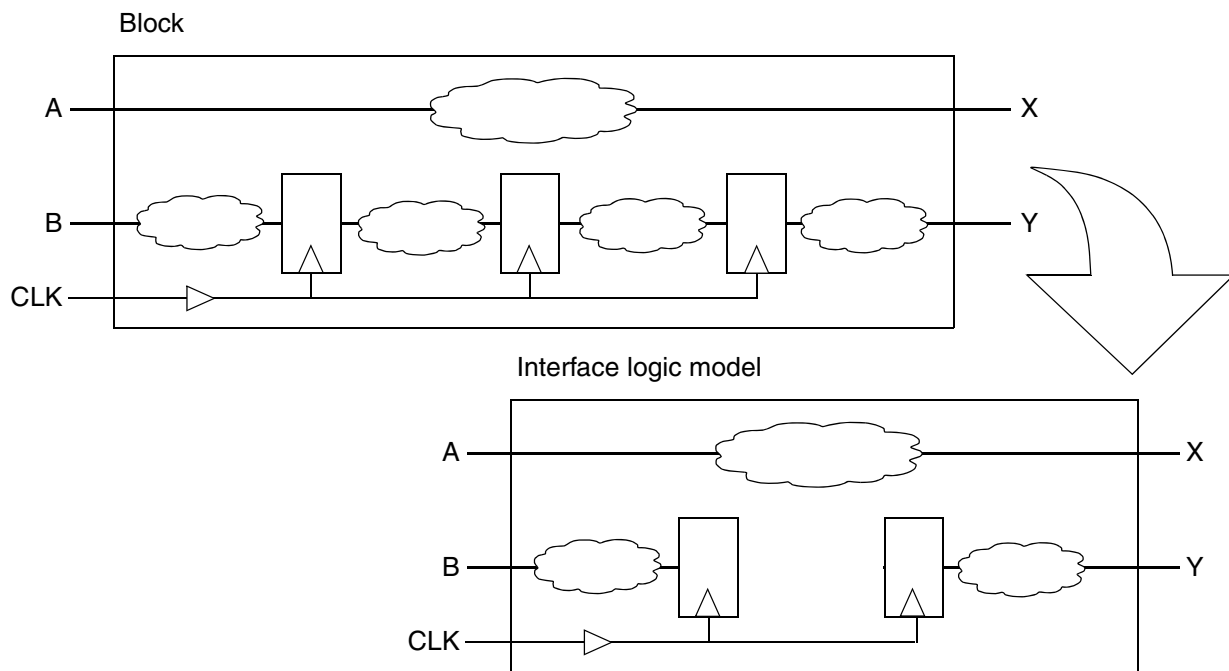
ILMs are used in IC Compiler to reduce the number of design objects and memory requirements when the tool performs top-level optimization on large designs. In an interface logic model, the *gate-level* netlist for a block is modeled by another *gate-level* netlist that contains only the needed interface logic of a block and possibly the logic you manually associate with the interface logic. All other logic is removed.

IC Compiler supports both abutted and non-abutted ILMs in IC Compiler hierarchical flows. (Channels can exist between hard and soft macros in non-abutted flows whereas they cannot exist in the abutted flow). The abutted flow supports automatic pin detection and RC extraction. Thus, abutted pins are seen by the extraction engine as being connected to one another, for example, one ILM abutted to another ILM.

You create an ILM from a block by using the `create_ilm` command. Various command options are available to control specific properties. You can create an ILM for a block that contains others ILM(s). This is known as a nested ILM.

[Figure 11-1](#) shows a simple case of a block and its interface logic model. For noncompact ILMs, the logic between the input port and all input boundary registers is preserved, as is the logic between all output boundary registers and the output port. Clock connections to the preserved registers are also kept. Logic associated with pure combinational input-port-to-output-port timing paths (A to X) is also preserved. Compact ILMs (default ILM creation) further reduce what is included in the ILM as only the interface logic involved in the critical interface timing paths (i.e. minimum rise, minimum fall, maximum rise, and maximum fall) is retained in the ILM.

Figure 11-1 A Block and Its Interface Logic Model



ILMs can be used in a hierarchical signal integrity flow. In this flow, the ILMs are shielded at the top level to prevent crosstalk between the ILM nets and top-level nets.

Benefits of Using ILMs

In addition to improved capacity and reduced optimization times for large designs, ILMs provide these benefits:

- ILMs preserve interface logic by discarding only that is not required for modeling boundary timing. Any subblock in the hierarchy that affects the boundary timing is retained. Therefore, ILMs provide highly accurate timing representations.
- Differences between the timing characteristics of an ILM and its original netlist are easy to debug because the needed interface logic is maintained.
- Required clock and data paths are preserved, and cell and net names are identical.
- The runtime for generating ILMs is short because this is a *structural* operation performed by analysis of circuit topology.

- Noncompact ILMs are context-independent because they are extracted from the original netlist and contain instantiations of the library cells that make up the interface logic.

Note:

Compact ILMs in non-multicorner-multimode flows can be context-dependent, if the top-level design has case analysis constraints or timing exceptions.

- ILM supports feedthrough nets. Feedthroughs can be on signal nets and/or clock nets.
- Back-annotated data that was applied to the original netlist is retained by the ILM. This back-annotated data is automatically propagated to the top level, thereby avoiding certain timing inaccuracies for nets fully enclosed within the ILM.
- ILMs are saved in the ILM view of a Milkyway database.
- ILM over-the-block routing resources can be established by creating a FRAM view for the ILM in the Milkyway database.

General Considerations When Using ILMs

When you use ILMs, keep the following points in mind:

- When ILMs are used, the amount of memory usage and runtime improvement depends on design style.
 - Designs with registered inputs and outputs will have the greatest reduction in size (original netlist compared to its ILM netlist).
 - Some design types, such as pure combinational logic blocks or latch-based designs do not show much reduction. The interface logic for such designs tends to contain much of the original design.
 - Compact ILMs, which are the default ILMs, significantly reduce what is retained in an ILM versus a noncompact ILM. Only the interface logic involved in the critical interface timing paths (minimum rise, minimum fall, maximum rise, and maximum fall from each input port or to each output port) is retained in a compact ILM.

If you are using a multicorner-multimode flow, the `create_ilm` command automatically detects the presence of multiple corner and multiple modes and retains the interface logic involved in the critical interface timing paths for each scenario.

- Only instances at the top level of the physical hierarchy can be modeled as ILMs, but the models can occur at any level of logic hierarchy.
- The `create_ilm` command automatically handles `dont_touch` subblocks; that is, you do not have to manually remove the `dont_touch` attribute on subblocks before creating an ILM for the block.

- You can flatten the logic of an ILM by first using the `ungroup -all -flatten` command to flatten the hierarchy of the original block. (Bear in mind that the `create_ilm` command maintains the hierarchy of the netlist it models.)
- The `dont_touch` attribute is automatically applied to all cells of the ILM, and the `is_interface_model` attribute is applied to the model.
- Optimization is not performed on the cells within an ILM.
- In non-multicorner-multimode flows, context-dependent ILMs can result when you generate a compact ILM (either `-compact all` or `-compact output`) under the following conditions:
 - You have set timing exceptions, such as `set_multicycle_path`, on the block
 - You have set case analysis constraints (`set_case_analysis` command) on the block

In these cases, you must set the same timing exceptions and case analysis constraints from the top level onto the ILM; otherwise, the ILM might not correctly represent the block-level design. For example, if different case analysis constraints are applied from the top level than those set on the block level prior to ILM creation, the ILM might not have all the minimum rise, minimum fall, maximum rise, and maximum fall timing paths that correspond to the top-level constraints that affect the ILM.

- By using the `-traverse_disabled_arcs` and `-compact none` options with the `create_ilm` command, you can generate a noncompact, context-independent ILM for a block that would otherwise have timing arcs or paths disabled by commands such as `set_case_analysis`, `set_false_path`, and `set_disable_timing`).

Note:

The `-traverse_disabled_arcs` option is not supported when creating compact ILMs. If you specify both `-compact all | output` and `-traverse_disabled_arcs`, the `create_ilm` command ignores the `-traverse_disabled_arcs` option.

- Timing exceptions specified on the core logic of a design (register-to-register paths) are lost when the design becomes an ILM, because the core logic is not part of the model.
- Apply only block-level constraints and exceptions at the block level, and apply the full design constraints and exceptions (the golden SDC file) at the top level.

Applying constraints and exceptions in this manner makes it unnecessary to use the `propagate_constraints` command.

Note:

If you apply constraints and exceptions at the block level and propagate them to the top level by using the `propagate_constraints` command, be aware that some constraints or exceptions might not be propagated to the top level.

- You can propagate block-level operating conditions to the current design. To do so, use the `propagate_constraints -operating_conditions` command.
- Generated clocks are treated like clock ports. Interface registers driven by a generated clock are retained in the model. However, internal registers driven by generated clocks are not retained. When the design contains generated clocks, the logic along the timing path between the master clock and the generated clock is also retained in the ILM.
- User-controlled handling of side-load cells can affect the timing of an interface path, even when side-load cells are not in the interface logic.
- By default, latches found in compact ILMs are treated as registers. However, in noncompact ILMs, latches found in interface logic are assumed to be potential time borrowers; thus, logic from I/O ports to flip-flops or output ports is identified as belonging to interface logic. You can specify the number of latch levels over which time borrowing can occur for latch chains that are a part of the interface logic of a noncompact ILM.
- Physical abstract information for an ILM comes from the FRAM view, which you create from the original block with the `create_macro_fram` command. If you do not create a FRAM view for the original block and select the FRAM view (with the `change_macro_view` command), IC Compiler treats the ILM cell as a full placement blockage and an all-metal layer blockage. In such a case, the tool cannot route over the ILM.
- The FRAM view of a block can enable over-the-block routing, depending on the options used with the `create_macro_fram` command.
- The `write_def` command writes the location and orientation of an ILM block to the output Design Exchange Format (DEF) file. No leaf cells beneath an ILM block are output to the DEF file because the ILM itself is treated as a leaf-level cell by the `write_def` command. This capability applies to top-level ILMs, buried ILMs (buried in the logic hierarchy but top-level cells in the physical hierarchy), and nested ILMs (ILMs that contain other ILMs).
- ILMs can support the hierarchical signal integrity flow if you use the `-include_xtalk` option with the `create_ilm` command to create an ILM that includes signal integrity data.

An ILM created in version A-2007.12 or earlier versions is not compatible with IC Compiler version B-2008.09 and vice versa. The following error message is issued if this situation occurs:

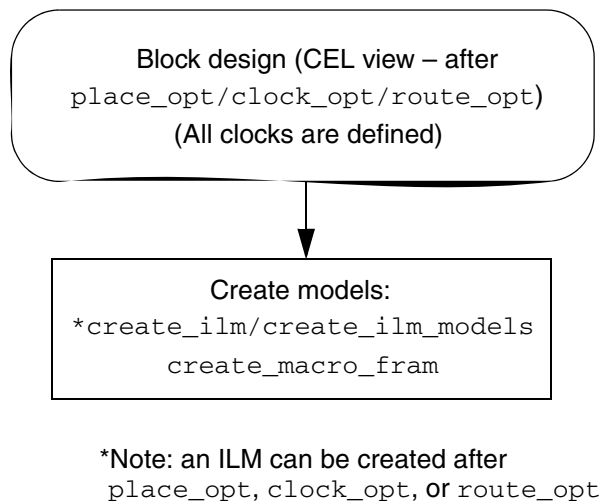
```
Error: The block ILM's data model version is older than what is supported
by this application. You must recreate the ILM using this version of the
application. (ILM-302)
```

Also, you cannot open a design saved in version B-2008.09 in version A-2007.12 or earlier versions to create an ILM for the design and then use the same ILM in version A-2007.12. You must use the ASCII interface for data transfer from version B-2008.09 to version A-2007.12 or earlier versions.

Creating ILMs

You use the `create_ilm` command to generate an ILM. Figure 11-2 shows the basic steps for creating ILMs in IC Compiler.

Figure 11-2 Basic Steps for Creating Models (FRAM and ILM) in IC Compiler



To create ILMs, follow these steps:

1. Determine which blocks you want to model as ILMs. Read in each implemented block design from the Milkyway design library (CEL view), then link each design.
2. For compact ILM creation to function appropriately, you must define all constraints (example: clocks, input delays, output delays, minimum and maximum delays to asynchronous pins, and etc.) for the design. Likewise, all timing exceptions, such as `set_multicycle_path` and `set_false_path`, that are applied to the compact ILM from the top-level must be specified at the block level prior to ILM creation, as these exceptions might determine what is retained in an ILM.

But for noncompact ILM creation you do not have to define the constraints for the design. It is recommended that you specify all clocks by using the appropriate commands, such as `create_clock` or `create_generated_clock`. If you do not specify the clocks, auto-clock detection is performed internally as a part of noncompact ILM creation.

3. Create the ILM for each block you chose. Use the `create_ilm` command with appropriate options as needed.

After command execution, the ILM is automatically saved to the ILM view. Do not use the `save_mw_cel` command to save the ILM. Note that the design in memory after `create_ilm` is the CEL view of the block, not the ILM view.

Note:

In special situations where you want to create a customized model that selectively includes (or excludes) certain cells, nets, and pins, you can separately use the `-identify_only` and `-extract_only` options with the `create_ilm` command. These options give you the flexibility to create an ILM for cases where you want to augment what would otherwise be included in the ILM produced by `create_ilm` without options. You should use the options only in such cases.

In particular, the `-identify_only` command option restricts the `create_ilm` command to identifying the circuit elements that are to be included in the ILM; the model is not created. You can use several other options with the `-identify_only` option to control what is included among the identified interface elements. See [“Controlling the Logic Included in an ILM” on page 11-10](#) and [“Summary of the create_ilm Command Options” on page 11-17](#).

Use the `get_ilm_objects` command to determine the objects that have been identified as interface logic objects. You can repeat this step, changing option settings as needed, until you obtain the desired collection of interface objects.

The `-extract_only` command option restricts the `create_ilm` command to extracting the currently identified interface logic objects and builds the ILM; it does not newly identify interface logic elements. Use this command after you are satisfied with the collection of interface logic objects.

You can create multiple ILMs from soft macros in the design by using the `create_ilm_models` command. The action of the command is the same as for `create_ilm`, except that you specify a list of macros in the command, which creates one ILM for each listed macro. For example, to create ILMs for soft macros `blk1` and `blk2`, use

```
create_ilm_models {blk1 blk2}
```

The `create_ilm_models` command has many of the same options as the `create_ilm` command to control the manner of model creation. These option settings apply to all the ILMs created by the command.

Before you can use `create_ilm_models`, CEL views must exist for the blocks that will be turned into interface logic models by the command.

Guidelines for Creating an ILM

When you create an ILM, observe these guidelines:

- By default, an ILM contains an abstracted clock tree and the logic in the critical timing paths (minimum rise, minimum fall, maximum rise, and maximum fall) from all input ports and to all output ports (`-compact all` option).

You can use command options to control the included clock tree and data paths. To keep the entire clock tree, use the `-keep_full_clock_tree` option. To keep the logic on the critical timing paths to the output ports, but not from the input ports, use the `-compact output` option. To disable compaction, use the `-compact none` option.

When you use the `-compact none` or the `-compact output` options, the `create_ilm` command automatically ignores the fanout from an input or inout port if the percentage of the total registers in the design in the transitive fanout of that port is greater than or equal to the value set by the `ilm_ignore_percentage` variable (the default is 25). The connectivity from automatically ignored input and inout ports to already identified interface logic, such as timing paths, is retained, as are the minimum and maximum paths (and associated connectivity) from the ignored input and inout ports. To disable the auto-ignore capability, use the `[-no_auto_ignore]` option.

- By default, only the logic in the timing critical paths (minimum and maximum) to output ports is included in the ILM (`-compact output` option).

To disable compaction, use the `[-compact none]` option. To include logic in the timing critical paths (minimum and maximum) from input ports and to output ports, use the `-compact all` option.

- To include all side-load cells in the ILM, use the `-include_side_load all` option. Using this option increases the model size but can improve model timing accuracy.
- You can use the `report_annotated_delay` command to check whether the necessary nets and cells are annotated for a postroute SDF flow. Note, however, that using parasitics with ILMs is the preferred way to annotate a design with postroute data.
- Save the parasitic information from the routed ILMs of a postroute flow in the ILM view. See [“Storing Parasitic Information in the ILM View” on page 11-25](#).
- How to handle “what if” constraints:

If you choose to apply logic constraints such as `set_case_analysis`, `set_disable_timing`, or `set_false_path` on a block, the disabled logic is pruned (removed) from the ILM because that logic is usually not needed. However, there may be applications (such as case analysis) for which you want to retain the disabled logic in the

ILM. To prevent pruning of disabled logic, use the `-traverse_disabled_arcs` option with the `create_ilm -compact none` command. The `-traverse_disabled_arcs` option works only for noncompact ILMs.

Note that if the `-traverse_disabled_arcs` option is used, you must apply case analysis constraints from the top-level design onto the block because constraints are not automatically propagated from the block level to the top level. Not doing so could lead to including additional timing paths that would need to be disabled for timing analysis when the ILMs are employed at the top level.

Alternatively, remove all case analysis constraints (`remove_case_analysis _all`) or do not apply such constraints to a block if you are not going to use the `-traverse_disabled_arcs` option with the `create_ilm -compact-none` command when you create the ILM for the block.

Note that either method creates an ILM that is independent of case analysis. Case analysis can then be applied from the top level to the ILM cell as desired for the particular application.

Automatic Checking as Part of Creating ILMs

When you run the `create_ilm` command, the following automatic checks are carried out:

- If the current design does not contain any sequential elements, a warning message is issued and the ILM is created. However, this ILM is the same size as the original design.
- All cells and ports of the current design are checked to ensure that they have defined locations. If not, the command terminates with an error message. No ILM is created.

If you want to disable these checks, set the variable `ilm_disable_ilm_checks` to true. The default for this variable is false.

Controlling the Logic Included in an ILM

Interface logic contains the circuitry from the following:

- Leaf cells and macro cells in timing paths that lead from input ports to output ports (combinational input-to-output paths)
- Leaf cells and macro cells in timing paths that lead from input ports to edge-triggered registers
- Leaf cells and macro cells in timing paths that lead from edge-triggered registers to output ports

- Macros that are not part of the interface timing paths but can be included in the model by using the `-keep_macros` option of the `create_ilm` command
- Abstracted clock trees that include the minimum and maximum clock paths, as well as the clock tree synthesis exceptions that are a part of these paths
- Clock-gating circuitry, if it is driven by external ports
- During the `create_ilm` command, the UPF objects that are required at the top-level are retained in the ILM. When the top-level netlist is linked, the UPF objects retained in the ILM are propagated to the top-level before loading of top-level power domain takes place.
- Any transparent latch encountered in a timing path from an input port to a register or from an output register to an output port (noncompact ILM only)

These latches are treated as combinational logic devices in noncompact ILMs and included in the interface logic. You can control this behavior by using the `-latch_level` option of the `create_ilm -compact none` command. (See [Figure 11-3 on page 11-12](#) and [Figure 11-4 on page 11-13](#).)

When you generate a compact ILM, either by not specifying the `-compact` option or by using the `-compact all` option, the `create_ilm` command ignores the `-latch_level` option.

If your design has time borrowing over the latches (`-latch_level` has a value greater than or equal to one), you should specify the `-latch_level` option and disable ILM compaction by using the `-compact none` option.

If your design does not have any time borrowing over the latches, you should use the default compaction (`-compact all`) because the first latch encountered in a timing path is treated as an edge-triggered cell and is considered an endpoint for a timing path from an input port or a startpoint for a timing path to an output port.

- If you use the `-traverse_disabled_arcs` option with the `create_ilm -compact none` command, the disabled timing arcs and paths of a block are ignored. Any interface logic belonging to these disabled arcs or paths is included in the model. Conversely, if you do not use this option, this logic is not included in the model.

Controlling the Fanin and Fanout of Chip-Level Networks for Noncompact ILMs

The fanin and fanout of a chip-level network, such as a scan-enable, set, or reset network, are not part of the ILM if the percentage of the total registers in the design in the transitive fanout of the input or inout port is greater than or equal to the value set by the `ilm_ignore_percentage` variable (the default is 25). If you want to exclude specific ports or both input and output ports, use the `-no_auto_ignore` option and explicitly specify the ignored ports by using the `-ignore_ports` option. The connectivity from all ignored ports

(whether auto-ignored or explicitly ignored) to already identified interface logic is retained. Minimum and maximum paths (and connectivity) from these ports are also included in the model.

Controlling the Number of Latch Levels for Noncompact ILMs

By default, all transparent latches are identified as part of the interface logic and are assumed to be potential time borrowers. Path tracing continues from the input ports through these latches until an edge-triggered register is encountered. Similarly, path tracing continues from the output registers through the latches to the output ports. You can control the number of latch levels included in the model by using the `-latch_level` option with the `create_ilm -compact none` command.

Figure 11-3 shows a default noncompact ILM model and latch levels. Figure 11-4 shows the results of specifying latch level 1 and latch level 0.

Figure 11-3 Interface Logic Model Noncompact Model and Latch Levels

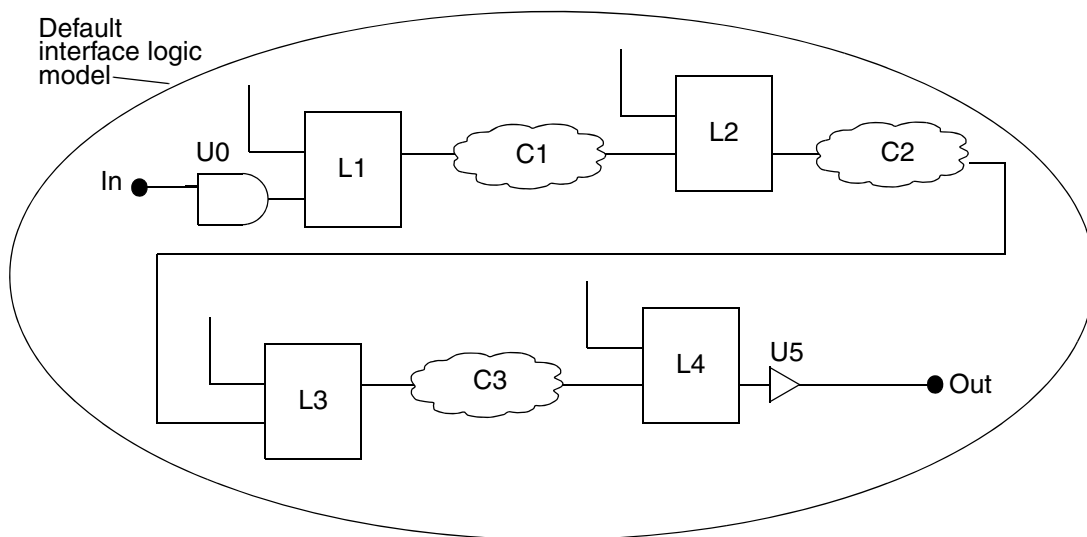
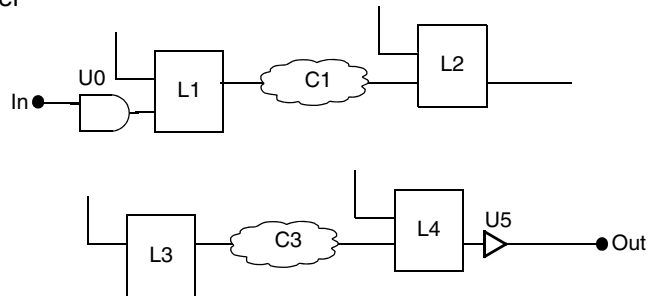
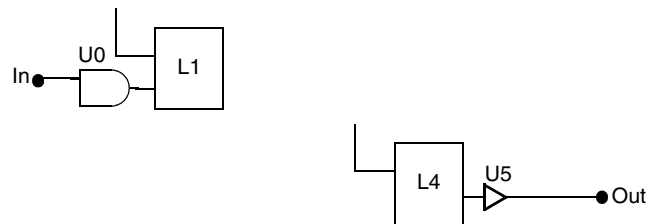


Figure 11-4 Results of Specifying Latch Level 1 and Latch Level 0

Interface logic model
latch levels = 1



Interface logic model
latch levels = 0



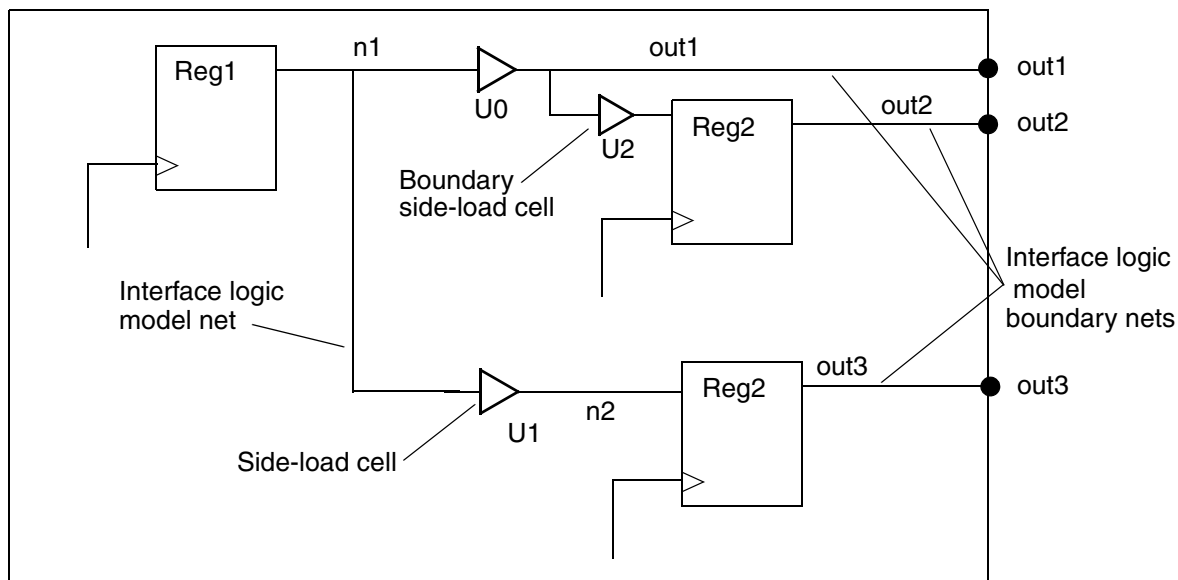
Controlling Side-Load Cells

Side-load cells are cells that can affect the timing of an interface path but are not directly part of the interface logic. By default, an ILM includes the side-load cells for nets that are connected to the ports of the ILM (the boundary side-load cells), which usually provides the best tradeoff between model size and accuracy.

You can control which side-load cells are included in the ILM by using the `-include_side_load side_load_type` option when you run the `create_ilm` command. To include only boundary side-load cells (the default), use the `boundary` side-load type. To extract the most accurate ILM, choose side-load type `all`. To create the smallest (least accurate) ILM, choose `none`.

Figure 11-5 shows an example of a side-load cell and a boundary side-load cell.

Figure 11-5 Side-Load Cell and Boundary Side-Load Cell



In [Figure 11-5](#), cell U1 is a side-load cell because it places a load capacitance on the interface logic net n1. Cell U2 is a boundary side-load cell because it places a load capacitance on the boundary interface logic net out1. Net out1 is considered a boundary net because it is directly connected to a port (out1).

Rotated and Mirrored ILMs

Even when multiple instances with different orientations of an ILM are instantiated at the top level, you need to create only one ILM. All eight orientations—north, flipped (mirrored) north, south, flipped (mirrored) south, east, flipped (mirrored) east, west, and flipped (mirrored) west—are allowed.

For detailed information, see [“Handling Rotated and Mirrored ILMs in the Top-Level Design” on page 11-27](#).

Note:

The orientations of the top-level instances must be defined in a floorplanning tool and output as a DEF file or as attributes in a Milkyway design in the ILM view.

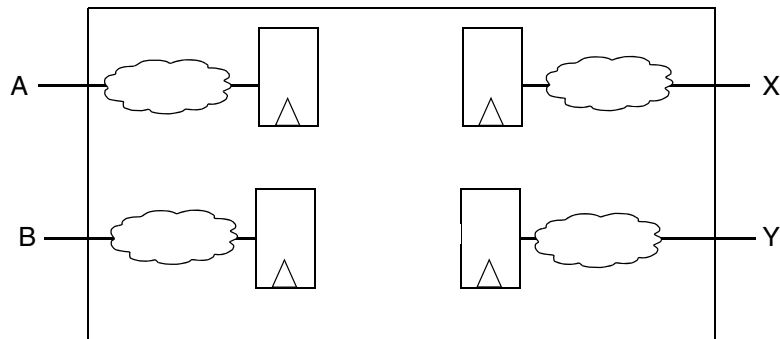
Nested ILM

A nested ILM is an ILM that contains one (or more) ILMs. Like a normal ILM, the interface timing paths of a nested ILM are visible to the top-level design. The `create_ilm` command includes only the interface logic of the original block in the ILM that gets created (only timing critical interface logic for the default compact ILM). If this ILM is instantiated in a block that will also become an ILM (nested ILM), the ILM creation process will only retain logic from the lower-level ILM that is involved in the interface paths (timing critical for compact ILM) of the upper-level ILM.

Creating Nested ILM

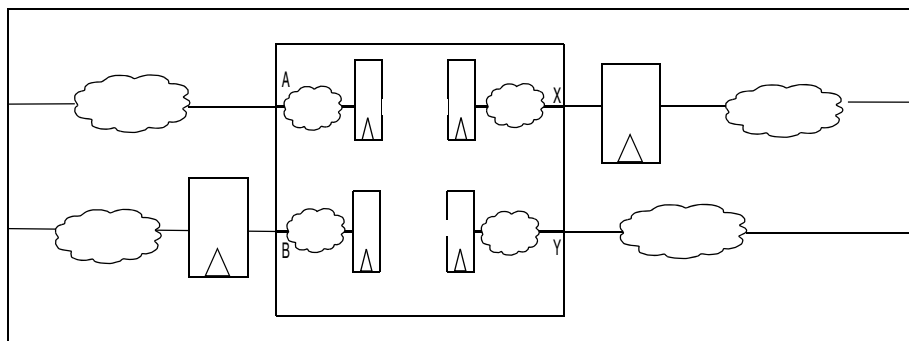
The following is an example of a nested ILM that uses the two macros BOT and MID. Macro BOT is nested in macro MID. For information on how to create an ILM, see [“Creating ILMs” on page 11-7](#). [Figure 11-6](#) displays the ILM for macro BOT.

Figure 11-6 Interface Logic Model for Macro BOT



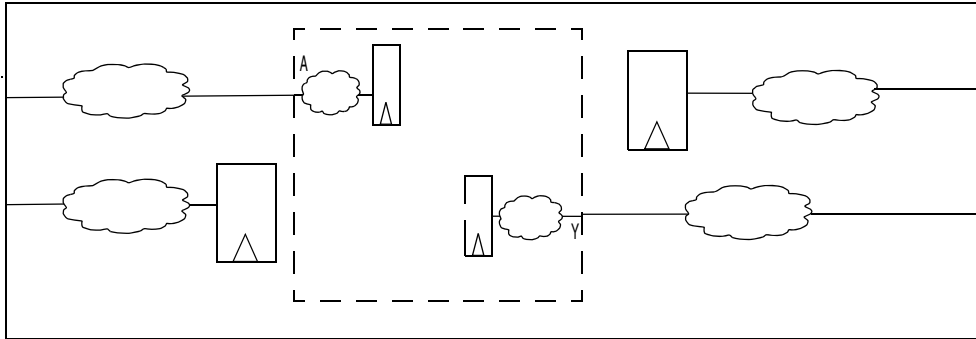
[Figure 11-7](#) shows the CEL view at the next level. It contains the macro BOT ILM.

Figure 11-7 MID Macro containing macro BOT ILM



When you run `create_ilm`, the ILM for macro MID is created. Figure 11-8 shows the MID ILM. The dotted line indicates the logical hierarchy. The logic connected to B and X pins of BOT ILM has been discarded.

Figure 11-8 Interface Logic Model for the MID Macro



Design Information Used With an ILM

Blockage, pin, and via information (BPV) is stored in the FRAM view of a block. The FRAM view is created by using the `create_macro_fram` command and is required if you want to route over the block (both virtual and real routing). Note that this view is always required for top-level global routing, track assign, and detail routing, regardless of whether you route over the ILM block.

Information such as the locations of ILM cells and ports, the net capacitance values, and net delays for nets internal to the model are stored in the ILM. For nets crossing the model boundary, IC Compiler reestimates the wire length and recomputes the boundary net delays when you run the `place_opt` command.

The IC Compiler hierarchical flow, which uses ILMs, supports UPF usage.

During the `create_ilm` command, the UPF objects that are required at the top-level are retained in the ILM. When the top-level netlist is linked, the UPF objects retained in the ILM are propagated to the top-level before loading of top-level power domain takes place.

UPF objects such as level shifter cells, isolation cells, power switch cells, netlist power pins, ports, nets and other strategy properties are maintained during the creation of an ILM.

Summary of the create_ilm Command Options

To use the `create_ilm` command with appropriate options, enter

```
icc_shell> create_ilm [options]
```

Table 11-1 summarizes the command options.

Table 11-1 *create_ilm Options*

To do this	Use this
Identify interface logic elements without building the ILM. Note: The next eight options help control which elements are identified as interface logic elements and can be included in the ILM. You can use these options with or without the <code>-identify_only</code> option.	<code>-identify_only</code>
Exclude the fanout and fanin logic from input and output of ports connected to chip-level nets (for example, scan enable and reset) to get the smallest ILM. See “Controlling the Fanin and Fanout of Chip-Level Networks for Noncompact ILMs” on page 11-11 .	<code>-ignore_ports</code>
Do not automatically exclude the fanout logic from input ports connected to a certain percentage or greater of the total number of registers in the design. See “Controlling the Fanin and Fanout of Chip-Level Networks for Noncompact ILMs” on page 11-11 .	<code>-no_auto_ignore</code>
Specify the number of latch levels for which time borrowing can occur for latch chains that are part of the interface logic. See “Controlling the Number of Latch Levels for Noncompact ILMs” on page 11-12 .	<code>-latch_level</code>
Include all macros. See “Controlling the Logic Included in an ILM” on page 11-10 .	<code>-keep_macros</code>
Include interface logic from disabled timing arcs and pins. See “Controlling the Logic Included in an ILM” on page 11-10 .	<code>-traverse_disabled_arcs</code>
Include the boundary cells of ignored ports.	<code>-keep_boundary_cells</code>
Include side-load cells of a specified type.	<code>-include_side_load</code>
Include parasitics in the ILM view. This option is automatically turned on if the block has routed nets. Retaining the parasitics in the ILM is required if you intend to perform top-level on-route optimization (<code>route_opt</code> or <code>psynopt -on_route</code>) using ILMs.	<code>-keep_parasitics</code>

Table 11-1 *create_ilm Options (Continued)*

To do this	Use this
Generate a signal-integrity-aware ILM; include crosstalk information, such as boundary net aggressor data, effective net resistance, total capacitance, and coupling capacitance.	-include_xtalk
Only extract and create an ILM from currently identified interface logic elements.	-extract_only
Print statistics about the number of design objects in the original design and the model netlist.	-verbose
Specify the compression strategy that should be used to reduce the size of ILM.	-compact output all none

Summary of the create_ilm_models Command Options

To use the `create_ilm_models` command with appropriate options, enter

```
icc_shell>create_ilm_models [options]
```

Table 11-2 summarizes these command options.

Table 11-2 *create_ilm_models Options*

To do this	Use this
Disable the automatic determination of input and inout ports whose fanout should be ignored when identifying interface logic (Noncompact ILM only).	-no_auto_ignore
Specify the number of logic levels over which time borrowing can occur for latch chains that are part of the interface logic (Noncompact ILM only).	-latch_level
Include detailed parasitics in the ILM view. This option is automatically turned on if the block has routed nets.	-keep_parasitics
Print comparative statistics about the number of design objects in the original design and in the ILM. The default is not to print comparative statistics.	-verbose
Store signal integrity (SI) information with the ILM view.	-include_xtalk

Table 11-2 create_ilm_models Options (Continued)

To do this	Use this
Specifies the technique to use to reduce the size of an ILM.	<code>-compact output all none</code>
Generate compact ILM using the transition time at the inputs of the blocks in the top design.	<code>-in_context</code>
Specify the list of soft macros for which ILM models need to be created. This list of one or more blocks is required input.	<code>macro_reference_list</code>

Reporting Information About ILMs

The commands and options listed in [Table 11-3](#) provide information about ILMs.

Table 11-3 Commands and Options That Report Information About Interface Logic Models

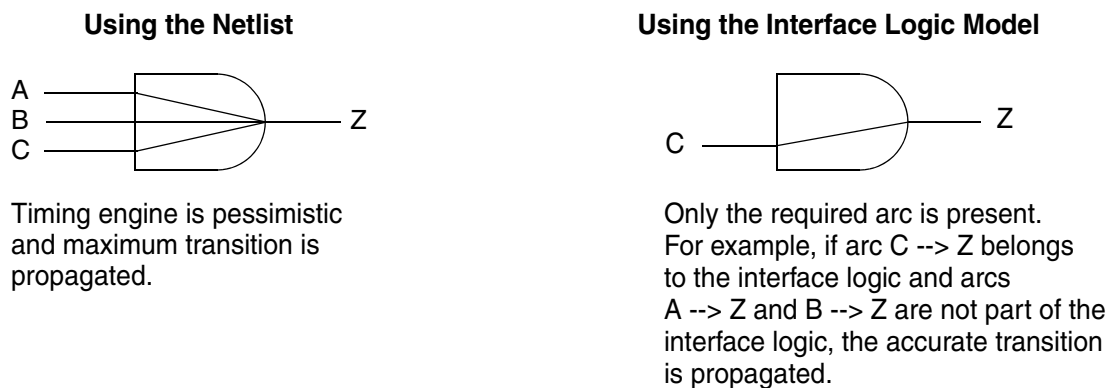
To report this	Use this
Information that the design is an ILM. See “Reporting Design Information” on page 11-20 .	<code>report_design</code>
Statistics for both the original netlist and the ILM netlist to let you know how much reduction occurred for each of the design objects and the total area reduction for ILM designs. See “Reporting Area Information” on page 11-20 .	<code>report_area</code>
The objects in the current design that are identified as belonging to interface logic.	<code>get_ilm_objects</code>
The ILM blocks that are defined as part of the current design. You can prevent this command from issuing messages by using the <code>-quiet</code> option.	<code>get_ilms</code>
Statistics for the original design and the ILM.	<code>create_ilm -verbose</code>

For the complete syntax, see the man pages.

Discrepancies between the original design and its interface logic model can arise because of the following:

- Whether or not you include all boundary side loads in the interface logic model. This choice affects the total capacitance on a node and the resultant path timing.
- Propagation of the actual cell interface timing arc instead of the worst-case, maximum transition, as shown in [Figure 11-9](#).

Figure 11-9 Comparison of Timing Propagation Between Netlist and Model



Reporting Design Information

The `report_design` command reports information that the design is an ILM if you are at the top of the interface logic model. Note the differences in the report results depending on the current design:

- If you have an ILM called ILM_TOP and ILM_TOP is the current design, the report provides the information that it is an ILM.
- If you have an ILM called ILM_TOP that has instances A, B, and C in it and you push into instance A and make it the current design, running `report_design` on design A reports that design A is a subdesign inside the ILM.

Reporting Area Information

The `report_area` command provides statistics for both the original netlist and the ILM netlist to let you know how much reduction occurred for each of the design objects and the total area reduction for ILM designs. However, if instance A is the current design and you run `report_area` on it, you see only the statistics for the ILM of design A. Compare the reports provided in [Example 11-1](#) and [Example 11-2](#).

Example 11-1 Pre-ILM-Creation Area Report When Instance A Is the Current Design

```
*****
Report : area
Design : A
Version: B-2008.09
Date   : Wed Oct  5 06:48:27 2005
*****
```

Library(s) Used:

slow (File: /remote/testcases7/65365/data/LIBS/slow.db)

```
Number of ports:      274
Number of nets:       1111
Number of cells:      134
Number of references:  17
```

```
Combinational area:   277140.500000
Noncombinational area: 152038.531250
Net Interconnect area: 162.984695
```

```
Total cell area:     429177.406250
Total area:          429340.406250
1
```

Example 11-2 Post-ILM-Creation Area Report When Instance A Is the Current Design

```
*****
Report : area
Design : A
Version: B-2008.09
Date   : Wed Oct  5 06:52:42 2005
*****
```

Library(s) Used:

slow (File: /remote/testcases7/65365/data/LIBS/slow.db)
Design is an Interface Logic Model

Design Information:

```
Number of ports:      274
Number of nets:       1111
Number of cells:      134
Number of references:  17
```

```
Combinational area:   277108.281250
Noncombinational area: 152038.531250
Net Interconnect area: 162.982513
```

```
Total cell area:     429146.812500
Total area:          429309.781250
```

Model Information:

```
Number of ports:      274
Number of nets:       557
Number of cells:      97
Number of references:  16
```

*Example 11-2 Post-ILM-Creation Area Report When Instance A Is the Current Design
(Continued)*

```
Combinational area:      37817.261719
Noncombinational area:   97321.593750
Net Interconnect area:    45.728115

Total cell area:         135140.203125
Total area:              135185.937500
1
```

Propagation of ILM Information to the Top-Level Design

IC Compiler automatically links in ILMs that are referenced in the top-level design. Specifically, whenever there is an unresolved reference in the design, the appropriate ILM is linked in from design/reference library.

IC Compiler first searches the explicitly specified `link_library` files, and then the Milkyway design and reference libraries for ILM views matching the unresolved reference name. If a `.db` file corresponding to a macro (an extracted timing model) is specified in the `link_library`, the `link` command chooses that model and does not search for the ILM in the Milkyway library, because higher precedence is given to explicitly specified files.

Also incorporated into the link process is automatic propagation of ILM data (placement, keepout, and delay information). Remember that if you want to route over the block, you must create and use FRAM views for the blocks that are modeled as ILMs.

The `link` command propagates UPF constraints stored in the ILM to the top-level before loading of the top-level power domain takes place.

The top-level constraints must be available for the `read_sdc` command. You run the `place_opt` command to place all the cells of the top level. After placing the top-level cells, you can run `clock_opt` and `route_opt`. Note that `route_opt` requires FRAM views to be used for all macros (that is, for all routed blocks modeled by ILMs). The following sections describe these steps in detail.

Creating and Propagating ILM Keepouts in IC Compiler

Before you instantiate the ILMs in the top-level design and run the `place_opt` command, IC Compiler needs information about the placement blockages and route guides (obstructions that are also known as placement keepouts and wiring keepouts, or collectively as keepouts) associated with an ILM. The top-level optimization commands use the keepout information to place the top-level cells, correctly estimate wire lengths for the top-level nets (before routing), and route the top level, using any over-the-block resources that are not blocked by the keepouts.

You run the `create_placement_blockage` and `create_route_guide` commands to create specific placement blockages and wiring keepouts for the block that you intend to model with an ILM. This keepout information is made available at the top level, *provided* you have also created a FRAM view for the block and changed the block's view from the CEL view to the FRAM view.

You use the `create_macro_fram` command to create the FRAM view for the block. For example,

```
icc_shell> create_macro_fram -library_name design_lib -cell_name block1
```

creates a FRAM view for block1. Then to change to that FRAM view, use the `change_macro_view` command. In this example,

```
icc_shell> change_macro_view -reference block1 -view FRAM
```

Manually defining the wiring and placement keepouts and using the block FRAM view causes over-the-block routing resources to be known and used during top-level operations. No top-level cells may be placed in the area of an ILM.

Note:

If you do not create a FRAM view for the block or if you fail to change to the block FRAM view, the keepout information is not available to the top level. In such a case, IC Compiler automatically creates an all-metal-layer routing blockage for the entire ILM, so that none of the ILM area is available for over-the-block routing. Likewise, none of the ILM area is available for over-the-block placement.

Manually Creating ILM Keepouts

You create placement and wiring keepout regions for the current design (the ILM block) with the following commands:

- The `create_placement_blockage` command for placement blockages (keepouts). For details, see the man page.
- The `create_route_guide` command for routing guides (keepouts). You can define route guides for specific regions of a routing layer or for entire routing layers. Use the `get_die_area` command to get the coordinates when you want to define a route guide for an entire routing layer. For details, see the man page.

The basic blockage shape is a rectangle; that is, the `create_placement_blockage` and `create_route_guide` commands define only rectangular regions. To create a rectilinear region, you must about two or more placement or routing blockages to create a "composite" blockage that defines the appropriate rectilinear region.

Note:

This method of creating rectilinear blockages and the corresponding rectilinear ILM shapes is not needed for rectilinear blocks that were created in JupiterXT. In such a case, IC Compiler recognizes and preserves the rectilinear shape when it creates the ILM for the block.

Automatically Derived ILM Keepouts

As noted previously, if you do not specify any placement blockages or route guides for an ILM, or if you do not create a FRAM view for a routed block (prior to ILM creation), or if you have not changed to the FRAM view for the block, IC Compiler automatically derives a default obstruction for the entire ILM block.

If you use automatically derived, default keepouts, you still need to define x- and y-coordinates for the ILM block in the top-level DEF file, because IC Compiler does not do block placement. The tool treats cells inside an ILM as fixed cells. It adjusts the locations of the cells within the ILM with respect to the coordinates and orientation that describe how the ILM is placed in the top-level floorplan.

You cannot view automatically derived ILM placement blockages and route guides in the IC Compiler GUI at the ILM level or at the top level where the ILMs are instantiated. Only user-specified placement blockages and route guides are visible in the IC Compiler GUI.

Creating Keepout Margins

You can use the `set_keepout_margin` command to create keepout margins around ILM instances or an ILM reference. For example, to put a 10-unit margin around the ILM instance `u1_ilm`, you would enter

```
icc_shell> set_keepout_margin -type hard -outer {10 10 10 10} u1_ilm
```

However, for an ILM reference, you also need to use the `get_ilms` command. For example, if `ilm_block` is the reference for the instance `u1_ilm`, you enter

```
icc_shell> set_keepout_margin -type hard -outer {10 10 10 10} \  
[get_ilms -reference ilm_block]
```

When you use this command, the keepout margin is placed around every instance of the `ilm_block` reference.

Storing Parasitic Information in the ILM View

When you create an ILM model for a routed block, IC Compiler automatically enables the `-keep_parasitics` option. This causes the `extract_rc` command to be run in the background and the ILM is back-annotated with the postroute detailed parasitic data of the original placed and routed block. The detailed parasitic data stored in the ILM is automatically used when you run top-level routing, the `extract_rc` command, or the `route_opt` command. For example,

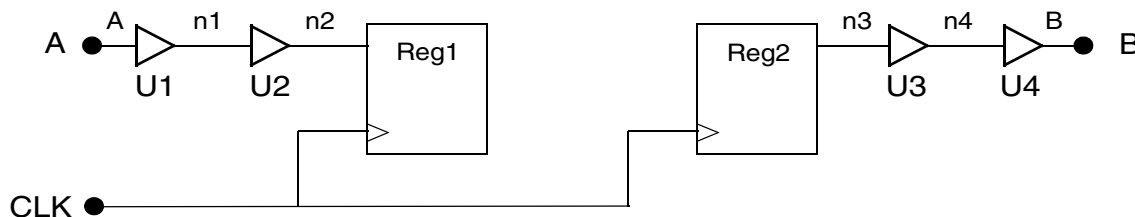
```
icc_shell> open_mw_cel routed_block1
icc_shell> create_ilm

icc_shell> open_mw_cel top
icc_shell> read_sdc top.sdc
icc_shell> route_global
icc_shell> extract_rc
icc_shell> report_timing
```

Examples of ILM Delay Estimation and Annotation

Preroute estimation: Figure 11-10 shows the ILM delay propagation for a preroute design. Note that cell delays are not propagated because these delays are recomputed by the tool during optimization of the top-level design.

Figure 11-10 ILM Delay Propagation: Preroute

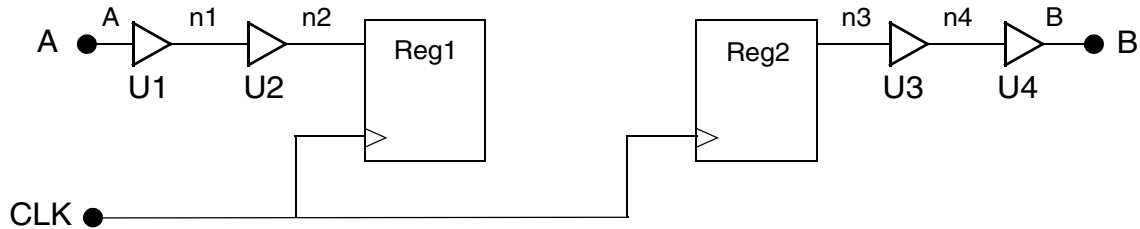


The delay information for ILM nets n1, n2, n3, and n4 is automatically propagated up to the top level.

Note: ILM cell delays are computed at the top level.

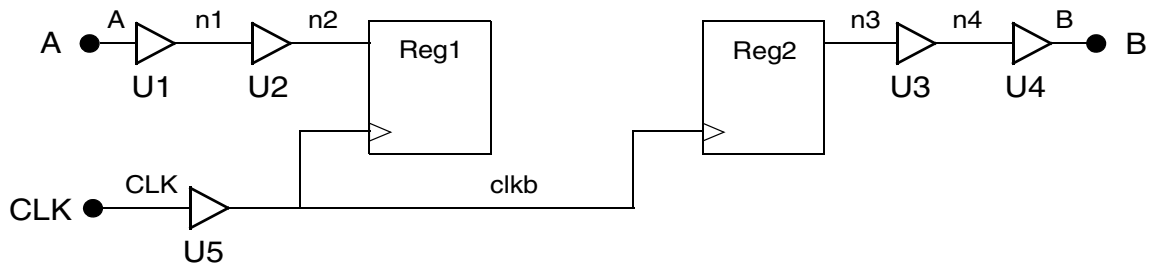
Postroute annotation: Figure 11-11 and Figure 11-12 show the ILM delay propagation for two designs in a postroute back-annotated flow. For the design shown in Figure 11-11, the delays for nets n1, n2, n3, and n4 and cells U2 and U3 are propagated. For the design shown in Figure 11-12, which includes a buffered clock signal, the delays for the clkb net and the Reg1 and Reg2 cells are included, as well as the delays for nets n1, n2, n3, and n4 and cells U2 and U3. Note that the delays for the boundary cells U1, U4, and U5 are not included in the default delay propagation.

Figure 11-11 ILM Delay Propagation: Back-Annotated Postroute



The delay information for ILM nets n1, n2, n3, and n4 and cells U2 and U3 is automatically propagated up to the top level.

Figure 11-12 ILM Delay Propagation: Back-Annotated Postroute



The delay information for ILM nets n1, n2, n3, n4, and clkb and cells U2, Reg1, Reg2, and U3 is automatically propagated up to the top level.

Handling Rotated and Mirrored ILMs in the Top-Level Design

IC Compiler supports rotated and mirrored ILMs while performing top-level optimization. The orientation information, as well as the leaf cell and port locations of the ILMs, is automatically propagated to the top level.

The eight allowed orientations are

- North (N)
- Flipped or mirrored north (FN)
- South (S)
- Flipped or mirrored south (FS)
- East (E)
- Flipped or mirrored east (FE)
- West (W)
- Flipped or mirrored west (FW)

These orientations are shown with their respective port locations in [Figure 11-13](#) through [Figure 11-20](#).

Figure 11-13 North (N) Orientation of an ILM

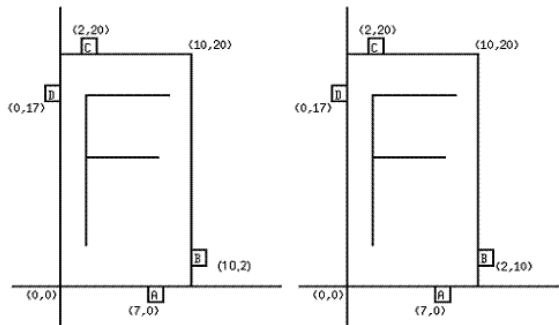


Figure 11-14 Flipped or Mirrored North (FN) Orientation of an ILM

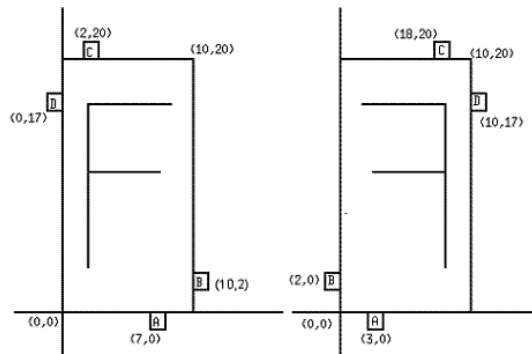


Figure 11-15 South (S) Orientation of an ILM

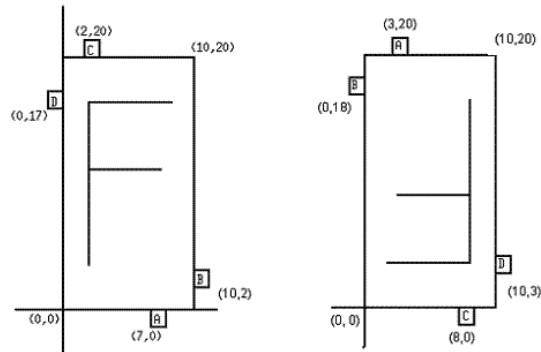


Figure 11-16 Flipped or Mirrored South (FS) Orientation of an ILM

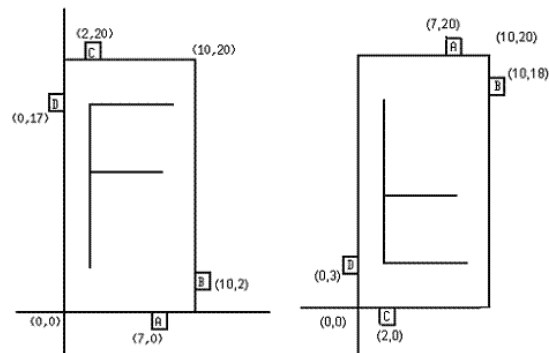


Figure 11-17 East (E) Orientation of an ILM

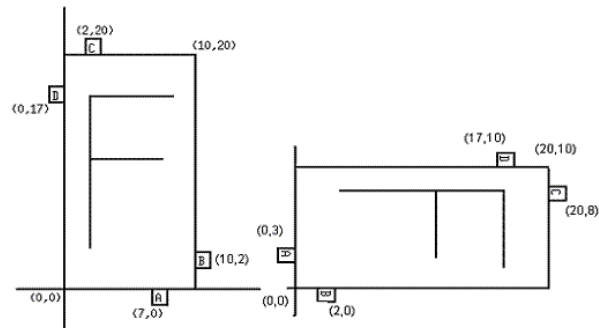


Figure 11-18 Flipped or Mirrored East (FE) Orientation of an ILM

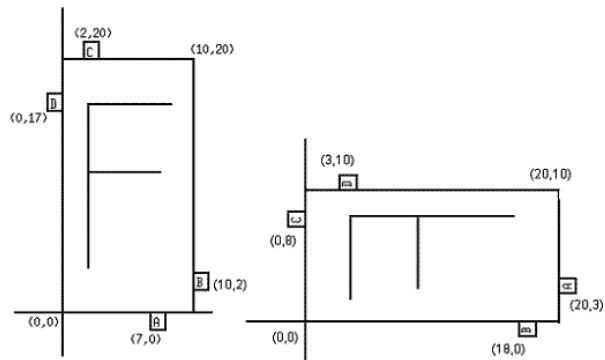


Figure 11-19 West (W) Orientation of an ILM

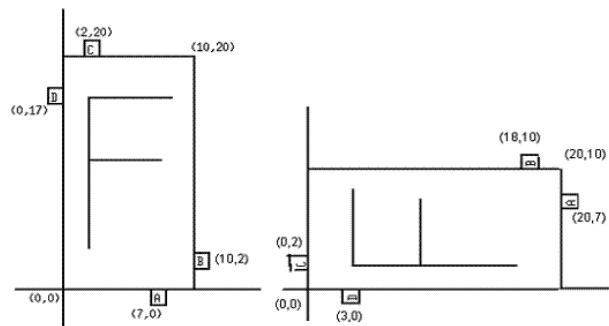
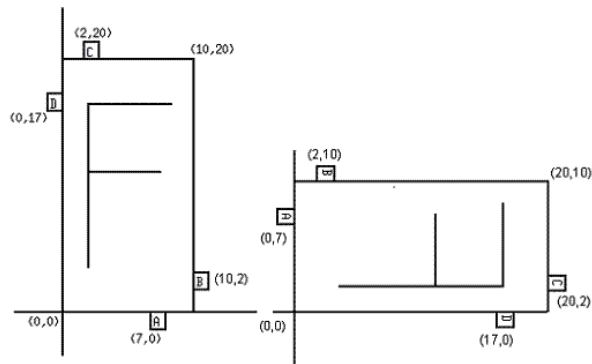


Figure 11-20 Flipped or Mirrored West (FW) Orientation of an ILM



The lower-left corner of the ILM is the actual location of the block. For example, if the location specified for the ILM block is (0,0), then the lower-left corner of the rotated block is (0,0). Notice how the port locations change with the various orientations.

Note:

To allow the E, FE, W, and FW orientations, you must set the `ilm_allow_all_rotations` variable to true (the default is false). The N, FN, S, and FS orientations are always allowed.

You do not need to extract a separate ILM for each orientation of a block because IC Compiler propagates all orientations of the ILM, along with the leaf cell locations and the port locations, to the top-level design. This capability of having a single ILM support eight orientations of the original block in the top-level design significantly eases the ILM usage flow.

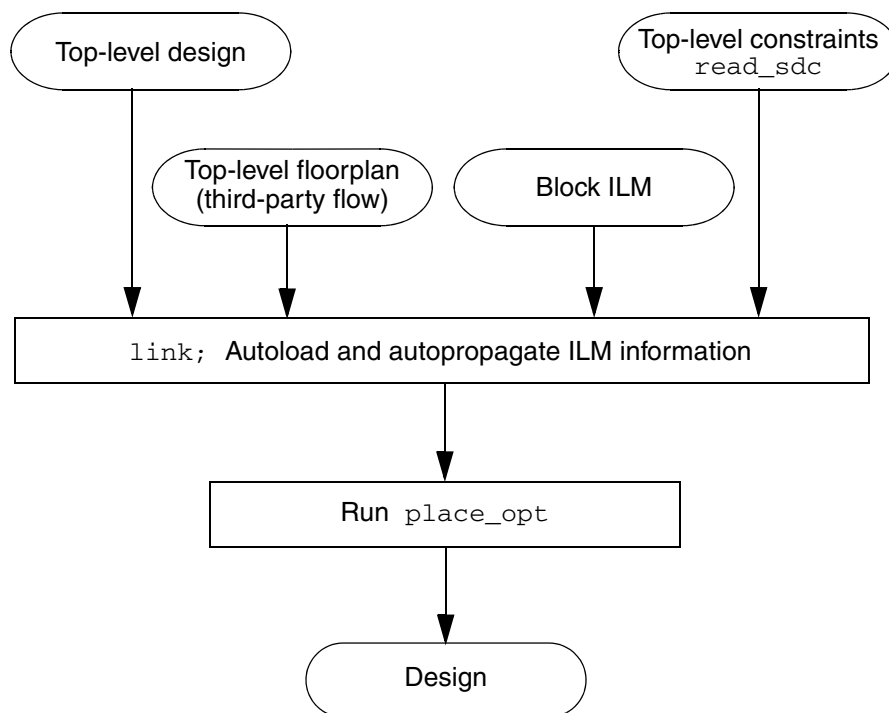
The orientation of an ILM block is defined in the floorplan. The floorplan can be specified as a DEF file or in the Milkyway database using attributes. When the ILM block is instantiated in the top level, the physical area that it occupies in the core area depends on the orientation of the block. The default orientation is N (north with 0-degree rotation).

Rotating a physical block is not supported in IC Compiler. Block rotation, which includes rotating site arrays, power plan, and so on, must be done in a floorplanning tool.

Instantiating and Using ILMs in the Top-Level Design

Figure 11-21 shows the `place_opt` flow for instantiating and using ILMs in IC Compiler.

Figure 11-21 `place_opt` Flow for Instantiating and Using ILMs



ILMs are stored in the ILM view of the Milkyway design library rather than in the CEL view. The models can be stored in either the current Milkyway design library or in a Milkyway reference library that is linked to the current Milkyway design library.

IC Compiler automatically links in ILMs that are referenced in the top-level design. Specifically, whenever there is an unresolved reference in the design, the appropriate ILM is linked in from the design/reference library.

IC Compiler first searches the explicitly specified `link_library` files, and then the Milkyway design and reference libraries for ILM views matching the unresolved reference name.

Note:

If a .db file corresponding to a macro (an extracted timing model) is specified in the `link_library`, the `link` command chooses that model and does not search for the ILM in the Milkyway library, because higher precedence is given to explicitly specified files.

For example, the following commands load the top-level design, including the ILMs:

```
icc_shell> open_mw_cel top_design
icc_shell> current_design top_design
icc_shell> link
```

ILMs can be viewed in the IC Compiler GUI.

Procedure for Using ILMs at the Top Level

To use ILMs in the top-level design, follow these steps:

1. Read in the top-level design. Use the `open_mw_cel` command.
2. Set the current design to the top-level design.
3. Link the design by using the `link` command.
4. Use the `read_sdc` command to read the top-level constraints.
5. For a third-party tool flow, read the floorplan data from the top-level DEF file. In a Synopsys tool flow, this information already resides in the Milkyway design library.
6. Use routing (FRAM) views for the ILM blocks for which you want placement and routing resources to be available at the top level (in other words, select the routing (FRAM) views for these blocks by using the `change_macro_view` command).

Viewing ILMs in the IC Compiler GUI

In the IC Compiler GUI, an ILM instance is displayed as a level of physical hierarchy (a rectangular or rectilinear shape with a fill pattern). The View Settings panel provides additional control over the ILM display.

- To display the ILM pins, select the Pin visibility option.
- To expand the ILM to show the leaf cells and macros within the ILM, type or select a positive integer in the “View level” box, and then click the Settings tab and select ILM in the Child View Name list.
- To reset the display to the default (unexpanded) view, type or select 0 in the “View level” box.

By default, when you change options on the View Settings panel, you must click Apply to update the layout view display.

How the `place_opt` Command Handles ILMs

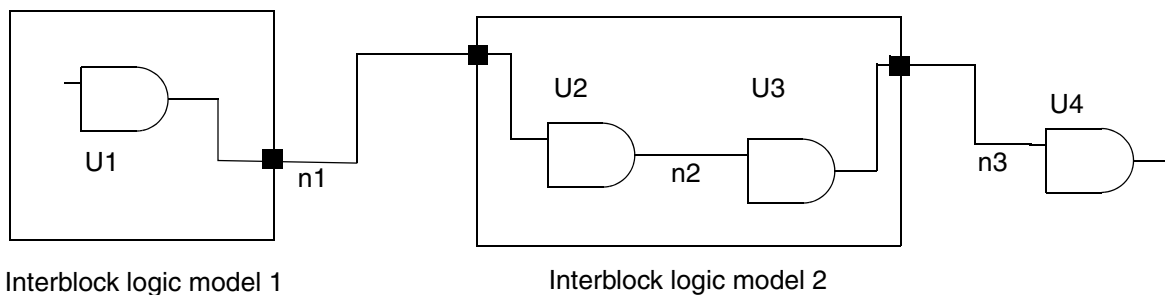
When you run the `place_opt` command, the tool uses and respects ILM objects such as placement blockages, route guides, keepout margins, leaf cell locations, pin locations, and delays (or parasitics) that have been propagated to the top-level design. The command recognizes any placement resources that have been defined by using the `create_macro_fram` command and changing to the FRAM view (`change_macro_view` command).

Keep the following interconnect-related points in mind:

- Because an ILM is a `dont_touch` cell, a path originating in an ILM and ending in an abutted ILM cannot be optimized at the top level. Optimization must be done at the block level to ensure that timing is met in a path spanning abutted ILMs.
- For nets crossing the ILM boundary, the ILM leaf cell pin locations and the ILM pin locations are used to compute the virtual routes in the preroute flow. See, for example, nets n1 and n3 in [Figure 11-22](#).
- For nets fully within an ILM, the capacitances are not reestimated. See net n2 in [Figure 11-22](#).

[Figure 11-22](#) shows interface logic model handling by IC Compiler.

Figure 11-22 Interface Logic Model Handling by IC Compiler



Other ILM Capabilities

The `get_location` command can return the coordinates for pins of an ILM instance. For example, to get the coordinates of the IN1 pin on the ILM instance `ILM_inst`, enter

```
icc_shell> get_location [get_pins ILM_inst/IN1]
```

You can perform on-route optimization on designs that contain ILMs.

How the `clock_opt` Command Handles ILMs

You can use interface logic models (ILMs) to increase the capacity and reduce the runtime for top-level clock tree synthesis. Before creating ILMs for use with top-level clock tree synthesis, you must perform clock tree synthesis on the blocks.

When you run the `clock_opt`, the command

- Identifies any ILMs inside a clock tree

When a clock defined at the top level goes through an ILM, IC Compiler inserts guide buffers before the ILM clock input pin and after the ILM clock output pins. The nets between the input guide buffer and output guide buffers are marked as don't buffer nets.

- Does not insert any cells beyond the input clock port of an ILM
- Honors explicit stop pins, exclude pins, and sink pins on an ILM port or inside an ILM
- Times the clock subtrees inside the ILM to calculate the phase and transition delays for the ILM

IC Compiler uses the timing information for the clock trees within the ILM to perform skew balancing and insertion delay minimization up to the ILM clock input pins and beyond the ILM clock output pins.

If there are multiple subtrees after an ILM, IC Compiler synthesizes each subtree independently and does not balance the insertion delay between them, which can result in large skew between them. To reduce this skew, run the `optimize_clock_tree` command after performing clock tree synthesis.

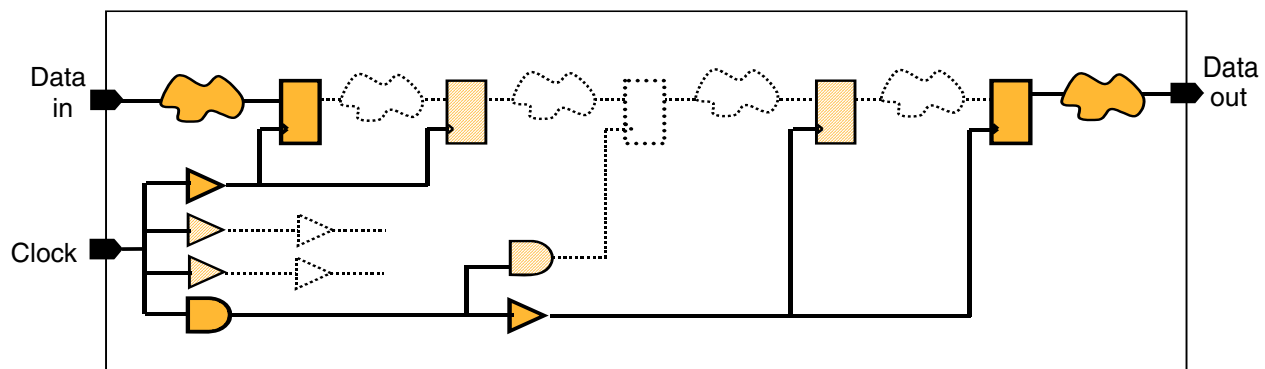
- Honors clocks or generated clocks defined on an ILM port or a pin internal to the ILM

When a clock is defined on an ILM input port or in an ILM, IC Compiler inserts guide buffers after the ILM clock output pins. Clock nets within the ILM (up to the guide buffers) are marked as don't buffer nets.

Note:

As a reminder, [Figure 11-23](#) shows the elements contained in an interface logic model, emphasizing the clock tree. The shaded elements are the elements on the clock path; the striped elements are the side-load cells. Unshaded elements are not included in the model.

Figure 11-23 Interface Logic Model



For more information, see [Chapter 7, “Clock Tree Synthesis.”](#)

How the route_opt Command Handles ILMs

When you run the `route_opt` command on a top-level design with instantiated ILMs, you must use FRAM views for all the blocks modeled by ILMs. In other words, you must use the `change_macro_view` command for the blocks. The tool treats the ILMs as placement blockages while routing resources are taken from the FRAM view of all the blocks. The tool also honors any keepout margins you have specified.

On-Route Capability With ILMs

You can perform late-stage top-level on-route optimization by running the `route_opt` or `psynopt -on_route` command. First, however, you must route the blocks. Second, you must create a FRAM view for the block (with the `create_macro_fram` command). Third, you extract the parasitic data and create an ILM for each block, keeping the parasitics. The ILM models and the parasitics are stored in the ILM view. Finally, you run either the `route_opt` or `psynopt -on_route` command on the top-level design.

For example, assuming block1 is already routed, enter

```
icc_shell> open_mw_cel block1
icc_shell> create_macro_fram -library_name design_lib -cell_name block1
icc_shell> extract_rc
icc_shell> create_ilm
icc_shell> close_mw_cel
icc_shell> open_mw_cel top
icc_shell> change_macro_view -reference block1 -view FRAM
icc_shell> link
icc_shell> read_sdc top.sdc
##
## Run the route command or the psynopt -on_route command
icc_shell> route_opt (or psynopt -on_route)
```

Hierarchical Signal Integrity Flow With ILMs

ILMs can be used in a hierarchical signal integrity flow. You can include signal integrity information in the ILM by using the `-include_xtalk` option with the `create_ilm` command. You can also use the hierarchical signal integrity flow in combination with the multicorner-multimode capability that is described later in this chapter.

ILM support for a hierarchical signal integrity flow requires the following steps:

- Use the IC Compiler design planning tool to carry out signal integrity budgeting.
- In the block-level optimization, use SI-aware optimization with the `route_opt` command to fix all block timing and signal integrity issues and provide block shielding for the ILM.

- At the block level, run the `create_ilm -include_xtalk` command to generate Milkyway signal-integrity-aware ILMs.
- At the top level, perform signal integrity analysis and optimization. For more information, see [Chapter 10, “Signal Integrity.”](#)

The following two scripts suggest how to carry out a signal integrity flow using ILMs:

Block level

```
open_mw_lib <design>
open_mw_cel <block>
place_opt
clock_opt
set_si_options ...
route_opt
save_mw_cel
create_ilm -include_xtalk
create_macro_fram
```

Top level

```
open_mw_lib <design>
open_mw_cel <top>
link
place_opt
clock_opt
set_si_options ...
route_opt
save_mw_cel
```

Multicorner-Multimode Scenarios and ILMs

ILMs are compatible with multicorner-multimode scenarios. You can apply multicorner-multimode constraints to an ILM and use that ILM in a top-level design.

The following requirements apply to using ILMs with multicorner-multimode scenarios:

- For each top-level scenario, an identically named scenario must exist in each of the ILM blocks used in the design. An ILM can have additional scenarios that are not used at the top level, as long as those scenarios do not have any additional TLUPlus file settings. In other words, the ILM can have additional modes but not additional parasitic corners (TLUPlus files).
- In a top-level design without multicorner-multimode scenarios, only ILMs without multicorner-multimode scenarios can be used.

- The number of TLUPlus files and the TLUPlus file names must match exactly between the top level and the block level. When you run the `extract_rc` command at the top level, it checks for consistency of TLUPlus files used at the top level and at the block level. Any reported errors must be fixed to prevent incorrect RC propagation for the ILM nets.
- Prior to running the `create_ilm` command, you must activate all the scenarios that are of interest to the top-level flow by using `set_active_scenarios` command. You must also run the `extract_rc` command after `set_active_scenarios` to ensure that appropriate timing information is stored in the ILM for each of the newly active scenarios.

For more information, refer to [Chapter 14, “Using Multicorner-Multimode Technology.”](#)

Creating ILMs for Multicorner-Multimode Usage

After you create ILMs with the `create_ilm` command and save them into the Milkyway database, you can use the ILMs as blocks at the top level.

The following is a summary of the usage flow for preparing an ILM block with multicorner-multimode scenarios:

1. Open the block-level design.
2. Remove all scenarios.

```
remove_scenario -all
```

3. Create scenarios for the block.

```
create_scenario s1
source ${macro}_s1.tcl
create_scenario s2
source ${macro}_s2.tcl
create_scenario CTS_only
source ${macro}_cts.sdc
set_cts_scenario CTS_only
set_active_scenarios {s1 s2...}
```

4. Perform placement and optimization.

```
place_opt
clock_opt
route_opt
```

5. Prepare the block for ILM creation.

```
set_active_scenarios { s1, s2, ...CTS_only}  
extract_rc
```

The `extract_rc` command is optional if you are using the `-keep_parasitics` option of `create_ilm`.

6. Create an ILM and save it into the Milkyway database.

```
create_ilm
```

The `create_ilm` command saves only the active scenarios to ILM view. The inactive scenarios are ignored. Ensure that the needed scenarios (for example, `CTS_only`, and so on) are activated prior to this step.

Using ILMs in Multicorner-Multimode Scenarios

The following is a summary of the usage flow for using an ILM block with multicorner-multimode scenarios at the top level:

1. Open the top-level design.

2. Remove all scenarios.

```
remove_scenario -all
```

3. Create scenarios for the top level.

```
create_scenario s1  
source ${top}_s1.tcl  
create_scenario s2  
source ${top}_s2.tcl  
create_scenario CTS_only  
source ${top}_cts.sdc  
set_cts_scenario CTS_only  
set_active_scenarios {s1, s2...}
```

The top-level scenarios must match the existing scenarios in the ILM blocks used in the design. All scenarios of the top level must be defined before the next step.

4. Link the design.

```
link
```

Any errors reported at this point must be fixed before the next step.

5. Perform optimization.

```
place_opt  
clock_opt  
route_opt
```

6. Save the top-level design into the Milkyway database.

```
save_mw_cel -as ${top}_routed
```

Summary of Commands for ILMs

[Table 11-4](#) lists commands for ILMs.

Table 11-4 Summary of Commands for Interface Logic Models

Command	Description
<code>create_ilm</code>	Determines the objects in the current design that are identified as belonging to interface logic; creates an ILM; and saves the model in memory, replacing the original design. The ILM in memory has the same design name as the original block. See “Summary of the create_ilm Command Options” on page 11-17 .
<code>get_die_area</code>	Provides the list of coordinates for the current design die area.
<code>get_ilms</code>	Returns a collection of ILMs defined in the current design. Use the <code>-quiet</code> option to prevent this command from issuing messages.
<code>get_ilm_objects</code>	Returns a collection of nets, cells, or pins that are identified as belonging to interface logic.

Note:

Using the FRAM view for the ILM blocks is recommended. To make the placement blockages, routing guides, and over-the-block routing resources available at the top level, use the `create_macro_fram` and `change_macro_view` commands.

12

Physical Datapath With Relative Placement

The IC Compiler physical datapath with relative placement capability provides a way for you to create structures in which you specify the relative column and row positions of instances. During placement and legalization, these structures, which are placement constraints called relative placement structures, are preserved and the cells in each structure are placed as a single entity. Relative placement is also called physical datapath and structured placement.

Note:

Relative placement is available only in the IC Compiler and IC Compiler-PC packages; it is not available in the IC Compiler-XP package.

The concepts and tasks necessary for doing relative placement within IC Compiler are described in these sections:

- [Introduction to Physical Datapath With Relative Placement](#)
- [Benefits of Relative Placement](#)
- [Limitations of Relative Placement](#)
- [Flow for Relative Placement](#)
- [Considerations for Using Relative Placement](#)
- [Creating Relative Placement Groups](#)
- [Adding Objects to a Group](#)
- [Placement of Relative Placement Groups](#)

- Postplacement Optimization of Relative Placement Groups
- Analyzing the Relative Placement Results
- Working With Relative Placement Groups in the GUI
- Querying Relative Placement Groups
- Saving Relative Placement Information
- Ignoring Relative Placement Constraints
- Removing Relative Placement Groups
- Changing Relative Placement Information
- Deriving Relative Placement Groups
- Summary of Relative Placement Commands

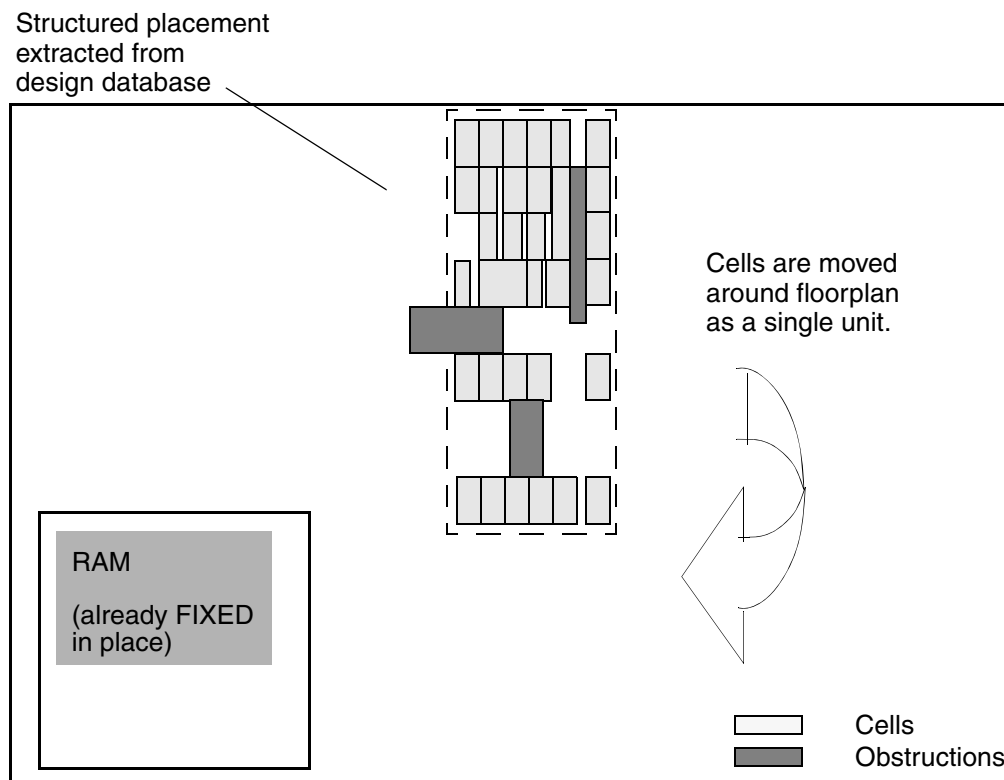
Introduction to Physical Datapath With Relative Placement

You do relative placement by using a set of dedicated commands to specify the relative placement constraints and apply them to a gate-level netlist, check the constraints, remove the constraints, or write the annotated relative placement constraints to a script.

Relative placement is usually applied to datapaths and registers, but you can apply it to any cells in your design, controlling the exact relative placement topology of gate-level logic groups and defining the circuit layout. You can use relative placement to explore QoR benefits, such as shorter wire lengths, reduced congestion, better timing, skew control, fewer vias, better yield, and lower dynamic and leakage power.

The relative placement constraints you create and annotate implicitly generate a matrix structure of the instances and control the placement of the instances. You use the resulting annotated netlist for physical optimization, during which IC Compiler preserves the structure and places it as a single entity or group, as shown in [Figure 12-1](#).

Figure 12-1 Relative Placement in a Floorplan



Benefits of Relative Placement

Relative placement provides these benefits:

- Provides a method for maintaining structured placement for legacy or intellectual property (IP) designs
- Handles flat and hierarchical designs

For complex designs, a typical design can have many engineers working on it and many blocks. Hierarchical relative placement makes it possible to place those blocks together relative to each other more easily. Any number of hierarchical levels is allowed.

- Reduces the placement search space in critical areas of the design, meaning greater predictability of QoR (wire length, timing, power) and congestion
- Is technology-independent
- Can improve routability

Limitations of Relative Placement

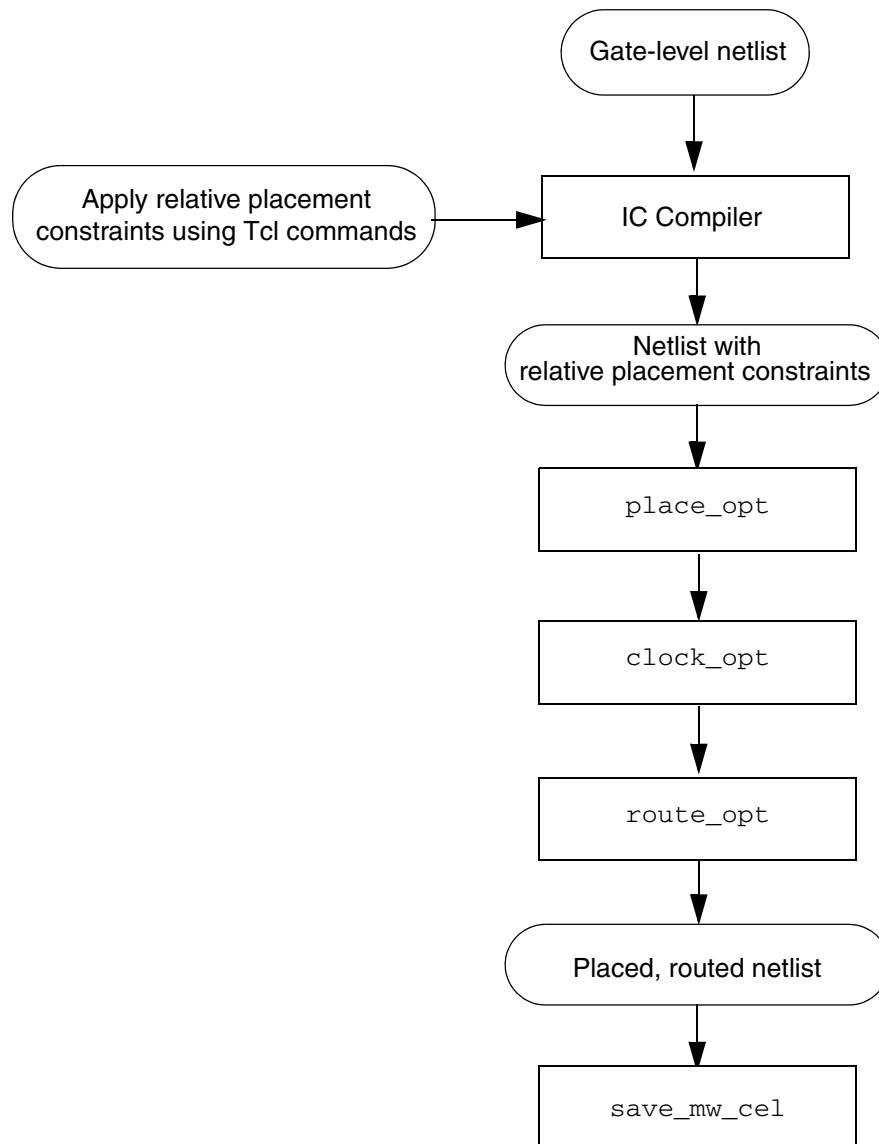
Relative placement has these limitations:

- When IC Compiler estimates that the size of a relative placement group is not suitable to the given floorplan, IC Compiler can fail to maintain the relative placement constraints during placement. To maintain relative placement information precisely, there must be enough space for relative placement groups without overlapping placement obstructions in the design floorplan.
- If your design contains multiheight cells, relative placement constraints might not be honored, because the legalizer might move relative placement cells to a valid site and leave an unintended gap.
- There is no limit on the number of cells in a relative placement group. However, if your design has many relative placement groups, coarse placement returns overlapping group locations at times, resulting in misalignment. In such cases, a warning appears after coarse placement.
- If you use move bounds to constrain relative placement, all cells in a relative placement group must be in the same move bound; otherwise the resulting QoR might not be acceptable.

Flow for Relative Placement

The relative placement flow is shown in [Figure 12-2](#).

Figure 12-2 Relative Placement Flow



The flow for using relative placement follows these major steps:

1. Read a gate-level netlist into IC Compiler.
2. Specify and apply the gate-level relative placement constraints.

3. Generate the placed netlist.
4. Perform clock tree synthesis with the relative placement structures fixed in place.
5. Perform routing with the relative placement structures fixed in place.

Methodology for the Relative Placement Flow

The methodology for the relative placement flow follows these major steps:

1. Read the gate-level netlist into IC Compiler by using the `open_mw_cel` command.
2. Define the relative placement constraints.
 - Create the relative placement groups by using the `create_rp_group` command.
See [“Creating Relative Placement Groups” on page 12-9](#).
 - Add relative placement objects to the groups by using the `add_to_rp_group` command.

See [“Adding Objects to a Group” on page 12-13](#).

IC Compiler annotates the netlist with the relative placement constraints.

3. Prevent relative placement cells from being removed during optimization by using `psynopt_option`. For example, enter

```
icc_shell> set_rp_group_options \  
-psynopt_option size_only \  
[get_rp_groups *]
```

See [“Preserving Relative Placement Cells During Optimization” on page 12-35](#).

4. Read floorplan information. For example, enter

```
icc_shell> read_def top.def
```

5. Perform placement for the design by using the `place_opt` command.

Note:

Before running `place_opt`, you can use the `create_placement` command to validate the relative placement results. After you achieve the desired relative placement results, run `place_opt` to perform placement and optimization.

6. Analyze the relative placement results.

You can analyze the relative placement results with either `icc_shell` or the IC Compiler GUI. See [“Analyzing the Relative Placement Results” on page 12-40](#).

If the relative placement is not what you want, modify the relative placement constraints and run this procedure again.

7. Perform clock tree synthesis and physical optimization with the relative placement structures fixed in place. For example, enter

```
icc_shell> set_rp_group_options -cts_option fixed_placement \  
[get_rp_groups *]  
icc_shell> clock_opt
```

See “Postplacement Optimization of Relative Placement Groups” on page 12-39.

8. Perform routing with the relative placement structures fixed in place. For example, enter

```
icc_shell> set_rp_group_options -route_opt_option fixed_placement \  
[get_rp_groups *]  
icc_shell> route_opt
```

See “Postplacement Optimization of Relative Placement Groups” on page 12-39.

Sample Script for a Relative Placement Flow

[Example 12-1](#) is a sample script for running a relative placement flow.

Example 12-1 Sample Script for the Relative Placement Flow

```
# Set library and design paths  
source setup.tcl  
open_mw_cel design_name  
current_design top  
link  
  
# Create relative placement constraints  
create_rp_group ...  
...  
add_to_rp_group ...  
...  
  
# Apply design constraints  
source constraints.tcl  
# Apply set_size_only on relative placement cells  
set_size_only [rp_group_references -leaf]  
  
# Read floorplan information  
read_def top.def  
  
# Perform coarse placement on the design  
place_opt
```

```
# Preserve relative placement during clock tree synthesis
set_rp_group_options -cts_option fixed_placement \
    [get_rp_groups *]

# Perform clock tree synthesis on the design
clock_opt

# Preserve relative placement during routing
set_rp_group_options -route_opt_option fixed_placement \
    [get_rp_groups *]

# Perform routing
route_opt
```

Considerations for Using Relative Placement

When you use relative placement, keep the following points in mind:

- Relative placement requires a gate-level netlist. The format can be any format that IC Compiler can read.
- A design can contain both structured and unstructured objects (leaf cells, keepouts, hierarchical groups), and you can control which cells are to be structured by including the cells you want to structure in a relative placement group.

You need to determine which portions of the design need to be structured because providing relative placement information for cells that would have been placed better by allowing IC Compiler to place the cells can produce poor results.

- Some designs, such as datapaths, are appropriate for structured placement, whereas others are more appropriate for usual placement by IC Compiler. You must ensure that relative placement is applicable to your design.
- Relative placement is supported in multivoltage designs. In such designs, a relative placement group cannot cross voltage regions. For more information about multivoltage designs, see [Chapter 13, "Multivoltage Design Flow."](#)

Creating Relative Placement Groups

A relative placement group is an association of cells, other groups, and keepouts. A group is defined by the number of rows and columns it uses.

The basic syntax for creating a relative placement group is

```
create_rp_group group_name
    [-design design_name
    [-columns col_cnt] [-rows row_cnt]]
```

IC Compiler creates a relative placement group named *design_name::group_name*, where *design_name* is the design specified by *-design*, or the current design if you do not use the *-design* option. You must use this name (or a collection of relative placement groups) when referring to this group in other relative placement commands.

If you do not specify any options, the tool creates a relative placement group that has one column and one row but does not contain any objects. To add objects (leaf cells, relative placement groups, or keepouts) to a relative placement group, use the `add_to_rp_group` command, which is described in [“Adding Objects to a Group” on page 12-13](#).

For example, to create a group named `designA::rp1`, having six columns and six rows, enter

```
icc_shell> create_rp_group rp1 -design designA -columns 6 -rows 6
```

[Figure 12-3](#) shows the positions of columns and rows in a relative placement group.

Figure 12-3 Relative Placement Column and Row Positions

row 5	0 5	1 5	2 5	3 5	4 5	5 5
row 4	0 4	1 4	2 4	3 4	4 4	5 4
row 3		1 3	2 3	3 3	4 3	5 3
row 2	0 2	1 2	2 2	3 2	4 2	5 2
row 1	0 1	1 1	2 1	3 1		5 1
row 0	0 0	1 0	2 0	3 0	4 0	5 0
	col 0	col 1	col 2	col 3	col 4	col 5

In this figure,

- Columns count from column 0 (the leftmost column).
- Rows count from row 0 (the bottom row).

- The width of a column is the width of the widest cell in that column.
- The height of a row is the height of the tallest cell in that row.
- It is not necessary to use all positions in the structure. For example, in this figure, positions 0 3 (column 0, row 3) and 4 1 (column 4, row 1) are not used.

In addition to the size of the relative placement group, you can also specify attributes of the group. [Table 12-1](#) describes the attributes that you can set on a relative placement group.

Table 12-1 Relative Placement Group Attributes

To do this	Use this option
Specify the anchor location. See “Anchoring Relative Placement Groups” on page 12-11.	-x_offset -y_offset
Specify the type of alignment used by the group. You can specify either bottom-left (the default), bottom-right, or bottom-pin. See “Aligning Leaf Cells Within a Column” on page 12-15.	-alignment
Specify the group alignment pin. See “Aligning Leaf Cells Within a Column” on page 12-15.	-pin_align_name
Specify the utilization percentage (default is 100 percent).	-utilization
Specify the clock tree synthesis preservation attribute. You can specify <code>fixed_placement</code> or <code>size_only</code> . See “Postplacement Optimization of Relative Placement Groups” on page 12-39.	-cts_option
Specify the physical optimization preservation attribute. You can specify <code>fixed_placement</code> or <code>size_only</code> . See “Postplacement Optimization of Relative Placement Groups” on page 12-39.	-psynopt_option
Specify the routing preservation attribute. You can specify <code>fixed_placement</code> or <code>in_place_size_only</code> . See “Postplacement Optimization of Relative Placement Groups” on page 12-39.	-route_opt_option
Ignore this relative placement group. See “Ignoring Relative Placement Constraints” on page 12-46.	-ignore

Table 12-1 Relative Placement Group Attributes (Continued)

To do this	Use this option
Allow keepouts over tap cells. See “Adding Keepouts” on page 12-32 .	<code>-allow_keepout_ over_tapcell</code>
Legalize relative placement groups. You can specify low, medium, or high. See “Movement Control When Legalizing Relative Placement Groups” on page 12-34 .	<code>-move_effort</code>

Anchoring Relative Placement Groups

By default, IC Compiler can place a relative placement group anywhere within the core area. You can control the placement of a top-level relative placement group by anchoring it.

To anchor a relative placement group, use the `create_rp_group` or the `set_rp_group_options` command with the `-x_offset` and `-y_offset` options. The offset values are float values, in microns, relative to the chip's origin.

If you specify both the x- and y-coordinates, the group is anchored at that location. If you specify only one coordinate, IC Compiler can determine the placement by maintaining the specified coordinate and sliding the group along the line passing through the unspecified coordinate.

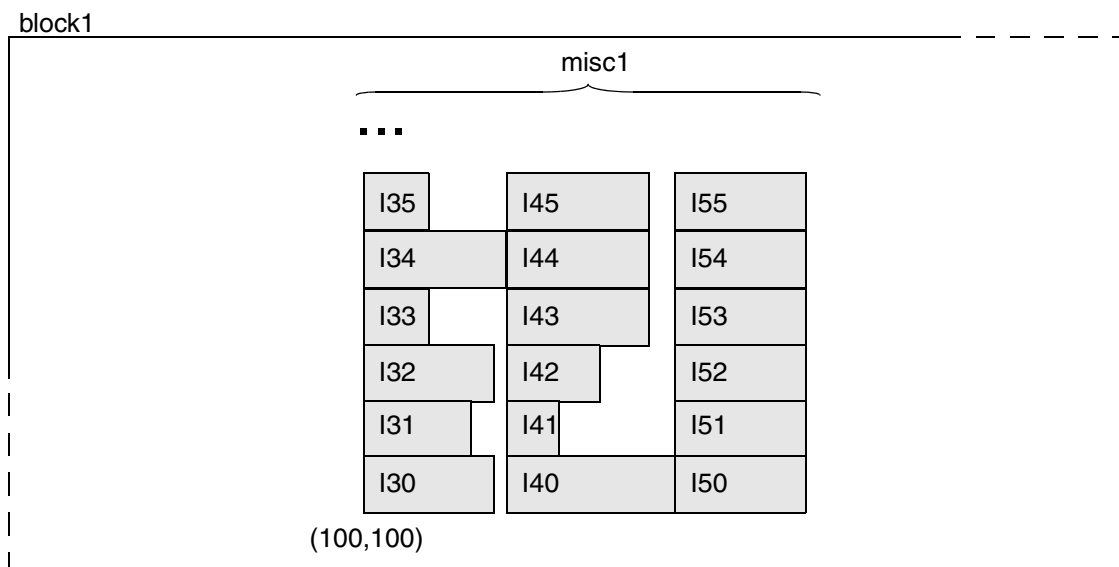
Note:

If you try to anchor a group that is not a top-level group or that causes an invalid placement, IC Compiler generates a warning and continues relative placement. If you specify an anchor point that is outside the design boundary (or an anchor point that causes cells in the relative placement group to be outside the design boundary), IC Compiler generates a warning message and clusters the cells inside the design boundary.

For example, to specify a relative placement group anchored at (100, 100), as shown [Figure 12-4](#), enter the following command:

```
icc_shell> create_rp_group misc1 -design block1 \  
          -columns 3 -rows 10 -x_offset 100 -y_offset 100
```

Figure 12-4 Anchored Relative Placement Group

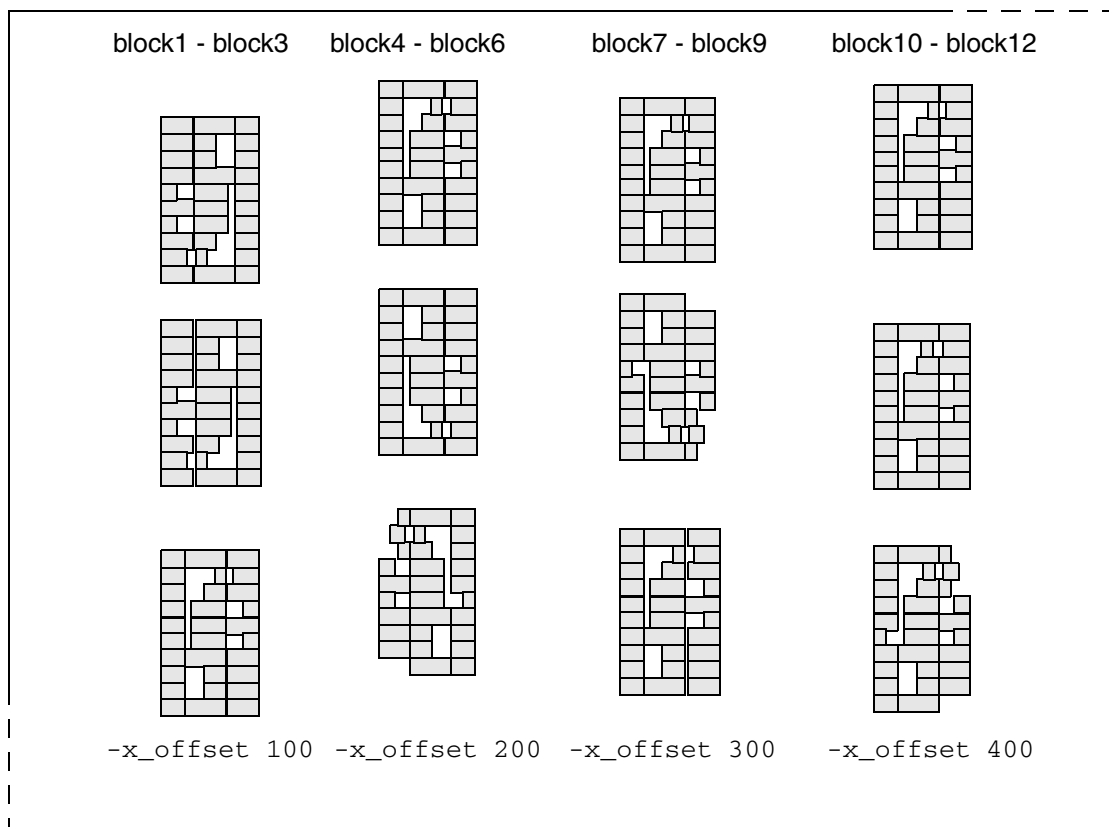


The script in [Example 12-2](#) defines the locations of 12 relative placement groups that are aligned and anchored vertically at four coordinates, as shown in [Figure 12-5](#). For brevity, not every group is listed in the example.

Example 12-2 Definitions for Locations of Vertically Aligned and Anchored Groups

```
create_rp_group block1 -design misc1 -columns 3 -rows 10 -x_offset 100
create_rp_group block2 -design misc1 -columns 3 -rows 10 -x_offset 100
create_rp_group block3 -design misc1 -columns 3 -rows 10 -x_offset 100
create_rp_group block4 -design misc1 -columns 3 -rows 10 -x_offset 200
create_rp_group block5 -design misc1 -columns 3 -rows 10 -x_offset 200
create_rp_group block6 -design misc1 -columns 3 -rows 10 -x_offset 200
...
create_rp_group block12 -design misc1 -columns 3 -rows 10 -x_offset 400
```

Figure 12-5 Relative Placement Groups Aligned and Anchored Vertically



Adding Objects to a Group

You can add leaf cells, other relative placement groups, and keepouts to relative placement groups (created with the `create_rp_group` command). You can add objects either in the relative placement hierarchy browser (for information about the relative placement hierarchy browser, see [“Using the Relative Placement Hierarchy Browser” on page 12-42](#)) or from the command line (using the `add_to_rp_group` command).

When you add an object to a relative placement group, keep the following points in mind:

- The relative placement group to which you are adding the object must exist.
- The object must be added to an empty location in the relative placement group.

Adding Leaf Cells

When you add leaf cells to a relative placement group, you specify their column and row position within the relative placement group. You can also specify how to orient the cells.

To add leaf cells in the relative placement hierarchy browser (Placement > New RP Hierarchy View),

1. Select the group you want to add to in the left pane.
2. Select the Grid tab in the right pane.
3. Select the desired location in the grid view.

When you add leaf cells, make sure that the selected location is empty.

4. Right-click and choose Add Object.

The Add Objects to RP dialog box appears.

5. Select Add Leaf Cell, specify the leaf cell name, and select the possible orientations.
6. Click OK.

Alternatively, you can use the `add_to_rp_group` command. The syntax for adding a leaf cell to a relative placement group by using `add_to_rp_group` is

```
add_to_rp_group group_list -leaf cell_name
                [-column col_number] [-row row_number]
                [-orientation direction]
                [-alignment {bottom-left | bottom-right} |
                -pin_align_name pin_name]
```

Specifying Orientation for Leaf Cells

You can specify orientations for leaf cells when you add them to a relative placement group, but you cannot specify an orientation for a relative placement group. If you do not specify a leaf cell orientation, IC Compiler assigns a default orientation, and then (by default) optimizes that orientation for wire length.

To specify the orientation for leaf cells,

- Select the possible orientations in the Add Objects to RP dialog box.
- or
- Include the `-orientation` option with a list of possible orientations when you add the cells to the group with the `add_to_rp_group` command.

The tool chooses the first legal orientation from the list of orientations you provide.

To prevent IC Compiler from automatically optimizing the orientation of cells without user-specified orientation specifications,

- Set the `physopt_rp_enable_orient_opt` variable to false.

By default, this variable is set to true and IC Compiler optimizes the orientation of cells without user-specified orientations.

Aligning Leaf Cells Within a Column

You can align the leaf cells in a column of a relative placement group by using the following alignment methods:

- Bottom left (default)
- Bottom right
- Pin alignment

Controlling the cell alignment can improve the timing and routability of your design.

Aligning by Bottom-Left Corners. By default, IC Compiler aligns the leaf cells by aligning the bottom-left corners. (To explicitly specify this alignment method, use the `-alignment bottom-left` option of the `create_rp_group` or `set_rp_group_options` command.)

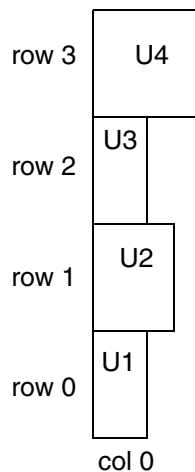
```
icc_shell> set_rp_group_options -alignment bottom-right [get_rp_groups *]
```

The script in [Example 12-3](#) defines a relative placement group that is bottom-left aligned. The resulting structure is shown in [Figure 12-6](#).

Example 12-3 Definition for Bottom-Left-Aligned Relative Placement Group

```
create_rp_group rp1 -design pair_design -columns 1 -rows 4
add_to_rp_group pair_design::rp1 -leaf U1 -column 0 -row 0
add_to_rp_group pair_design::rp1 -leaf U2 -column 0 -row 1
add_to_rp_group pair_design::rp1 -leaf U3 -column 0 -row 2
add_to_rp_group pair_design::rp1 -leaf U4 -column 0 -row 3
```

Figure 12-6 Bottom-Left-Aligned Relative Placement Group



Aligning by Bottom-Right Corners. To align a group by aligning the bottom-right corners, use the `-alignment bottom-right` option of the `create_rp_group` or `set_rp_group_options` command.

```
icc_shell> set_rp_group_options -alignment bottom-right
```

Note:

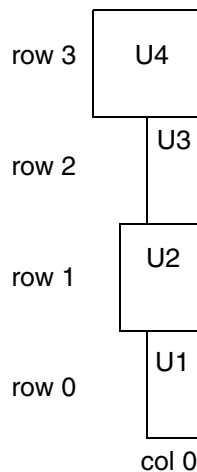
For hierarchical relative placement groups, the bottom-right alignment does not propagate through the hierarchy.

The script in [Example 12-4](#) defines a relative placement group that is bottom-right aligned. The resulting structure is shown in [Figure 12-7](#).

Example 12-4 Definition for Bottom-Right-Aligned Relative Placement Group

```
create_rp_group rp1 -design pair_design -columns 1 -rows 4 \  
-alignment bottom-right  
add_to_rp_group pair_design::rp1 -leaf U1 -column 0 -row 0  
add_to_rp_group pair_design::rp1 -leaf U2 -column 0 -row 1  
add_to_rp_group pair_design::rp1 -leaf U3 -column 0 -row 2  
add_to_rp_group pair_design::rp1 -leaf U4 -column 0 -row 3
```

Figure 12-7 Bottom-Right-Aligned Relative Placement Group



Aligning by Pin Location. To align a group by pin location, use the `-alignment bottom-pin` and `-pin_align_name align_pin` options of the `create_rp_group` or `set_rp_group_options` command.

```
icc_shell> set_rp_group_options -alignment bottom-pin  
-pin_align_name align_pin
```

IC Compiler looks for the specified alignment pin in each cell in the column. If the alignment pin exists in a cell, the cell is aligned by use of the pin location. If the specified alignment pin does not exist in a cell, the cell is aligned at the bottom-left corner and IC Compiler generates an information message. If the specified alignment pin does not exist in any cell in the column, IC Compiler generates a warning message.

If you specify both pin alignment and cell orientation, IC Compiler resolves potential conflicts as follows:

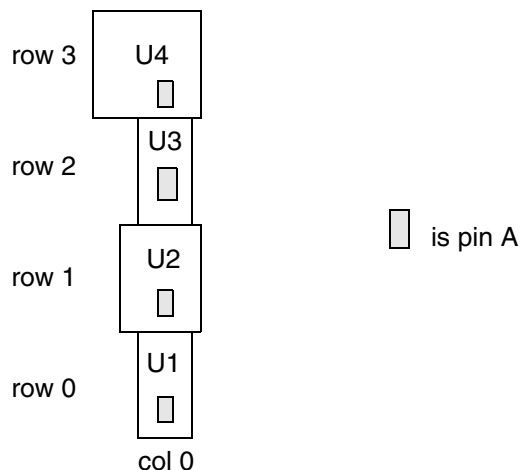
- User specifications for cell orientation take precedence over the pin alignment done by IC Compiler.
- Pin alignment done by IC Compiler takes precedence over the cell orientation optimization done by IC Compiler.

The script in [Example 12-5](#) defines a relative placement group that is aligned by pin A. The resulting structure is shown in [Figure 12-8](#).

Example 12-5 Definition for Relative Placement Group Aligned by Pins

```
create_rp_group rp1 -design pair_design -columns 1 -rows 4 -pin_align_name A
add_to_rp_group pair_design::rp1 -leaf U1 -column 0 -row 0
add_to_rp_group pair_design::rp1 -leaf U2 -column 0 -row 1
add_to_rp_group pair_design::rp1 -leaf U3 -column 0 -row 2
add_to_rp_group pair_design::rp1 -leaf U4 -column 0 -row 3
```

Figure 12-8 Relative Placement Group Aligned by Pins



When you specify an alignment pin for a group, the pin applies to all cells in the group. You can override the group alignment pin for specific cells in the group by specifying the `-pin_align_name` option when you use the `add_to_rp_group` command to add the cells to the group.

Note:

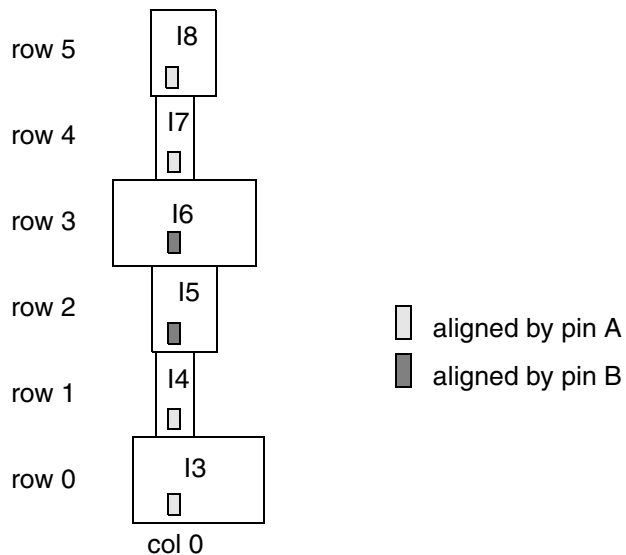
You cannot specify a cell-specific alignment pin when you add a leaf cell from the relative placement hierarchy browser.

The script in [Example 12-6](#) defines relative placement group `misc1`, which uses pin A as the group alignment pin; however, instances `I5` and `I6` use pin B as their alignment pin, rather than the group alignment pin. The resulting structure is shown in [Figure 12-9](#).

Example 12-6 Definition for Aligning a Group and Leaf Cells by Pins

```
create_rp_group misc1 -design block1 -columns 3 -rows 10 -pin_align_name A
add_to_rp_group block1::misc1 -leaf I3 -column 0 -row 0
add_to_rp_group block1::misc1 -leaf I4 -column 0 -row 1
add_to_rp_group block1::misc1 -leaf I5 -column 0 -row 2 -pin_align_name B
add_to_rp_group block1::misc1 -leaf I6 -column 0 -row 3 -pin_align_name B
add_to_rp_group block1::misc1 -leaf I7 -column 0 -row 4
add_to_rp_group block1::misc1 -leaf I8 -column 0 -row 5
```


Figure 12-9 Relative Placement Group Aligned by Pins

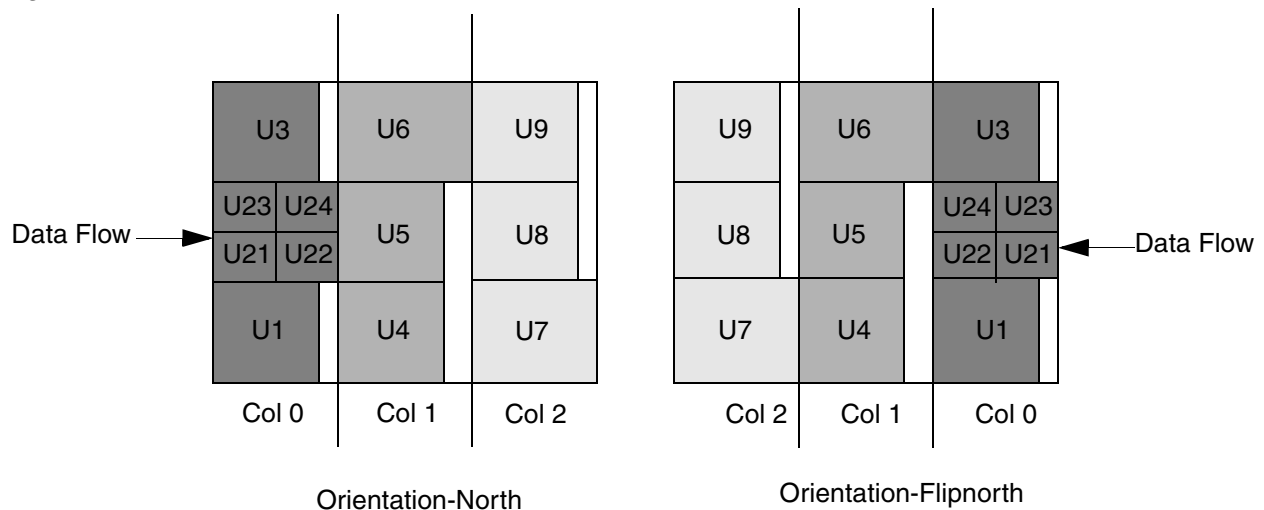


Automatic Orientation of Relative Placement Groups Based on Data Flow

The orientation of relative placement groups is set automatically by the tool according to the data flow to minimize wire length. If the data flow of the relative placement group is from left to right, the first column is placed on the left side and subsequent columns are placed towards the right. This orientation is considered as north (N). If the data flow of the relative placement groups is from right to left, the first column is automatically placed at the right side and subsequent columns are placed towards left: that is, the orientation of the block is flipped. This orientation is considered as flip-north (FN).

In [Figure 12-10](#), U21 and U22, U23, and U24 are four cells in a relative placement group. In case of flip-north orientation, the relative placement group is automatically flipped.

Figure 12-10 Auto-Orientation of Relative Placement Blocks

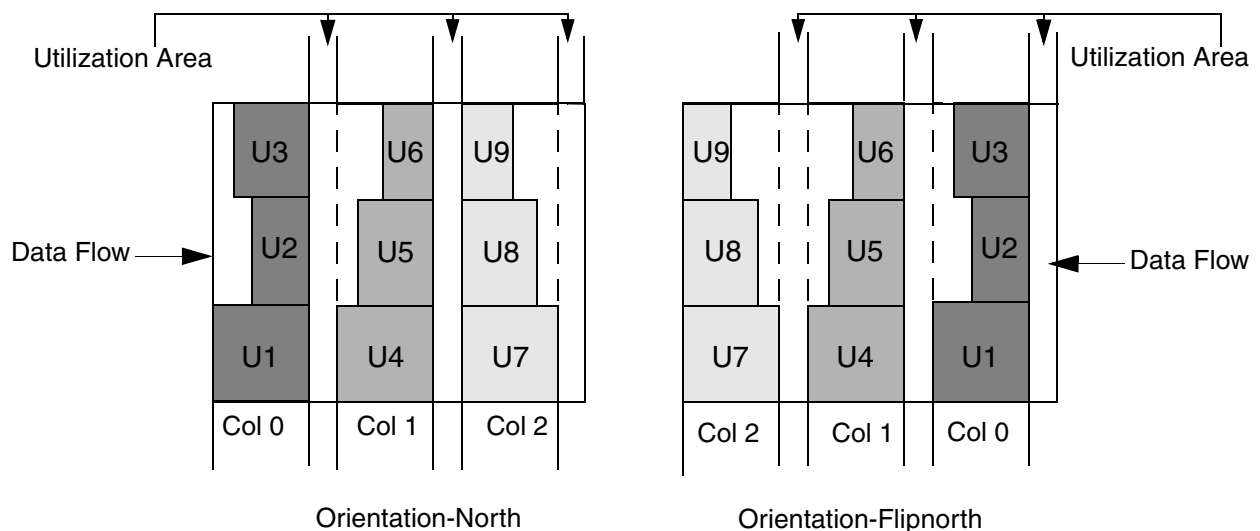


The automatic orientation of relative placement groups is supported for designs with hierarchical relative placement groups. The change in orientation is propagated down to the lowest level in hierarchy.

In case of flip-north orientation, the relative placement constraints, such as alignment, utilization, and so on are preserved according to the specifications that you provide.

Figure 12-11 shows an example in which the utilization constraints and the alignment constraints (align to the right or left edge) are preserved with flip-north orientation.

Figure 12-11 Preservation of Constraints



Adding Relative Placement Groups

Hierarchical relative placement allows relative placement groups to be embedded within other relative placement groups. The embedded groups then are handled similarly to leaf cells.

You can use hierarchical relative placement to simplify the expression of relative placement constraints. With hierarchical relative placement, you do not need to provide relative placement information multiple times for a recurring pattern.

There are two methods for adding a relative placement group to a hierarchical group:

- Include the group

If the relative placement group to be added is in the same design as its parent group, it is an included group. You can include groups in either flat or hierarchical designs. See [“Including Relative Placement Groups” on page 12-22](#).

- Instantiate the group

If the relative placement group to be added is in an instance of a subdesign of its parent group, it is an instantiated group. You can instantiate groups only in hierarchical designs. See [“Instantiating Relative Placement Groups” on page 12-24](#).

The following sections describe these aspects of hierarchical relative placement groups:

- [Benefits of Hierarchical Relative Placement Groups](#)
- [Including Relative Placement Groups](#)
- [Instantiating Relative Placement Groups](#)
- [Using Hierarchical Relative Placement for Straddling](#)
- [Using Hierarchical Relative Placement for Compression](#)
- [Effect of Ungrouping on Hierarchical Relative Placement](#)
- [Effect of Uniquifying on Hierarchical Relative Placement](#)

Benefits of Hierarchical Relative Placement Groups

Using hierarchical relative placement provides these benefits:

- Allows you to organize your relative placement in a manner that is easier to maintain and understand. For example, you can create the relative placement group to parallel your Verilog or VHDL organization.
- Allows reuse of a repeating placement pattern, such as an adder.
- Can reduce the number of lines of relative placement information you need to write.
- Allows integrating blocks.
- Provides flexibility for the configuration you want.

Including Relative Placement Groups

To include relative placement groups in the relative placement hierarchy browser (Placement > New RP Hierarchy View),

1. Select the group you want to add to in the left pane.
2. Select the Grid tab in the right pane.
3. Select the desired location in the grid view.

To include relative placement groups, make sure that the selected location is empty.

4. Right-click and choose Add Object.

The Add Objects to RP dialog box appears.

5. Specify the row and column position, specify the alignment method, select Add Hierarchical RP, and specify the relative placement group name.
6. Click OK.

Alternatively, you can use the `add_to_rp_group` command. The syntax for including a group in a hierarchical group by using `add_to_rp_group` is

```
add_to_rp_group hier_group -hierarchy group_name
                  [-column col_number] [-row row_number]
                  [-alignment bottom-left | bottom-right]
```

The group specified in the `-hierarchy` option must be in the same design as the hierarchical group in which you are including it.

When you include a relative placement group in a hierarchical group, it is as if the included group is directly embedded within its parent group. An included group can be used only in a group of the same design and only once. However, a group that contains an included group can be further included in another group in the same design or can be instantiated in a group of a different design.

The script in [Example 12-7](#) creates a hierarchical group (rp4) that contains three included groups (rp1, rp2, and rp3). Groups rp1, rp2, rp3, and rp4 are all in design top. The contents of groups rp1, rp2, and rp3 are treated as leaf cells when they are included in group rp4. You can further include group rp4 in another group in design top, or you can instantiate group rp4 in a group of a different design.

The construction of the resulting hierarchical relative placement group is shown in [Figure 12-12](#).

Example 12-7 Including Groups in a Hierarchical Group

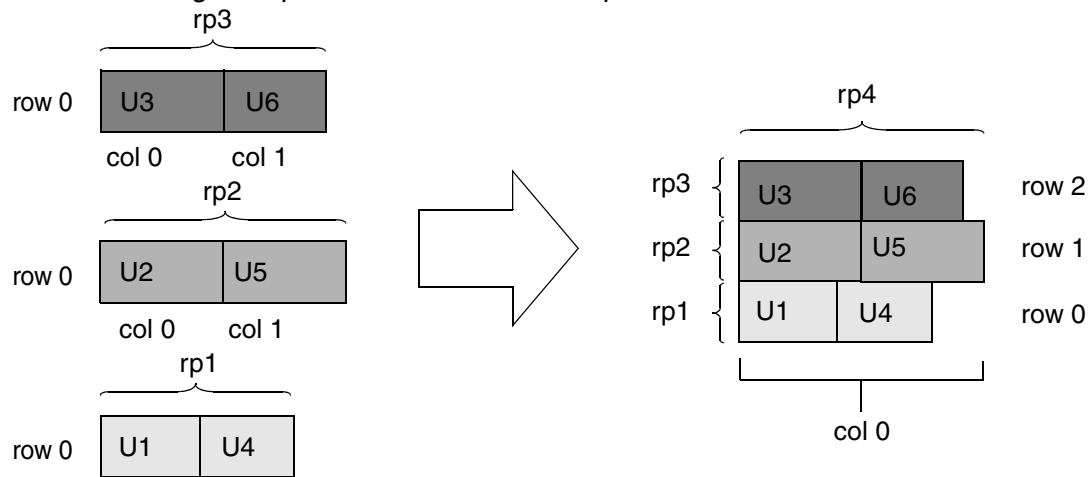
```
create_rp_group rp1 -design top -columns 2 -rows 1
  add_to_rp_group top::rp1 -leaf U1 -column 0 -row 0
  add_to_rp_group top::rp1 -leaf U4 -column 1 -row 0

create_rp_group rp2 -design top -columns 2 -rows 1
  add_to_rp_group top::rp2 -leaf U2 -column 0 -row 0
  add_to_rp_group top::rp2 -leaf U5 -column 1 -row 0

create_rp_group rp3 -design top -columns 2 -rows 1
  add_to_rp_group top::rp3 -leaf U3 -column 0 -row 0
  add_to_rp_group top::rp3 -leaf U6 -column 1 -row 0

create_rp_group rp4 -design top -columns 1 -rows 3
  add_to_rp_group top::rp4 -hierarchy top::rp1 \
    -column 0 -row 0
  add_to_rp_group top::rp4 -hierarchy top::rp2 \
    -column 0 -row 1
  add_to_rp_group top::rp4 -hierarchy top::rp3 \
    -column 0 -row 2
```

Figure 12-12 Including Groups in a Hierarchical Group



Instantiating Relative Placement Groups

The syntax for instantiating a group in a hierarchical group is

```
add_to_rp_group hier_group
    -hierarchy group_name -instance instance_name
    [-column col_number] [-row row_number]
```

Note:

You cannot instantiate relative placement groups from the relative placement hierarchy browser; you must do this from the command line.

The group specified in the `-hierarchy` option must be defined in the reference design of the instance specified in the `-instance` option. In addition, the specified instance must be in the same design as the hierarchical group in which you are instantiating the specified group.

Note:

If you define a relative placement group in a subdesign that has been instantiated multiple times, only those instances that are instantiated in a hierarchical relative placement group have relative placement. Instances that are not instantiated in a hierarchical relative placement group do not have relative placement.

Using an instantiated group is a useful way to replicate relative placement information across multiple instances of a design and to create relative placement relationships between those instances. An instantiated group can be used multiple times and in multiple places. For example, use instantiated groups for these cases:

- You have multiple relative placement layouts you want to use for different instances of a design.
- You have only one layout but want to specify relative placement between instances of that layout or between instances and other cells and groups.

The script in [Example 12-8](#) creates a hierarchical group (rp2) that contains three instances of group rp1. Group rp1 is in the design pair_design and includes leaf cells U1 and U2. Group rp2 is a hierarchical group in the design mid_design that instantiates group rp1 three times (mid_design must contain at least three instances of pair_design). Group rp2 is treated as a leaf cell. You can instantiate rp2 multiple times and in multiple places, up to the number of times mid_design is instantiated in your netlist.

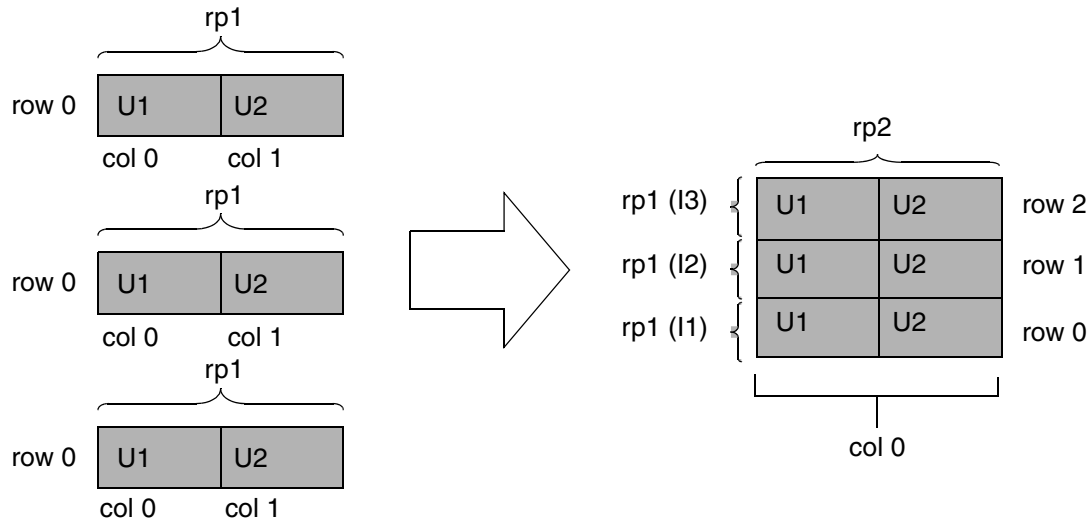
The construction of the resulting hierarchical relative placement group is shown in [Figure 12-13](#).

Example 12-8 Instantiating Groups in a Hierarchical Group

```
create_rp_group rp1 -design pair_design -columns 2 -rows 1
  add_to_rp_group pair_design::rp1 -leaf U1 -column 0 -row 0
  add_to_rp_group pair_design::rp1 -leaf U2 -column 1 -row 0

create_rp_group rp2 -design mid_design -columns 1 -rows 3
  add_to_rp_group mid_design::rp2 \
    -hierarchy pair_design::rp1 -instance I1 -column 0 -row 0
  add_to_rp_group mid_design::rp2 \
    -hierarchy pair_design::rp1 -instance I2 -column 0 -row 1
  add_to_rp_group mid_design::rp2 \
    -hierarchy pair_design::rp1 -instance I3 -column 0 -row 2
```

Figure 12-13 Instantiating Groups in a Hierarchical Group



Using Hierarchical Relative Placement for Straddling

A cell can occupy multiple column positions or multiple row positions, which is known as straddling. To define straddling, use the method for including a relative placement group (see [“Including Relative Placement Groups” on page 12-22](#)).

Figure 12-14 shows a relative placement group in which cells straddle columns (instance U2) and rows (instance U7).

Figure 12-14 Hierarchical Relative Placement Group With Straddling

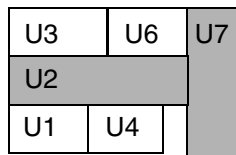
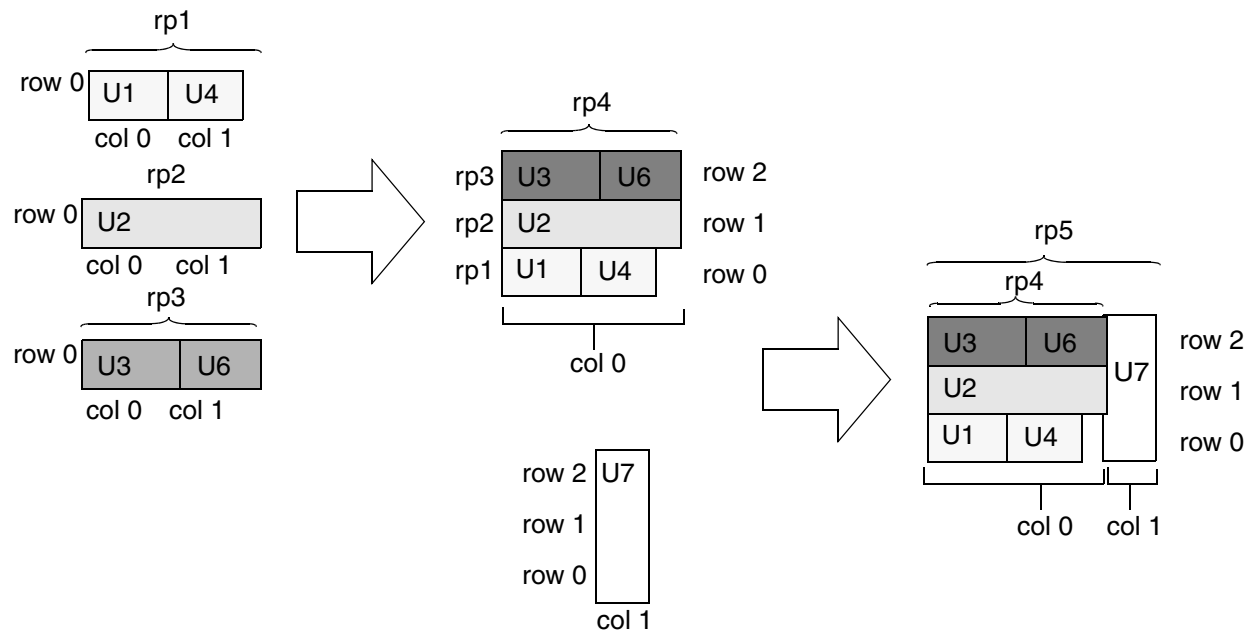


Figure 12-15 shows the process of using hierarchical relative placement to build this structure. First, define relative placement groups that contain the leaf cells: rp1 contains U1 and U4, rp2 contains U2, and rp3 contains U3 and U6. Then define a group (rp4) that contains these groups. Finally, define a group that contains the hierarchical group rp4 and the leaf cell U7. The resulting group includes both the column and the row straddle.

Example 12-9 shows the commands used in this process.

Figure 12-15 Straddling With Hierarchical Relative Placement



Example 12-9 Straddling With Hierarchical Relative Placement

```
create_rp_group rp1 -design top -columns 2 -rows 1
  add_to_rp_group top::rp1 -leaf U1 -column 0 -row 0
  add_to_rp_group top::rp1 -leaf U4 -column 1 -row 0

create_rp_group rp2 -design top -columns 1 -rows 1
  add_to_rp_group top::rp2 -leaf U2 -column 0 -row 0

create_rp_group rp3 -design top -columns 2 -rows 1
  add_to_rp_group top::rp3 -leaf U3 -column 0 -row 0
  add_to_rp_group top::rp3 -leaf U6 -column 1 -row 0

create_rp_group rp4 -design top -columns 1 -rows 3
  add_to_rp_group top::rp4 -hierarchy top::rp1 -column 0 -row 0
  add_to_rp_group top::rp4 -hierarchy top::rp2 -column 0 -row 1
  add_to_rp_group top::rp4 -hierarchy top::rp3 -column 0 -row 2

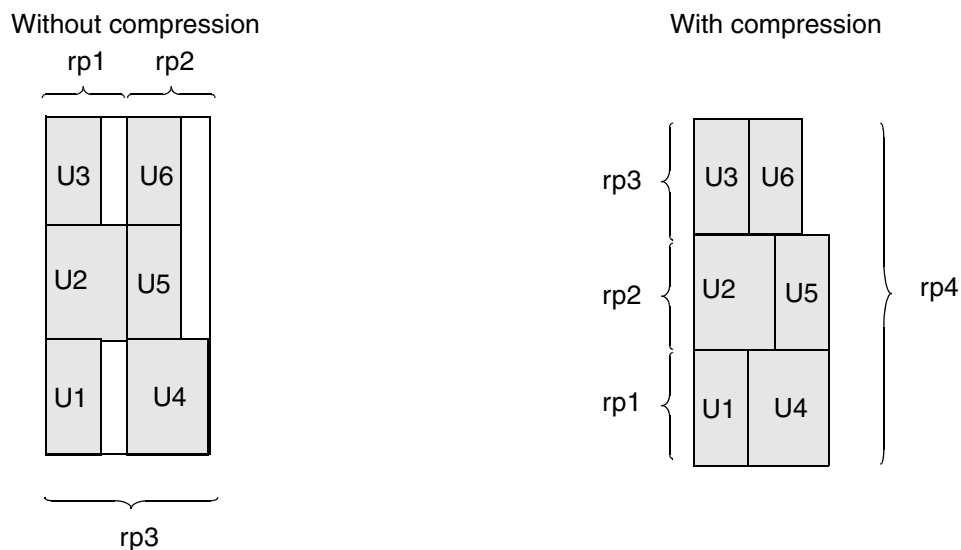
create_rp_group rp5 -design top -columns 2 -rows 1
  add_to_rp_group top::rp5 -hierarchy top::rp4 -column 0 -row 0
  add_to_rp_group top::rp5 -leaf U7 -column 1 -row 0
```

Using Hierarchical Relative Placement for Compression

By default, construction for relative placement aligns cells from their bottom-left corner. Compression removes empty space in rows to create a more compact structure. The compressed columns are no longer aligned, and utilization is higher in the area of the compressed cells. To define a compressed structure, use the method for including a relative placement group (see [“Including Relative Placement Groups” on page 12-22](#)).

[Figure 12-16](#) shows the same cells aligned without compression and with compression. To define the compressed structure, first define relative placement groups that contain the rows of leaf cells: rp1 contains U1 and U4, rp2 contains U2 and U5, and rp3 contains U3 and U6. Then define a group (rp4) that contains these groups. [Example 12-10](#) shows the commands used to build the compressed structure.

Figure 12-16 Bottom-Left Alignment Construction and Compression



Example 12-10 Compression With Hierarchical Relative Placement

```
create_rp_group rp1 -design top -columns 2 -rows 1
  add_to_rp_group top::rp1 -leaf U1 -column 0 -row 0
  add_to_rp_group top::rp1 -leaf U4 -column 1 -row 0

create_rp_group rp2 -design top -columns 2 -rows 1
  add_to_rp_group top::rp2 -leaf U2 -column 0 -row 0
  add_to_rp_group top::rp2 -leaf U5 -column 1 -row 0

create_rp_group rp3 -design top -columns 2 -rows 1
  add_to_rp_group top::rp3 -leaf U3 -column 0 -row 0
  add_to_rp_group top::rp3 -leaf U6 -column 1 -row 0

create_rp_group rp4 -design top -columns 1 -rows 3
  add_to_rp_group top::rp4 -hierarchy top::rp1 -column 0 -row 0
  add_to_rp_group top::rp4 -hierarchy top::rp2 -column 0 -row 1
  add_to_rp_group top::rp4 -hierarchy top::rp3 -column 0 -row 2
```

Alternatively, you can accomplish compression by using bit-stack placement for flat relative placement groups. Set the `physopt_use_bit_stacked_placement` variable to true (the default is false). Setting this variable to true causes the empty space to be removed during placement, compressing the group as shown in [Figure 12-16](#). The columns are no longer aligned, and utilization is higher.

Effect of Ungrouping on Hierarchical Relative Placement

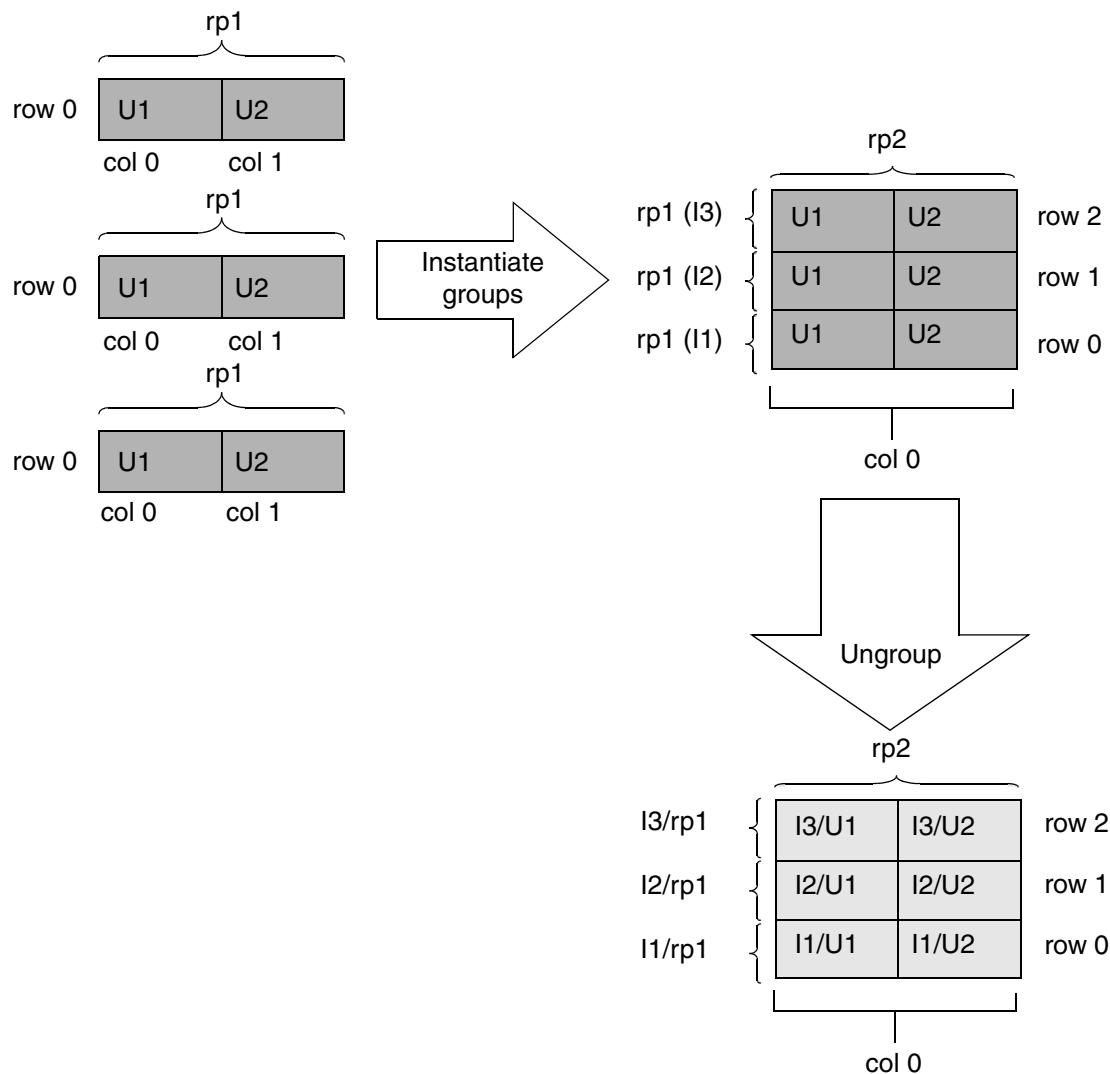
The `ungroup` command changes the hierarchical relative placement structure. When you ungroup the current design, use `ungroup -flatten -all`, to ensure that the relative placement gets flattened properly.

After you ungroup, instantiated relative placement groups are converted to included relative placement groups, because the design is flattened and all the groups are now of the same design. Instantiation of hierarchical modules no longer exists.

Relative placement groups affected by an `ungroup` command are renamed to show the path to the group before flattening, followed by a slash (/) and the original group name. If this results in a name collision, a numbered suffix is added to create a unique name.

For example, [Figure 12-17](#) shows a hierarchical relative placement group that contains multiple instantiations of group `rp1` before and after ungrouping. Before ungrouping, the relative placement definition is as shown in [Example 12-11](#). After ungrouping, the relative placement definition is as shown in [Example 12-12](#).

Figure 12-17 Instantiating Groups in a Hierarchical Group



Example 12-11 Instantiated Groups Before Ungrouping

```
create_rp_group rp1 -design pair_design -columns 2 -rows 1
add_to_rp_group pair_design::rp1 -leaf U1 -column 0 -row 0
add_to_rp_group pair_design::rp1 -leaf U2 -column 1 -row 0

create_rp_group rp2 -design mid_design -columns 1 -rows 3
add_to_rp_group mid_design::rp2 -hierarchy pair_design::rp1 -instance I1 \
-column 0 -row 0
add_to_rp_group mid_design::rp2 -hierarchy pair_design::rp1 -instance I2 \
-column 0 -row 1
add_to_rp_group mid_design::rp2 -hierarchy pair_design::rp1 -instance I3 \
-column 0 -row 2
```

Example 12-12 Instantiated Groups After Ungrouping

```
create_rp_group I1/rp1 -design mid_design -columns 2 -rows 1
  add_to_rp_group mid_design::I1/rp1 -leaf I1/U1 -column 0 -row 0
  add_to_rp_group mid_design::I1/rp1 -leaf I1/U2 -column 1 -row 0

create_rp_group I2/rp1 -design mid_design -columns 2 -rows 1
  add_to_rp_group mid_design::I2/rp1 -leaf I2/U1 -column 0 -row 0
  add_to_rp_group mid_design::I2/rp1 -leaf I2/U2 -column 1 -row 0

create_rp_group I3/rp1 -design mid_design -columns 2 -rows 1
  add_to_rp_group mid_design::I3/rp1 -leaf I3/U1 -column 0 -row 0
  add_to_rp_group mid_design::I3/rp1 -leaf I3/U2 -column 1 -row 0

create_rp_group rp2 -design mid_design -columns 1 -rows 3
  add_to_rp_group mid_design::rp2 -hierarchy mid_design::I1/rp1 -column 0 -row 0
  add_to_rp_group mid_design::rp2 -hierarchy mid_design::I2/rp1 -column 0 -row 1
  add_to_rp_group mid_design::rp2 -hierarchy mid_design::I3/rp1 -column 0 -row 2
```

Effect of Uniquifying on Hierarchical Relative Placement

The `uniquify` command can change each instantiation of hierarchical relative placement structures.

For example, uniquifying

```
create_rp_group grp_ripple -design ripple
...
create_rp_group grp_top -design top -columns 1 -rows 2
  add_to_rp_group top::grp_top -hierarchy ripple::grp_ripple -instance u1 \
    -column 0 -row 0
  add_to_rp_group top::grp_top -hierarchy ripple::grp_ripple -instance u2 \
    -column 0 -row 1
```

results in

```
create_rp_group grp_ripple_1 -design ripple_1
...
create_rp_group grp_ripple_2 -design ripple_2
...
create_rp_group grp_top -design top -columns 1 -rows 2
  add_to_rp_group top::grp_top -hierarchy ripple_1::grp_ripple_1 \
    -instance u1 -column 0 -row 0
  add_to_rp_group top::grp_top -hierarchy ripple_2::grp_ripple_2 \
    -instance u2 -column 0 -row 1
```

Adding Keepouts

You can add soft or hard keepouts within relative placement groups either in the relative placement hierarchy browser or from the command line.

Both hard and soft keepouts prevent the placement of relative placement cells in that location during both placement and legalization. The differences between the two keepout types are

- The handling of non-relative-placement cells during legalization

During legalization, non-relative-placement cells can be placed in soft keepouts, but not in hard keepouts.

- The handling of the keepout width and height during optimization

For hard keepouts, the width and height do not change as a result of optimization performed on the relative placement group (however, its location might change). Upsizing of relative placement cells can cause an increase in column width.

For soft keepouts, the width and height might change as a result of optimization performed on the relative placement group. The column width is maintained, even during upsizing of relative placement cells.

- The ability to place keepouts over tap cells

Soft keepouts can be placed over tap cells.

By default, hard keepouts cannot be placed over tap cells.

To allow placement of hard keepouts over tap cells for a relative placement group, use the `-allow_keepout_over_tap true` option of the `create_rp_group` or `set_rp_group_options` command.

When you specify keepouts, keep the following points in mind:

- You can specify whether the keepout is a hard or soft keepout.

If you do not specify the keepout type, IC Compiler creates a hard keepout.

- You can specify the width and height of a keepout.

- The unit of width for a keepout is the number of placement sites. If you do not specify the width, the default width is the width of the widest cell in that column.
- The unit of height for a keepout is one row. If you do not specify the height, the default height is the height of the tallest cell in that row.

- You can add a cell and a keepout at the same position in the relative placement group.

When a cell and a keepout occupy the same position, the cell is placed first and the keepout is placed to the right of the cell.

To add keepouts in the relative placement hierarchy browser (Placement > New RP Hierarchy View),

1. Select the group you want to add to in the left pane.
2. Select the Grid tab in the right pane.
3. Select a location in the grid view.

Keepouts can be added in open or occupied locations.

4. Right-click and choose Add Object.

The Add Objects to RP dialog box appears.

5. Specify the keepout name, width, height, and type.
6. Click OK.

Alternatively, you can use the `add_to_rp_group` command. The syntax for adding a keepout by using `add_to_rp_group` is

```
add_to_rp_group group_list -keepout keepout_name
                [-column column_number] [-row row_number]
                [-width value] [-height value] [-type hard | soft]
```

For example, to create a hard keepout named gap1, enter

```
icc_shell> add_to_rp_group TOP::misc -keepout gap1 \
            -column 0 -row 2 -width 15 -height 1
```

Movement Control When Legalizing Relative Placement Groups

You can control the movement of the relative placement group during legalization by using the `-move_effort` option. Coarse placement estimates an initial location for every top-level relative placement group. The `-move_effort` option controls the extent to which a relative placement group can be moved from its initial location to preserve the relative placement without violating relative placement constraints. The size of the region searched for placement of a relative placement group is progressively reduced as you reduce the effort from high to medium to low.

For example, to place relative placement groups close to the initial location returned by the coarse placement, enter

```
icc_shell> create_rp_group grp_ripple -design ripple -move_effort low
```

The `set_rp_groups_options` command also supports the `-move_effort` option.

For more information about the `-move_effort` command, see the man page.

Placement of Relative Placement Groups

During placement and legalization, the structure of the relative placement groups is preserved and the cells in each group are placed as a single entity. The following sections describe these aspects of relative placement:

- [Preserving Relative Placement Cells During Optimization](#)
- [Performing Relative Placement in a Design Containing Obstructions](#)
- [Performing Relative Placement in a Design Containing Tap Cells](#)
- [Using Move Bounds to Constrain Relative Placement](#)
- [Adding Incremental Relative Placement Groups](#)

Preserving Relative Placement Cells During Optimization

During optimization, relative placement cells might be optimized away, thereby disturbing the relative placement groups. To preserve the attributes, you can specify `fixed_placement` or `size_only`. The `fixed_placement` attribute keeps the relative placement structures fixed. In comparison, the `size_only` attribute prevents relative placement cells from being removed, but it allows sizing during optimization.

For example, to set the `size_only` attribute on all relative placement cells during optimization, enter

```
icc_shell> set_rp_group_options -psynopt_option size_only [get_rp_groups *]
```

If relative placement cells are upsized or downsized, the relative placement cell alignment is maintained for placement.

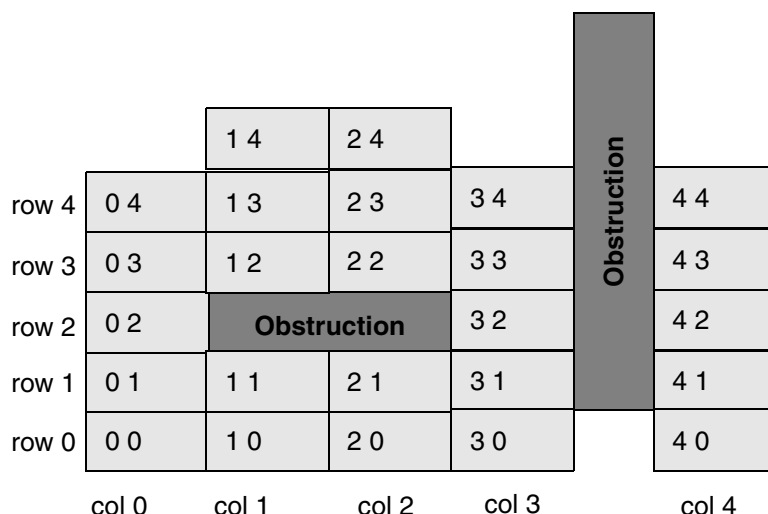
Performing Relative Placement in a Design Containing Obstructions

During placement, relative placement groups avoid placement blockages (obstructions) that are defined in the DEF file or created by the `create_placement_blockage` command. A relative placement group can be broken into pieces that straddle obstructions, yet maintain the relative placement structure.

If the height of the obstruction is below a certain threshold, the relative placement cells are shifted vertically; otherwise, the relative placement cells are shifted horizontally.

Figure 12-18 shows the placement of relative placement cells in a design containing obstructions that were either defined in the DEF file or created by `create_placement_blockage`. The obstruction in columns one and two is below the threshold, so IC Compiler shifts the cells vertically. The obstruction in column four is greater than the threshold, so IC Compiler shifts the cells horizontally.

Figure 12-18 Relative Placement in a Design Containing Obstructions



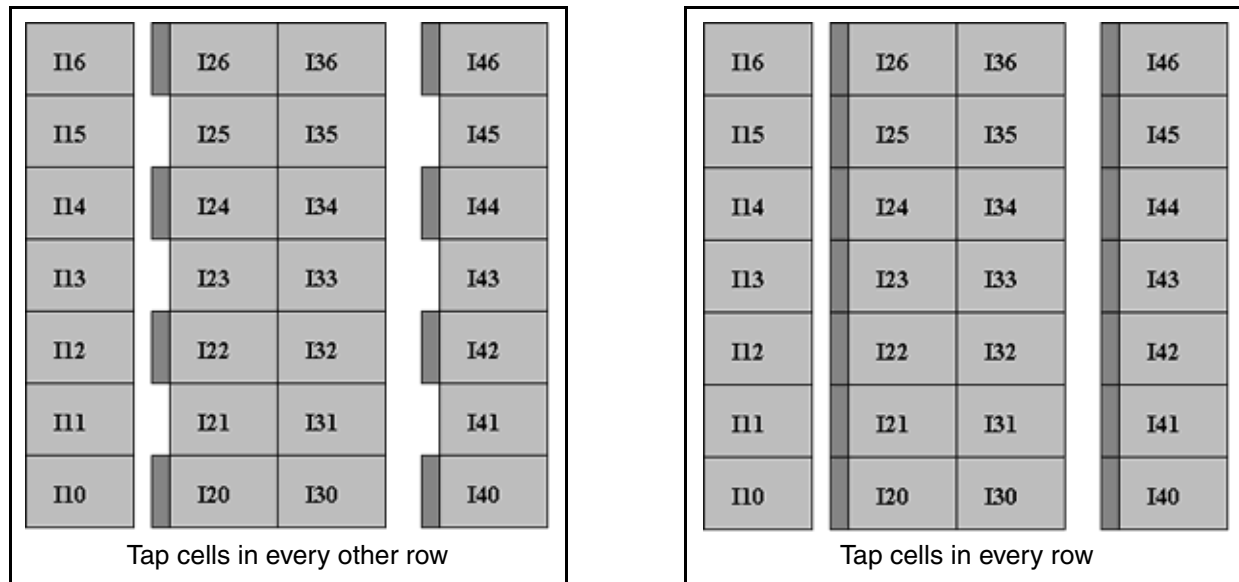
Performing Relative Placement in a Design Containing Tap Cells

Tap cells are special nonlogic cells with well and substrate ties. Tap cells can be defined in the DEF file or created by a tap cell command such as `add_tap_cell_array`. For more information about tap cells, see [Chapter 9, “Chip Finishing and Design for Manufacturing.”](#)

During placement, the cells or columns of a relative placement group are automatically shifted to avoid tap cells while maintaining the relative positions of the cells in the relative placement group.

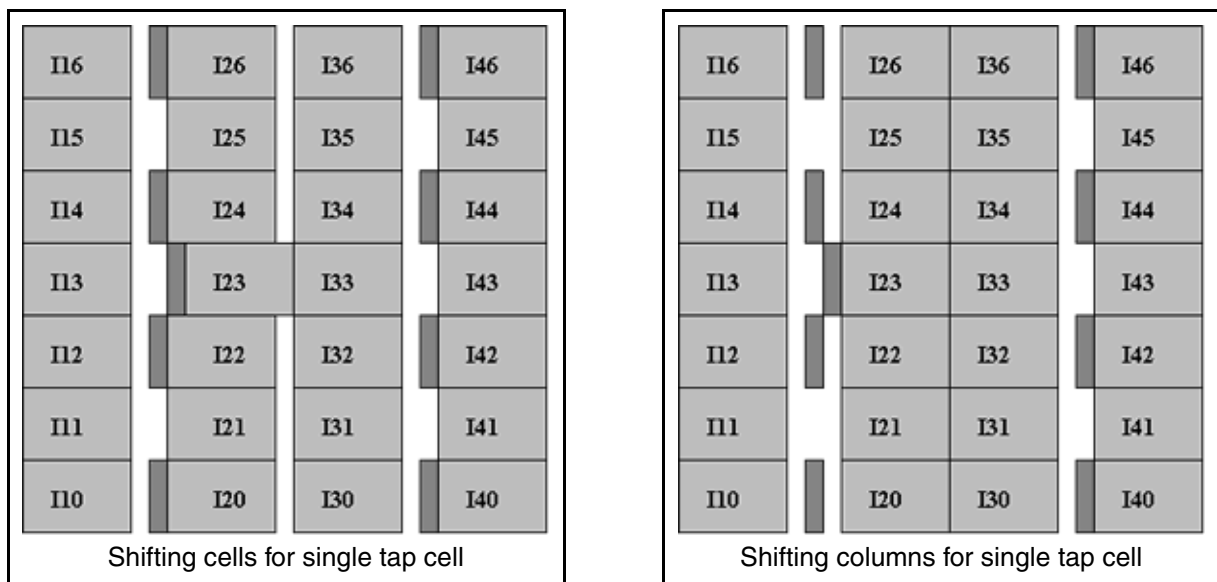
[Figure 12-19](#) shows how placement of a relative placement group is affected by a tap cell array. The design on the left contains tap cells in every other row, and the design on the right contains tap cells in every row. In both cases, the columns of the relative placement group are shifted to avoid the tap cells but the alignment of the relative placement group is maintained.

Figure 12-19 Shifting Columns to Avoid Tap Cell Array



If your design contains a single tap cell in a column, rather than a tap cell array, the default placement behavior is to shift individual cells of the relative placement group to avoid the tap cells (as shown in the design on the left in [Figure 12-20](#)). To shift the entire column instead of a single cell, set the `rp_shift_column_for_fixed_cells` variable to true. This results in the placement shown in the design on the right in [Figure 12-20](#).

Figure 12-20 Avoiding Single Tap Cells



Using Move Bounds to Constrain Relative Placement

You can constrain the placement of relative placement cells by defining move bounds with fixed coordinates (relative placement does not support group bounds). Both soft bounds and hard bounds are supported for relative placement cells, and both rectangular bounds and rectilinear bounds are supported.

Caution!

If you use move bounds to constrain relative placement, all cells in a relative placement group must be in the same move bound; otherwise the resulting QoR might not be acceptable.

To constrain relative placement by using move bounds, use the `create_bounds` command and specify the individual cell names as provided in an `add_to_rp_group` command as the command argument. You cannot specify a relative placement group as the command argument.

For example, enter

```
icc_shell> create_bounds -coordinates {100 100 200 200} U1 U2 U3 U4
```

For details about the `create_bounds` command, see the man page.

Adding Incremental Relative Placement Groups

You can create relative placement groups of cells in an already placed design and place the relative placement groups incrementally where the `refine_placement` command does not disturb the initial placement significantly. Note that if you select cells that are placed far apart in the initial placement for the same relative placement group, performing incremental relative placement might degrade the QoR.

The following script shows an example of how to add incremental relative placement groups:

```
icc_shell> create_placement
icc_shell> legalize_placement
icc_shell> create_rp_group new_rp -design design1 -columns 1 -rows 4
icc_shell> add_to_rp_group design1::new_rp -leaf U1 -column 0 -row 0
icc_shell> add_to_rp_group design1::new_rp -leaf U2 -column 0 -row 1
...
icc_shell> refine_placement
```

For more information about the `refine_placement` command, see [“Refining Placement” on page 4-42](#).

Postplacement Optimization of Relative Placement Groups

During postplacement optimization, relative placement cells can be modified, moved, or removed. When a relative placement cell is modified or moved, the relative placement structure can be disturbed. When a relative placement cell is removed during optimization, the relative placement information attached to the instance is also removed, disrupting the relative placement structure.

To preserve the relative placement structures during various postplacement optimization processes, use the `create_rp_group` or `set_rp_group_options` command to specify the `fixed_placement` keyword with the appropriate option (as shown in Table 12-2). To prevent relative placement cells from being removed (but allow sizing) during optimization, set the `size_only` keyword with the appropriate option.

These options mark the relative placement cells as `fixed_placement`, `size_only`, or `in_place_size_only` during the affected commands.

Table 12-2 Options to Preserve Relative Placement Structures

Process	Option	Affected commands
Clock tree synthesis	-cts_option	clock_opt compile_clock_tree optimize_clock_tree Note: The <code>compile_clock_tree</code> and <code>optimize_clock_tree</code> commands do not support the <code>size_only</code> keyword.
Physical optimization	-psynopt_option	psynopt
Routing	-route_opt_option	route_opt psynopt -on_route

For clock tree synthesis and physical optimization, you specify the preservation attribute by using `fixed_placement` or `size_only`. For routing, you specify the preservation attribute by using `fixed_placement` or `in_place_size_only`.

To size the relative placement cells during clock tree synthesis and physical optimization, enter

```
icc_shell> set_rp_group_options -cts_option size_only  
-psynopt_option size_only [get_rp_groups *]
```

To size the relative placement cells during postroute optimization, enter

```
icc_shell> set_rp_group_options -route_opt_option in_place_size_only \  
[get_rp_groups *]
```

Analyzing the Relative Placement Results

The following sections explain methods for analyzing your relative placement results:

- [Checking Relative Placement Constraints](#)
- [Locating a Relative Placement Group](#)

Checking Relative Placement Constraints

To check whether the relative placement constraints have been met, run the `check_rp_groups` command.

The `check_rp_groups` command checks for the following failures:

- One or more cells in the relative placement group cannot be legally placed.
- The relative placement group cannot be placed as a whole.
- The height or width of the relative placement group is greater than the height or width of the core area.
- The user-specified orientation cannot be met.

If a failure prevents the group from being placed as a single entity (as defined by the relative placement constraints), the failure is considered a critical failure. If the failure does not prevent placement but causes the relative placement constraints to be violated, the failure is considered noncritical. The generated report contains separate sections for critical and noncritical failures.

The following failures can also occur, but `check_rp_groups` does not check for them:

- The keepouts are not created correctly in the relative placement group.
- Violations occur during bit-stacked placement.

You can check all relative placement groups (by specifying the `-all` option) or you can specify which relative placement groups to check. By default, the report is output to the screen. To save the generated report to a file, specify the file name by using the `-output` option.

For example, to check the relative placement constraints for groups compare17::seg7 and compare17::rp_group3 and save the output in a file called rp_failures.log, run the following command:

```
icc_shell> check_rp_groups "compare17::seg7 compare17::rp_group3" \
            -output rp_failures.log
```

The generated report is similar to this:

```
*****
Report : The RP groups, which could not be placed.
Version: 2005.12
Date   : Tue Dec 27 16:32:02 2005
No. of RP groups:1
*****
RP GROUP: compare17::seg7
-----
ERROR: Could not get clean area for placing RP group compare17::seg7.

*****
Report : The RP groups, not meeting all constraints but placed.
Version: 2005.12
Date   : Tue Dec 27 16:32:02 2005
No. of RP groups:1
*****
RP GROUP: compare17::rp_group_3
-----
WARNING: Could not set user specified orientation for cell u[8].
```

You can also check the relative placement constraints in the GUI. To do so, choose Placement > Check RP Groups. When you check the constraints in the GUI, all relative placement groups are checked. In the generated report, critical failures are indicated by an exclamation mark (!). When you select reported violations, the location of the violation is displayed in the layout view.

Locating a Relative Placement Group

You use the `get_location` command with its `-rp_group` option to get the grid location of a relative placement cell or group.

The syntax is

```
get_location cell_or_group_list
            -rp_group containing_rp_group
```

These commands return a list of the x- and y-values of each object specified within the containing relative placement group you specify.

If a group is instantiated multiple times within the group that contains it, a sublist of all locations is returned within the returned list of locations for each object.

If a group does not contain any of the objects, a message appears.

For example, to get the location of cell U11 in relative placement group my_group, enter the following commands:

```
icc_shell> set cell_rp_location \  
[get_location [get_cell U11]-rp_group my_group]  
icc_shell> set x_rp_loc [lindex $cell_rp_location 0]  
icc_shell> set y_rp_loc [lindex $cell_rp_location 1]
```

Working With Relative Placement Groups in the GUI

The GUI provides two methods of viewing the relative placement groups:

- Using the relative placement hierarchy browser
- Using the layout view

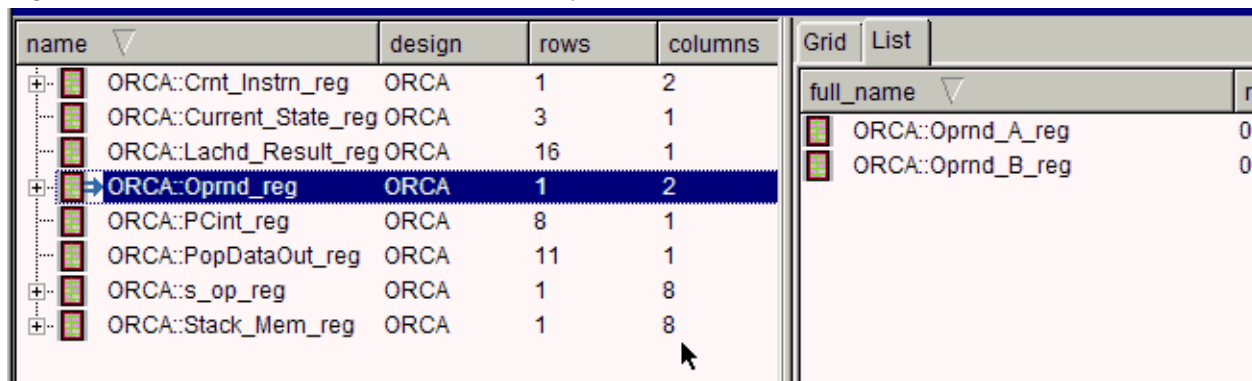
Using the Relative Placement Hierarchy Browser

The relative placement hierarchy browser displays information about the relative placement groups in the current design.

To open the relative placement hierarchy browser, choose Placement > New RP Hierarchy View.

As shown in [Figure 12-21](#), the left pane of the relative placement hierarchy browser lists the relative placement groups in the current design. When you select a relative placement group in the left pane, its contents are listed in the right pane. When you select a group in the relative placement hierarchy browser (in either the right or left pane), it is highlighted in the layout view.

Figure 12-21 Relative Placement Hierarchy Browser



Using the Layout View

Relative placement groups are visible and selectable in the layout view. In addition, the layout view provides a relative placement visual mode for displaying the relative placement groups.

To use relative placement visual mode,

1. Invoke relative placement visual mode by choosing Placement > Color By RP Groups.

The Visual Mode panel appears. The relative placement groups are listed by number, along with the number of cells in each group. If the design or the relative placement constraints have changed since you last invoked relative placement visual mode, this information might be out-of-date.

2. Update the relative placement information by clicking Reload in the Visual Mode panel.

The Update Relative Placement Visual Mode dialog box appears.

3. Select “Color by RP groups” in the Update Relative Placement Visual Mode dialog box and click OK.

Each relative placement group is displayed in a different color in the layout view. If the design has not been placed, you will not see the relative placement groups in the layout view but the information in the visual mode panel will be correct.

Querying Relative Placement Groups

Table 12-3 lists the commands available for querying particular types of relative placement groups to annotate, edit, and view. For detailed information, see the man pages.

Table 12-3 Commands for Querying Relative Placement Groups

To return collection handles for this	Use this
Relative placement groups that match certain criteria	<code>get_rp_groups</code>
All or specified relative placement groups and the included and instantiated groups they contain in their hierarchies	<code>all_rp_groups</code>
All or specified relative placement groups that contain included or instantiated groups	<code>all_rp_hierarchicals</code>
All or specified relative placement groups that contain included groups	<code>all_rp_inclusions</code>

Table 12-3 Commands for Querying Relative Placement Groups (Continued)

To return collection handles for this	Use this
All or specified relative placement groups that contain instantiated groups	<code>all_rp_instantiations</code>
All or specified relative placement groups that directly embed leaf cells (added to a group by <code>add_rp_group -leaf</code>) or instantiated groups (added to a group by <code>add_rp_group -hierarchy -instance</code>) in all or specified relative placement groups in a specified design or the current design	<code>all_rp_references</code>
Directly embedded included groups (added to a group by <code>add_rp_group -hierarchy</code>) in all or specified groups	<code>rp_group_inclusions</code>
Directly embedded instantiated groups (added to a group by <code>add_rp_group -hierarchy -instance</code>) in all or specified groups	<code>rp_group_instantiations</code>
Directly embedded leaf cells (added to a group by <code>add_rp_group -leaf</code>), directly embedded included cells that contain hierarchically instantiated cells (added to the included group by <code>add_rp_group -hierarchy -instance</code>), or both in all or specified relative placement groups	<code>rp_group_references</code>
Create a collection of the specified relative placement group keepouts	<code>get_rp_group_keepouts</code>

By default, a maximum of 100 objects is displayed when you use one of these commands at the command prompt. To change this maximum, set the `collection_result_display_limit` variable.

For example, to create a collection of relative placement groups that start with the letter *g* in a design that starts with the letter *r*, enter

```
icc_shell> get_rp_groups r*::g*
{ripple::grp_ripple}
```

To set the utilization to 95 percent for all relative placement groups, enter

```
icc_shell> set_rp_group_options [all_rp_groups] -utilization 0.95
```

Saving Relative Placement Information

The relative placement information is automatically saved in the Milkyway design database when you save the design in Milkyway format (`save_mw_cel` command).

You can also save the relative placement information to a file, creating a Tcl script for re-creating relative placement groups and their objects on the same design. To do this, use the `write_rp_groups -output script` command. You must either specify which relative placement groups to write commands for or specify `-all` (to write commands for all relative placement groups).

For example, to save all the relative placement groups to disk, remove the information from the design, and then re-create the information on the design, enter

```
icc_shell> get_rp_groups
{mul::grp_mul ripple::grp_ripple example3::top_group}
icc_shell> write_rp_groups -all -output my_groups.tcl
1
icc_shell> remove_rp_groups -all -quiet
1
icc_shell> get_rp_groups
Error: Can't find objects matching '*'. (UID-109)
icc_shell> source my_groups.tcl
{example3::top_group}
icc_shell> get_rp_groups
{example3::top_group ripple::grp_ripple mul::grp_mul}
```

By default, the `write_rp_groups` command writes out commands for creating the specified relative placement groups and to add leaf cells, hierarchical groups, and keepouts to these groups. The commands for generating subgroups within hierarchical groups are not written. You can modify the default behavior by using the options described in [Table 12-4](#).

Table 12-4 write_rp_groups Options

To do this	Use this
Write all the relative placement groups within the hierarchy of the relative placement groups. If you omit this option, subgroups are not written.	<code>-hierarchy</code>
Write only <code>create_rp_group</code> commands to the script.	<code>-create</code>
Write only <code>add_to_rp_group -leaf</code> commands to the script.	<code>-leaf</code>
Write only <code>create_rp_group -keepout</code> commands to the script.	<code>-keepout</code>

Table 12-4 *write_rp_groups Options (Continued)*

To do this	Use this
Write only <code>create_rp_group -hierarchy -instance</code> commands to the script.	<code>-instance</code>
Write only <code>create_rp_group -hierarchy</code> commands to the script.	<code>-include</code>

Ignoring Relative Placement Constraints

You can direct the tool to ignore relative placement constraints annotated to the design, for example, when you want to confirm that relative placement is helpful to the QoR. You use the `-ignore` option with either the `create_rp_group` or `set_rp_group_options` command to skip structured placement for relative placement groups you specify.

When you direct the tool to ignore relative placement constraints, it places the parts of the relative placement groups as if the group had no relative placement information.

To remove the ignore attribute and cause the tool to consider the relative placement constraints, use the `remove_rp_group_options -ignore` command.

Removing Relative Placement Groups

You can remove relative placement groups, either in the relative placement hierarchy browser or from the command line.

To remove relative placement groups in the relative placement hierarchy browser,

1. Select the groups you want to remove in the left pane.
2. Right-click and choose Delete Selected RP Groups from the pop-up menu.

For more information about the relative placement hierarchy browser, see [“Using the Relative Placement Hierarchy Browser” on page 12-42](#).

To remove relative placement groups from the command line, run the `remove_rp_groups` command. You can remove all relative placement groups (by specifying the `-all` option), or you can specify which relative placement groups to remove. If you specify a list of relative placement groups, only the specified groups (and not groups included or instantiated within the specified group) are removed. To remove the included and instantiated groups of the specified groups, you must specify `-hierarchy`.

Note:

When you remove a relative placement group, the memory it occupies is freed to be reused by this same process. However, memory is not returned to the operating system until you exit `icc_shell`.

For example, to remove the relative placement group named `grp_ripple` and confirm its removal, enter

```
icc_shell> get_rp_groups
{mul::grp_mul ripple::grp_ripple example3::top_group}
icc_shell> remove_rp_groups ripple::grp_ripple
Removing rp group 'ripple::grp_ripple'
1
icc_shell> get_rp_groups *grp_ripple
Error: Can't find object 'grp_ripple'. (UID-109)
icc_shell> remove_rp_groups -all
Removing rp group 'mul::grp_mul'
Removing rp group 'example3::top_group'
1
```

Changing Relative Placement Information

You can change a relative placement group. For example, you can

- Change the attributes of a relative placement group (anchor location, alignment, utilization, preservation during clock tree synthesis, and preservation during routing)
- Rename a group
- Remove objects from a group
- Change objects within a group

Reporting the Attributes of a Relative Placement Group

To report the attributes of one or more relative placement groups (as set by the `create_rp_group` or `set_rp_group_options` command), use the `report_rp_group_options` command.

```
icc_shell> report_rp_group_options rp_groups
```

You can also use the `report_attribute` command to report the attributes of all relative placement groups in your design.

```
icc_shell> report_attribute -application -class rp_group rp_group
```

In addition to the attributes reported by the `report_rp_group_options` command, the `report_attribute` command also lists the number of rows and columns of each relative placement group.

Changing the Attributes of a Relative Placement Group

You can change the attributes of an existing group in either of the following ways:

- Set the attribute values.
- Reset the attributes to their default values (by using the `remove_rp_group_options` command).

Setting Relative Placement Group Attributes

You can set relative placement group attributes either in the relative placement hierarchy browser or from the command line.

To set relative placement group attributes in the relative placement hierarchy browser,

1. Select the groups you want to modify in either the left or right pane (relative placement groups are identified by the  icon).
2. Right-click and choose “Set RP Options.”

The “Set RP Options” dialog box appears.

3. Update the attributes, as described in [Table 12-5](#).

For more information about the relative placement hierarchy browser, see [“Using the Relative Placement Hierarchy Browser” on page 12-42](#).

To set relative placement group attributes from the command line, run the `set_rp_group_options` command. You must specify the group name and at least one option; otherwise, this command has no effect.

For example, to set the utilization to 80 percent for the `block1::misc1` group, enter

```
icc_shell> set_rp_group_options block1::misc1 -utilization .80
{block1::misc1}
```

The command returns a collection of relative placement groups for which attributes have been changed. If no attributes change for any object, an empty string is returned.

[Table 12-5](#) describes the relative placement group attributes and the Set RP Options dialog box fields or command options used to change them.

Table 12-5 Relative Placement Group Attributes

Attribute	Dialog box fields	Command line options
Alignment method See “Aligning Leaf Cells Within a Column” on page 12-15.	Alignment	-alignment
Group alignment pin See “Aligning Leaf Cells Within a Column” on page 12-15.	Pin alignment	-pin_align_name
Anchor location See “Anchoring Relative Placement Groups” on page 12-11.	X offset Y offset	-x_offset -y_offset
Utilization percentage	Utilization	-utilization
Routing preservation You can specify <code>fixed_placement</code> or <code>in_place_size_only</code> . See “Postplacement Optimization of Relative Placement Groups” on page 12-39.	Route_opt Option	-route_opt_option
Clock tree synthesis preservation You can specify <code>fixed_placement</code> or <code>size_only</code> . See “Postplacement Optimization of Relative Placement Groups” on page 12-39.	CTS option	-cts_option
Physical optimization preservation You can specify <code>fixed_placement</code> or <code>size_only</code> . See “Postplacement Optimization of Relative Placement Groups” on page 12-39.	Psynopt option	-psynopt_option
Legalizing relative placement groups. You can specify low, medium, or high. See “Movement Control When Legalizing Relative Placement Groups” on page 12-34.	move_effort	-move_effort

Table 12-5 Relative Placement Group Attributes (Continued)

Attribute	Dialog box fields	Command line options
Ignore relative placement constraints See “Ignoring Relative Placement Constraints” on page 12-46 .	Ignore placement	-ignore
Allow keepouts over tap cells See “Adding Keepouts” on page 12-32 .	Allow keepout over tapcell	-rp_allow_keepout_ over_tapcell

Removing Relative Placement Group Attributes

You can remove relative placement group attributes either in the relative placement hierarchy browser or from the command line.

To remove relative placement group attributes in the relative placement hierarchy browser (Placement > New RP Hierarchy View),

1. Select the groups you want to modify in either the left or right pane (relative placement groups are identified by the  icon).
2. Right-click and choose “Set RP Options.”

The “Set RP Options” dialog box appears.

3. Remove the attributes by setting them to empty values.

To remove relative placement group attributes from the command line, run the `set_rp_group_options` command. You must specify the group name and at least one option; otherwise, this command has no effect.

For example, to remove the `x_offset` attribute for the `block1::misc1` group, enter

```
icc_shell> remove_rp_group_options block1::misc1 \
-x_offset
{block1::misc1}
```

The command returns a collection of relative placement groups for which attributes have been changed. If no attributes change for any object, an empty string is returned.

[Table 12-6](#) describes changes you can make with the `remove_rp_group_options` command.

Table 12-6 remove_rp_group_options Options

To do this	Use this option
Reset the anchor location. Removing an offset attribute resets that coordinate to 0. See “Anchoring Relative Placement Groups” on page 12-11 .	<code>-x_offset</code> <code>-y_offset</code>
Remove the Ignore setting for this relative placement group. See “Ignoring Relative Placement Constraints” on page 12-46 .	<code>-ignore</code>

Renaming a Group

You cannot rename a group directly. If you need to rename a group, you must remove it, and then create a new group that duplicates the removed group but has the new name.

Removing Objects From a Group

You can remove objects from a relative placement group, either in the relative placement hierarchy browser or from the command line.

To remove objects from a relative placement group in the relative placement hierarchy browser (Placement > New RP Hierarchy View),

1. Select the objects you want to remove in the right pane (relative placement groups are identified by the  icon, leaf cells are identified by the  icon, keepouts are identified by the  icon, and locations containing both a leaf cell and a keepout are identified by the  icon).
2. Invoke the pop-up menu by right-clicking.
3. Right-click and choose “Remove Selected Cells and RP Groups” or “Remove Selected Keepouts”.

For more information about the relative placement hierarchy browser, see [“Using the Relative Placement Hierarchy Browser” on page 12-42](#).

To remove objects from a relative placement group from the command line, use the `remove_from_rp_group` command. You can remove leaf cells (`-leaf`), included groups (`-hierarchy`), instantiated groups (`-hierarchy -instance`), and keepouts (`-keepout`).

When you remove objects from a group, the space previously occupied by the removed objects is left unoccupied.

For example, to remove leaf cell `carry_in_1` from `grp_ripple`, enter

```
icc_shell> remove_from_rp_group ripple::grp_ripple -leaf carry_in_1  
1
```

Changing Specific Objects Within a Group

You can change a specific cell within a group. For example, you can move a cell from one position to another or change its alignment pin name or orientation.

To change an object within a group,

1. Remove the object from the group by using the `remove_from_rp_group` command.
2. Reinsert the object by using the appropriate `add_to_rp_group` command.

Deriving Relative Placement Groups

You can derive relative placement groups from your design by either specifying the cells contained in the group or specifying the physical location of the group.

Note:

IC Compiler does not derive hierarchical relative placement groups. Derived relative placement groups are always flat.

You can use this capability to migrate and reuse structured netlists.

To derive relative placement groups from your design,

- Use the `extract_rp_group` command, which generates a script containing `create_rp_group` commands for the extracted groups.
- Use the `order_rp_groups` command to order the extracted groups to create a linear placement of groups.

This capability does not generate all possible relative placement groups but can serve as a starting point in many cases. You can refine the resulting groups by editing the script generated by `extract_rp_group`.

Extracting a Relative Placement Group

The `extract_rp_group` command helps you create relative placement structures at an abstract level. You can extract a group by either specifying the cells in the group (with the `-objects` option) or specifying a bounding box from which to extract the group (with the `-coordinates` option).

The `extract_rp_group` command annotates the extracted relative placement group to the design and generates the `create_rp_group` and `add_to_rp_group` commands to define the relative placement group. By default, these commands are written to the screen. To write the commands to a script file, use the `-output` option. To append the commands to an existing script file, specify the `-append` option in addition to the `-output` option. You can refine the extracted group by editing the script file.

By default, IC Compiler creates a single column with one row for each object in the group. To change the default behavior, specify the desired number of rows (with the `-rows` option) and columns (with the `-columns` option). Each column is described as a new relative placement group that, initially, is randomly ordered. You can specify an order by using the `order_rp_groups` command.

For example, to create a relative placement group for a bank of registers whose names start with `Crnt_Instrn_1_reg`, annotate the group to the design, and save the script to a file called `rp.tcl`, enter

```
icc_shell> extract_rp_group \  
-group_name Crnt_Instrn_1_reg \  
-objects [get_cells -hierarchical Crnt_Instrn_1_reg*] \  
-output rp.tcl
```

To create a relative placement group containing the cells within the bounding box {0 0 100 100} and annotate the group to the design, enter

```
icc_shell> extract_rp_group -group_name ph1 \  
-physical -coordinates { 0 0 100 100 }
```

Alternatively, you can use the GUI to extract a relative placement group by choosing Placement > Extract RP Group.

Ordering Extracted Relative Placement Groups

You can use the `order_rp_groups` command to create a linear placement of the extracted groups.

The `order_rp_group` command generates the `create_rp_group` and `add_to_rp_group` commands to define the hierarchical relative placement group. By default, these commands are written to the screen. To write the commands to a script file, use the `-output` option. To append the commands to an existing script file, specify the `-append` option in addition to the `-output` option. You can refine the group by editing the script file.

For example, to define a hierarchical relative placement group for a related set of register banks, enter

```
icc_shell> extract_rp_group \  
-group_name Crnt_Instrn_1_reg \  
-objects [get_cells -hierarchical Crnt_Instrn_1_reg*] \  
-output Crnt_Instrn.tcl  
icc_shell> extract_rp_group \  
-group_name Crnt_Instrn_2_reg \  
-objects [get_cells -hierarchical Crnt_Instrn_2_reg*] \  
-output Crnt_Instrn.tcl -append  
icc_shell> order_rp_groups \  
-group_name Crnt_Instrn_reg \  
{ORCA::Crnt_Instrn_1_reg ORCA::Crnt_Instrn_2_reg} \  
-output Crnt_Instrn.tcl -append
```

Summary of Relative Placement Commands

[Table 12-7](#) shows some of the key commands used to perform relative placement.

Table 12-7 Relative Placement Commands

Command	Described in section
<code>create_rp_group</code>	See “Creating Relative Placement Groups” on page 12-9.
<code>add_to_rp_group</code>	See “Adding Objects to a Group” on page 12-13.
<code>check_rp_groups</code>	See “Checking Relative Placement Constraints” on page 12-40.
<code>write_rp_groups</code>	See “Saving Relative Placement Information” on page 12-45.
<code>remove_rp_groups</code>	See “Removing Relative Placement Groups” on page 12-46.
<code>report_rp_group_options</code>	See “Reporting the Attributes of a Relative Placement Group” on page 12-47.
<code>set_rp_group_options</code>	See “Changing the Attributes of a Relative Placement Group” on page 12-48.
<code>remove_rp_group_options</code>	See “Removing Relative Placement Group Attributes” on page 12-50.
<code>remove_from_rp_group</code>	See “Removing Objects From a Group” on page 12-51.
<code>extract_rp_group</code>	See “Extracting a Relative Placement Group” on page 12-53.
<code>order_rp_groups</code>	See “Ordering Extracted Relative Placement Groups” on page 12-54.

13

Multivoltage Design Flow

This chapter describes the multivoltage design flow in IC Compiler.

This chapter contains the following sections:

- [Multivoltage Designs](#)
- [High-Level Design Flow for Multivoltage Designs](#)
- [UPF Flow for Multivoltage Designs](#)
- [Defining Power Domains and Supply Networks](#)
- [Defining and Using Voltage Areas](#)
- [Handling Level Shifters and Isolation Cells](#)
- [Handling Always-On Logic](#)
- [Reporting Commands](#)
- [Sample IC Compiler Script for UPF Flow](#)
- [Voltage-Area-Aware Capabilities](#)
- [Placing and Optimizing the Design](#)
- [Checks and Reporting Functions for Multivoltage Designs](#)
- [Non-UPF Flow For Multivoltage Designs](#)

Multivoltage Designs

IC Compiler supports multivoltage, multisupply, and mixed multivoltage-multisupply designs. In multivoltage designs, the subdesign instances, or blocks, operate at different voltages. In multisupply designs, the block voltages are the same, but the blocks can be powered on and off independently. (In this user guide, unless otherwise noted, the term *multivoltage* includes multisupply and mixed multisupply-multivoltage designs).

You use the Unified Power Format (UPF) to specify the power supply intent for your design. UPF is an industry standard supported by Accellera. UPF supports the ability to specify the power intent for your design early in the design process. IC Compiler supports the use of UPF for the entire design flow. For more details on the low-power flow and various Synopsys tools that support UPF, see the *Synopsys Low-Power Flow User Guide*.

Alternatively, you can also specify the power intent in the RTL file. For more details about specifying power intent in the RTL file, see [“Non-UPF Flow For Multivoltage Designs” on page 13-52](#).

Multivoltage designs typically make use of the following features:

- Power domain – a grouping of logic hierarchies comprising a block
- Voltage area – physical placement area for the cells of a power domain
- Special cells – level shifters and isolation cells used to connect the drive and load pins across different power domains

Power Domains

Power consumption can be reduced in a multivoltage design by making use of power domains that are independently powered up and down including domains that are defined to have always-on relationships relative to each other.

Specifically, a power domain is defined as a grouping of one or more *logic* hierarchies comprising a design block that share the following properties:

- Primary voltage states or voltage range (specifically, the same operating voltage)
- Power net hookup requirements
- Power-down control and acknowledge signals (if any)
- Power switching style
- Process, voltage, and temperature (PVT) operating condition values (all cells of the power domain except level shifters)

- Same set or subset of nonlinear delay model (NLDM) target libraries
- A *physical* voltage area into which the cells of these logic hierarchies are to be placed
- Using a .ddc file
- Writing directly into the Milkyway database
- Writing a Verilog output file and a constraints script file.

Note:

The Verilog file format does not save the power domain information.

Defining Power Domains

You have the following three different ways of defining power domains for your design:

- Define power domains in a UPF file.

You specify all the power information for your design in a separate file, the UPF file. You read this UPF file in IC Compiler, along with your design. For more details, see [“UPF Flow for Multivoltage Designs” on page 13-9](#).

- Define power domains in the RTL design.

If you use Design Compiler for logic synthesis, you can also define power domains in the RTL code and use the `compile_ultra` or the `compile` commands of Design Compiler to synthesize your design. For further information, see [“Non-UPF Flow For Multivoltage Designs” on page 13-52](#)

To save your synthesized design and the power constraints in a file for use in the IC Compiler flow, follow one of the methods mentioned below:

- Write the design in the .ddc file format
- Write the design in the Milkyway database
- Write the design in the Verilog format and the constraints in a separate file. In the Verilog format, the power constraints are not saved. These have to be saved separately in a constraints file.
- Define power domains in IC Compiler

If you use a third-party logic synthesis tool that does not support power domain definitions either in a UPF file or in the RTL code, use the `create_power_domain` command in IC Compiler to define the power domains. You create these power domains in IC Compiler in the same way as you do in Design Compiler. (See the multivoltage chapter in the *Power Compiler User Guide* for details.) Power domains created in IC Compiler are stored in the Milkyway database.

For further information, see [“Non-UPF Flow For Multivoltage Designs” on page 13-52.](#)

Voltage Areas

As previously noted, a voltage area is the physical placement area of a logic block (comprising one or more logic hierarchies) operating under a single voltage level. Except for level-shifter cells, all cells in the hierarchies associated with a voltage area operate at the same process, voltage, and temperature (PVT) operating condition values.

You can create voltage areas in IC Compiler by using the `create_voltage_area` command. Voltage areas must be defined before running the `psynopt` or `place_opt` command.

Note:

If you use the floorplanning method to define voltage areas, the floorplan must contain a default voltage area for the top-level cells and at least one other voltage area. If you directly create the voltage areas by using the `create_voltage_area` command, the tool automatically derives a default voltage area for the top-level leaf cells and level shifters that do not belong to any block.

Physically nested voltage areas are supported, but intersecting voltage areas are not supported. This property of the tool has certain implications for how you decompose intersecting voltage areas into areas that do not intersect. The recommended way of creating physically nested voltage areas is to have one voltage area completely encompassing the other.

For further information, see [“Defining and Using Voltage Areas” on page 13-22](#) and [“Handling Nested Voltage Areas” on page 13-28.](#)

Associating Power Domains and Voltage Areas

There must be an exact one-to-one relationship between logic power domains and physical voltage areas. Design Compiler and IC Compiler can automatically align the logic hierarchies of the power domains with their voltage areas if you provide the corresponding power domain name as an argument in each `create_voltage_area` command. In this case, the power domain name and the voltage area name are identical.

If you do not make these specifications, you are responsible for ensuring that the logic hierarchies are correctly aligned, as well as being correctly associated with the appropriate operating conditions.

Level-Shifter and Isolation Cells

Level-shifter cells operate across voltage differences, connecting drive and load pins of cells belonging to different power domains (and placed in different voltage areas). The basic function of a level shifter is to connect the power domains by stepping the voltage up or down as needed. IC Compiler supports both simple buffer-type level shifters and enable-type level shifters, which are essentially buffer-type level shifters with an additional enable pin. Enable-type level shifters are used when the power domains must be powered on and off independently. A design can contain both types of level shifters.

By using the `insert_level_shifters` command, you can explicitly insert buffer-type level shifters into an unmapped or mapped netlist at any point in the design flow before IC Compiler places the cells of a design. In an *unmapped* design, buffer-type level shifters are automatically inserted as needed when you run the `compile_ultra` or `compile` command. If you intend to explicitly insert level shifters, it is strongly recommended that you run the command as early in the flow as possible.

No command is available that lets you directly insert enable-type level shifters. Usually these level shifters are instantiated at the RTL level or represented by generic library (GTECH) cells, before the unmapped design is compiled. During logic synthesis, existing isolation cells are sometimes replaced with enable-type level shifters. (Since enable-type level shifters are buffers, they can also be used as isolation cells.)

You use isolation cells to isolate the input and output signals of the shut-down power domains. These cells can be instantiated at the RTL level of the design description, or you can run the `insert_isolation_cell` command to insert them directly. (Note that isolation cells are sometimes replaced by enable-type level shifters during logic synthesis.)

IC Compiler places level shifters and isolation cells near the voltage area boundaries, usually at the top level of the design. However, if the port voltages are appropriately specified, these cells can be placed inside the voltage areas. For some designs, there are advantages to placing these cells inside the voltage areas.

For further information, see [“Handling Level Shifters and Isolation Cells” on page 13-31](#).

Basic Library Requirements for Multivoltage Designs

The libraries must conform to the Liberty open library rules. Design Compiler, Automated Chip Synthesis, and IC Compiler support multiple libraries characterized at different voltages.

Target library cells are selected on the basis of matching operating conditions between library cells and voltage area. The selection of these cells can be further restricted by using the `set_target_library_subset` command. (For further information about target library subsets, see [“Target Library Subsetting” on page 13-8.](#))

Note that k-factors are not supported for multivoltage designs and are ignored if present in the libraries.

Physical Synthesis

You use the `place_opt` command in IC Compiler to accomplish the physical synthesis phase of the multivoltage design flow. During this phase, the gate-level designs are read into memory along with the floorplan. The multivoltage design steps are

- (Optional) Insert buffer-type level shifters if they were not inserted during the logic synthesis phase of the flow.
- Transfer the power domain objects into the Milkyway database (Design Compiler to IC Compiler flow) or create power domains and power domain objects in IC Compiler (third-party synthesized design to IC Compiler flow).
- Create voltage areas and associate or align the logic hierarchies, and write the information into the Milkyway database. (Optional) Create power wells within given voltage areas as always-on placement sites for single-power standard cells.
- Run the `place_opt` command to perform cell placement in the corresponding voltage areas of the power domains and optimize the design. This includes always-on optimization of the paths that must receive power when shut-down power domains are powered down.

High-Level Design Flow for Multivoltage Designs

IC Compiler support for multivoltage designs is part of a larger design flow involving several Synopsys tools. Support for multivoltage designs is automatically enabled across the entire Galaxy Design platform.

The Galaxy Design platform tools recognize a design as a multivoltage design and utilize multivoltage capabilities if any of the following conditions are true:

- A UPF command is encountered.
- Power domains are defined.
- Instance-based operating conditions are specified.
- Target library subsetting is used.
- Multiple link libraries are used.
- Multiple target libraries are used.

If none of these conditions hold, the tools assume the design is not a multivoltage design.

Figure 13-1 shows the Galaxy Design platform high-level flow for multivoltage designs. This is a complete RTL-to-sign-off design flow. To understand how to use the individual tools of the design flow, consult the appropriate user guides.

Figure 13-1 Galaxy Design Platform High-Level Multivoltage Design Flow

Multiple Libraries	Logic Synthesis (Design Compiler or (Automated Chip Synthesis))	Initial mapping RTL budgeting Apply operating conditions to design parts Bottom-up compile method
	Hierarchical Planning (IC Compiler)	Create physical partition Create voltage areas Voltage area location based on virtual flat Power routing
	Physical Synthesis (IC Compiler)	Create voltage areas as needed Run <code>place_opt</code> Run <code>report_timing -volt</code>
	MV Implementation (IC Compiler)	Run clock tree synthesis Post-clock tree synthesis optimization Autoroute Postroute optimization
	Parasitic Extraction (Star-RCXT)	Parasitic extraction
	Sign-Off (PrimeTime)	Signal integrity timing sign-off Run <code>check_timing -inc signal_level</code>
	Formal Verification (Formality)	RTL versus gates matching

Target Library Subsetting

When you need to restrict the target library cells eligible for optimizing the hierarchical cells of a block, use the `set_target_library_subset` command.

The command syntax is

```
set_target_library_subset {library_list} -object_list {cell_list}
```

where

- *library_list* is a list of target library file names, all of which must also be listed in the `target_library` variable.
- *cell_list* is a list of hierarchical cells (blocks or top level) for which the target library subset is used.

To use this command at the top level, you must include the `-top` option.

Applying this command to a hierarchical cell or to the top-level design enforces this library restriction on all lower cells in the hierarchy, except for those cells that have a different library subset constraint explicitly set on them.

Note:

When you use the `set_target_library_subset` command, you do not need to uniquify the designs before using the command. (Note, however, that if you intend to use the `insert_level_shifters` command as part of the logic synthesis flow in Design Compiler, you do need to run `uniquify` first.)

To remove a target library subset constraint, use the `remove_target_library_subset` command with the appropriate library list and cell list.

You can check for errors and conflicts introduced by target library subsetting by using either the `check_mv_design -target_library_subset` command or the `check_target_library_subset` command. The command checks for

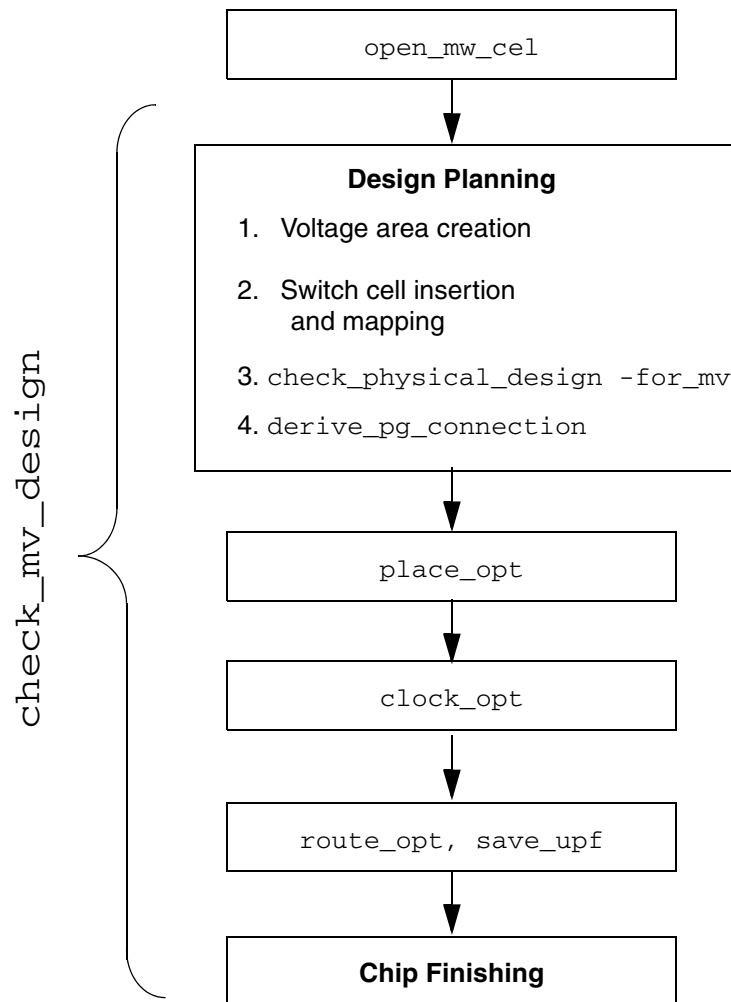
- Conflicts between target library subsets and the global `target_library` variable
- Conflicts between operating condition and target library subset
- Conflicts between the library cell of a mapped cell and target library subset

Use the `report_target_library_subset` command with the appropriate library cell list to find out which target library subsets have been defined both for the hierarchical cells and at the top level.

UPF Flow for Multivoltage Designs

Figure 13-2 shows an overview of the UPF flow for multivoltage designs.

Figure 13-2 IC Compiler Multivoltage UPF Flow



The steps involved in the UPF flow are described below:

1. Invoke IC Compiler
2. Use the appropriate target libraries that comply with the power and ground pin Liberty library syntax.

For additional information about the power and ground pin library syntax, see the “Advanced Low Power Modeling” chapter in the *Library Compiler User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries*.

3. Read the multivoltage design.
4. Read the UPF file using the `load_upf` command.

Use the UPF commands to specify the power intent or power constraints for your multivoltage design in the UPF file. Use the `load_upf` command to read this file in IC Compiler. The `load_upf` command executes the UPF commands in the specified UPF file.

This UPF file can also be used for RTL simulation and equivalence checking.

5. Use the `set_operating_condition` command to set the operating condition on the top level of the design hierarchy and to derive the process and temperature conditions for the design. Use the `set_voltage` command to set the voltage values for the power or ground supply nets. If your setup has both `set_operating_condition` and `set_voltage` commands, IC Compiler uses an internal precedence rule to derive an appropriate operating condition.

Note:

If you do not set the top-level operating condition, you will encounter a MV-028 error message.

6. Use the `derive_pg_connection` command to connect all unconnected power and ground pins. Use the `derive_pg_connection -create_net` command to create the missing power nets. To override the existing power connection and use the power intent defined in the UPF file, use the `derive_pg_connection -reconnect` command. Use this command before using any of the optimization commands. For more information about the `derive_pg_connection` command, see the “Power Planning” chapter in the *IC Compiler User Guide: Design Planning*.

If the floorplan is read from a DEF file, the port names in the DEF file and the UPF file should be consistent. If not, the port names in the DEF file take precedence over the port names in the UPF file. However, for boundary ports, IC Compiler honors both DEF file and UPF file definitions.

After the completion of implementation, use the `derive_pg_connection` command prior to writing the power and ground netlist to a file.

You can use the `check_mv_design -power_nets` command to check for mismatches in the power and ground pin connections.

7. Use the `map_power_switch` command to map the power switch to the technology library cell. Instantiate the power switch cell in the floorplan using the `add_header_footer_cell_array` command. Use the UPF supported command, `create_power_switch` to create a power switch.

8. It is recommended that the design being read into IC Compiler already contain special cells such as the retention registers, isolation cells, and level-shifter cells.

Before you run the optimization command, `place_opt`, it is recommended to use the `check_mv_design -verbose` command. This command checks for the violations in the multivoltage designs. With the `-verbose` option this command gives a detailed report of the violations.

9. After the optimization steps, use the `save_upf` command to save and write the UPF file updated by IC Compiler.

Compared to the UPF file that you read into the tool, the UPF file written by IC Compiler contains the following additional information:

- The following comment on the first line of the UPF file:

```
#Generated by IC Compiler(B-2008.09) on Mon Aug 4 14:26:58 2008
```

- Explicit power connections to special cells such as level shifters and isolation cells.
- Any additional UPF commands that you specified at the command prompt in the IC Compiler session.

Defining Power Domains and Supply Networks

This section describes how to define power domains and supply networks using the UPF commands. This section contains the following subsections:

- [Power Domains](#)
- [Hierarchy and Scope](#)
- [Creating Power Domains](#)
- [Power and Ground Supply Network](#)
- [Creating a Power Switch](#)
- [Adding Port State Information to Supply Ports](#)
- [Creating a Power State Table](#)

- [Defining the States of the Supply Nets](#)
- [Connecting Power and Ground Nets](#)

Power Domains

Multivoltage designs contain design partitions which have specific power behavior compared to the rest of the design. A power domain is a basic concept in the Synopsys low-power infrastructure, and it drives many important low power features across the flow.

By definition, a power domain is a logic grouping of one or more logic hierarchies in a design that shares the same power characteristics, including:

- Power net hookup requirements
- Power down control and acknowledge signals, if any
- Power switching style

Thus, a power domain describes a design partition, bounded within logic hierarchies, which has a specific power behavior with respect to the rest of the design.

A power domain has the following characteristics:

- Name
- Level of hierarchy or scope where the power domain is defined or created
- The set of design elements that comprises the power domain
- An associated set of supply nets that are allowed to be used within the power domain
- Primary power supply and ground nets
- Synthesis strategies for isolation, level-shifters, and retention registers

Note:

A power domain is strictly a logic construct; not a netlist object. For more details on the concept of power domain see the “Power Intent Specification” chapter in the *Synopsys Low-Power Flow User Guide*.

Hierarchy and Scope

Logical hierarchy, where the power domain is created, is called the scope of the power domain. Design elements that belong to a power domain are said to be in the extent of the power domain. For more details, see the “Power Intent Specification” chapter in the *Synopsys Low-Power Flow User Guide*.

In Design Compiler and IC Compiler, in UPF mode, you can use the `set_scope` command to specify the scope or level of hierarchy. The `set_scope` command sets the scope or the level of hierarchy to the specified scope. When no instance is specified, the scope is set to the top level of the design hierarchy. Alternately, you can also use the `current_instance` command to specify the current scope. However, in the power context, the `set_scope` command is preferred.

You should explicitly specify the scope using the `set_scope` or the `current_instance` command. Unless explicitly specified, IC Compiler uses the current scope or current level of hierarchy when you define objects. For more details on scope, see the “Power Intent Specification” chapter in the *Synopsys Low-Power Flow User Guide*.

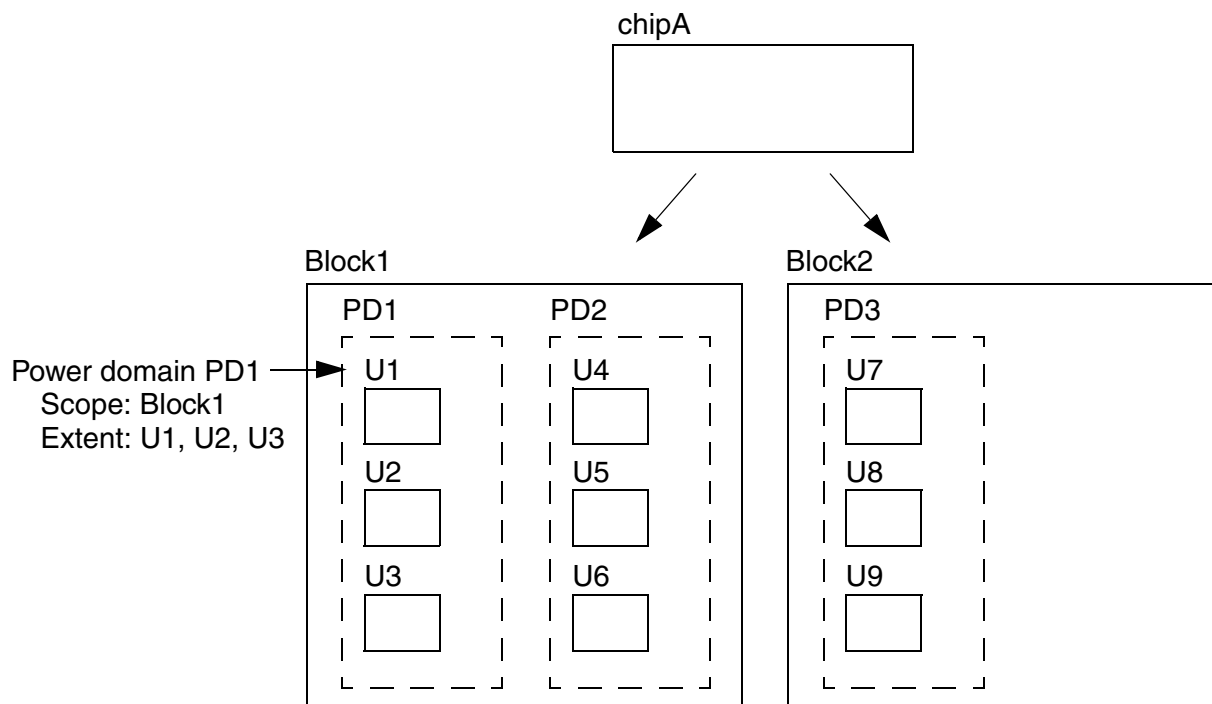
Creating Power Domains

Use the `create_power_domain` command to create a power domain with the specified name. The syntax of the `create_power_domain` command is as follows:

```
create_power_domain [-elements element_list] [-include_scope]
[-scope instance_name] domain_name
```

Use the `-element` option to specify the list of hierarchical cells that are added as the extent of the power domain. The `-include_scope` option specifies that all the elements in the current scope share the primary supply of the power domain, but are not necessarily added as the extent of the power domain. Use the `-scope` option to specify the logical hierarchy or the scope at which the power domain is to be defined. *domain_name* is the name of the power domain to be created.

Figure 13-3 Defining a Power Domain and Scope



The following example creates the PD1 and PD2 power domains shown in [Figure 13-3](#).

```
create_power_domain -elements {U1 U2 U3} -scope Block1 PD1
create_power_domain -elements {U4 U5 U6} -scope Block1 PD2
```

Alternatively, you can use the `set_scope` command to first set to the desired scope and then create the power domain:

```
set_scope Block1
create_power_domain -elements {U1 U2 U3} PD1
create_power_domain -elements {U4 U5 U6} PD2
```

You can use the `-include_scope` option to include all the elements in the specified scope to share the supply of the power domain.

```
create_power_domain -include_scope Block2 PD3
```

In this case, the U9 element shares the supply of power domain PD3, though U9 is not explicitly mentioned to be part of the PD3 power domain.

Power and Ground Supply Network

Each power domain has a supply network consisting of supply nets and supply ports. The supply network can contain power switches. The supply network is used to specify the power and ground net connections for a power domain. A supply net is a conductor that carries a supply voltage or ground. A supply port is a power supply connection point between the inside and outside of the power domain. Supply ports serve as the connection points between supply nets. A supply net can carry a voltage supply from one supply port to another.

When used together, the power domain and supply network objects allow you to specify the power management intentions of the design.

Every power domain must have one primary power supply and one primary ground. In addition to the primary power and ground nets, a power domain can have any number of additional power supply and ground nets.

Creating Supply Ports

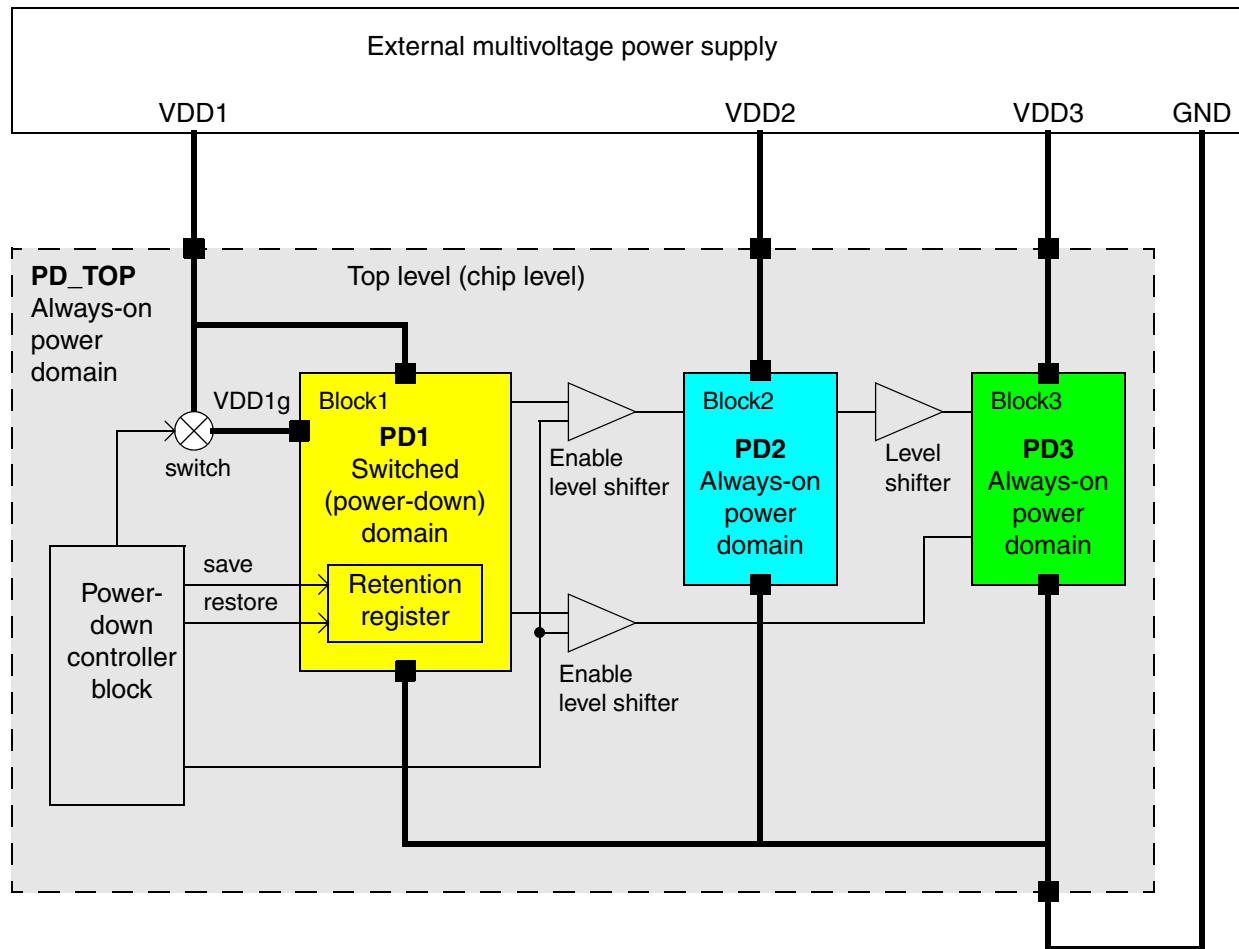
To create the supply and ground ports, use the `create_supply_port` command.

The syntax of the `create_supply_port` command is as follows:

```
create_supply_port supply_port_name [-domain power_domain_name ]  
[-direction in | out]
```

The *supply_port_name* specified should be unique at the level of hierarchy where it is defined. The `-domain` option is used to create the supply port in the scope of the submodule. The supply port name should be a simple non hierarchical name. When the `-domain` option is not used, the supply port is created in the current scope or level of hierarchy and all power domains in the current scope can use the created port.

Figure 13-4 Power Intent Specification



The following example shows creation of the ports shown in Figure 13-4:

To create the supply ports VDD1, VDD2 and VDD3 and GND at the top level of the design hierarchy or power domain PD_TOP, use the command as follows:

```
create_supply_port VDD1
create_supply_port VDD2
create_supply_port VDD3
create_supply_port GND
```

To create the supply ports VDD1, VDD1g and GND in the power domain PD1, use the create_supply_port command as follows:

```
create_supply_port VDD1 -domain PD1
create_supply_port VDD1g -domain PD1
create_supply_port GND -domain PD1
```

To create the supply ports VDD2, GND in the power domain PD2, and VDD3, GND in power domain PD3 respectively, use the `create_supply_port` command as follows:

```
create_supply_port VDD2 -domain PD2
create_supply_port GND -domain PD2
create_supply_port VDD3 -domain PD3
create_supply_port GND -domain PD3
```

Note:

Connectivity is not defined when the supply port is created. To define connectivity use the `connect_supply_net` command.

Creating Supply Nets

A supply net connects supply ports. Use the `create_supply_net` command to create a supply net. The syntax of the `create_supply_net` command is as follows:

```
create_supply_net -domain domain_name [-reuse] supply_net_name
```

The supply net is created in the same scope or logical hierarchy as the specified power domain. When the `-reuse` option is used, the specified supply net is not created; instead an existing supply net with the specified name is reused.

```
create_supply_net GND_NET
create_supply_net -domain PD1 GND_NET
create_supply_net -domain PD2 GND_NET
create_supply_net -domain PD3 GND_NET
```

When a supply net is created, it is not considered a primary power supply or ground net. To make a specific power supply or ground net of a power domain into a primary supply or ground net, use the command `set_domain_supply_net`.

Connecting Supply Nets

This command connects the supply net to the specified supply ports or pins. The connection can be within the same level of hierarchy or to ports or pins down the hierarchy. The syntax of the `connect_supply_net` command is as follows:

```
connect_supply_net supply_net_name -ports list
```

The following example depicts the use of the `connect_supply_net` command to connect supply nets to various supply ports in different levels of hierarchy or power domains.

```
connect_supply_net GND_NET -ports GND
connect_supply_net GND_NET -ports {B1/GND B2/GND B3/GND} GND
```

Specifying the Primary Supply Net for a Power Domain

Use the `set_domain_supply_net` command to define the primary power supply net and primary ground net for a power domain. The syntax of the `set_domain_supply_net` command is as follows:

```
set_domain_supply_net -primary_power_net supply_net_name
-primary_ground_net supply_net_name domain_name
```

Every power domain must have one primary power and one ground connection. When a supply net is created, it is not a primary supply net. You must use the `set_domain_supply_net` command to designate the specific supply net as the primary supply net for the power domain. If a power or ground pin of a cell in a power domain is not explicitly connected to any supply net, that power or ground pin of the cell is assumed to be connected to the primary power or ground net of the power domain to which the cell belongs.

When in the scope of Top, you can use the following command to designate VDD and GND nets as the primary power and ground net, respectively, of the power domain PD_TOP.

```
set_domain_supply_net -primary_power_net VDD -primary_ground_net GND PD_TOP
```

Creating a Power Switch

The `create_power_switch` command creates an instance of power switch in the scope of the specified power domain. The power switch has at least one input supply port and one output supply port. When the switch is off, the output supply port is shut down and has no power.

The syntax of the `create_power_switch` command is as follows:

```
create_power_switch -domain domain_name
-output_supply_port {port_name supply_net_name}
-input_supply_port {port_name supply_net_name}
-control_port {port_name net_name}
-on_state {state_name port_name {boolean_function}}
[-ack_port {port_name net_name {boolean_function}}]
[-ack_delay {port_name delay}]
[-off_state {state_name {boolean_function}}]
switch_name
```

For more details of the various options, see the command man page.

The following simple power switch definition is the definition for the power switch shown in [Figure 13-4](#):

```
create_power_switch SW1 -domain PD_TOP \  
  -output_supply_port {SWOUT VDD1g} \  
  -input_supply_port {SWIN1 VDD1} \  
  -control_port {CTRL swctl} \  
  -on_state {ON VDD1 {!swctl}}
```

Adding Port State Information to Supply Ports

The `add_port_state` command adds state information to a supply port. This command specifies the name of the supply port and the possible states of the port. The first state specified is the default state of the supply port. The port name can be a hierarchical name. Each state is specified as a state name and the voltage level for that state. The voltage level can be specified as a single nominal value, set of three values (minimum, nominal, and maximum), or the keyword `off` to indicate the “off” state. The state names are also used to define all possible operating states in the power state table. The syntax of the `add_port_state` command is as follows:

```
add_port_state port_name {-state {name nom | min nom max | off}} *
```

*Asterisk indicates an item that can be repeated in the command.

A power switch supply port is considered a supply port because it is connected by a supply net, so it can be specified as the supply port in the `add_port_state` command. Similarly, a RTL port can be made a supply port by this command or by the `connect_supply_net` command. Note that supply states specified at different supply ports are shared within a group of supply nets and supply ports directly connected together. However, this sharing does not happen across a power switch.

Creating a Power State Table

A power state table (PST) defines the legal combination of states that can exist simultaneously during the operation of the design. A power state table is a set of power states of a design in which each power state is represented as an assignment of power states to individual power nets. A power state table for a design captures all the possible operational modes of the design in terms of power supply levels. Given a power state table, a power state relationship including voltage and relative always-on relations can be inferred between any two power nets. The power state table is used by the synthesis tool for analysis, synthesis, and optimization of the multivoltage design.

The `create_pst` command creates a new power state table and assigns a name to the table. The command lists the supply ports or supply nets in a particular order. The `add_port_state` defines the name of the possible states for each supply port.

The `create_pst` command can only be used at the top-level scope. The power switch supply ports are considered supply ports because they are connected by supply nets, so they can be listed as supply nets in `create_pst` command.

A supply port and a supply net can have the same name, even when they are unconnected. If such a name is listed in the `create_pst` command, it is assumed to represent the supply port and not the supply net. The syntax of the `create_pst` command is as follows:

```
create_pst table_name -supplies list
```

Defining the States of the Supply Nets

The `add_pst_state` command defines the states of each of the supply nets for one possible state of the design. The syntax of the command is as follows:

```
add_pst_state -pst pst_name -state list state_name
```

The command must specify the name of the state, the name of the power state table previously created by the `create_pst` command, and the states of the supply ports in the same order as listed in the `create_pst` command.

The listed states must match the supply ports or nets listed in the `create_pst` command in the corresponding order. For a group of supply ports and supply nets directly connected together, the allowable supply states are derived from the shared pool of supply states commonly owned by the members of the group.

The following example creates a power state table, defines the states of the supply ports, and lists the allowed power states for the design.

```
create_pst pt -supplies { PN1 PN2 SOC/OTC/PN3 }
add_port_state PN1 -state { s88 0.88 }
add_port_state PN2 -state { s88 0.88 } -state { s99 0.99 }
add_port_state SOC/OTC/PN3 -state { s88 0.88 } -state { pdown off }
add_pst_state s1 -pst pt -state { s88 s88 s88 }
add_pst_state s2 -pst pt -state { s88 s88 pdown }
add_pst_state s3 -pst pt -state { s88 s99 pdown }
```

Connecting Power and Ground Nets

To connect the power and ground pins defined in your power domain to the appropriate power and ground supply ports of your design, use the `derive_pg_connection` command. By default this command works only on the unconnected power and ground nets. Use the `-reconnect` option to force the command to work on the entire power network by reconnecting the already-connected power and ground pins as well as connecting the unconnected pins. Use this command before using any of the optimization commands. This command appropriately connects the power and ground pins defined in the design hierarchy based on the power and ground details defined in the power domains in the UPF file.

The automatic connection behavior for power pins is as follows:

- For a power pin with an exception connection defined, the power net specified in the exception is used.
- For cells with a single power pin that are used as always-on cells, the backup power net is used.
- For cells with a single power pin that are *not* used as always-on cells, the primary power net is used.
- For multithreshold-CMOS switch cells with multiple power pins, the power pins are connected to the power domain power nets of the same type – that is, primary power pin to primary power net, backup power pin to backup power net, and internal power pin to internal power net.
- In the shut-down power domains of an multithreshold-CMOS design, the primary power pins of the standard cells are connected to the internal power net of the shut-down domain. The primary power net of the shut-down power domain is connected only to the primary pin of the switch cell.
- For boundary cells with multiple power pins (level shifters and isolation cells), the power pins are connected to the power nets of the related power domains – that is, the load pin of the boundary cell is connected to the power net of the driver's power domain, and the drive pin of the boundary cell is connected to the power net of the load's power domain.
- For cells with multiple power pins that are neither boundary cells nor multithreshold-CMOS switch cells (for example, multi-rail always-on cells, retention registers, macro cells), the power pins are connected to the appropriate power nets – that is, the primary and backup power pins are connected to the primary and backup power nets, respectively.

Note:

The rules applicable for the automatic connection of ground pins are similar to those for the automatic connection of power pins.

To verify the power connections, use the `check_mv_design -power_net` command.

For more information, see the man pages.

Defining and Using Voltage Areas

A voltage area is a placement area for one or more logical partitions operating at the same voltage. The cells of the logical partition are associated with the partition's voltage area and are constrained to placement within that area.

Use the `create_voltage_area` command to create voltage areas and align the hierarchies, as described in the *IC Compiler User Guide: Design Planning*.

You can define voltage areas at any level of a logic hierarchy. Both rectangular and rectilinear voltage areas are allowed. Voltage areas cannot consist of disjoint rectilinear and rectangular shapes.

Physically nested voltage areas are supported, provided the inner voltage is completely contained within the outer voltage area. These nested voltage areas correspond to logically nested power domains. Voltage areas that would overlap physically have to be resolved into nonoverlapping, abutted rectangular or rectilinear subareas. How you construct voltage areas in this case is discussed in [“Handling Nested Voltage Areas” on page 13-28](#).

Voltage areas can be explicitly associated with existing power domains, or they can be created without specifying any explicit association with the power domains. They can also be created when no power domains have been defined for the design.

The logic hierarchies are assigned to specific voltage areas when you create or update these voltage areas by specifying a cell list in the `create_voltage_area` command. Care must be taken to align the hierarchies correctly. The tool checks the operating condition voltage of each module against the voltage area specification to ensure that all modules inside the voltage area operate at the correct voltage.

For designs with power domains, cell alignment can be assured by specifying the name of the associated power domain when you run the `create_voltage_area` command. Use the `-power_domain` option to specify the name. In this case, you do not specify a voltage area name or the list of cells in the `create_voltage_area` command. (The cell list is provided in the power domain definition.)

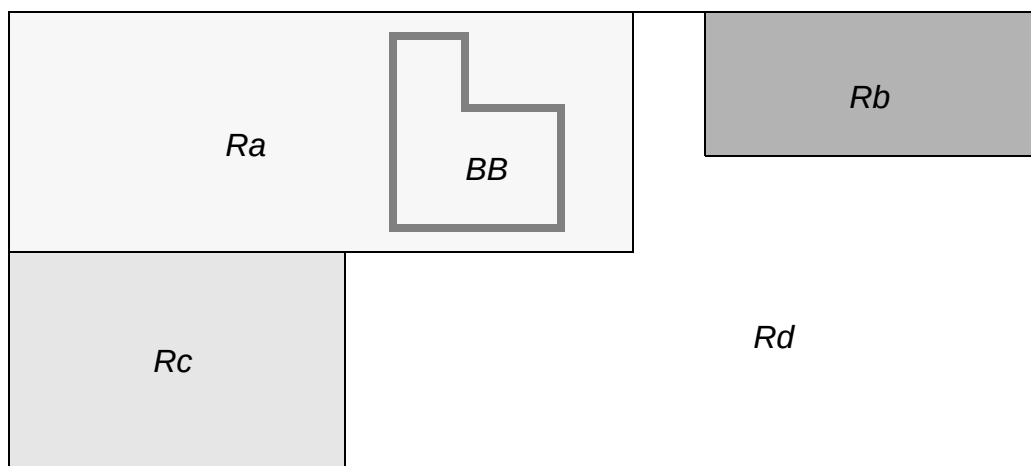
The tool derives a default voltage area for placement of cells not assigned to other voltage areas. The default voltage area can contain top-level leaf cells and buffer-type level shifters, as well as the blocks.

Level shifters and isolation cells can have site heights that differ from the standard cell site height by either integer multiples (homogeneous sites) or noninteger multiples (heterogeneous sites). IC Compiler supports the placement of both types of sites.

Using heterogeneous sites for certain level shifters and isolation cells can help improve utilization. However, if the design has overlapping rows, set the `physopt_check_site_overlap` variable to false before running placement and legalization.

Figure 13-5 shows a multivoltage design divided into several voltage areas.

Figure 13-5 Multivoltage Design With Several Voltage Areas



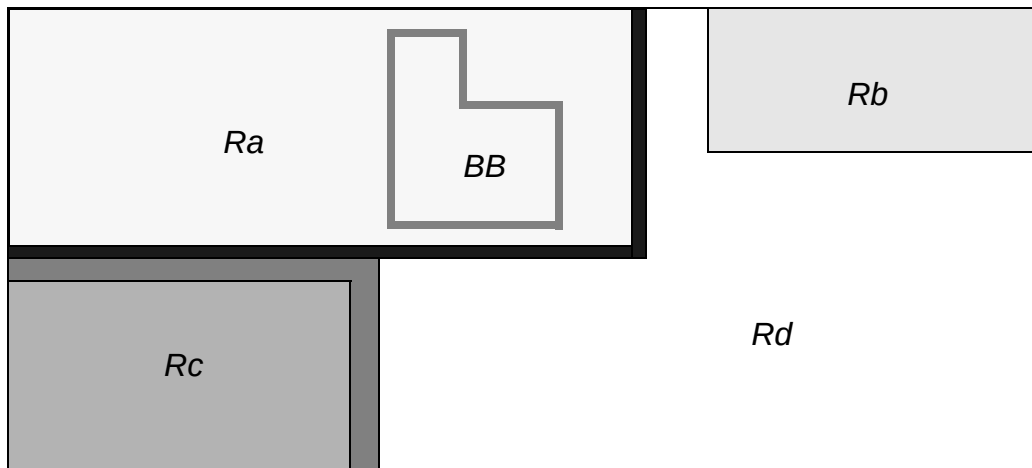
In this design, the floorplan has four voltage areas: Ra, Rb, Rc, and Rd. The default voltage area is Rd. Each voltage area has a corresponding logical partition, A, B, C, and D, that aligns with its respective voltage area. A hard bound can be defined inside a voltage area but not across voltage areas. In this example, a hard bound, BB, is defined inside voltage area Ra.

Special always-on site rows can be defined within a given voltage area. The standard cells placed in these site rows serve as the always-on cells of the shut-down power domain associated with the given voltage area.

Optionally, guard bands can be specified for some or all voltage areas to prevent shorts. Guard bands define hard keepout margins surrounding the voltage areas in which no cells, including level shifters and isolation cells, can be placed. Guard bands can be defined when creating voltage areas automatically or when updating the voltage areas.

Figure 13-6 shows a floorplan similar to Figure 13-5 but with guard bands protecting voltage areas Ra and Rc.

Figure 13-6 Multivoltage Design With Several Voltage Areas and Guard Bands



Creating Voltage Areas

You create voltage areas by using the `create_voltage_area` command. With this command you specify the exact coordinates of the voltage area.

If the design contains power domains, you can use the `-power_domain` option with the corresponding power domain name in the `create_voltage_area` command. Then the power domain name is also the voltage area name. For this case, do not specify a cell list in the `create_voltage_area` command. The cells that were specified as belonging to the power domain are the cells that IC Compiler places in the given voltage area. Using the `-power_domain` option ensures correct alignment of the logic hierarchies with the voltage areas.

If the design does not contain power domains or you do not use the `-power_domain` option, you must then specify a voltage area name and provide the list of cells that are to be placed in the given voltage area. In this case, you are responsible for making sure that the hierarchies align correctly with their voltage areas.

For an example on how to use the `create_voltage_area` command without using the `-power_domain` option, see the script provided in [Example 13-1 on page 13-25](#). For an example that does use the `-power_domain` option, see the script provided in [Example 13-4 on page 13-54](#).

Note:

You can specify guard bands when creating voltage areas by using the `-guard_band_x` and `-guard_band_y` options with the `create_voltage_area` command. See [“Including Guard Bands With Voltage Areas” on page 13-26](#).

After you have defined a voltage area and associated cells, you can add or remove cells from the voltage area by using the `update_voltage_area` command.

Note the following limitations:

- Intersecting voltage areas are not supported.
- All voltage area geometries must be mutually exclusive.
- An error occurs when you use values with the `-coordinate` option that is too small for all the cells to fit.

The script in [Example 13-1](#) shows you how to create voltage areas manually without specifying power domains.

Example 13-1 Manual Voltage Area Specification

```
# Block1 is VoltageArea1 and Block2 is VoltageArea2 #
##
create_voltage_area -coordinate {...} -name VoltageArea1
update_voltage_area -name VoltageArea1 Block1
create_voltage_area -coordinate {...} -name VoltageArea2
update_voltage_area -add -name VoltageArea2 Block2
report_voltage_area -all
```

Updating Voltage Areas in a Design

After you have created voltage areas and associated hierarchical cells with them, you can use the `update_voltage_area` command to add or remove cells from specified voltage areas. You can use the `get_voltage_area` command instead of explicitly specifying voltage area names. By adding or removing cells, you can adjust the utilization of the voltage area. However, you might prefer to change the size of the voltage areas and keep the utilization the same.

Note:

You can specify guard bands when updating voltage areas by using the `-guard_band_x` and `-guard_band_y` options with the `update_voltage_area` command. See the next section, [“Including Guard Bands With Voltage Areas.”](#)

Including Guard Bands With Voltage Areas

You can use guard bands to ensure that no shorts occur at the boundaries of the voltage areas. Guard bands define hard keepout margins surrounding the voltage areas. No cells, including level shifters and isolation cells, can be placed within the guard band margins.

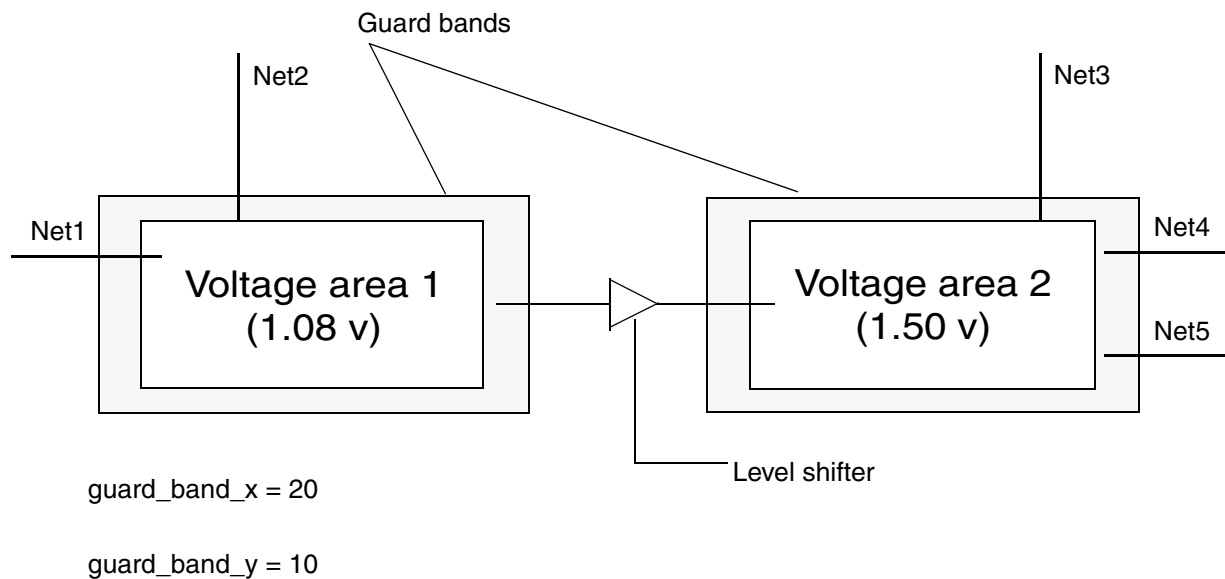
For example, using guard bands is recommended when the rows are the same for all the voltage areas. The guard bands guarantee that the cells in different voltage areas are separated so that power planning does not introduce shorts.

You can include guard bands when you create or update voltage areas. You generate guard bands by using the `-guard_band_x` and `-guard_band_y` options with any of the following three commands:

- `create_voltage_area -guard_band_x x_width`
 `-guard_band_y y_width`
 [user input values defining a voltage area]
- `update_voltage_area -guard_band_x x_width`
 `-guard_band_y y_width`
 [user input values specifying voltage area
 and updating hierarchical cells list]

Figure 13-7 shows two voltage areas with guard bands of different widths. For both voltage areas, the x-width is 20 and the y-width is 10. Notice that the x-width defines the horizontal width of the vertical part of the guard band, and the y-width defines the vertical width of the horizontal part.

Figure 13-7 Two Voltage Areas With Guard Bands



In the IC Compiler GUI, voltage areas are outlined. Spaces or margins between voltage areas indicate the presence of guard bands.

When generating guard bands by using the `create_voltage_area` or `update_voltage_area` commands, you should be aware of the following conditions:

- If some coordinates of a voltage area are defined outside the core area, the command does not generate the voltage area and issues a warning and error message.
- If a voltage area intersects an existing voltage area, the command does not create the voltage area and issues an error message.
- If a guard band of a voltage area overlaps another voltage area, the command generates the guard band and issues a warning message.
- The x (y) guard band width applies to both the left and right (top and bottom) side of a voltage area.
- The IC Compiler GUI can highlight the guard bands.
- Overlapping guard bands are not allowed and generate warning messages.
- A guard band that lies outside the core area does not generate a warning message.

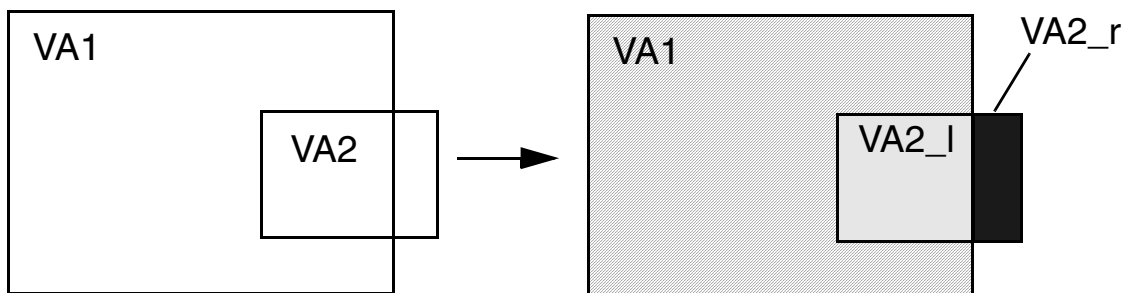
Handling Nested Voltage Areas

Voltage areas can be physically nested corresponding to the nested relationship of the logic hierarchy of nested power domains. You still use the `create_voltage_area` command to define the individual voltage areas. A given voltage area can contain one or more nested voltage areas. Note that a voltage area must completely contain its nested voltage areas. Intersecting voltage areas are not supported.

In the case where logically nested hierarchies would lead to overlapping voltage areas, you can use the `create_voltage_area` command to resolve the overlapping areas by defining a number of voltage areas consisting of separate, *abutted* rectangles or rectilinear shapes. Each shape would share at least one boundary with another shape of the given voltage area.

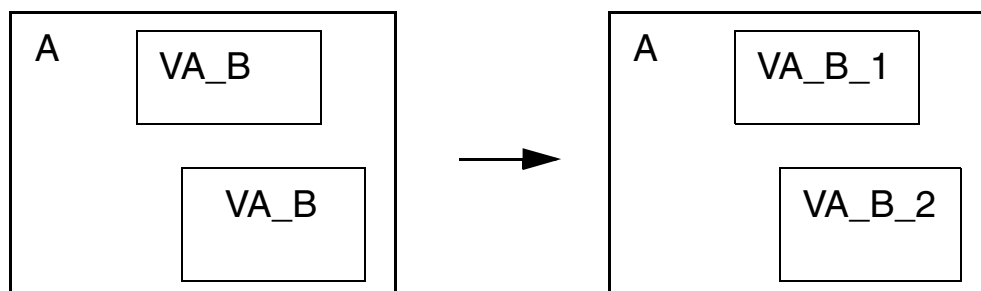
Figure 13-8 shows an example of areas intersecting and how you might resolve them into three physically abutted, separate voltage areas, VA1, VA2_l, and VA2_r. Voltage area VA1 is an eight-sided rectilinear area, and voltage areas VA2_l and VA2_r are four-sided rectangles.

Figure 13-8 Example of Physically Resolved Voltage Areas



A voltage area with disjoint shapes, in which no boundary is shared, is not allowed. You can change a disjoint voltage, such as VA_B shown in Figure 13-9, into two voltage areas with unique names, such as VA_B_1 AND VA_B_2.

Figure 13-9 Example of a Resolved Disjoint Voltage Area



Voltage Area Support for Macros

Macro cells can be listed as belonging to power domains and can be physically located within the corresponding voltage areas. If a macro is not listed in a power domain, it can be explicitly listed in the `create_voltage_area` command.

Note:

Care must be taken to define whether a given macro cell is internally isolated or requires external isolation. Internally isolated macros should have the `connected_to_iso_cell` attribute set on the appropriate library pin; otherwise the tool assumes external isolation might be required and adds an isolation cell to the appropriate input net. Usually, if attribute marking is required, it is done as part of logic synthesis.

Because a macro cell is a physical cell, having height, width, and location, a voltage area is not created if the specified voltage area would not completely contain the macro cell. Instead, an error message is issued. Also, if a given macro cell already belongs to a voltage area, a new voltage area is not created if you run the `create_voltage_area` and list the macro cell. An error message is issued.

The `report_voltage_area` command reports all the top-level voltage area blocks, including macros. If a macro cell is within a hierarchical block that belongs to a voltage area, that macro is not part of the top-level blocks and is not reported.

Defining Always-On Power Wells Within Voltage Areas

To create an always-on well within a voltage area, use the `create_bounds` command with the `-exclusive` option to define a special placement area or well for specific cells within a voltage area. You define the location and dimensions of an exclusive move bound as well as providing the list of cells to be placed within the exclusive move bound. The geometry of the exclusive bound must fit within the voltage area. No cells except the specified cells can be placed within the bound.

Once the exclusive move bounds are defined, you use the `set_power_guide` command to identify these bounds as power guides. Use the `unset_power_guide` command to “remove” a move bound as a power guide. You can determine the already defined power guides by using the `report_power_guide` command.

You use the exception connection mechanism to connect these cells to the backup power net. The cells then function as always-on cells, so you can use single-power, standard cells.

Removing Voltage Areas From the Design

Use the `remove_voltage_area` command to remove all voltage areas or specified voltage areas from the design. This includes removing any move bounds and guard bands belonging to the removed voltage area. If you remove a logically nested voltage area, the subhierarchy logic inherits the voltage properties of the parent hierarchy.

Reporting on the Voltage Areas of a Design

You use the `report_voltage_area` command to obtain information about existing voltage areas. The report includes voltage area name, the list of hierarchical cells associated with the voltage area, the voltage area geometry, area, utilization, and guard band information if guard bands are present.

Handling Level Shifters and Isolation Cells

To properly handle level shifters and isolation cells, you should be aware of the following properties and commands:

- [Level-Shifter and Isolation Cell Requirements](#)
- [Level-Shifter Commands](#)
- [Level-Shifter Threshold \(Automatic and User-Specified\)](#)
- [Inserting Buffer-Type Level Shifters](#)
- [Inserting Isolation Cells and Enable-Type Level Shifters](#)
- [Placing Level Shifters](#)

Level-Shifter and Isolation Cell Requirements

The following level-shifter and isolation cell requirements should be observed:

- Input and output pin voltages are the same for isolation cells.
- Input and output pin voltages are different for level shifters, although they can connect driver and load pins operating at the same voltage (property of buffers).
- Only buffer-type or enable-type level-shifter library cells are currently supported by IC Compiler. (Enable-type level shifters are like buffer-type level shifters except they have an additional enable pin.)
- Buffer-type and enable-type level-shifter library cells must have the library attribute `is_level_shifter` set to true.
- Enable-type level shifters must also have the `level_shifter_enable_pin` attribute on the enable pin.
- Isolation library cells must have the library attribute `is_isolation_cell` set to true.
- Isolation cells must have the `isolation_cell_enable_pin` attribute on the enable pin.
- Level shifters and isolation cells are selected by IC Compiler from the target libraries. Therefore, at least one of the libraries must contain the required cells.
- You can define special sites (for multiheight cells) where IC Compiler places the level shifters and isolation cells when you run the `place_opt` command. Otherwise, the tool attempts to place the level shifters and isolation cells near the voltage area boundaries.

Level-Shifter Commands

These commands can be used to manage the level shifters in your design:

- `check_mv_design` with the `-level_shifters` option checks and reports net, cell, and environment violations with respect to level shifters. You can run the command at any point in the flow. (This command is preferred over the `check_level_shifters` command.)
- `insert_level_shifters` – Inserts *only* buffer-type level shifters to accommodate voltage mismatches on nets, according to the level-shifter strategy and threshold settings. This command can be used anywhere in the flow, but it is usually best to run the command as early in the flow as possible (usually prior to logic synthesis).

Note:

If the design is synthesized using Design Compiler, *automatic* level-shifter insertion is done as part of the `compile_ultra` or `compile` command process. In this case, you do not need to use the `insert_level_shifter` command unless level-shifter modifications are necessary.

- `set_level_shifter_strategy` – Sets the strategy used for adjusting voltage levels in the design. To use voltage step-up level shifters only, set the strategy rule to `low_to_high`; to use voltage step-down level shifters only, set the strategy rule `high_to_low`. To use both kinds of level shifters, set the rule to `all`. If you do not want to use the default (`all`), you should set the strategy before level shifters are inserted during compile or before using the `insert_level_shifters` command.
- `set_level_shifter_threshold` – Use this command if you do not want the tool automatically to determine the threshold levels. The threshold sets the minimum allowed voltage difference between source and sink, beyond which voltage differences are adjusted by insertion of level shifters. If you do not want to use the default (0 threshold voltage difference and 0 percentage difference), you should set the threshold before level shifters are inserted during compile or before using the `insert_level_shifters` command. See [“Level-Shifter Threshold \(Automatic and User-Specified\)” on page 13-33](#).
- `remove_level_shifters` – Removes all level shifters in a design except those marked with the `dont_touch` attribute. Use this command as needed to correct or redefine the use of level shifters in the design.

Level-Shifter Threshold (Automatic and User-Specified)

A level-shifter threshold strategy must be defined before buffer-type level shifters are inserted (manually or automatically) and checked. Two methods for defining the threshold strategy are available to you:

- The tool's automatic threshold capability, which uses the default threshold values to determine voltage mismatched nets requiring buffer-type level-shifter insertion. (No user-defined specifications are needed.)
- User-specified threshold conditions for nets with significant voltage mismatches. Threshold conditions are defined by using the `set_level_shifter_threshold` command.

Automatically Determined Level-Shifter Threshold

The tool can automatically determine level-shifter thresholds if the target library cells have the following attributes (usually described by equations involving the rail voltages, for example, $V_{DD1}+0.5$):

- V_{ih} – lowest input voltage for logic 1
- V_{il} – highest input voltage for logic 0
- V_{imax} – maximum input voltage for logic 1
- V_{imin} – minimum input voltage for logic 0
- V_{oh} – lowest output voltage for logic 1
- V_{ol} – highest output voltage for logic 0
- V_{omax} – maximum output voltage for logic 1
- V_{omin} – minimum output voltage for logic 0

These voltage parameters are used to determine output and input voltage bands (ranges) for both logic 1 and logic 0. For logic 1, the input voltage band is $(V_{imax}-V_{ih})$ and the output voltage band is $(V_{omax}-V_{oh})$. Similarly, for logic 0, the input voltage band is $(V_{il}-V_{imin})$ and the output voltage band is $(V_{ol}-V_{omin})$.

Therefore, for any pair of driver and load pins, if the output voltage band of the driver pin is completely overlapped by (contained within) the input voltage band of the load pin, no level shifter is required. This behavior holds for both logic 1 and logic 0.

Conversely, a sufficient voltage difference between the driver and load pins exists so that buffer-type level-shifter insertion is required if any of the following conditions hold:

- Driver pin $V_{omax} > \text{Load pin } V_{imax}$
- Driver pin $V_{omin} < \text{Load pin } V_{imin}$
- Driver pin $V_{oh} < \text{Load pin } V_{ih}$
- Driver pin $V_{ol} > \text{Load pin } V_{il}$

For any of these four conditions, the driver voltage band does not fall completely within the load voltage band, and buffer-type level shifters are needed.

User-Specified Level-Shifter Threshold

You can override the default threshold conditions by using the `set_level_shifter_threshold` command to specify the minimum voltage difference beyond which the voltage is to be adjusted by use of a buffer-type level shifter. Use the `-voltage` option to specify an absolute voltage difference between source and sink voltages, or use the `-percent` option to specify the percentage by which the source and sink voltages must differ.

You can specify both threshold options. In this case, buffer-type level shifters are inserted when either threshold condition is met. For example, to specify both a 0.5 voltage threshold and a 5 percent threshold, enter

```
icc_shell> set_level_shifter_threshold -voltage 0.5 -percent 5
```

Note:

The percentage is computed as $100 \times (\text{absolute value of the difference between the driver voltage and the load voltage}) / \text{driver voltage}$.

Inserting Buffer-Type Level Shifters

If buffer-type level shifters were not inserted automatically during logic synthesis or modifications are needed, you can use the `insert_level_shifters` command to insert these level shifters in the current design. Command options include `-preserve`, `-all_clock_nets`, `-clock_net`, and `-verbose`.

Using this command, level shifters are inserted on nets where significant voltage differences occur between drive pins and load pins. The voltage differences are determined either automatically by the tool or according to voltage threshold conditions that you define by using the `set_level_shifter_threshold` command. Also, you can accept the default level-shifter strategy, which allows buffer-type level shifters that step up the voltage and those that step it down, or you can use the `set_level_shifter_strategy` command to control

whether only step-up or step-down buffer-type level shifters are to be used. This command automatically assigns `dont_touch` attributes to the nets connecting ports and buffer-type level shifters.

Note:

Although you can manually insert buffer-type level shifters at a number of points in the multivoltage design flow, it is recommended that you insert them early in the flow – preferably before logic synthesis – or let logic synthesis automatically insert them. The latest they should be inserted is before you run the `place_opt` command in IC Compiler.

By default, level shifters are not inserted on clock nets. Use the `-all_clock_nets` or `-clock_net list_of_clock_nets` option if you want level shifters inserted on clock nets.

To obtain information about all the level shifters available in the target libraries, use the `-verbose` option. The information reported includes library name and type, level-shifter cell name, operating conditions, input and output voltages, process ID, temperature, and tree type. Also, the number of inserted level shifters is reported.

In general terms, IC Compiler goes through the following steps to insert buffer-type level shifters into voltage-mismatched nets:

- Identifies nets in the design with voltage mismatches that meet the automatically determined or user-defined threshold
- Analyzes the target library for available buffer-type level shifters, given the specified level-shifter rule

Note:

IC Compiler accesses only the target library buffer-type level-shifter cells that are marked with the library attributes specified in the Synopsys Liberty library model.

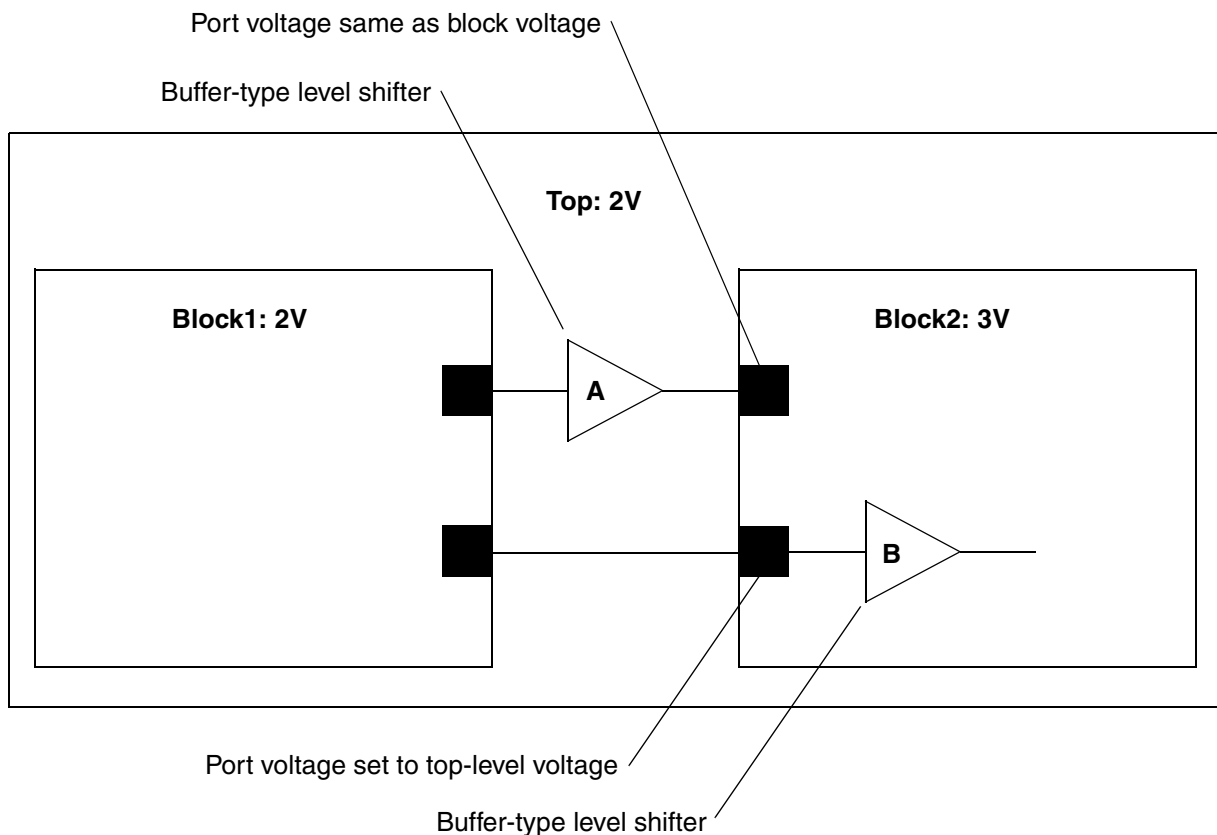
- Instantiates the proper buffer-type level shifter, and sets a `size_only` attribute on the level shifters and a `dont_touch` attribute on the port-to-level-shifter nets
- Instantiates buffer-type level shifters in clock nets as needed but only if the `-all_clock_nets` or the `-clock_net` option is specified

The buffer-type level shifters are inserted between the blocks if the operating voltage is set on the instance and not on any of its ports. However, if an operating voltage different from the block operating voltage is set on a port of the block, the buffer-type level shifter for that port is inserted inside the block. This behavior is the same for both automatic insertion during compile and manual insertion through the `insert_level_shifters` command.

You annotate operating voltages on the instances or their hierarchical ports by using the `set_operating_conditions` command for blocks. To have a buffer-type level shifter inserted inside a block, use the `set_operating_conditions -object_list` command to provide a port list and operating voltage setting for these ports that is different from the block voltage.

Figure 13-10 shows an example of buffer-type level-shifter insertion at both the top level for a block port that has the default block voltage and within a block for a port that is set at the top-level voltage.

Figure 13-10 Buffer-Type Level-Shifter Insertion at the Top Level and Block Level



Note:

Buffer-type level shifters are marked with the `size_only` attribute and therefore can be optimized as part of the physical synthesis phase (`place_opt` command).

Inserting Isolation Cells and Enable-Type Level Shifters

Isolation cells and enable-type level shifters are used to selectively shut off the input side of the voltage interface of a voltage area. Isolation cells do not shift the voltage. In general enable-type level shifters are used to step the voltage up or down, but they can also be used as isolation cells. (During logic synthesis, the `compile_ultra` or `compile` command can replace isolation cells with enable-type level shifters if the appropriate cells are available in the libraries.)

Isolation cells can be instantiated at the RTL level, or they can be manually inserted by using the `insert_isolation_cell` command at various points in the flow. Enable-type level shifters cannot be inserted by a user command. They can be instantiated only at the RTL level or inserted automatically during compile to replace already present isolation cells. It is recommended that isolation cells and enable-type level shifters cells be instantiated at the RTL level; otherwise formal verification of the logic is likely to fail.

For isolation cells, the `is_isolation_cell` and `isolation_cell_enable_pin` library attributes should be set on these cells and their enable pins. Similarly, the `is_level_shifter` and the `level_shifter_enable_pin` library attributes must be set on the enable-type level shifter cells and their enable pins. After these cells are instantiated in the netlist, they should be marked with the `size_only` attribute before logic optimization.

If you instantiate GTECH isolation cells, they can be mapped, retargeted, and sized during synthesis. (GTECH isolation cells can be instantiated in the RTL netlist when you run HDL Compiler Presto Verilog or Presto VHDL. See the appropriate reference manual.)

There are a number of ways in which isolation cells and enable-type level shifters might be connected to the input and output nets of the voltage areas. It is possible to introduce redundant isolation cells accidentally into the RTL description of the design or to fail to specify necessary cells. You can use the `check_isolation_cells` command to check for these conditions.

Placing Level Shifters

Level shifters (both types) can be placed at regular sites (single-height cells) or at special sites (multiheight cells) with multiple power rails. For regular sites, you are responsible for connecting secondary power to the level shifters.

If special sites are needed for level shifters, these sites must be instantiated in the floorplan at locations where these level shifters could be placed. Such sites should usually be defined along the voltage area boundaries. They can overlap with the base site array.

Level-shifter placement is performed when you run the `place_opt` command or when you issue the `create_placement` and `legalize_placement` commands. If the library attribute `is_level_shifter` is set to true for the level shifters, the tool attempts to place the level-shifter cells at legal sites nearest the voltage area boundaries. (This same placement process holds for isolation cells that have their `is_isolation_cell` attribute set to true.)

Handling Always-On Logic

Multivoltage designs typically have power domains that are shut down and powered up during the operation of the chip while other power domains remain powered up. The control nets that connect cells in an always-on power domain to cells within the shut-down power domain must remain on during shut down. These paths are referred to as always-on paths.

The target library can have special cells for use on the always-on paths. Alternatively you can use the `always-on` attribute on the library cells and library cell pins for the tool to use these cells and pins during always-on optimization. The following sections describe the always-on cells and pins, the rules you must follow to mark the `always-on` attribute on library cell pins and how to set the `always-on` attribute on various library and design objects.

Marking Always-On Cells of Pins

The always-on paths terminate on the cell pins inside the shut-down power domain. IC Compiler recognizes these pins as always-on pins. The tool then determines the always-on paths by tracing back from the always-on pins along the fanin logic cone.

The cells of the always-on pins can be of the following two types:

- Special, dual-power cells: isolation cells, enable-type level shifters, multithreshold-CMOS switch cells, retention registers, special RAMs, and so on. The tool automatically infers that the enable pins of special cells such as isolation cells and enable-type level shifters are always-on. For other types of cell, the always-on pin must be marked manually. The dual-power cells can be placed anywhere in the shut-down power domain.
- Single-power, standard cells whose placement is confined to the site rows of special always-on power wells within the shut-down block's voltage area. The tool treats the input pins of these cells as always-on because placement of such cells is restricted to always-on power wells. The pins do not need to be explicitly marked.

You can specify which type of always-on cell the tool uses in a given shut-down power domain with the `set_always_on_strategy` command. The command options are `single_power` or `dual_power` cells.

Rules for Marking Always-On Pins

To mark the pin of a cell that is not an isolation cell or an enable-type level shifter, as always-on you must follow these rules:

- Any input pin of a library cell can be marked always-on.
- Any input pin of a library cell instance can be marked always-on.
- Either the input or output pin of a hierarchical cell can be marked always-on.

You can mark the pins of the special cell by defining a .lib file with the appropriate attribute settings, which you should then compile using Library Compiler to obtain an output .db file, or by specifying the attributes in a script that you run in every session. For more details on marking cells and pins as always-on, see the Advanced Low Power Modeling chapter in the *Library Compiler User Guide: Modeling Timing, Signal Integrity, and Power in Technology Libraries*.

Marking Always-On Library Pins

If you want to mark a pin of a library cell as always-on, you can mark the pin in a .lib file or use the `set_attribute` command. For example, to mark the input pin of the library cell A, run the following command:

```
icc_shell> set_attribute [get_lib_pins lib_name/A/in] always_on true
```

By default, the tool assumes the enable pin of special cells such as isolation cells or level-shifters as always-on. You do not need to mark these pins as always-on.

Use the `get_attribute` command to query the always-on attribute on the library cell pin.

Marking Pass-Gate Library Pins

In the current implementation, the tool has the ability to prevent always-on cells from connecting to cells with pass-gate inputs. An always-on buffer should not drive a gate that has pass transistors at the inputs (pass-gate). Pass-gate input cells should be driven by a standard cell in a shut-down power domain. Therefore, if your library contains any of these cells, you must mark them as pass-gates in each session.

For example, to mark pin A of the multiplexer cell MUX1, run the following command:

```
icc_shell> set_attribute [get_lib_pins lib_name/MUX1/A] pass_gate true
```

Use the `get_attribute` command to query the pass_gate attribute on the library cell pin.

Marking Leaf Cell Instance Pins As Always-On

You can also mark the pins of leaf cell instances of a library cell as always-on.

For example, to mark pin A of the leaf cell U1/U2/A, run the following command:

```
icc_shell> set_attribute [get_pins U1/U2/A] always_on true
```

By specifically marking the instance pin as always-on instead of the library cell pin, you can use the library cell pin as an always-on pin in one instance but not as an always-on pin in another instance.

Use the `get_attribute` command to query the always-on attribute on the pin of the leaf cell instances.

Marking Library Cells for Always-On Optimization

The always-on synthesis in IC Compiler takes place only when the target library contains an always-on inverter and always-on buffer. To use a specific library cell in the optimization of always-on paths within the shut-down power domains, you mark the cell with the `always_on` attribute. The tool uses only always-on cells to optimize the always-on paths within the shut-down power domains. (Unmarked cells are used outside the shut-down power domains.)

Note that marking a library cell as always-on means that the cell is not used in the optimization of other kinds of paths. If you want to use a library cell in both always-on paths and paths that can be shut-down, you should mark only the instances of the library cell that occur in the shut-down power domains.

Reporting Commands

report_power_domain

The `report_power_domain` command reports the details of the specified power domain. The syntax of the `report_power_domain` command is as follows:

```
report_power_domain [-scope instance_name] [power_domains]
```

When the power domain name is not specified, all power domains in the design are reported.

Note:

The `report_power_domain` command is support in the non-UPF mode as well, but the syntax is different.

report_power_switch

The `report_power_switch` command reports all the power switches in the specified power domain. The syntax of the `report_power_switch` command is as follows:

```
report_power_switch -domain [domain_name]
```

Use the `-domain` option to specify the power domain for which power switches are to be reported. If the specified power domain does not exist in the current scope, the `report_power_switch` command fails.

report_isolation_cell

The `report_isolation_cell` command reports all the isolation cells in the current scope. With the `-strategy` option, this command reports the details of the isolation strategy.

Note:

IC Compiler reports the isolation strategy only when you have transferred your synthesized design from Design Compiler to IC Compiler using the `.ddc` or the Milkyway file formats.

report_retention_cell

The `report_retention_cell` command reports the retention cells in the design. With the `-verbose` option this command reports the list of the retention cells, the save and restore signals and the retention strategy.

Note:

IC Compiler reports the retention strategy only when you have transferred your synthesized design from Design Compiler to IC Compiler using the `.ddc` or the Milkyway file formats.

Sample IC Compiler Script for UPF Flow

The following is an example script for power intent specification using the UPF commands.

Example 13-2 A sample UPF Script for power intent specification

```
# Use load_upf command to load this file

set_scope
name_format -isolation_prefix "ISO_" -level_shift_prefix "LS_"
create_power_domain TOP
create_supply_port VDD -domain TOP
create_supply_port VDDI -domain TOP
create_supply_port VDDG -domain TOP
create_supply_port VDDM -domain TOP
create_supply_port VSS -domain TOP
add_port_state VDD -state {active_state 1.0}
add_port_state VDDI -state {high_state 1.0} \
    -state {low_state 0.8} -state {off_state off}
add_port_state VDDG -state {high_state 1.0} \
    -state {low_state 0.8} -state {off_state off}
add_port_state VDDM -state {active_state 1.0}
create_supply_net VDD -domain TOP
create_supply_net VDDI -domain TOP
create_supply_net VDDIS -domain TOP
create_supply_net VDDG -domain TOP
create_supply_net VDDM -domain TOP
create_supply_net VSS -domain TOP
set_domain_supply_net TOP -primary_power_net VDD -primary_ground_net VSS
create_power_switch my_sw -input_supply_port {vin VDDI} \
    -output_supply_port {vout VDDIS} -control_port {ctrl inst_on} \
    -control_sense high -on_state {state1 VDDG {inst_on}}
map_power_switch my_sw -domain TOP -lib_cells HDRDID2HVT
connect_supply_net VDDI -ports {VDDI inst_sw/vin }
connect_supply_net VDDIS -ports {inst_sw/vout InstDecode/VDDIS}
connect_supply_net VDD -ports {VDD GPRs/VDD MULT/VDD}
connect_supply_net VDDG -ports {VDDG GPRs/VDDG}
connect_supply_net VDDM -ports {VDDG Multiplier/VDDM}
connect_supply_net VSS -ports {VSS InstDecode/VSS GPRs/VSS Multiplier/VSS}
create_pst chiptop_pst -supplies {VDD A/VDDMS B/VDDIS C/VDDGS}
add_pst_state s0 -pst myblock_pst -state {1.0 0.8 0.8 1.0}
add_pst_state s1 -pst myblock_pst -state {1.0 1.0 1.0 1.0}
add_pst_state s2 -pst myblock_pst -state {1.0 off 1.0 1.0}
add_pst_state s2 -pst myblock_pst -state {1.0 1.0 off off}
```


The following is an example of a IC Compiler script for the UPF flow.

Example 13-3 A Sample IC Compiler Script for the UPF Flow

Invoke IC Compiler

```
##open design ddc
read_ddc design.ddc
save_mw_cel -as floorplan
save_upf out.upf
link
link_physical
## derive pg connections
derive_pg_connection -create_net/reconnect

## floorplan steps
read_io_constraints design.ios

## Initialize_floorplan
create_voltage_area -power_domain PD1 -guard_band_x 1 -guard_band_y 1
check_mv_design -verbose

##Power plan and secondary power pin routing steps
create_rectangular_rings -nets {VDDGS VDDMS}
create_rectangular_rings -nets {VSS }
create_power_straps -direction vertical -start_at x \
-nets {VSS VDD VDDI} -layer M6 -width 2
create_power_straps -direction horizontal -{VSS VDD VDDI ...} \
-layer M5 -width 2
preroute_standard_cells -mode net \
                        -net VDDG \
                        -h_layer M5 \
                        -v_layer M6 \
                        -h_width 0.38 \
                        -v_width 0.38

## Switch cell mapping and insertion
map_power_switch -domain top -lib_cell HDRSODHVT <name_of_switch>
add_header_footer_cell_array -lib_cell \
"name_of_switch"-orientation N =x 4.2 -y 3.6 -prefix "inst_header"

##Connect switch cells in daisy/hfn mode
connect_power_switch -source inst_on -port_name on -mode hfn \
-verbose

##Implementation steps
place_opt
clock_opt
route_opt
```

Sample IC Compiler Script for UPF Flow

13-43

```

##Chip finishing
set_cell_vt_type -vt_type "VtType1"
set_vt_filler_rule
insert_stdcell_filler -cell_without_metal
check_mv_design

## Write results out
save_upf out2.upf
## The Power Ground Netlist is written
write_verilog -pg -output_net_name_for_tie out.v

## This is the non power ground netlist
write_verilog filename.v

```

Voltage-Area-Aware Capabilities

The `place_opt`, `clock_opt`, and `route_opt` commands are voltage area aware, which means that these commands can be used on multivoltage designs. Other important capabilities include

- Automatic high-fanout synthesis
- Virtual hierarchy routing
- Maximum net length optimization
- Voltage-area-based optimization
- Voltage-area-based clock tree synthesis and optimization
- Voltage-area-based global routing
- Voltage-area-based power and ground net associations
- Voltage-area-based standard-cell filler cell insertion
- Interface logic model (ILM) hierarchical flow
- Multivoltage relative placement capability
- Hierarchical signal integrity flow
- Multicorner-multimode designs

Automatic High-Fanout Synthesis

This capability is part of the `place_opt` command. Buffer trees are built with dedicated subtrees for the fanouts in each voltage area. In particular, automatic high-fanout synthesis selects buffers according to the associated operating condition. Buffers are inserted and placed correctly.

Virtual Hierarchy Routing

The virtual route estimator takes the presence of voltage areas into account when you invoke the `run_router` command. This estimator observes the following constraints:

- Virtual routes for nets connecting cells within a voltage area do not go outside the voltage area.
- Virtual routes for nets connecting cells outside a voltage area detour around the voltage area.
- Virtual routes for nets crossing a voltage area do not zigzag in and out of the voltage area: the number of boundary crossings is minimized.

Also, routing-driven synthesis, such as `max_net_length`, uses voltage-area-aware routing.

For information on the `run_router` command and virtual route estimation, see the man page.

Maximum Net Length Optimization

The maximum net length optimization carried out by the `psynopt` command takes into account the presence of voltage areas, basically routing around them. Routes for nets connecting two cells within a voltage area stay inside the voltage area, and routes for nets crossing into a voltage area do not excessively zigzag in and out of the voltage area.

Voltage-Area-Based Optimization

Cell placement and buffer optimization is one-pass voltage area aware. When the tool buffers a net that crosses voltage areas, the net is divided into multiple segments, each of which is confined within a single voltage area (including the default voltage area), and buffering is confined to the segments. The insertion of new buffers does not introduce additional zigzag paths that cross the voltage area boundaries.

For designs with multiple voltage areas, the tool ensures that a tie cell in one voltage area does not feed a cell in another voltage area. It is recommended that you ensure constant nets remain within their respective voltage area boundaries.

If the `direct_power_rail_tie` attribute has been set on a library pin, instances of this pin do not connect to constant signals by connecting to a tie-off cell. Instead, such pins are connected to generic constant signals so that during power routing these pins can be connected directly to power rails.

Voltage-Area-Based Clock Tree Synthesis and Optimization

Clock tree synthesis and optimization are voltage area aware; they honor the following constraints:

- Sink pins are separated and clustered by voltage areas so that clock subtrees are built for each voltage area.
- A guide buffer is inserted for the set of sink pins for each voltage area to ensure that any subsequent levels of clustering do not mix pins from different voltage areas.

After the clock subtrees are built for each voltage area, clock tree synthesis can proceed in the usual manner, joining the subtrees at the root of the clock net. In addition to the synthesis of the initial clock tree, the preceding constraints that are listed above are also honored by several clock tree optimization techniques, such as buffer relocation, buffer sizing, gate relocation, and delay insertion.

Voltage-Area-Based Global Routing

As with virtual hierarchy routing, global routing also takes the presence of voltage areas into account. The global router and the detail router observe the following constraints:

- Routes for nets connecting cells within a voltage area do not go outside the voltage area.
- Routes for nets connecting cells outside a voltage area detour around the voltage areas.
- Routes for nets crossing a voltage area boundary do not zigzag in and out of the voltage area. The number of boundary crossings is minimized.

Voltage-Area-Based Power and Ground Net Associations

You can assign specific power and ground tie-off nets to voltage areas by using the `set_power_net_to_voltage_area` command. This command allows you to link and unlink these nets to the given voltage areas. Using the `-report` option, you can determine the power and ground nets currently associated with a voltage area. In addition, the CEL consistency checker automatically checks the multivoltage (voltage area) tie-off cells against the assigned settings.

Voltage-Area-Based Standard-Cell Filler Cell Insertion

The `insert_stdcell_filler` command is voltage area aware. By specifying the `-voltage_area` option, you can specify the particular voltage areas in which to insert filler cells in the empty standard cell row sites.

Interface Logic Model Hierarchical Flow

Interface logic models (ILMs) are supported in multivoltage design optimization flows. ILMs are used to model the interface logic of the *mapped* lower-level blocks of a hierarchical design. In general, the noninterface logic is excluded from the model, although you can specify options that allow certain noninterface nets and cells to be included. For detailed information on ILMs, see [Chapter 11, “Interface Logic Modeling”](#).

The following relationships between voltage areas and ILMs are supported:

- A voltage area of the top-level design can contain ILMs. The combined physical area of the ILMs must be less than or equal to the area of the voltage area.
- An ILM at the top level can contain voltage areas.

In both cases, the operating conditions applied to the block (`set_operating_conditions` command) before you create the ILM for the block (`create_ilm` command) are not visible at the top level. At the top level, use the `propagate_constraints` command with the `-operating_conditions` option to propagate the operating conditions placed on the ILM design, cells, and ports, and on the hierarchical pins inside the ILM.

Note that propagation of voltage areas inside an ILM is not supported. Therefore, you must propagate the operating conditions of the hierarchical cells within the voltage area by using the `propagate_constraints` command, or you must reapply the operating conditions from the top level.

The region utilization calculation skips the placeable area or cell area occupied by any ILM contained in the region. That is, only the cell area occupied by the top-level cells and the actual placeable area (region area minus the area occupied by the ILMs) are used in the calculation. If the entire region is occupied by ILMs, the placeable area for the top-level cells is zero, and region utilization is reported as zero.

Level shifters are kept only if they are an actual part of the interface logic. Those level shifters past the first registers of a block are not retained. For level shifters that become part of the ILM, the leaf cells connected to the input and output nets of the level shifter are also retained, as well as the nets. You can use the `check_level_shifter` command to determine whether any output nets of the level shifters are floating.

Multivoltage Relative Placement Capability

You can use the relative placement capability with multivoltage designs.

First, you group selected cells of a voltage area into a relative placement grouping. You can then place them as a single structure within the voltage area. The flow is identical to a typical relative placement flow, and the entire relative placement feature set is available.

However, you should be aware of these limitations:

- A relative placement group should not span voltage areas. If it is necessary to span voltage areas, break the relative placement group into multiple relative placement groups, each contained within its voltage area.
- In situations where a relative placement group is placed very close to a voltage area boundary, legalization might place part of the relative placement outside the multivoltage area. When this happens, the relative placement group is broken in a way that respects the voltage area constraint.

Relative placement is most often applied to physical datapaths. It provides a way for you to create structures in which you specify the relative column and row positions of instances with respect to each other. During placement and legalization, these structures are preserved, and the cells in each structure are placed as a single entity.

Hierarchical Signal Integrity Flow

The hierarchical signal integrity flow can handle multivoltage designs, that is, designs with voltage areas. You do not have to run any special commands or set any variables to enable this capability. As with all signal integrity flows, you enable signal integrity functionality by running the command `set_si_options` with the appropriate options.

Multicorner-Multimode Designs

Multivoltage designs support multicorner-multimode, concurrent optimization. For information about multicorner-multimode functionality and usage, see [Chapter 14, “Using Multicorner-Multimode Technology”](#).

Placing and Optimizing the Design

The `place_opt` command, while placing and optimizing the locations of the cells of the design, honors the power supply intent specified in the UPF file. The tool places level shifters and isolation cells near the boundaries of defined voltage areas. No buffering is done between the placed level shifter and the edge of the voltage area.

To perform always-on synthesis, the tool first identifies the always-on paths connecting the drive pins in the always-on power domains to the always-on load pins in the shut-down power domains. Always-on paths can be determined only if the tool finds always-on pins in the shut-down power domains. The enable pins of the special dual-power cells and the input pins of the single-powered, standard cells placed in always-on power wells are recognized as always-on pins. You can manually mark other pins as always-on if needed.

Once the tool has established the always-on paths, it carries out optimizations appropriate to these paths, such as sizing and buffering. The tool uses ordinary cells for the portion of the paths outside the shut-down power domain. Inside the shut-down power domains, the tool optimizes, using cells that you have specially marked as always-on.

The tool uses the following rules during optimization:

- Always-on logic is mapped only with always-on cells.
- Always-on cells are used only in the shut-down power domains.
- Any gate with input pass gate pins that is driven by an always-on cell is swapped with a functionally equivalent cell that does not have pass gate pins at the inputs.

Note:

After completing always-on synthesis, you can determine the always-on logic by using the `get_always_on_logic` command. This command shows you the nets and cells on the always-on paths of the design.

Handling Isolation Cells and Always-On Clock Paths During Clock Tree Synthesis

When performing clock tree synthesis, both the `compile_clock_tree` and `optimize_clock_tree` commands honor the isolation cells in the clock path and perform synthesis on the always-on clock paths.

To ensure correct isolation between shutdown and power-up power domains, clock tree synthesis does not insert buffers between the isolation cells and the power domain boundary.

The tool uses the following approach during clock tree synthesis on the always-on clock paths:

- Insert dual-power always-on cells in the shutdown power domain.
- All the buffers inserted on the always-on paths are dual-power always-on cells, and these cells are in the powered-down power domain.
- The always-on cells are always inserted in the correct power domain, that is, the always-on region.
- Ensure that redundant dual-power always-on cells are not inserted in the powered-down region or power domain.

Alternatively, you create always-on islands in the shutdown power domain and the tool during clock tree synthesis, inserts buffers in these always-on islands. This ensures that the always-on paths are not disturbed during clock tree synthesis.

Note:

To avoid long runtimes, before performing clock tree synthesis, you should manually insert buffers at the top level if your clock path at the top level traverses from the output of one module to the input of another module.

It is recommended that you use the `check_mv_design` command before and after the clock tree synthesis process.

The following example shows the usage of the `check_mv_design` command before and after the clock tree synthesis process.

```
open_mw_lib library
open_mw_cel cel
remove_clock -all
create_clock -name clk -period 10 -waveform {0.0 2.0 4.0 6.0}
set_clock_tree_options -root [get_ports clk] \
-max_transition 0.5 -reference_per_level {BUF2 INV} -target_skew 0.1
check_mv_design
compile_clock_tree -clock_tree clock_name
check_mv_design
save_mw_cell -as post_cts
```

Checks and Reporting Functions for Multivoltage Designs

IC Compiler supports several checks and reporting functions for multivoltage designs. You can use these to check for the following:

- List operating conditions and P, V, T values
- Report different operating conditions and P, V, T values set on different blocks
- Ensure that each block operates at its specified P, V, T values and operating condition
- Report or check for any target library subsetting used in the multivoltage design flow
- Ensure that each physical voltage area corresponding to a specific power domain has the same operating condition

Multivoltage Specific Checks

You use the `check_physical_design -for_mv` command to check the validity of the constraints that you specify. This command requires that the floor plan is loaded and the Milkyway CEL view is open. You can get more elaborate details using this command with the `-verbose` option in combination with the `-for_mv` option. This command analyzes the design and performs the following multivoltage specific checks:

- Site rows of voltage area that contain the site types as required by the target library cells
- Voltage areas with missing guard bands
- Absence of the associated power domain for voltage area
- Cells having fixed placement that lies outside the parent voltage area
- The report and count of special cells such as isolation cells and level-shifter cells in the design

- Special cells such as isolation cells and level-shifter cells that are fixed or inside a relative placement block
- Designs that have both always-on strategies defined
- Utilization of always-on power wells to determine the size of the power wells

The following is a list of reporting commands that support multivoltage designs.

- `report_operating_conditions`
- `report_target_library_subset`
- `check_mv_design`
- `check_target_library_subset`
- `check_isolation_cells`
- `report_cell`
- `check_design`
- `report_voltage_area`
- `report_timing`
- `report_delay_calculation`
- `report_hierarchy`

Many of these reports have been enhanced to provide the process, voltage, and temperature (PVT) values of the operating environment of the blocks. For information on these commands, see the appropriate man pages.

Non-UPF Flow For Multivoltage Designs

This section describes the non-UPF flow for multivoltage designs. To invoke IC Compiler in non-UPF mode use the `-non_upf_mode` option of the `icc_shell` command.

Using Power Domains

In the Design Compiler to IC Compiler flow, power domains and power domain objects are created in Design Compiler before compiling the design. You can output the compiled design and power domain data as a `.ddc` file, or write it directly into the Milkyway database, or write out Verilog and constraint script files. IC Compiler can process all three types of output, automatically transferring the power domain objects into the Milkyway database.

Power Domain Commands

The principal power domain commands are

- `create_power_domain` – use to create a power domain; arguments include `domain_name` and the options `-power_down`, `-power_down_control`, `-power_down_ack`, and `-object_list` (of hierarchical cells).
- `create_power_net_info` – use to specify the abstract power or ground net objects; arguments include `power_net_name` and the options `-power`, and `-gnd`.
- `connect_power_domain` – use to connect power and ground nets to power domains; arguments include `-object_list` (of power domains), `-primary_power_net`, `-primary_ground_net`, `-backup_power_net`, `-backup_ground_net`, `-internal_power_net`, and `-internal_ground_net`.
- `connect_power_net_info` – use to specify exceptional power net hookups; arguments include `power_net_name` and the options `-object_list` (of leaf cells intended for exceptional power net connections), `-power_pin_name` (of the leaf cells).

Other power domain commands include `remove_power_domain`, `remove_power_net_info`, `report_power_domain`, `report_power_net_info`, `disconnect_power_net_info`. For a detailed discussion on all the above commands and how to use them, see both the man pages and the multivoltage chapter in the *Power Compiler User Guide*.

In a third-party flow, in which the design is not synthesized using Design Compiler, you must create the power domains in IC Compiler. However, you use the same commands as listed previously and in the same way. Refer to the man pages and the multivoltage chapter in the *Power Compiler User Guide*.

Connecting Power and Ground Nets

The specifications you provide with the `create_power_domain`, `create_power_net_info`, and `connect_power_domain` commands are sufficient to enable the tool to connect the power and ground nets of a power domain. Use the `derive_pg_connection` command to connect the power and ground nets of the power domains automatically. Use this command before you run any of the optimizations. This command has similar behavior in the default and the non-UPF mode. For more details, See [“Connecting Power and Ground Nets” on page 13-21](#).

Sample IC Compiler Script For Non-UPF Flow

[Example 13-4](#) provides a script with all the steps required in the physical synthesis phase of the multivoltage design flow. This design uses multiple NLDM libraries. The script assumes that two power domains, MV_BLOCK1 and MV_BLOCK2, have already been defined for the design.

Example 13-4 IC Compiler Multivoltage Non-UPF Flow

```
## Invoke icc_shell
## Create (and open) Milkyway library with tech file
## and reference library names

create_mw_lib MWDL -tech tech.tf \
    -mw_reference_library "$MWRL_stdcell $MWRL_ls" -open

set mw_power_net VDD
set mw_ground_net VSS
set mw_power_port VDD
set mw_ground_port VSS

set search_path "$. ./lib ./src ./db ./scripts $search_path "
set target_library " \
    max_v108_t125.db \
    max_v132_t125.db \
    max_v70_t125.db \
    max_v85_t125.db \
    max_v95_t125.db \
    slow_ls.db "

set link_library " * $target_library "

## Read in logic design
read_verilog ./netlist.v
current_design ${block_name}
link

## Optional step, correct unit site names to match techfile
set mw_site_name_mapping "unit_nsm unit"

## Read in physical pin locations
read_def ./${block_name}.def

link
link_physical

source -verbose -echo ./${block_name}.SDC
```

Example 13-4 IC Compiler Multivoltage Non-UPF Flow (Continued)

```
## Create power net information and power domains
create_power_net_info -power VDD
create_power_net_info -power VDDH
create_power_net_info -power VDDS
create_power_net_info -ground VSS

create_power_domain TOP
connect_power_domain TOP \
    -primary_power_net VDD \
    -primary_ground_net VSS

create_power_domain MV_BLOCK1 \
    -object_list [get_cell -hier* -filter "@ref_name == BLOCK1"] \
    -power_down
connect_power_domain MV_BLOCK1 \
    -primary_power_net VDD \
    -internal_power_net VDDS \
    -primary_ground_net VSS

create_power_domain MV_BLOCK2 \
    -object_list [get_cell -hier* -filter "@ref_name == BLOCK2"] \
    -power_down
connect_power_domain MV_BLOCK2 \
    -primary_power_net VDDH \
    -primary_ground_net VSS

## Create voltage areas using bounding box method
create_voltage_area -power_domain MV_BLOCK1 \
    -coord {387 453 820 820} \
    -guard_band_x 5 \
    -guard_band_y 5 \

create_voltage_area -power_domain MV_BLOCK2 \
    -coord {387 52 820 340} \
    -guard_band_x 5 \
    -guard_band_y 5 \
```

Example 13-4 IC Compiler Multivoltage Non-UPF Flow (Continued)

```
## Check to make sure all voltage areas have been created
report_voltage_area -all

derive_pg_connection -tie
## Your libraries must contain power/ground pin definitions when using the
## following command
derive_pg_connection -reconnect -verbose

check_mv_design

place_opt

check_legality
check_mv_design -power_nets

report_change_list
change_names -rule verilog -hierarchy
write_verilog icc_placed.v
write_def -output icc_placed.def
write_script -output icc_placed.wscr
save_mw_cel -as icc_placed
exit
```

14

Using Multicorner-Multimode Technology

This chapter describes the IC Compiler support for multicorner-multimode technology.

This chapter discusses the following topics:

- [Basic Multicorner-Multimode Concepts](#)
- [Basic Multicorner-Multimode Flow](#)
- [Setup Considerations](#)
- [Multicorner Library Considerations](#)
- [Using Multicorner-Multimode Commands](#)
- [Multicorner-Multimode Script Example](#)

Basic Multicorner-Multimode Concepts

Present-day designs are often required to operate under multiple operating conditions (“corners”) and in multiple modes. Such designs are referred to as multicorner or multimode or multicorner-multimode designs. These designs can involve numerous operating condition corners and many modes. IC Compiler employs a truly concurrent, multi-scenario method to analyze and optimize these designs across all design corners and modes of operation.

Scenario Definition

A *scenario* is a mode or corner or a combination of both that can be analyzed and optimized.

A *mode* is defined by a set of clocks, supply voltages, timing constraints, and libraries. It can also have annotation data, such as SDF or parasitics files.

A *corner* is defined as a set of libraries characterized for process, voltage, and temperature variations. Corners are not dependent on functional settings; they are meant to capture variations in the manufacturing process, along with expected variations in the voltage and temperature of the environment in which the chip will operate.

You create scenarios by using the `create_scenario` command.

Concurrent Multicorner-Multimode Optimization and Timing Analysis

Concurrent multicorner-multimode optimization works on the worst violations across all scenarios, eliminating the convergence problems observed in sequential approaches. Optimization is performed for DRC, setup and hold, or only setup. Concurrent hold-only optimization for preroute designs (that is, `place_opt` and `psynopt` hold-only optimization) is supported, as well as concurrent preroute and postroute hold fixing when you use the `-effort high` option with the `route_opt` command.

Timing analysis is carried out on all scenarios concurrently, and costing is measured across all scenarios for timing and DRC. As a result, the timing and constraint reports show worst-case timing across all scenarios.

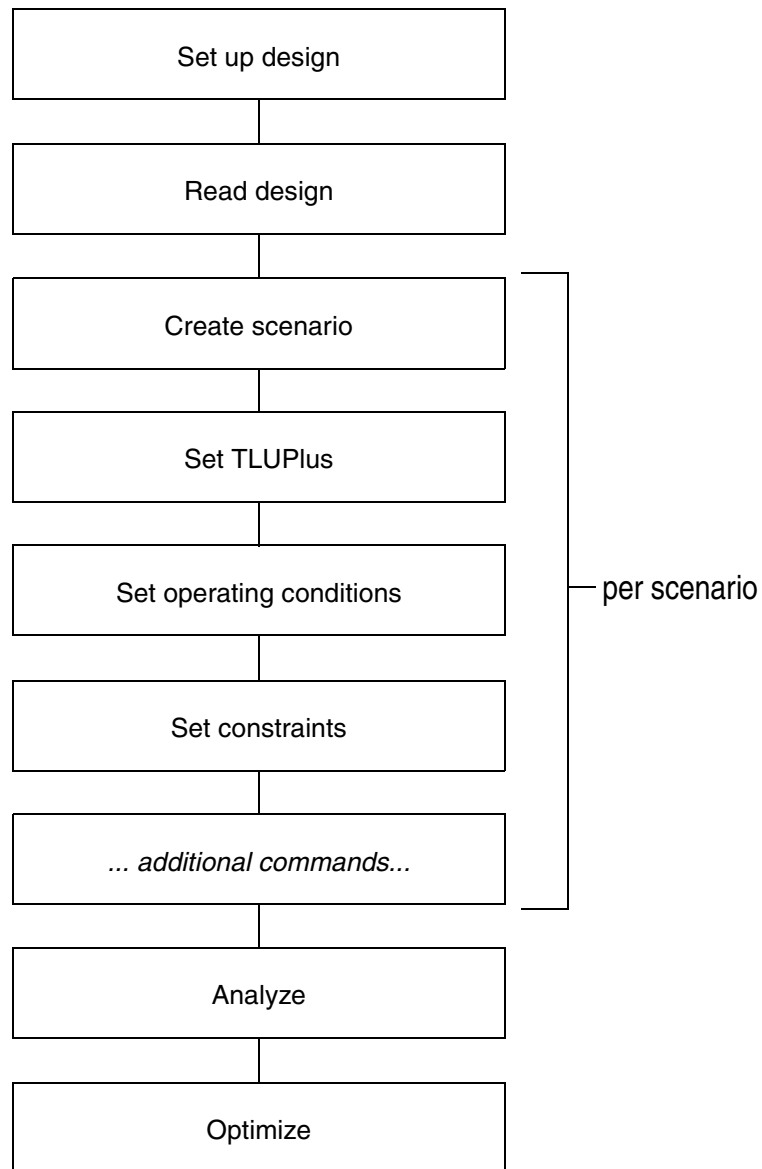
Timing analysis can be performed in one of two methods:

- Traditional min/max methodology
- Early-late analysis, such as that in PrimeTime, utilizing the on-chip variation (OCV) switch in the `set_operating_conditions` command.

Basic Multicorner-Multimode Flow

Figure 14-1 shows the basic multicorner-multimode flow.

Figure 14-1 Basic Multicorner-Multimode Flow



As shown in [Figure 14-1](#), a scenario definition usually includes commands that specify the TLUPlus libraries, operating conditions, and constraints. However, other commands can be included. For example, for multimode scenarios, you can use the `set_max_leakage_power` command to control power leakage on a per-scenario basis or the `read_sdf` command to set the correct net RC and pin-to-pin delay information in the respective scenarios.

Setup Considerations

Multicorner-multimode scenarios support

- Up to one hundred TLUPlus files (minimum and maximum)
- Minimum and maximum libraries, or multivoltage libraries

IC Compiler uses the nominal process, voltage, and temperature (PVT) values to group the libraries into different sets. Libraries with the same PVT values are grouped into the same set. For each scenario, the PVT of the maximum operating condition is used to select the appropriate set.

To allow for temperature scaling, the TLUPlus files must contain the `GLOBAL_TEMPERATURE`, `CRT1`, and (optionally) `CRT2` variables. The following example is an excerpt from a TLUPlus file:

```
TECHNOLOGY = 90nm_lib
GLOBAL_TEMPERATURE = 105.0
CONDUCTOR metal8 {THICKNESS= 0.8000
    CRT1=4.39e-3 CRT2=4.39e-7
...

```

The TLUPlus file settings, which you specify by using the `set_tlu_plus_files` command, must be made explicitly for each scenario. If a TLUPlus setup is not correct, an error similar to the following is issued:

```
Error: tlu_plus files are not set in this scenario s1.
      RC values will be 0.

```

Also, the design's operating condition must be set for each scenario. If it is not defined, MV-020 and MV-21 warnings similar to the following are issued:

```
icc_shell> set_operating_conditions SLOW_95 -lib max_v95_t125
icc_shell> create_scenario s1
Warning: Any existing scenario-specific constraints
         are discarded. (MV-020)
icc_shell> report_timing
Warning: No operating condition was set inscenario s1 (MV-021)

```

Multicorner Library Considerations

To handle libraries appropriately for the multicorner-multimode design flow, you should consider the topics discussed in the following sections:

- [Link Libraries With Equal Nominal PVT Values](#)
- [Distinct PVT Requirements](#)
- [Unsupported k-factors](#)
- [Automatic Detection of Driving Cell Library](#)
- [Relating the Minimum Library to the Maximum Library](#)
- [Unique Identification of Libraries Based on File Names](#)

Link Libraries With Equal Nominal PVT Values

The link library list is a list of all of the libraries that are to be used for linking the design for all scenarios. Because you can use several libraries with any specific scenario, such as a standard cell library and a macro library, IC Compiler automatically groups the libraries in the link library list into sets. The tool also identifies the set of link libraries that must be used with each scenario.

The library grouping is based on their PVT values. Libraries with the same PVT values are grouped into the same set. The tool uses the PVT value of a scenario's maximum operating condition to select the appropriate library set for the scenario.

When the tool cannot find a suitable cell in any of the specified libraries, it reports a warning message as follows:

```
Error: cell TEST_BUF2En_BUF1/Z (inx4) is not characterized
      for 0.950000V, process 1.000000,
      temperature -40.000000. (MV-001)
```

When you encounter a MV-001 error in your flow, verify that the operating conditions and library setup are right. If you do not correct this error, optimization is not performed.

Link Library Example

The following example table shows how the library linking scheme works.

Table 14-1 shows the libraries in the link library list, their nominal PVT values, and the operating condition (if any) specified in each library. The design has instances of combinational, sequential, and macro cells.

Table 14-1 Link Libraries With PVT and Operating Conditions

Link library (in order)	Nominal PVT	Operating conditions in library (PVT)
Combo_cells_slow.db	1/0.85/130	WORST (1/0.85/100)
Sequentials_fast.db	1/1.30/100	None
Macros_fast.db	1/1.30/100	None
Macros_slow.db	1/0.85/130	None
Combo_cells_fast.db	1/1.30/100	BEST (1/1.3/130)
Sequentials_slow.db	1/0.85/130	None

To create a scenario s1 with the cell instances linked to the Combo_cells_slow, Macros_slow, and Sequential_slow libraries, you run

```
icc_shell> create_scenario s1
icc_shell> set_operating_conditions -max WORST -library Combo_cells_slow.db
```

Note that providing the `-library` option in the `set_operating_conditions` command merely helps the tool identify the correct PVT for the operating conditions. The PVT of the maximum operating condition is used to find the correct matches in the link library list during linking.

Using this linking scheme, you can link libraries that do not have operating condition definitions. The scheme also provides the flexibility of having multiple library files (for example, one for standard cells, another for macros, and so forth).

Inconsistent Libraries Warning

When you use multiple libraries, if library cells with the same name are not functionally identical or do not have identical sets of library pins with the same name and order, a warning is issued, stating that the libraries are inconsistent.

Distinct PVT Requirements

If the maximum libraries associated with each corner (scenario) do not have distinct PVT values, the cell instances are linked incorrectly and results in incorrect timing values. This happens because the nominal PVT values, that are used to group the link libraries into sets, group the maximum libraries of different corners into one set. Consequently, the cell instances are linked to the first cell with a matching type in that set (for example, the first AND2_4), even though the `-library` option is specified for each of the scenario-specific `set_operating_conditions` commands. That is, the `-library` option locates the operating condition and its PVT but not the library to link.

The following two tables and the following script demonstrate the problem:

Table 14-2 shows the libraries in the link library, listed *in order*, their nominal PVT; and the operating condition that is specified in each library.

Table 14-2 Link Libraries With PVT and Operating Conditions

Link library (in order)	Nominal PVT	Operating conditions in library (PVT)
Ftyp.db	1/1.30/100	WORST (1/1.30/100)
Typ.db	1/0.85/100	WORST (1/0.85/100)
TypHV.db	1/1.30/100	WORST (1/1.30/100)
Holdtyp.db	1/0.85/100	BEST (1/0.85/100)

Table 14-3 and the script commands that follow show the operating condition specification for each of the scenarios.

Table 14-3 Scenarios and Their Operating Conditions

	Scenarios			
	s1	s2	s3	s4
Max Opcond (Library)	WORST (Typ.db)	WORST (TypHV.db)	WORST (Ftyp.db)	WORST (Typ.db)
Min Opcond (Library)	None	None	None	BEST (HoldTyp.db)

```

create_scenario s1
set_operating_conditions WORST -library Typ.db
create_scenario s2
set_operating_conditions WORST -library TypHV.db
create_scenario s3
set_operating_conditions WORST -library Ftyp.db
create_scenario s4
set_operating_condition \
    -max WORST -max_library Typ.db \
    -min BEST -min_library HoldTyp.db

```

The tool groups the Ftyp.db, and TypHV.db libraries into a set with Ftyp.db as the first library in the set. Therefore, the cell instances in scenario s2 are not linked to the library cells in TypHV.db, as intended. Instead, they are linked to the library cells in the Ftyp.db library (assuming that all the libraries include the library cells required to link the design).

Ambiguous Libraries Warning

When you use multiple libraries, if any of the libraries with same-name cells have the same nominal PVT, a warning is issued, stating that the libraries are ambiguous. The warning also states which libraries are being used and which are being ignored.

Unsupported k-factors

Multicorner-multimode design libraries do not support the use of k-factor scaling. Therefore, the operating conditions that you specify for each scenario must match the nominal operating conditions of one of the libraries in the link library list.

Automatic Detection of Driving Cell Library

In multicorner-multimode flow, the operating condition setting is different for different scenarios. To build the timing arc for the driving cell, different technology libraries are used for different scenarios. You can specify the library using the `-library` option of the `set_driving_cell` command. But specifying the library is optional because the tool can automatically detect the driving cell library.

When you specify the library using the `-library` option of the `set_driving_cell` command, the tool searches for the specified library in the link library set. If the specified library exists it is used. If the specified library does not exist in the link library, the tool issues the UID-993 error message as follows:

```
Error: Cannot find the specified driving cell in memory.(UID-993)
```

When you do not use the `-library` option of the `set_driving_cell` command, the tool searches all the libraries for the matching operating conditions. The first library in the link library set, that matches the operating condition is used. If no library in the link library set matches the operating condition, the first library in the link library set, that contains the matching library cell is used. If no library in the link library set contains the matching library cell, the tool issues the UID-993 error message.

Relating the Minimum Library to the Maximum Library

The `set_min_library` command is not scenario-specific. This implies that if you use this command to relate a minimum library to a particular maximum library, that relationship applies to all scenarios.

Table 14-4 Unsupported Multiple Minimum Library Configuration

	Scenarios	
	s1	s2
Max library	Slow.db	Slow.db
Min library	Fast_0yr.db	Fast_10yr.db

For example, you could not relate two different minimum libraries – say, `Fast_0yr.db` and `Fast_10yr.db` – with the maximum library, `Slow.db`, in two separate scenarios. The first minimum library you specify would apply to both scenarios. [Table 14-4](#) shows the *unsupported* configuration.

Note, however, that a minimum library can be associated with multiple maximum libraries. As shown in the [Table 14-5](#) example, the minimum library `Fast_0yr.db` is paired with both the maximum library `Slow.db` of scenario 1 and the maximum library `SlowHV.db` of scenario 2.

Table 14-5 Supported Min-Max Library Configuration

	Scenarios	
	s1	s2
Max library	Slow.db	SlowHV.db
Min library	Fast_0yr.db	Fast_0yr.db

Unique Identification of Libraries Based on File Names

Two libraries with the same name can be uniquely identified if their file names, which precede the library names (colon-separated), are unique. For example, the library ABC.db:stdcell (where ABC.db is the library file name and stdcell is the library name) is identifiable with respect to the library DEF.db:stdcell.

However, two libraries that have the same file name and library name but reside in different directories are not uniquely distinguishable. The following two libraries are not uniquely distinguishable:

```
/remote/snps/testcase/LIB/fast/ABC.db
```

```
/remote/snps/testcase/LIB/slow/ABC.db
```

Using Multicorner-Multimode Commands

The command groups used in multicorner-multimode flows are discussed in the following sections:

- [Scenario Management](#)
- [Multicorner-Multimode Optimization](#)
- [Using Scenario Reduction](#)
- [Preroute Hold Fixing](#)
- [Using the Clock Tree Synthesis Scenario](#)
- [Using Interface Logic Models With Multicorner-Multimode Scenarios](#)
- [Reporting Commands for Multicorner-Multimode](#)

Scenario Management

You use the following commands to create and manage scenarios:

- `create_scenario`
- `current_scenario`
- `all_scenarios`
- `set_active_scenarios`

- `all_active_scenarios`
- `set_scenario_options`
- `remove_scenario`
- `set_tlu_plus_files`
- `write_parasitics`
- `save_mw_cel -scenario`

create_scenario

Use this command to create a new scenario in `icc_shell`. When the first scenario is created, all previous scenario-specific constraints are removed from the design and this warning is issued:

```
icc_shell> create_scenario s1
Warning: Any existing scenario-specific constraints
         are discarded. (MV-020)
Current scenario is: s1
```

current_scenario

Use this command to specify the name of the current scenario. You use this command in combination with many `icc_shell` commands. Without arguments, the command returns the name of the current scenario.

Note that the `current_scenario` command merely specifies the focus scenario and that when you define more than one scenario, the tool performs concurrent analysis and optimization across all scenarios, independent of the current scenario setting. The following example highlights usage:

```
icc_shell> create_scenario s1
Warning: Any existing scenario-specific constraints
         are discarded. (MV-020)
Current scenario is: s1
icc_shell> create_scenario s2
Current scenario is: s2
icc_shell> current_scenario
Current scenario is: s2
```

all_scenarios

Use this command to display the scenarios currently defined in the `icc_shell`:

```
icc_shell> all_scenarios
s1 s2
```

set_active_scenarios

Use this command to specify which scenarios the tool should include for analysis and optimization. Scenarios that you consider not relevant in a given optimization context can be ignored, which allows memory recovery and runtime reduction.

all_active_scenarios

Use this command to display the currently active scenarios.

set_scenario_options

This command restricts the current scenario to be optimized for one specific cost, either leakage power or hold time. Set the `-leakage_only` option to true to optimize the current scenario only for leakage power. Set the `-hold_only` option to true to fix all the hold delays in the current scenario. Restricting the optimization to one specific cost yields you faster results. The `-hold_only` and the `-leakage_only` options are mutually exclusive. IC Compiler issues an error message if you use both the options.

When you set the `-leakage_only` option to true, the tool considers only the leakage constraints and ignores all other constraints such as the maximum and minimum delay and design rule checks. To restrict the current scenario to leakage power optimization, use this command as follows:

```
icc_shell> set_scenario_options -leakage_only true
```

To fix the hold delays in the current scenario, set the `-hold_only` option to true. When you set this option, the tool ignores the setup delay, design rule checks, and so on during the optimization process. Because all other constraints are ignored, the hold-fixing process for the current scenario completes faster. With the `-hold_only` option set to true, the tool honors the leakage constraints set on the scenario only if the leakage consideration does not adversely affect the hold delay and the effect on the setup delay is not a concern. To fix the hold delays on the current scenario, use this command as follows:

```
icc_shell> set_scenario_options -hold_only true
```

set_tlu_plus_files

Use this command to set the TLUPlus files in the scenario specified by the `current_scenario` command. Each scenario must have a set of TLUPlus files specified, or an error is reported:

```
Error: tlu_plus files are not set in this
      scenario <name>. RC values will be 0.
```

write_parasitics

Use this command to write out parasitics for the current scenario:

```
icc_shell> write_parasitics -format sbpf -output s1.sbpf
icc_shell> write_parasitics -format spef -output s1.spef
```

save_mw_cel -scenario

Use this command to save a design and its scenarios in Milkyway format.

```
icc_shell> save_mw_cel -scenario s1 -as cel_s1
Information: Saved design named cel_s1. (UIG-5)
```

You can save multiple scenarios:

```
icc_shell> save_mw_cel -scenario {s1 s2} -as cel_s1_s2
Information: Saved design named cel_s1_s2. (UIG-5)
```

If you do not specify a scenario, all scenarios are saved.

At present, there is no comparable `open_mw_cel -scenario` command. The design is read in with whatever scenarios were previously saved.

Multicorner-Multimode Optimization

Table 14-6 shows supported multicorner-multimode flows:

Table 14-6 Supported Multicorner-Multimode Flows

Functionality	Preroute placement optimization	Postroute optimization
Multimode	Yes	Yes
Multicorner	Yes	Yes

The `place_opt` and `psynopt` commands are fully supported in both multimode and multicorner flows.

The `route_opt` and `psynopt -on_route` commands are fully supported in both multimode and multicorner flows.

All features of multivoltage designs and related multivoltage commands are supported in both multicorner and multimode flows.

Signal integrity analysis and optimization are supported in both multicorner and multimode flows. Leakage power optimization is supported only in multimode flows.

The various `route_XXX` commands are supported. However, timing-driven options are supported only for the `route_opt` command.

The `clock_opt` command is not currently supported. You must perform clock tree synthesis sequentially on a per-scenario basis.

Scaling Resistance and Capacitance

Preroute Estimation

Use the `set_delay_estimation_options` command to set the scaling factors for resistance and capacitance for the current scenario. To set the scaling factors for a newly created scenario, use this command soon after you create the scenario.

Use the `report_delay_estimation_options` command to see the scaling factors set for the current scenario.

Postroute Extraction

Use the `set_extraction_options` command to set the resistance, capacitance, and coupling capacitance scaling values for the current scenario. The following options can be specified for the current scenario:

- `-max_res_scale`
- `-min_res_scale`
- `-max_cap_scale`
- `-min_cap_scale`
- `-max_ccap_scale`
- `-min_ccap_scale`

Use the `report_extraction_options` command to check the values set for the current scenario. Use the `extract_rc` command to see the effect of the scaling on the design's parasitics.

For more information, see the man pages for these commands.

Filtering Coupling Capacitances

The tool supports coupling capacitance filtering across scenarios. After you have defined each scenario along with its TLUPlus files and operating conditions, do the following steps:

- `set_si_options -delta_delay true`
- `route_opt`
- `extract_rc -coupling_cap`
- `report_timing -scenario [all_scenarios]`

With signal integrity options set, the `route_opt` command takes into account coupling capacitance effects across the nets. In multicorner-multimode flows, the tool's extraction engine generates the parasitics per corner from the TLUPlus and operating conditions pair values. To model the coupling capacitance effects during extraction, you must use the `-coupling_cap` option of the `extract_rc` command.

Using Scenario Reduction

During concurrent analysis and optimization, you can significantly reduce memory usage and runtime by limiting the number of active scenarios to those that are “essential” or “dominant.” An essential or dominant scenario has the worst slack among all the scenarios for at least one of its constrained objects. (Constrained objects can include delay constraints associated with a pin, the DRC constraints for a net, leakage power, and so on.) Any scenario for which none of its constrained objects is the worst slack value is a dominated scenario.

Scenario reduction analysis is always based on the current state of the design – that is, the design's current placement, routing, and clock tree state. In general, restricting concurrent analysis and optimization to a subset of dominant scenarios does not lead to significant QoR degradation.

There are two methods for accomplishing scenario reduction:

- User-controlled scenario reduction

In this method, you use the `get_dominant_scenarios` command to establish the list of dominant scenarios from among the currently active scenarios. To make this set the active set of scenarios, run the `set_active_scenarios [get_dominant_scenarios]` command. All subsequent commands work with this subset of active scenarios until you run the `set_active_scenarios [get_dominant_scenarios]` command again. It is possible that the list of dominant scenarios could change at this point.

If you want to control the set of scenarios that the `get_dominant_scenarios` command analyzes, use the `-scenarios list_of_scenarios` option. This list can include both active and inactive scenarios. The command finds the dominant set from the specified list.

- Automatic scenario reduction during core command execution

If you set the variable `mcm ENABLE_HIGH_CAPACITY_FLOW` to true (default is false), the `place_opt`, `clock_opt`, and `route_opt` commands automatically determine the set of dominant scenarios at particular points in their flow and activate these scenarios. During the execution of basic commands, the last set of dominant scenarios is active.

The set of active scenarios immediately after a core command runs might be a subset of the scenarios that were active immediately before the command was run. A message at the end of the command lists both sets of scenarios. If necessary, you can always reset the list of active scenarios by using the `set_active_scenarios` command.

You can control the effort level of scenario reduction with the `mcm HIGH_CAPACITY_EFFORT_LEVEL` variable. Allowed values range from 0 (the default) to 10. The tool considers all active scenarios in the reduction effort when you use the default value. As you increase the effort level, the scenario reduction method becomes more aggressive, adjusting its criteria to allow more scenarios to be removed from the dominant scenario set. This can lead to further memory and runtime reductions, but it can also imply that fewer violations are fixed during optimization.

For more information on scenario reduction, see the appropriate man pages.

Preroute Hold Fixing

Concurrent, preroute hold fixing is fully supported across all scenarios in multicorner-multimode designs. Hold fixing is done in the standard way, using the `set_fix_hold` and `set_fix_hold_options` commands. In the `place_opt` flow, both setup and hold optimizations are carried out concurrently when you run the `place_opt` command. The `route_opt` flow requires that you use the `-effort high` option with the `route_opt` command to obtain concurrent setup and hold optimizations.

Using the Clock Tree Synthesis Scenario

To perform clock tree synthesis in a multicorner-multimode design, you must define a scenario that is used specifically for clock tree synthesis. You might name this scenario CTS. In the clock tree synthesis scenario, you select a set of clocks that need to be synthesized and put the constraints for these clocks into a single constraints file, which is then used to run clock tree synthesis or clock tree optimization.

To identify this scenario as the clock tree synthesis scenario, use the `set_cts_scenario` command. For example, `set_cts_scenario CTS`. Then you use the `set_active_scenarios` command to list the scenarios you want to activate for optimization, but not necessarily including the clock tree synthesis scenario.

When you run clock tree synthesis (`clock_opt` command), only the scenario identified as the clock tree synthesis scenario is active. That is, IC Compiler first deactivates all scenarios except the clock tree synthesis scenario. After clock tree synthesis, design information (such as ideal clock latencies, propagated clocks, and so on) is automatically updated in the design scenarios, so that subsequent optimizations (such as `route_opt`) can be performed on all the active scenarios.

To return the clock tree synthesis scenario, use the `get_cts_scenario` command.

Using Interface Logic Models With Multicorner-Multimode Scenarios

The multicorner-multimode feature is enabled by default when you create an ILM. You can apply multicorner-multimode constraints to a block, perform physical synthesis, and create an ILM for the block. The ILM created can be used in the top-level design.

The following are the requirements when you use ILMs in a multicorner-multimode flow:

- For each top-level scenario, an identically named scenario must be defined in each of the ILM blocks used in the design. An ILM can have additional scenarios that are not used at the top level as long as those scenarios do not have any additional TLUPlus file settings. In other words, the ILM can have additional modes, but not additional parasitic corners (TLUPlus files).
- In a top-level design without multicorner-multimode scenarios, only ILMs without multicorner-multimode scenarios can be used.

For detailed information on using ILMs, including script examples, see [Chapter 11, “Interface Logic Modeling.”](#)

Reporting Commands for Multicorner-Multimode

This section describes commands that you can use for reporting multicorner-multimode designs.

report_scenarios Command

The `report_scenarios` command reports the scenario setup information for multicorner-multimode designs. This command reports all the defined scenarios, the clock tree synthesis scenario, and the leakage-only scenario in the design. The scenario-specific information includes the technology library used, the operating condition, and the TLUPlus files.

The following example shows a report generated by the `report_scenarios` command:

```
*****
Report : scenarios
Design : DESIGN1
scenario(s) : SCN1
Version: A-2007.12-ICC
Date    : Thu Nov 08 20:55:59 2007
*****

All scenarios (Total=4): SCN1 SCN2 SCN3 SCN4
All Active scenarios (Total=1): SCN1
Current scenario      : SCN1
CTS Scenario          : SCN3

Scenario #0: SCN1 is active.
Scenario options:
Has timing derate: No
Library(s) Used:
  technology library name (File: library.db)

Operating condition(s) Used:
  Analysis Type      : bc_wc
  Max Operating Condition: library:WCCOM
  Max Process        : 1.00
  Max Voltage         : 1.08
  Max Temperature: 125.00
  Min Operating Condition: library:BCCOM
  Min Process         : 1.00
  Min Voltage         : 1.32
  Min Temperature: 0.00

Tlu Plus Files Used:
  Max TLU+ file: tlu_plus_file.tf
  Tech2ITF mapping file: tf2itf.map

Number of leakage-only scenario(s): 0
```


Reporting Commands That Support the -scenario Option

Some reporting commands in IC Compiler support the `-scenario` option to report scenario-specific information. You can specify a list of scenarios to the `-scenario` option and the tool reports scenario details for the specified scenarios.

The following reporting commands support the `-scenario` option:

- `report_timing`
- `report_timing_derate`
- `report_clock`
- `report_path_group`
- `report_extraction_options`
- `report_tlu_plus_files`
- `report_constraint`

Commands That Report the Current Scenario

The following reporting commands report scenario-specific details for the current scenario. The header section of the report contains the name of the current scenario. No additional options are required to report the scenario-specific details of the current scenario.

- `report_annotated_check`
- `report_annotated_delay`
- `report_annotated_transition`
- `report_attribute`
- `report_case_analysis`
- `report_clock_gating_check`
- `report_clock_timing`
- `report_clock_tree`
- `report_clock_tree_power`
- `report_crpr`
- `report_delay_calculation`
- `report_delay_estimation_options`
- `report_disable_timing`

- `report_ideal_network`
- `report_internal_loads`
- `report_latency_adjustment_options`
- `report_net`
- `report_noise`
- `report_power`
- `report_power_calculation`
- `report_signal_em`
- `report_timing_derate`
- `report_timing_requirements`
- `report_transitive_fanin`
- `report_transitive_fanout`

Reporting Examples

This section contains sample reports for some of the multicorner-multimode reporting commands.

report_qor

This command reports the QoR for each scenario.

```
*****
Report : qor
Design : DESIGN1
Version: X-2005.12-ICC1.1
Date   : Thu Dec 15 20:55:59 2005
*****
```

```
Scenario 's1'
Timing Path Group 'reg2reg'
-----
Levels of Logic:           33.00
Critical Path Length:      694.62
Critical Path Slack:       -144.52
Critical Path Clk Period:  650.00
Total Negative Slack:      -4533.01
No. of Violating Paths:    136.00
-----
```

```
Scenario 's2'
Timing Path Group 'reg2reg'
-----
Levels of Logic:           32.00
Critical Path Length:      393.61
Critical Path Slack:        61.18
Critical Path Clk Period:  500.00
Total Negative Slack:        0.00
No. of Violating Paths:     0.00
-----
```

report_timing -scenario [all_scenarios]

This command reports timing results for the active scenarios in the design. You can specify a list of scenarios with the `-scenario` option. When the `-scenario` option is not specified only the current scenario is reported.

```

Report : timing
        -path full
        -delay max
        -max_paths 1
Design : DESIGN1
Version: X-2005.12-ICC1.1
Date   : Tue Dec 13 18:46:08 2005
*****

* Some/all delay information is back-annotated.

# A fanout number of 1000 was used for high fanout net computations.

Startpoint: TEST_BUF2En
            (input port clocked by clk)
Endpoint:  TEST1/TEST2_SYN/latch_3
            (non-sequential rising-edge timing check clocked by clk)
Scenario:  s1
Path Group: clk
Path Type: max
Point      Incr      Path      Lib:OC
-----
clock clk (rise edge)                0.00      0.00
clock network delay (propagated)      0.00      0.00
input external delay                 450.00    450.00 f
TEST_BUF2En (in)                      0.00    450.00 f stdcell_typ:WORST
TEST_BUF2En_BUF1/Z (inx4)             9.75    459.75 r stdcell_typ:WORST
U468/Z (inx10)                       10.21    469.96 f stdcell_typ:WORST
TEST_BUF2En_BUF/Z (inx11)             8.74    478.70 r stdcell_typ:WORST
U293/Z (inx11)                        9.30    488.00 f stdcell_typ:WORST
TEST1/TEST2_SYN/U74963/Z (nr2x4)      12.78    500.78 r stdcell_typ:WORST
U31662/Z (inx4)                      10.58    511.37 f stdcell_typ:WORST
TEST1/TEST2_SYN/U75093/Z (aoi21x6)    18.98    530.34 r stdcell_typ:WORST
U42969/Z (nd2x6)                     14.16    544.51 f stdcell_typ:WORST
TEST1/TEST2_SYN/U53046/Z (inx8)       13.35    557.86 r stdcell_typ:WORST
U2765/Z (inx8)                       11.48    569.33 f stdcell_typ:WORST
U32442/Z (inx6)                       7.61    576.94 r stdcell_typ:WORST
U33615/Z (nd2x3)                      18.14    595.09 f stdcell_typ:WORST
U32269/Z (nd2x6)                      8.74    603.82 r stdcell_typ:WORST
TEST1/TEST2_SYN/clk_gate/EN (cklan2x1) 0.00    603.82 r stdcell_typ:WORST

data arrival time                      603.82
clock clk (rise edge)                 650.00    650.00
clock network delay (propagated)      0.00    650.00
TEST1/TEST2_SYN/clk_gate/CLK (cklan2x1) 0.00    650.00 r
library setup time                    -56.25    593.75
data required time                     593.75
-----
data required time                     593.75
data arrival time                      -603.82
-----
slack (VIOLATED)                      -10.07

```

report_constraint

This command reports the constraints of each scenario along with the multi-scenario costs. The `-all_violators` and `-verbose` options of this command are not available in multicorner-multimode flows. Therefore, only the current scenario is reported.

```
*****
Report : constraint
Design : DESIGN1
Scenarios: 0, 1
Version: X-2005.12-ICC1.1
Date   : Tue Dec 13 18:46:10 2005
*****
```

Group (max_delay/setup)	Cost	Weight	Weighted Cost	Scenario
CLK	10.07	1.00	10.07	s1
in2out	372.89	1.00	372.89	s1
in2reg	199.73	1.00	199.73	s1
reg2out	467.99	1.00	467.99	s1
reg2reg	171.16	1.00	171.16	s1
default	0.00	1.00	0.00	s1
CLK	90.60	1.00	90.60	s2
in2out	474.97	1.00	474.97	s2
in2reg	166.88	1.00	166.88	s2
reg2out	326.46	1.00	326.46	s2
reg2reg	0.00	1.00	0.00	s2
default	0.00	1.00	0.00	s2
max_delay/setup			4404.52	

...

Constraint	Multi-Scenario Cost
multiport_net	0.00 (MET)
min_capacitance	0.00 (MET)
max_transition	45.28 (VIOLATED)
max_fanout	150.00 (VIOLATED)
max_capacitance	0.00 (MET)
max_delay/setup	4404.52 (VIOLATED)
critical_range	4404.52 (VIOLATED)
min_delay/hold	0.00 (MET)
max_area	714233.56 (VIOLATED)

report_tlu_plus_files

This command reports the TLUPlus files associations; it shows each minimum and maximum TLUPlus and layer map file per scenario:

```
icc_shell> current_scenario s1
Current scenario is: s1

icc_shell> report_tlu_plus_files
Max TLU+ file: /snps/testcase/s1max.tluplus
Min TLU+ file: /snps/testcase/s1min.tluplus
Tech2ITF mapping file: /snps/testcase/tluplus_map.txt
```

Multicorner-Multimode Script Example

The following script shows the multicorner-multimode scenario capabilities in design placement, clock tree synthesis, and routing.

```
#.....path settings.....

set search_path ". $DESIGN_ROOT $lib_path/dbs \
    $lib_path/mwlibs/macros/LM"
set target_library "stdcell.setup.ftyp.db \
    stdcell.setup.typ.db stdcell.setup.typhv.db"
set link_library [concat * $target_library \
    setup.ftyp.130v.100c.db setup.typhv.130v.100c.db \
    setup.typ.130v.100c.db]
set_min_library stdcell.setup.typ.db -min_version \
    stdcell.hold.typ.db

#.....Milkyway setup.....
#.....load design.....

create_scenario s1
set_operating_conditions -library $lib_path/dbs \
    /stdcell.setup.typ.db:stdcell_typ WORST
set_tlu_plus_files -max_tluplus design.tlup \
    -tech2itf_map layermap.txt
read_sdc s1.sdc

create_scenario s2
set_operating_conditions -library $lib_path/dbs \
    /stdcell.setup.ftyp.db:stdcell_ftyp WORST
set_tlu_plus_files -max_tluplus design.tlup \
    -tech2itf_map layermap.txt
read_sdc s2.sdc

.....
```

```
create_scenario cts
set_operating_conditions -library $lib_path/dbs \
    /stdcell.setup.ftyp.db:stdcell_ftyp WORST
set_tlu_plus_files -max_tluplus design.tlup \
    -tech2itf_map layermap.txt
read_sdc scts.sdc

set_cts_scenario cts
set_active_scenarios {s1 s2 ... }

place_opt
clock_opt
route_opt

report_qor
report_constraints
report_timing -scenario [all_scenarios]
```


15

ECO Flow

An engineering change order (ECO) is a small, incremental change made to a complete or nearly complete chip layout. You can use ECOs to make small logic changes and to fix timing, noise, and crosstalk violations without re-running logic synthesis for the whole design.

This chapter describes the IC Compiler ECO flow in the following sections:

- [ECO Flow Overview](#)
- [ECO Key Features](#)
- [ECO Limitations](#)
- [Inserting Spare Cells](#)
- [Creating ECO Changes and Updating the Design](#)
- [Updating Placement](#)
- [Updating Routing](#)

ECO Flow Overview

IC Compiler provides the following ECO flow:

- Unconstrained ECO flow

Use this flow if your design has not been taped out yet. In this flow, you can add new cells and move or delete existing cells. You don't need to have spare cells in the design.

Note:

The Freeze Silicon ECO flow is not supported. It will be available in a future release. You use this flow if your cell placement is fixed. In this flow, you only change the metal and via mask patterns because all cell placement is fixed. You use spare cells to make any functional design change. Spare cells are logic gates you add to the design before running the ECO process. These spare gates should be appropriately preplaced to minimize metal changes. This type of change is usually only performed after tape-out.

The Freeze Silicon ECO flow requires you to add and place spare cells in advance, in anticipation of possible ECO needs, as described in the next section, [“Inserting Spare Cells” on page 15-3](#). Otherwise, the unconstrained and freeze silicon flows are very similar, consisting of the following steps:

1. Set silicon status.

The default silicon status is unfreeze.

2. Create your ECO changes and add them to your design. You can do this by using a Verilog netlist or Tcl commands. See [“Creating ECO Changes and Updating the Design” on page 15-8](#).
3. Update the placement with the ECO changes. See [“Updating Placement” on page 15-12](#).
4. Update the routing with the ECO changes. See [“Updating Routing” on page 15-13](#).

ECO Key Features

Key ECO features are the following:

- PG/tie net support
For details, see [“Power and Ground \(PG\) Support” on page 15-10](#).
- Physical-only support
For details, see [“ECO on Physical-Only Cells” on page 15-10](#)

ECO Limitations

The `eco_netlist` command, used to read in the Verilog ECO netlist, does not support the following:

- The Freeze Silicon ECO flow
- Multivoltage and UPF flows
- Bus manipulation
- Clock tree synthesis (CTS)
- Scan chains

For `eco_netlist` usage details, see [“Creating ECO Changes and Updating the Design” on page 15-8](#).

In addition to the `eco_netlist` method of ECO implementation, you can use Tcl commands to change your netlist. Note the following Tcl limitation:

- Tcl commands do not support the Freeze Silicon ECO flow.

For Tcl usage details, see [“Creating ECO Changes and Updating the Design”](#).

Inserting Spare Cells

Spare cells are used in the Freeze Silicon ECO flow.

Spare cells accommodate late-arriving design changes. You insert them before or after placement and use them when placement is fixed. With spare cells, you can perform simple logic changes by updating only one or a few metal and via masks, thereby saving the considerable expense of generating a whole new set of silicon-layer masks.

Spare cells are most useful when they are physically near the location of the logic that needs to be changed. Therefore, spare cells need to be dispersed across the chip rather than gathered in one or several tight locations.

To add and place spare cells in a design, do one of the following:

- Manually instantiate the spare cells into the Verilog netlist

Do this before reading the netlist into IC Compiler. There is no RTL description for spare cells; you must instantiate all spare cells. See [“Inserting Spare Cells Using a Verilog Netlist” on page 15-4](#).

- Use the `insert_spare_cells` command

The `insert_spare_cells` command only works for a legalized placement. This command inserts a specified number of specified library cells into a legally placed design. The new cells can be evenly distributed and legalized. These additional cells do not affect existing placement. The command does not remove any existing spare cells. For details, see [“Inserting Spare Cells Using `insert\_spare\_cells`” on page 15-8](#).

These methods are described in the following sections.

Inserting Spare Cells Using a Verilog Netlist

There is no limit to the number of spare cells you can add to the design. This allows you the flexibility to break up the spare cells so that they can be easily scattered or be located near specific logic that might be more likely to require an ECO.

Usually the spare cells are instantiated and grouped in separate hierarchical blocks. However, if you want to place a group of spare cells near specific logic, you can break the hierarchical blocks into multiple blocks. This enables you to physically place a group of spare cells near its related logic.

The spare cell inputs can be connected to logic level high or low. To prevent the gate from oscillating (creating noise and consuming power), tie the inputs to ground or power, as appropriate. The spare gates are usually independent, but you can connect them together to form other functions. In any case, you must instantiate each spare cell in the netlist.

Note that each spare cell block can be used only once within the design.

After you read in the netlist, you can place the cells in the following ways:

- Automatically place the spare cells during placement
- Manually place the spare cells before placement
- Manually place the spare cells after placement

These methods are described in the following sections.

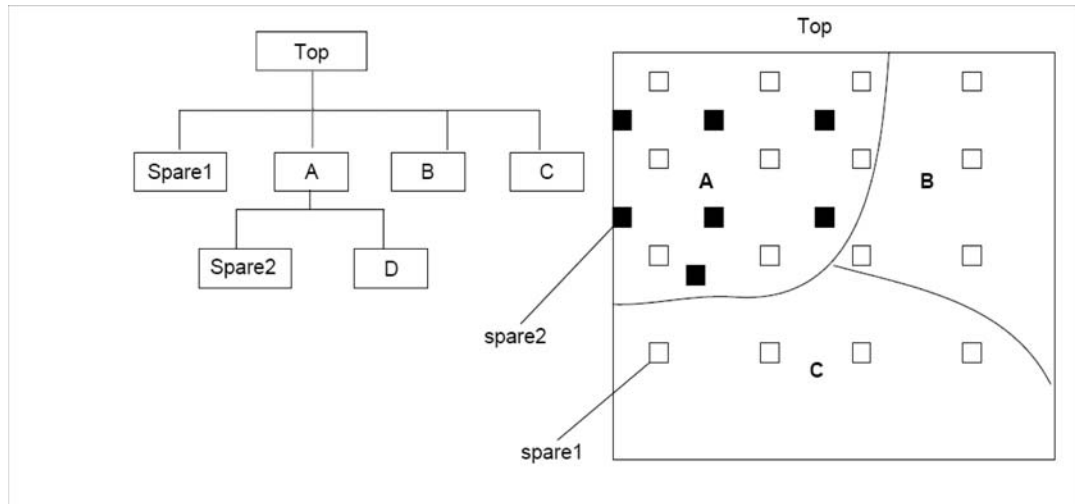
Automatically Distribute the Spare Cells During Placement

Note:

Automatic spare cell placement only works for spare cells that have all inputs tied low/high and output ports that float. Any spare cells that are connected to signal nets (such as sequential cells connected to a clock) cannot be distributed automatically. You need to insert these spare cells manually, as described in the next section.

After you add your spare cells into the Verilog netlist and import the netlist into IC Compiler, the tool will distribute the spare cells automatically during placement. This placement is hierarchically aware, so you can insert spare cells in a logic submodule instead of across the entire design. In this case, when IC Compiler places these spare cells, instead of placing the cells across the entire floorplan, the placement is limited to the area that the submodule covers. Consider the spare cell placement in [Figure 15-1](#).

Figure 15-1 Spare Cells Inserted in a Submodule



In [Figure 15-1](#),

- The spare cells in module Top/spare1 are distributed across the placed area of Top (A, B, and C).
- The module Top/A contains a submodule, spare2, which contains only spare cells. The spare cells in spare2 are placed only in the area in which Top/A is placed.

When you know you are likely to do an ECO on module A, you want some of the known spare cells to be placed only in that module. It is not necessary for 100 percent of these spare cells to be placed in that module; some spare cells can spill over to other regions.

When you want IC Compiler to place the spare cells automatically, keep the following in mind:

- If you apply `dont_touch` to the spare cells, the spare cells will not be removed during optimization.
- You must connect the spare cell inputs to low or high. The tool identifies spare cells automatically by checking the inputs. If the inputs of spare cells are not tied low or high, they are not treated as spare cells and placement may place them in one tight location.
- Set the spare cells to soft-fixed once the spare cells are distributed. With the soft-fixed attribute, they can still be moved slightly.

Manually Distribute the Spare Cells After Placement

If you want to manually control spare cell placement, use the `spread_spare_cells` command.

The `spread_spare_cells` command evenly distributes specified spare cells over a specified region. Manual cell spreading means that the spare cells are not placed automatically during placement; instead, they are placed either before or after placement. This is useful when you want to control the spare cell location.

To manually place spare cells after placement, use the following steps:

1. Apply the `is_spare_cell` attribute to the spare cells.

To do this, you can run the `set_attribute` command. For example,

```
set_attribute [get_cells block1/spare*] is_spare_cell true
```

If you don't apply this attribute, the `spread_spare_cells` command will not work.

2. Apply the `dont_touch` attribute and the `fixed` attribute to all the spare cells.

If you don't set the cell to fixed status, the placement will place all the spare cells into a small tight location and waste the placement area.

3. Run the placement optimization.
4. Remove the `fixed` attribute from spare cells.

If you don't remove the `fixed` attribute, the spare cells cannot be distributed.

5. Run the `spread_spare_cells` command.

Note: The `spread_spare_cells` command accepts collections which means you can pass any collection variable from other Tcl commands. The `spread_spare_cells` command distributes the specified spare cells into the specified region. It derives the location for each of the spare cells but does not perform legalization on the distributed spare cells. You will need to run incremental legalization for these spare cells. The command works for both rectangular and rectilinear areas.

6. Run incremental legalization.

Incremental legalization attempts to maintain the existing placement. It is highly recommended that you run the incremental legalization.

7. Set the spare cells to soft-fixed.

This maintains the spare cell placement location and also allows the cell to be moved slightly, if necessary, later in further optimization. Do this with the `set_attribute` command.

[Example 15-1](#) shows a script that manually places spare cells after regular placement.

Example 15-1 *Manually Place Spare Cells After Placement*

```
source icc.setup
open_mw_lib MDBLIB
open_mw_cel top
set_attribute [get_cells *spares*] is_spare_cell true
set_dont_touch_placement [get_cells *spares*]
place_opt
remove_dont_touch_placement [get_cells spare*]
spread_spare_cells [get_cells spare*] -bbox {{20 20} {999 999}}
legalize_placement -incr
set_dont_touch [get_cells *spares*] true
set_attribute [get_cells *spares*] is_soft_fixed true
clock_opt
route_opt
...
```

Manually Place the Spare Cells Before Placement

[Example 15-2](#) shows how to manually place spare cells before regular placement.

Example 15-2 *Manually Place Spare Cells Before Placement*

```
open_mw_cel top
set_attribute [get_cells spares*] is_spare_cell true
spread_spare_cells [get_cells spare*] -bbox {{5 5} {2000 2000}}
set_attribute [get_cells spares*] is_soft_fixed true
set_dont_touch [get_cells spares*]
place_opt
clock_opt
route_opt
```

Inserting Spare Cells Using `insert_spare_cells`

The `insert_spare_cells` command inserts a specified number of instances of a specified library cell as spare cells in an already legalized design. You can only run this command on a legalized placement. When you insert spare cells in a given hierarchy by using the `-sub_design` option, the spare cells are placed in the rectangular area that encloses all of the cells belonging to the specified subdesign so that the spare cells are closer to those cells. [Example 15-3](#) shows how to use the `insert_spare_cells` command.

Example 15-3 `insert_spare_cell`

```
...
open_mw_cel top
place_opt
insert_spare_cells -lib_cell {INV8 DFF1} -cell_name spares -num_instances 300
set_attribute [get_cells spares*] is_soft_fixed true
set_dont_touch [get_cells spares*] true
```

Note:

Remember to set the spare cells to `soft_fixed` and `dont_touch` after running the `insert_spare_cells` command.

Creating ECO Changes and Updating the Design

For the unconstrained ECO flow, IC Compiler provides two ways to create ECO changes and add them to your design:

- [Updating With a Verilog Netlist](#)
- [Updating With Tcl Commands](#)

The Freeze Silicon ECO flow is not supported. It will be supported in a future release.

In general, use a complete Verilog netlist for major changes and use the Tcl commands for small changes. Both methods are described in the next sections.

Updating With a Verilog Netlist

Use the `eco_netlist` command to import your ECO netlist into IC Compiler. You can also import your ECO netlist with the GUI. From the GUI, choose ECO > ECO by Verilog. The ECO by Verilog dialog box appears. The default cell is the current cell.

When creating the ECO netlist, it is helpful to note instance and master names of the new added cells so that you can check the resulting update.

When you execute `eco_netlist`, the tool reads the ECO Verilog netlist file and updates the top-level design. It will first read in the Verilog file and expand the hierarchical design into flat cells with the name of the top module. It then compares the current design cell with the expanded ECO cell and makes the changes in the design cell. After the flat design is updated, the command will delete the original hierarchy data and rebuild the hierarchy using preservation information from specified Verilog files. The ECO change list will be attached to the ECO history.

Both the command line and the GUI method provide the following options:

- `eco_netlist -by_verilog_file`
Use this option to specify your Verilog ECO file.
- `eco_netlist -compare_pg`
Use this option to perform ECO changes on PG ports and nets.
- `eco_netlist -physical`
Use this option to perform ECO changes on physical-only cells.

[Example 15-4](#) shows basic `eco_netlist` usage.

Example 15-4 Using eco_netlist

```
eco_netlist -by_verilog_file myNewVerilog.v
```

[Example 15-5](#) shows `eco_netlist` usage when your ECO contains PG ports and nets.

Example 15-5 ECO with PG Ports and Nets

```
eco_netlist -compare_pg -by_verilog_file myNewVerilog.v
```

[Example 15-6](#) shows `eco_netlist` usage when your ECO contains physical cells.

Example 15-6 ECO with Physical Cells

```
eco_netlist -physical -by_verilog_file myNewVerilog.v
```

For command details, see the `eco_netlist` man page.

Important:

You should make a back-up copy of the cell before importing the Verilog netlist.

Power and Ground (PG) Support

If your netlist has PG ports and nets and you execute `eco_netlist -compare_pg`, the ECO engine performs changes on all PG ports/nets and considers the new Verilog netlist as golden. When you use this switch, the PG ports in the golden Verilog netlist will override any PG information contained earlier in the Milkyway database.

If your netlist has PG ports and nets and you execute `eco_netlist` without the `-compare_pg` switch, the tool only performs the logical ECO. The golden Verilog file's PG data will have no effect on the existing PG data in the Milkyway database.

ECO on Physical-Only Cells

IC Compiler identifies physical-only cells automatically by their cell type. You do not need to mark these cells in any special way before applying the ECO.

If your golden Verilog netlist contains physical-only cells and you execute `eco_netlist -physical`, the ECO engine recognizes physical-only cells from the Verilog netlist and saves them in the Milkyway database.

Updating With Tcl Commands

IC Compiler provides the following Tcl commands that help you change your netlist. Three of these commands are available from the GUI: `insert_buffer`, `remove_buffer`, and `size_cell`. For command details, see the man pages.

Note:

To make netlist changes with hierarchical engineering change order (HECO) commands instead of Tcl commands, see SolvNet article 019642, "Mapping HECO Commands to IC Compiler Tcl Commands."

Port Operations

You can use Tcl commands to perform the following port operations:

- Add new port/hierarchical port

`create_ports`: creates ports in the current design or its subdesign

- Remove port/hierarchical port

`remove_port`: removes ports from the current design or its subdesign

Net Operations

You can use Tcl commands to perform the following net operations:

- Add new flat/hierarchical net

`create_net`: creates nets in the current design or its subdesign

- Remove flat/hierarchical net

`remove_net`: removes nets from the current design or its subdesign

- Rename flat/hierarchical net

`change_names`: renames nets in the current design or its subdesign

Connection Operations

You can use Tcl commands to perform the following connection operations:

- Connect a flat/hierarchical net with a port/hierarchical port instance

`connect_net`: connects a specified net to specified pins or ports

- Disconnect a port/hierarchical port instance from a flat/hierarchical net

`disconnect_net`: disconnects a specified net from specified pins or ports

Instance Operations

You can use Tcl commands to perform the following instance operations:

- Add new flat/hierarchical cell instance

`create_cell`

- Add new buffer

`insert_buffer`

- Remove flat/hierarchical cell instance

`remove_cell`

- Remove buffer

`remove_buffer`

- Resize flat cell instance

`size_cell`

Example

The following example uses Tcl commands to size cell “block1/U2” from reference “BUFX2” to “BUFX8” and insert a buffer with reference “BUFFX16”.

```
size_cell block1/U2 class/BUFX8
insert_buffer {block1/U2/Z} class/BUFFX16
```

Updating Placement

Once you have performed the functional ECO using `eco_netlist`, the next step is to legalize your ECO changes.

Note:

The ECO engine, which drives the `eco_netlist` command, does not currently support the Freeze Silicon ECO flow. The `eco_netlist` command only supports the unconstrained ECO flow. In this flow the placement is not fixed. You can add new cell instances and remove deleted cells. Legalization derives the locations for new cell instances.

To update placement, use the `legalize_placement -eco -incr` command. The `-eco` option tells the tool to derive locations for new added cells. The `-incr` option tells the tool to move as few cells as possible because most of the cells have already been placed and legalized.

Example

```
.....
open_mw_lib MDBLIB
open_mw_cel top
eco_netlist -by_verilog_file eco.v
legalize_placement -eco -incr
```

Updating Routing

There are several options for routing the ECO placement. You can manually connect the ECO routing when the change is very minor or if the routing is limited to a few layers. When the change is major, use the `route_eco` command which does the following:

- Performs the routing for broken nets, new added nets, deleted nets, obsolete nets, and ripped-up unused sections of changed wires
- Attempts to preserve the critical routes and fix the DRC violations
- Goes through global routing, track assignment, and detail routing

The `route_eco` command has many powerful options that make routing intelligent. For example, it can freeze certain layers or vias and skip global routing or track assignments. This command only routes modified nets, regional based ECO routing (specified areas), and distributed regions based on ECO routing. The following example performs ECO routing by using the options `-auto`, `-search_repair_loop`, `-utilize_dangling_wires`, `-scope`, and `-reroute`.

Example

```
icc_shell> route_eco -auto -search_repair_loop 5 -utilize_dangling_wires \  
            -scope local -reroute modified_nets_first_then_others
```


16

Sign-Off Driven Design Closure

This chapter describes how to reduce timing violations during final sign-off by using IC Compiler sign-off driven design closure.

In general, the implementation engine (IC Compiler) and the sign-off engine (Star-RCXT and PrimeTime) are well correlated with each other. However, small variations can occur because

- PrimeTime and Star-RCXT, as sign-off tools, are more accurate.
- PrimeTime has reduced signal integrity pessimism.
- IC Compiler may have implementation margins.
- The implementation engine (IC Compiler) may have fewer corners.

These differences can cause timing violations during final sign-off. To reduce violations, you should use IC Compiler sign-off driven design closure. In this flow, IC Compiler automatically runs the sign-off tools (PrimeTime and Star-RCXT) to find violations. You can enable IC Compiler to perform optimizations automatically to fix these violations, or you can manually fix the violations.

This chapter describes IC Compiler sign-off driven design closure in the following sections:

- [IC Compiler Sign-Off Driven Design Closure Overview](#)
- [Key Features](#)
- [IC Compiler Sign-Off Driven Design Closure Methodology](#)

- [Usage Guidelines](#)
- [Limitations](#)
- [Troubleshooting and Problem Solving](#)

IC Compiler Sign-Off Driven Design Closure Overview

The starting point for IC Compiler sign-off driven design closure is a completely routed database from IC Compiler. This flow uses two key commands: `run_signoff` and `signoff_opt`. The first command runs analysis and is the basis of the manual flow. For details, see [“Manual IC Compiler Sign-Off Driven Design Closure” on page 16-6](#). The second command runs analysis and optimization and is the basis of the automatic flow. For details, see [“Automatic IC Compiler Sign-Off Driven Design Closure” on page 16-4](#).

During the analysis process, IC Compiler automatically runs Star-RCXT to perform a complete parasitic extraction on the database and stores the results as an SBPF (Synopsys Binary Parasitic Format) file which is transparent to the user. To build the netlist and load the constraints and parasitic information, IC Compiler automatically runs PrimeTime. PrimeTime performs a full timing update and passes the timing information back to IC Compiler.

Running IC Compiler sign-off driven design closure increases your memory requirements. The `signoff_opt` command automatically saves one snapshot of the design even if you do not specify the `-snapshot` option. The tool automatically saves the best iteration, which is used as the final solution. For disk space requirement guidelines, see [“Usage Guidelines” on page 16-9](#).

Key Features

IC Compiler sign-off driven design closure supports the following key features:

- Multicorner-Multimode (MCMM)

Sign-off driven design closure supports multicorner-multimode technology (MCMM). When you use sign-off with this technology, IC Compiler runs PrimeTime distributed multi-scenario analysis (DMSA) and multiple Star-RCXT sessions, one for each parasitic corner. Under PrimeTime, a unique extraction corner requires a mixture technology file plus an operating temperature. This is similar to how IC Compiler defines a corner by using a TLUPlus file plus an operating temperature. To use sign-off multicorner-multimode technology, follow the guidelines in [“Multicorner-Multimode Guidelines” on page 16-14](#).

- Unified Power Format (UPF) support

IC Compiler sign-off driven design closure supports designs with UPF constraints automatically. IC Compiler will automatically pass UPF constraints to PrimeTime for analysis when you execute either `run_signoff` or `signoff_opt`. The `signoff_opt` flow with UPF constraints is supported for both multicorner-multimode (MCM) and best case/worst case (BCWC) analysis. For IR drop analysis, you need to convert the `set_rail_voltage` commands to `set_voltage` commands manually so that you can use them in the UPF mode.

- Zroute support

IC Compiler sign-off driven design closure can use either Zroute or the classic router for ECO route. To invoke Zroute during `signoff_opt`, execute `set_route_mode_options -zroute true`.

- Bottleneck analysis support

IC Compiler sign-off driven design closure supports PrimeTime's bottleneck analysis. To enable this feature, execute `set signoff_bottleneck_analysis 1`.

- Path-based analysis support

To improve timing analysis accuracy, IC Compiler sign-off driven design closure supports the PrimeTime path-based analysis feature. To enable this feature, execute `run_signoff -path_based_analysis` or `signoff_opt -path_based_analysis`.

- Advanced on-chip variation modeling (AOCVM) and on-chip variation (OCV) support

On-chip variation (OCV) is supported in `signoff_opt`. If you enable crosstalk analysis, OCV is automatically supported. If you do not enable crosstalk analysis, you need to enable OCV analysis by using `set_operating_conditions -analysis_type on_chip_variation`

For AOCVM support details, see [“General Guidelines” on page 16-11](#).

- Customized PrimeTime and Star-RCXT settings

When you use IC Compiler sign-off driven design closure, IC Compiler runs PrimeTime and Star-RCXT automatically using each tool's default settings. If you want to customize these settings, use the methods described in [“PrimeTime and Star-RCXT Customizing Options” on page 16-17](#).

IC Compiler Sign-Off Driven Design Closure Methodology

This section describes the IC Compiler sign-off driven design closure methodology which consists of the following:

- [Automatic IC Compiler Sign-Off Driven Design Closure](#)
- [Manual IC Compiler Sign-Off Driven Design Closure](#)

Automatic IC Compiler Sign-Off Driven Design Closure

The automatic flow uses the `signoff_opt` command to run analysis and optimizations, such as buffer insertion, buffer removal, and cell sizing, which fix the violations reported during analysis. You can choose three effort levels of optimization: low, medium, or high. For details, see the `signoff_opt` man page.

To run the automatic flow, follow these steps:

1. Set up the PrimeTime and Star-RCXT tools.

Use the `set_primetime_options` and `set_starrcxt_options` commands to specify global options. You can also use the GUI. Choose Signoff > Optimize. The Signoff Optimize dialog box appears. You can verify your options by using the `report_primetime_options` and `report_starrcxt_options` commands. For details, see [“PrimeTime and Star-RCXT Version Requirements” on page 16-10](#) and [“General Guidelines” on page 16-11](#).

2. If you are using multicorner-multimode technology, see [Example 16-2](#) and complete these steps:

- Set up your host farm by using the `add_distributed_host` command. You can verify your host options by using the `report_distributed_host` command.
- Create your scenarios by using the `create_scenario` command.

For each scenario, specify the scenario-specific operating conditions, the TLUPlus files, the SDC file, and the `max_nxtgrd_file`. Note that the `min_nxtgrd_file` is not supported.

For details, see [“Multicorner-Multimode Guidelines” on page 16-14](#).

3. Execute `signoff_opt`.

You can also use the GUI. Choose Signoff > Optimize. The Signoff Optimize dialog box appears. Choose the appropriate effort level.

4. Review any violations.

Use the `report_timing`, `report_constraints` or `report_qor` commands as needed.

5. To fix any remaining violations, follow the guidelines in the manual flow. See [“Manual IC Compiler Sign-Off Driven Design Closure” on page 16-6](#).

[Example 16-1](#) shows a sample script.

Example 16-1 Using signoff_opt

```
open_mw_cel post_route_opt
link

set_primetime_options
set_starrcxt_options

report_primetime_options
report_starrcxt_options
signoff_opt

report_timing
report_constraints -all_violators

run_signoff -signoff_analysis false

save_mw_cel -as signoff
```

[Example 16-2](#) shows a sample script that uses multiple scenarios.

Example 16-2 Using signoff_opt With Multiple Scenarios

```
set pt_dir /tools/primetime/B-2008.09/amd64/syn/bin
set star_dir /tools/star_rcxt/B-2008.09/amd64_star-rcxt/bin

# global settings
set_primetime_options -exec_dir $pt_dir
set_starrcxt_options -exec_dir $star_dir

report_primetime_options
report_starrcxt_options

add_distributed_hosts -target all -farm lsf -setup_path /lsf/bin

report_distributed_hosts
```

```
# Create 3 scenarios with scenario specific settings
create_scenario maxmax
set_operating_conditions -analysis_type on_chip_variation Worst
set_tlu_plus_files -max_tluplus $max_tlu_file -tech2itf_map $map_file
set_primetype_options -sdc_file $max_sdc_file
set_starrcxt_options -max_nxtgrd_file $max_grd_file

create_scenario minmin
set_operating_conditions -analysis_type on_chip_variation Best
set_tlu_plus_files -max_tluplus $min_tlu_file -tech2itf_map $map_file
set_primetype_options -sdc_file $min_sdc_file
set_starrcxt_options -max_nxtgrd_file $min_grd_file

create_scenario nomnom
set_operating_conditions -analysis_type on_chip_variation Nom
set_tlu_plus_files -max_tluplus $nom_tlu_file -tech2itf_map $map_file
set_primetype_options -sdc_file $nom_sdc_file
set_starrcxt_options -max_nxtgrd_file $nom_grd_file

# run signoff optimization
signoff_opt
```

Manual IC Compiler Sign-Off Driven Design Closure

Manual IC Compiler sign-off driven design closure uses the `run_signoff` command to run analysis. After the analysis, you manually make optimizations to fix the violations reported during analysis. For example, you manually edit the netlist to do different optimizations such as buffer insertion, buffer removal, or cell sizing. Both IC Compiler and PrimeTime simultaneously update the netlist with your changes, and PrimeTime runs an incremental timing update to verify the effects of the changes.

To run the manual flow, do the following:

1. Set up the PrimeTime and Star-RCXT tools.

Use the `set_primetype_options` and the `set_starrcxt_options` commands to specify global options. You can also use the GUI. Choose Signoff > Analysis. The Run Signoff dialog box appears. You can verify your options by using the `report_primetype_options` and `report_starrcxt_options` commands. For details, see [“PrimeTime and Star-RCXT Version Requirements” on page 16-10](#) and [“General Guidelines” on page 16-11](#)

2. If you are using multiple scenarios, that is, multicorner-multimode technology, see [Example 16-4](#) and follow these steps:

- Set up your host farm by using the `add_distributed_host` command. You can verify your host options by using the `report_distributed_host` command.
- Create your scenarios by using the `create_scenario` command.

For each scenario, specify the scenario-specific operating conditions, the TLUPlus files, the SDC file, and the `max_nxtgrd_file`. Note that the `min_nxtgrd_file` is not supported. For details, see [“Multicorner-Multimode Guidelines” on page 16-14](#).

3. Execute `run_signoff`.

You can also use the GUI by choosing Signoff > Analysis. The Run Signoff dialog box appears. Select the full analysis option.

4. Review the violations.

Use the `report_timing` and `report_constraints` commands as needed.

5. Manually fix the violations.

Use the `get_alternative_lib_cells`, `size_cell`, `insert_buffer`, and `remove_buffer` commands as needed. You can run these commands from the GUI ECO menu. Other netlist editing commands, such as `create_cell`, are not supported in the IC Compiler sign-off driven design closure flow. From the Size Cell dialog box, you can preview slack improvement. For details, see [“Troubleshooting and Problem Solving” on page 16-21](#).

6. Run incremental legalization and incremental routing. This will legalize any new added cells to legal locations and do ECO routing.

7. Run incremental analysis.

Use the `run_signoff -incremental` command. From the GUI, choose Signoff > Analysis. The Run Signoff dialog box appears. Select the incremental analysis option.

8. Review violations. Repeat steps 3 through 6 until all violations are fixed.

Example 16-3 shows a sample script.

Example 16-3 Using run_signoff

```
set pt_dir /tools/primetime/B-2008.09/amd64/syn/bin
set star_dir /tools/star_rcxt/B-2008.09/amd64_star-rcxt/bin
set_starrcxt_options -exec_dir $star_dir -map_file $MAP \
                    -max_nxtgrd_file $MAX_GRD \
                    -max_nxtgrd_file $MIN_GRD \
set_primetime_options -exec_dir $pt_dir -sdc_file $SDC
report_starrcxt_option
report_primetime_options
run_signoff
report_qor
report_timing
size_cell ...
insert_buffer ...
legalize_placement -eco
route_eco -auto
run_signoff -incremental
report_qor
report_timing
```

Example 16-4 shows a sample script for a design that uses multiple scenarios.

Example 16-4 Using run_signoff With Multiple Scenarios

```
set pt_dir /tools/primetime/B-2008.09/amd64/syn/bin
set star_dir /tools/star_rcxt/B-2008.09/amd64_star-rcxt/bin

open_mw_cel post_route_opt
link

# global settings
set_primetime_options -exec_dir $pt_dir
set_starrcxt_options -exec_dir $star_dir
                    $map_file

report_primetime_options
report_starrcxt_options

add_distributed_hosts -target primetime -farm lsf -setup_path /lsf/bin

report_distributed_hosts

# Create 3 scenarios with scenario specific settings
create_scenario maxmax
    set_operating_conditions -analysis_type on_chip_variation Worst
    set_tlu_plus_files -max_tluplus $max_tlu_file -tech2itf_map $map_file
    set_primetime_options -sdc_file $max_sdc_file
    set_starrcxt_options -max_nxtgrd_file $max_grd_file

create_scenario minmin
    set_operating_conditions -analysis_type on_chip_variation Best
    set_tlu_plus_files -max_tluplus $min_tlu_file -tech2itf_map $map_file
    set_primetime_options -sdc_file $min_sdc_file
    set_starrcxt_options -max_nxtgrd_file $min_grd_file
```

```

create_scenario nomnom
  set_operating_conditions -analysis_type on_chip_variation Nom
  set_tlu_plus_files -max_tluplus $nom_tlu_file -tech2itf_map $map_file
  set_primetype_options -sdc_file $nom_sdc_file
  set_starrcxt_options -max_nxtgrd_file $nom_grd_file

# run signoff optimization

run_signoff

report_timing
report_constraints -all_violators

get_alternative_lib_cells
size_cell
insert_buffer

legalize_placement -eco
route_eco -auto

run_signoff -incremental
report_timing

report_qor

run_signoff -signoff_analysis false

save_mw_cel -as final

```

Usage Guidelines

Before running IC Compiler sign-off driven design closure, you need to be familiar with the following usage guidelines:

- [PrimeTime and Star-RCXT Version Requirements](#)
- [Entry Requirements for IC Compiler Sign-Off Driven Design Closure](#)
- [General Guidelines](#)
- [Scenario Differences Between IC Compiler and PrimeTime](#)
- [Multicorner-Multimode Guidelines](#)
- [PrimeTime and Star-RCXT Customizing Options](#)

These guidelines are described in the next sections.

PrimeTime and Star-RCXT Version Requirements

For best results, use the following sign-off tools when running IC Compiler sign-off driven design closure:

- PrimeTime version B-2008.06 or later

Default options of PrimeTime and the PrimeTime SI default filters are used in the sign-off driven design closure flow. If you want to use non-default settings, see [“PrimeTime and Star-RCXT Customizing Options” on page 16-17](#).

- Star-RCXT version B-2008.06 or later

Default options of Star-RCXT are used in the sign-off driven design closure flow. These include MODE 200, accept Star-RCXT run directory, operating temperature – ON, topological reduction – ON, and writing subnode information - ON. If you want to use non-default options, see [“PrimeTime and Star-RCXT Customizing Options” on page 16-17](#).

Entry Requirements for IC Compiler Sign-Off Driven Design Closure

When you use IC Compiler sign-off driven design closure, you should be finished with any functional ECO changes, and your design should meet the following conditions:

- There are no routing shorts. Routing shorts will cause the tool to issue an error message and exit the `signoff_opt` command, unless you specify the `ignore_design_readiness` option.
- The number of routing DRC violations must be less than 0.1 percent of the total nets or less than 1000 nets. Otherwise, the tool will issue an error message and exit the `signoff_opt` command unless you specify the `-ignore_design_readiness` option.
- The number of unrouted nets must be less than 0.1 percent of the total nets or less than 1000 nets. Otherwise, the tool will issue an error message and exit the `signoff_opt` command unless you specify the `-ignore_design_readiness` option.
- The number of logic DRC violations must be less than 0.1 percent of the total nets or less than 1000 nets. Otherwise, the tool will issue an error message and exit the `signoff_opt` command unless you specify the `-ignore_design_readiness` option.
- The number of violating paths should be less than 1000; otherwise, the tool will generate a warning.
- Violating endpoints should be less than 1000; otherwise, the tool will generate a warning.

You might not get the best QoR if your design does not meet the above requirements when you execute `signoff_opt`. To prevent the tool from exiting with an error message, execute `signoff_opt -ignore_design_readiness`.

General Guidelines

General IC Compiler sign-off driven design closure guidelines are as follows:

1. Sign-off driven design closure requires additional memory because `signoff_opt` automatically saves one snapshot of the design. Use the following guidelines for memory requirements:
 - In the run directory, you need free space of at least twice the CEL size.
 - In the directory that contains the Milkyway CEL view, you need free space of at least twice the CEL size.
 - If you use the `-snapshot` option of either the `run_signoff` command or the `signoff_opt` command, you will need about the same CEL size for both the run and the CEL view directory per iteration.

If you are using multicorner-multimode technology, you will need additional memory. For details, see [“Multicorner-Multimode Guidelines” on page 16-14](#).

2. To improve QoR, you can enable the tool to fix static noise by executing `set_si_options -static_noise TRUE`. When you use this command, the following commands will be PrimeTime based after `run_signoff` and `signoff_opt`:
`report_noise`, `report_noise_parameters`,
`report_noise_violation_sources`, `report_si_aggressor_exclusion`,
`report_si_bottleneck`, `report_si_delay_analysis`,
`report_si_double_switching`, and `report_si_noise_analysis`.
3. To improve timing analysis, use the PrimeTime path-based analysis feature. To enable this feature, execute `run_signoff -path_based_analysis` or `signoff_opt -path_based_analysis`.
4. To achieve better correlation with the stand-alone sign-off tools, set Star-RCXT REDUCTION to YES. For the best QoR, IC Compiler, by default, sets Star-RCXT REDUCTION to TOPO when you execute `signoff_opt`.
5. After you run `signoff_opt`, you need to re-insert the filler cells. Any filler cells that overlap with new or changed cells are automatically removed during `signoff_opt`. Generally, it is best to run `signoff_opt` before chip finishing. But, if there is a significant timing impact during the chip finishing process, you need to run `signoff_opt` after chip finishing.

6. When using metal filling, you must execute `signoff_opt` after metal fill if the metal fill has significant timing impact; otherwise, you can execute `signoff_opt` before fill. When metal fill is done before `signoff_opt`, use the following guidelines:
 - a. Metal fill must be done in the FILL view, not in the CEL view. To do this, use the `-out design_name.FILL` option of the `insert_metal_filler` command:


```
insert_metal_filler ... -out design_name.FILL
```
 - b. Metal fill must be made floating. Notice that the FILL view only supports floating metal filling. To do this, use the `-tie_to_net` option of `insert_metal_filler` as follows:


```
insert_metal_filler ... -tie_to_net none
```
 - c. Exit `signoff_opt` after ECO routing. To do this, run `signoff_opt` with `-only_psyn` as follows:


```
signoff_opt -only_psyn
```
 - d. Remove overlap between signal routes and fill metals. To do this, manually update metal fill by using `trim_fill_eco` as follows:


```
trim_fill_eco -input design_name.FILL -output design_name.FILL
```
 - e. Repeat Step (c) and (d) as needed.

When you use this flow, you lose the capability to run the incremental Star-RCXT and incremental PrimeTime features of `signoff_opt` because in each iteration, full extraction and full timing analysis will be performed.
7. To exit the PrimeTime timing engine and return to the IC Compiler timing engine, execute `run_signoff -signoff_analysis false`.
8. By default, the tool always runs incremental extraction and incremental PrimeTime analysis if possible. If you specify the `-full_extract` option, you must use the `full_analysis` option. However, if you use the `-full_analysis` option, you do not need to use `full_extract`. When `full_analysis` is used without `full_extract`, the tool will run Star-RCXT with incremental extraction and create a full netlist output. A warning is issued for runtime increases.
9. If you execute `run_signoff` prior to `signoff_opt`, you can use `signoff_opt -skip_initial_analysis` to use the previous analysis results. In this case, the previous `run_signoff` must be done on a full design and you need to make sure nothing is changed between the `run_signoff` and `signoff_opt`. Note that you will still need to run Star-RCXT to create the netlist output (but no actual extraction is needed).
10. IC Compiler sign-off driven design closure supports the advanced on-chip variation modeling (AOCVM) feature in PrimeTime. You enable AOCVM by using the `-aocvm` option with `run_signoff` or `signoff_opt`. You must use a saved session or the `-specific_file` option when running IC Compiler sign-off driven design closure with AOCVM. See [“PrimeTime and Star-RCXT Customizing Options” on page 16-17](#). Use

`report_aocvm` to report AOCVM related information. For timing reports, use `report_timing -aocvm` after `run_signoff`. Note that this is PrimeTime syntax. The `report_timing` command from IC Compiler does not have the `-aocvm` option.

11. You cannot change the timing constraints after you execute the `run_signoff` command. To change timing constraints, exit `run_signoff` by using `run_signoff -signoff_analysis false`, as shown below:

```
run_signoff
report_timing
run_signoff -signoff_analysis false
set_false_path
...
```

Scenario Differences Between IC Compiler and PrimeTime

When creating scripts for IC Compiler sign-off driven design closure, you need to understand how IC Compiler and PrimeTime handle scenarios differently. Both tools define scenarios the same way by using the `create_scenario` command, but each tool uses a different command to specify the scenarios to be used in the current run. IC Compiler uses the `set_active_scenario` command; PrimeTime uses the `current_session` command. Keep the following in mind:

- The scenario concept is the same for both IC Compiler and PrimeTime.
- The actual user interface commands have differences.
- IC Compiler's *active scenario* is similar to PrimeTime's *session*.
- IC Compiler does not support the PrimeTime `current_session`. You need to use `set_active_scenario` instead.
- Each PrimeTime process reads only one corner for `read_parasitics`.
- IC Compiler invokes multi-corner `read_parasitics`.

After you execute the IC Compiler `current_scenario` command, most reporting commands and the `read_parasitic` command are only applied to the specified current scenario. All other commands, such as optimization and netlist editing commands, are applied to all scenarios. The IC Compiler `current_scenario` command does not support the following PrimeTime commands:

- `current_scenario -all`
- `current_scenario {Func Test}`

After you execute the PrimeTime `current_scenario` command, all commands are applied to the specified current scenario. Even optimization commands, such as `size_cell`, will only change the given scenario(s). The PrimeTime `current_scenario` command supports the following:

- `current_scenario -all`
- `current_scenario {Func Test}`

After you execute the IC Compiler `run_signoff` command, the `current_scenario` behavior will be changed to update both IC Compiler and PrimeTime.

- `current_scenario -all`

No change will occur to the current scenario in IC Compiler.

Command focus is changed to include all scenarios in PrimeTime distributed multi-scenario analysis.

- `current_scenario Func`

Executing `current_scenario Func` changes the current scenario to Func in IC Compiler.

Executing `current_scenario Func` changes the command focus to Func in PrimeTime distributed multi-scenario analysis.

Multicorner-Multimode Guidelines

Use the following guidelines when running IC Compiler sign-off driven design closure with multicorner-multimode technology:

1. Using IC Compiler sign-off driven design closure in multicorner-multimode requires additional memory space. Use the Unix `top` command to determine the memory size. As a general rule, you need 10 GB memory to run IC Compiler for a one million instance design for a single scenario. For a multicorner-multimode design with X instances and Y scenarios, the memory requirement is:

$10 \text{ GB} / (1 \text{ MB} / X) * Y * 25\%$, assuming Y is ≥ 4

This requirement only considers the IC Compiler process. PrimeTime and Star-RCXT processes are assumed to be run with distributed hosts. If you use GRD/LSF for your main IC Compiler process, memory size needs to be bigger because you are sharing with other users. Not enough memory will cause excessive memory swap and increase runtime.

2. When using GRD for your distributed processes, you must submit the job as a binary job by using the “-b y” option. Otherwise, PrimeTime will fail. The following example uses the “-b y” option.

```
add_distributed_hosts -farm grd \  
-setup_path ... -num_of_hosts 8 \  
-options {-cwd -P bnormal -b y -l arch=glinux,cputype=amd64}
```

These options instruct the `qsub` command to run the specified script directly without making an intermediate copy, which speeds up the submission of the jobs. For details, see the `qsub` man page.

3. IC Compiler sign-off driven design closure uses a distributed processing environment. To set up this environment, you need to understand that, by default, IC Compiler sign-off driven design closure uses the remote shell (rsh) program to invoke Star-RCXT and PrimeTime processes. To use the C-shell (csh), create a *runcmd* file as follows:

```
#!/bin/csh -f  
$argv &
```

Then specify the *runcmd* file with `add_distributed_hosts` as follows:

```
add_distributed_hosts -submission_script runcmd ...
```

To use the secure shell protocol (ssh), use the `-enable_ssh` option as follows:

```
add_distributed_hosts -enable_ssh
```

Note that the secure shell protocol (ssh) must be configured without using an explicit user name and password. This is a PrimeTime distributed multi-scenario analysis limitation.

4. If you use `add_distributed_hosts` multiple times, the number of hosts is cumulative. For example,

```
add_distributed_hosts -farm lsf -num_of_hosts 4  
add_distributed_hosts -farm now -num_of_hosts 2 unix1  
add_distributed_hosts -farm now -num_of_hosts 2 unix2
```

IC Compiler runs the first four processes on LSF hosts, the next two processes on the machine “unix1,” and the last two processes on machine “unix2.” If the number of scenarios is more than the number of hosts specified, the remaining scenarios are run on the local host. This behavior is different from the PrimeTime distributed multi-scenario analysis behavior which will not use local hosts for remaining scenarios. Instead, PrimeTime will wait for the specified hosts to finish and then use those hosts to run the remaining scenarios.

5. To use scenario reduction, you must manually set the dominant scenarios based on your `run_signoff` results. Scenario reduction is not done automatically by `signoff_opt`.

6. After you run the initial analysis (`run_signoff`),
 - Determine the dominant scenario by using `get_dominant_scenario`.
 - To reduce the number of scenarios, specify the active scenarios by using `set_active_scenario`.
7. IC Compiler does not support the PrimeTime `current_session` command. Use `set_active_scenario` instead.
8. All active scenarios in IC Compiler, that is, those specified by `set_active_scenario`, will be analyzed in PrimeTime distributed multi-scenario analysis. Inactive scenarios in IC Compiler will not be analyzed in PrimeTime.

Use a one-to-one mapping between active scenarios in IC Compiler to scenarios in PrimeTime. Note the following:

- IC Compiler does not split a scenario based on minimum and maximum conditions.
 - For each scenario, you must use on-chip variation (OCV) and you cannot use `set_min_library`, minimum TLUPlus, minimum `nxtgrd`, minimum images for PrimeTime, or minimum images for Star-RCXT.
9. You must redefine all best case/worst case scenarios.

Because IC Compiler does not split a scenario based on minimum and maximum conditions, you must redefine all your best case/worst case scenarios as follows:

- Consider the following best case/worst case scenario in IC Compiler multicorner-multimode:

```
set_tlu_plus_files -max_tlup max.tlup \
                  -min_tlup min.tlup
set_min_library -min fast.db slow.db
set_operating_conditions -analysis_type bcwc
```

You need to replace this scenario with two scenarios when running IC Compiler sign-off driven design closure with multicorner-multimode technology.

- Scenario worst case (WC)

```
set_tlu_plus_file -max_tlup max.tlup
# Use slow.db as the link_library
set_operating_conditions -analysis_type on_chip_variation
```

- Scenario best case (BC)

```
set_tlu_plus_file -max_tlup min.tlup
# Use fast.db as the link_library
set_operating_conditions -analysis_type on_chip_variation
```

10. After you execute `run_signoff`, you cannot activate a scenario that was inactive before you executed `run_signoff`. If you try to do so, IC Compiler will issue a PSYN-432 warning message.

11. IC Compiler disables `create_scenario` and `remove_scenario` after you execute `run_signoff` or `signoff_opt`. To enable these commands, execute `run_signoff -signoff_analysis false`

PrimeTime and Star-RCXT Customizing Options

When you use IC Compiler sign-off driven design closure, IC Compiler runs PrimeTime and Star-RCXT automatically using each tool's default settings. If you want to customize these settings, you can use

- Saved sessions
- The `common_file` option to the `set_primetime_options` command
- The `specific_file` option to the `set_primetime_options` command
- `set_starrcxt_options -option_file`
- `set_starrcxt_options -mode`

These methods are described in the following paragraphs.

Saved Sessions

To create a saved session in PrimeTime, use the `save_session` command in PrimeTime and specify this saved session file name by using the `-max_image` option or the `-min_image` option of the `set_primetime_options` command, as shown in [Example 16-5](#). If you want to verify your option settings, use the `report_primetime_options` command, as shown in [Example 16-6](#).

You must add the following command at the beginning of the PrimeTime script in order to generate a PrimeTime saved session for `signoff_opt`:

```
set_program_options -enable_eco
```

Without this command, IC Compiler cannot use the saved session for incremental analysis.

Starting with version A-2007.12-SP4, `signoff_opt` will check if the specified PrimeTime saved session has ECO enabled. If ECO is not enabled, IC Compiler will generate a PSYN-441 warning and behave as if `-only_psyn` is specified.

To use a saved session of Star-RCXT, specify the run directory name in the `set_starrcxt_options`, as shown in [Example 16-5](#). If you want to verify your option settings, use the `report_starrcxt_options` command, as shown in [Example 16-6](#).

Example 16-5 Specified Saved PrimeTime and Star-RCXT Runs

```
set SDC pt.sdc
set pt_dir /tools/primetime/B-2008.09/amd64/syn/bin
set star_dir /tools/star_rcxt/B-2008.09/amd64_star-rcxt/bin
set_starrcxt_options -exec_dir $star_dir \
                    -max_nxtgrd_file $MAX_GRD \
                    -min_nxtgrd_file $MIN_GRD \
                    -max_image star_max \
                    -min_image star_min \
                    -map_file $MAP
set_primetime_options -exec_dir $pt_dir -sdc_file $SDC \
                    -max_image pt_max \
                    -min_image pt_min

report_starrcxt_option
report_primetime_options
```

Example 16-6 Report Saved Sessions

```
icc_shell> report_starrcxt_option
*****
Report : StarRCXT_options
Design : RISC_CORE
Version: B-2008.09-ICC
Date   : Mon Aug 4 23:18:51 2008
*****

StarRCXT exec dir: /tools/star_rcxt/2008.09/B-2008.06/amd64_star-rcxt/bin
StarRCXT max nxtgrd file: /libraries/tcbn90g/tsmcn90_7lm.nxtgrd
StarRCXT min nxtgrd file: /libraries/tcbn90g/tsmcn90_7lm.nxtgrd
StarRCXT map file: /libraries/tcbn90g/tcbn90_7lm.map
StarRCXT max image: star_max
StarRCXT min image: star_min

icc_shell> report_primetime_options*
*****
Report : PrimeTime_options
Design : RISC_CORE
Version: B-2008.09-ICC
Date   : Mon Aug 4 23:19:05 2008
*****

PrimeTime exec dir: / tools/primetime/B-2008.06/amd64/syn/bin
PrimeTime SDC file: /home/signoff/risc_core/pt.sdc
PrimeTime max image: pt_max
PrimeTime min image: pt_min
```

set_primetime_options -common_file

When you use `set_primetime_options -common_file`, one file is used for all PrimeTime runs (BC/WC, and all scenarios). This file is sourced into PrimeTime before the netlist loads and is used for setting up Tcl variables.

Only the following variables are allowed; otherwise the tool reports a PSYN-387 error:

- All RC variables
- All case analysis variables
- All CCS variables
- All path-based analysis (PBA) variables
- All report variables
- All timing variables except `timing_save_pin_arrival_and_slack`
- All signal integrity (SI) variables except `si_enable_analysis`
- All parasitic variables except `read_parasitics_load_locations`
- All variation variables except `variation_enable_analysis` and `variation_derived_scalar_attribute_mode`

set_primetime_options -specific_file

When you use `set_primetime_options -specific_file`, you can have up to two files {`max_file min_file`} in best-case/worst-case (BC/WC) mode. The second file is used for the PrimeTime minimum corner in BC/WC analysis. If only one file is specified, the same file will be used for both corners. This file is sourced into PrimeTime after the netlist loads. You can have one specific file per corner when using multicorner-multimode technology in the IC Compiler sign-off driven design closure flow.

Only these commands are allowed; otherwise the tool reports a PSYN-388 error:

- All report commands
- All check commands
- All update commands
- All set commands except `set_units` and `set_unix_variable`
- All reset commands except `reset_design`
- `group_path`
- `scale_parasitics` and `complete_net_parasitics`
- `read_aocvm` and `remove_aocvm`
- `create_correlation` and `create_variation`
- `define_scaling_lib_group`

set_starrcxt_options -option_file

To customize Star-RCXT options, use `set_starrcxt_options -option_file`. This is a file you create that contains a subset of the Star-RCXT command file.

Note:

The commands listed below are reserved and will be automatically set by IC Compiler. Do not put any of these commands in the option file; otherwise, the tool returns a PSYN-171 error.

- MILKYWAY_DATABASE
- TCAD_GRD_FILE
- MAPPING_FILE
- COUPLE_TO_GROUND
- STAR_DIRECTORY
- NETLIST_FORMAT
- TARGET_ANALYSIS
- EXTRA_GEOMETRY_INFO
- NETLIST_NODE_SECTION
- NETLIST_TAIL_COMMENTS
- NETLIST_FILE
- BLOCK
- INCREMENTAL
- NETLIST_INCREMENTAL
- HIERARCHICAL_SEPARATOR
- MILKYWAY_EXPAND_HIERARCHICAL_CELLS
- FSCOMPARE_FILE_PREFIX
- FS_PREPROCESSOR_STAGE
- SENSITIVITY
- MULTI_CPU_ON_LICENSE_ONLY

set_starrcxt_options -mode

You can specify a Star-RCXT MODE by executing `set_starrcxt_options -mode`. The valid values are 100, 150, 200, and 400. The default value is 200. MODE establishes the tradeoff between runtime and accuracy. See the man page for details. You can also provide the MODE by using `set_starrcxt_options -option_file`.

Limitations

IC Compiler sign-off driven design closure limitation(s) are as follows:

- IC Compiler sign-off driven design closure uses a distributed processing environment when running multicorner-multimode technology. By default, the flow uses the remote shell (rsh) program to invoke Star-RCXT and PrimeTime processes. If you use the secure shell protocol (ssh), you must configure ssh without using an explicit user name and password. This is a PrimeTime distributed multi-scenario analysis limitation. For details, see [“Multicorner-Multimode Guidelines” on page 16-14](#).

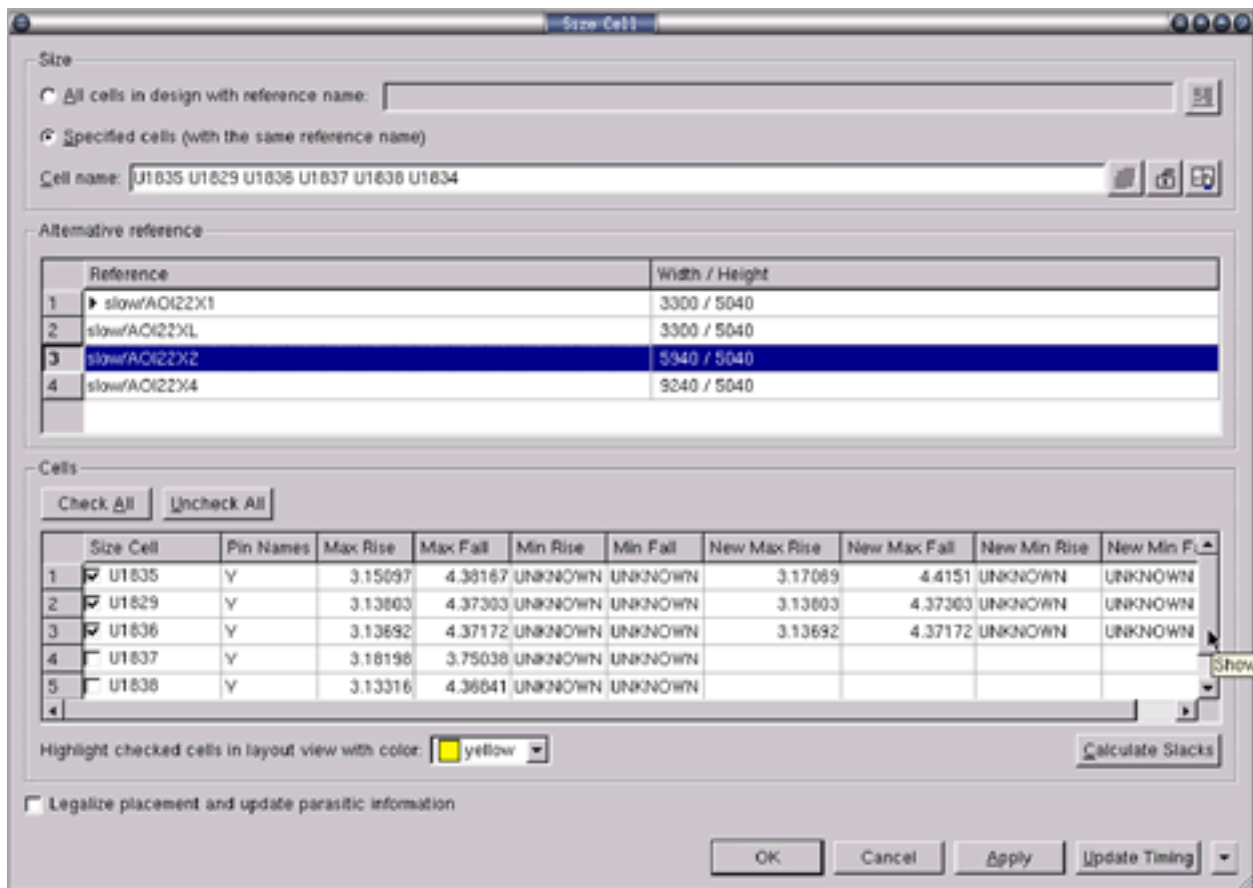
Troubleshooting and Problem Solving

Troubleshooting guidelines are as follows:

1. To help you find problems early and quickly, you can check your design by executing `run_signoff -check_only`. In this case you do not actually invoke Star-RCXT and PrimeTime.
2. If IC Compiler sign-off driven design closure is producing poor QoR, your design may not be meeting the entry requirements. For detailed requirements, see [“Entry Requirements for IC Compiler Sign-Off Driven Design Closure” on page 16-10](#).
3. If your IC Compiler sign-off driven design closure runs are excessively slow, you may not have enough memory. For memory requirements, see [“Usage Guidelines” on page 16-9](#) and [“Multicorner-Multimode Guidelines” on page 16-14](#).
4. If you are getting warnings about not being able to load the PrimeTime or Star-RCXT tools, you may not be using the correct Star-RCXT or PrimeTime versions. For tool version requirements, see [“PrimeTime and Star-RCXT Version Requirements” on page 16-10](#).
5. If you are having library consistency problems, do the following:
 - Use `get_libs *` or `write_link_library` before `signoff_opt` to ensure library consistency between IC Compiler and PrimeTime.

6. After you save the Milkyway CEL view and reload it, you may have inconsistencies in your timing reports between IC Compiler, PrimeTime, and Star-RCXT. If this happens, execute `run_signoff` again. The timing in IC Compiler is different from the timing in PrimeTime and Star-RCXT, shown by the previous `run_signoff` session, until you run `run_signoff` again.
7. If you are having difficulties adjusting slack, you can preview slack improvements in the GUI by using the Size Cell dialog box. Choose ECO > Size Cell. The Size Cell dialog box appears. Choose an alternative reference cell to size the cells in the design that have the same reference name. From this dialog box you can calculate slack values for alternative reference cells. This enables you to see how much slack can be improved with the new reference cell before you make actual changes to the netlist. Figure 16-1 shows the current slack values and the new values for an alternative reference cell.

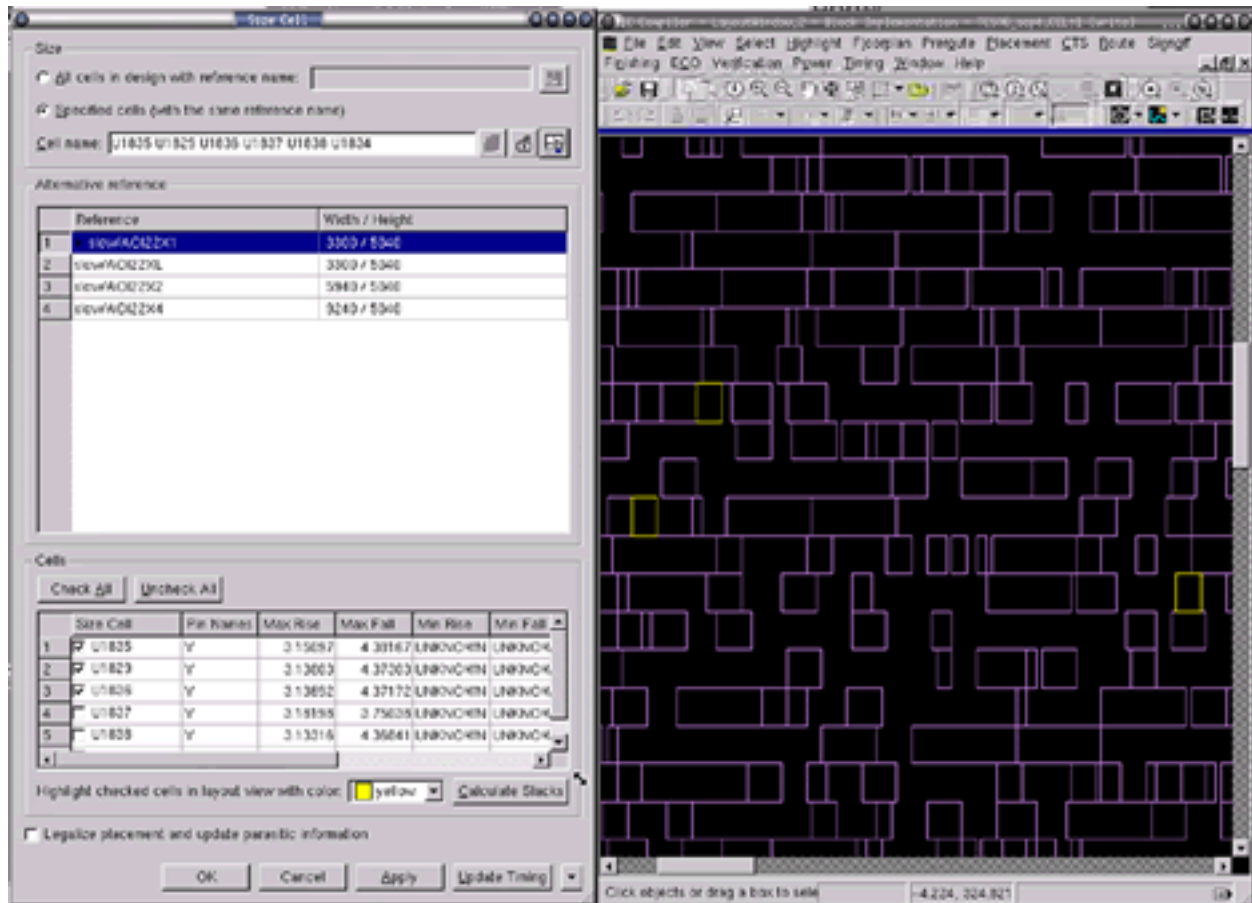
Figure 16-1 Slack Preview Using the GUI



From the Size Cell dialog box, you can highlight checked cells in layout view to see the locations of these checked cells. You can select different colors for highlighting.

Figure 16-2 highlights checked cells in yellow.

Figure 16-2 Highlighting Cell Locations



A

Using GUI Tools

The IC Compiler GUI provides a variety of tools for viewing, analyzing, and editing your design. This appendix provides an overview of these tools. For detailed information about these tools, see the online Help.

This appendix contains the following sections:

- [Viewing a Design](#)
- [Examining Design Objects](#)
- [Visualizing the Physical Design](#)
- [Analyzing the Physical Layout](#)
- [Editing the Physical Layout](#)
- [Analyzing Clock Tree Synthesis Results](#)
- [Analyzing Timing Paths](#)

Viewing a Design

The IC Compiler GUI displays graphic representations of design data in layout and schematic views. To help you visualize a design, the GUI provides tools and commands that you can use to examine the data and select objects of interest in a layout or schematic view.

- The layout view is the focal point of the IC Compiler GUI. It displays graphic representations of design objects in a single, flat view of the physical design. You can open multiple layout views to simultaneously work with different areas of the design.
- Schematic views display schematic representations of specific types of design data. You can use different types of schematic views to examine logic designs (hierarchical cells), selected timing paths (including fanin and fanout logic), and the fanin or fanout networks for selected clock trees.

The GUI provides interactive left-button mouse tools that you can use to view and probe a design. In a layout or schematic view, you can

- Traverse (pan) the design and magnify or shrink (zoom) areas of interest
- Display (query) object properties for individual objects (such as design and timing attribute values)
- Select or deselect individual objects or objects in rectangular areas of the design

Optionally, in a layout view you can add the objects you select to the current selection instead of replacing the objects in the current selection.

- Color highlight individual objects or objects in rectangular areas of the design

Optionally, in a layout view you can highlight the nets connected to objects you highlight and display object properties for an object when you highlight it.

In a layout view, you can also

- Select or deselect objects along a straight line
- Highlight objects with brightness in their normal colors, by dimming unhighlighted objects
- Measure distances by drawing rulers

In addition, the GUI provides other ways to view, select, and highlight objects or timing paths by choosing commands on the View, Select, and Highlight menus.

For information about performing these tasks, see the following sections:

- [Selecting a Mouse Tool](#)
- [Previewing Objects in a Layout View](#)
- [Selecting Objects](#)
- [Highlighting Objects](#)
- [Viewing Object Information](#)
- [Magnifying and Traversing the Design](#)
- [Drawing Rulers](#)
- [Redrawing the Active View](#)

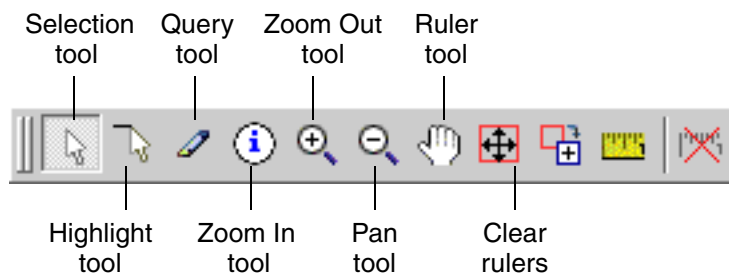
Selecting a Mouse Tool

Mouse tools control the actions performed in graphic views when you click or drag the pointer with the left mouse button. IC Compiler provides mouse tools for design visualization that you can use to

- Magnify or traverse the view in a layout, schematic, or histogram view
- Select, highlight, or query objects in a layout or schematic view
- Draw rulers in a layout view

The visualization mouse tools are available in all top-level GUI window. You can enable these mouse tools by clicking buttons on the Mouse Tools toolbar or by choosing commands on the Mouse Tools menu (choose View > Mouse Tools to display the menu). [Figure A-1](#) shows the full version of this toolbar that is available in the layout window.

Figure A-1 Full Mouse Tools Toolbar



[Figure A-2](#) shows the abbreviated version of the toolbar that is available in the main window, the timing analysis window, and the interactive CTS window.

Figure A-2 Abbreviated Mouse Tools Toolbar

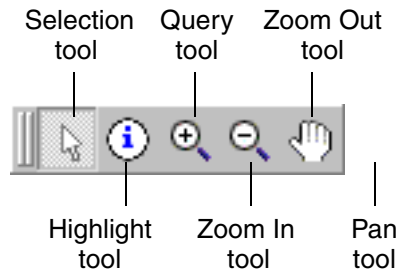


Table A-1 describes the tasks you can perform with the tools on the Mouse Tools toolbar.

Table A-1 Mouse Tools

To perform this task	Do this mouse tool
To set the left mouse button to select objects by clicking or dragging with the Selection tool,	Click the  button, or choose View > Mouse Tools > Selection tool.
To set the left mouse button to highlight objects by clicking or dragging with the Highlight tool,	Click the  button, or choose View > Mouse Tools > Highlight Tool.
To set the left mouse button to display object information by clicking objects with the Query tool,	Click the  button, or choose View > Mouse Tools > Query Tool.
To set the left mouse button to magnify the design by clicking or dragging with the Zoom In tool,	Click the  button, or choose View > Mouse Tools > Zoom In Tool.
To set the left mouse button to shrink the design by clicking or dragging with the Zoom Out tool,	Click the  button, or choose View > Mouse Tools > Zoom Out Tool.
To set the left mouse button to scroll or navigate through the design by dragging the Pan tool,	Click the  button, or choose View > Mouse Tools > Pan Tool.
To set the left mouse button to draw rulers by clicking with the Ruler tool,	Click the  button, or choose View > Mouse Tools > Rule Tool.

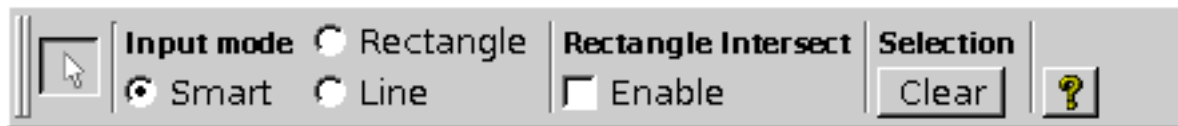
Note:

IC Compiler also provides mouse tools that you can use to perform various layout editing tasks. For details about these mouse tools, see [“Editing a Design” on page A-54](#).

The Selection tool is the default mouse tool. When you open a layout window or a new layout view, schematic view, or histogram view, the Selection tool is the active mouse tool. When you enable a mouse tool in the active view, the tool remains active until you enable a different mouse tool. When you change the mouse tool, you change it only in the active view.

When you enable the Selection tool, Highlight tool, or Ruler tool for use in a layout view, options for using the tool appear on the Mouse Tool Options toolbar. [Figure A-3](#) shows the default options when the Selection tool is the active mouse tool.

Figure A-3 Selection Tool Options on Mouse Tool Options Toolbar



You can use the Zoom In tool, the Zoom Out tool, or the Pan tool as nested tools when you are performing an operation with another mouse tool. When you finish the nested tool operation, the left mouse button returns to the previous tool and you can resume the operation that you were performing with that tool.

For more information about using mouse tools, see the IC Compiler online Help.

Previewing Objects in a Layout View

If you need to click an object in an area of the design where objects are densely packed or overlapped, you can preview information about the objects at a location before clicking the object you need. The tool displays the information in an InfoTip. InfoTips are enabled by default.

When you move the pointer over the active layout view, the status bar displays the relative coordinates of the pointer position. When you move the pointer over an object, the tool highlights the object in the preview color, which is the same as the select color (white by default) but with thinner lines. You can hold the pointer over an object to view its name, attribute values, and other information. If more than one object occupies the same location, you can display information sequentially for each object.

- To display the information for the Next object in the sequence, press F1.
- To display the information for the previous object in the sequence, press Shift-F1.

When the tool displays the information for the object you need, click the object. This preview mechanism is available for the Selection tool, the Highlight tool, the Query tool, and most of the layout editing mouse tools.

You can control the location where the InfoTips appear. By default, an InfoTip floats near the object origin. You can also disable InfoTips if, for example, you do not want to view them while traversing the design.

To set InfoTip options for the active layout view,

1. On the View Settings panel, click the Settings tab and then click the View tab.
2. Set options as needed.
 - To enable or disable InfoTips, select or deselect the Show InfoTip option.
A check mark on the option indicates that InfoTips are enabled.
 - To set the location for the InfoTips, select an option in the InfoTip Location list.
You can set the InfoTips to float near the object origin (the default) or to appear at the bottom of the layout view.
3. Click Apply.

Alternatively, you can enable or disable InfoTips by choosing View > InfoTip. A check mark beside the command on the View menu indicates that InfoTips are enabled;

Selecting Objects

You can use the Selection tool and the Select menu commands to select objects or timing paths in the design.

- Use the Selection tool () to select objects interactively in the active layout or schematic view.
- Use commands on the Select menu to select objects by name or type, select timing paths, or select fanin or fanout objects.

When you use the Selection tool, you can fine-tune selections by adding or removing objects in existing selections. Before you can select an object in a layout view, it must be enabled for display and selection.

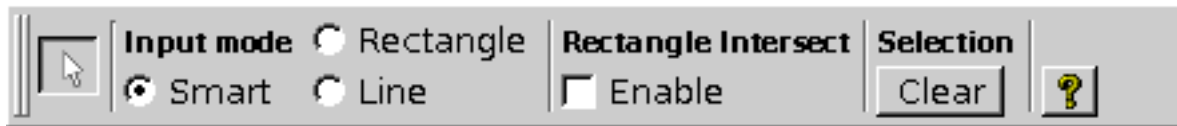
The GUI displays selected objects in the selection color, white by default, and cross-selects the objects in all other open views in which they appear. The object names appear in the selection list.

To activate the Selection tool,

- Click the  button on the Mouse Tools toolbar, or choose View > Mouse Tools > Selection Tool.

When you enable the Selection tool in a layout view, options for using the tool appear on the Mouse Tool Options toolbar. [Figure A-4](#) shows the default options when the Selection tool is the active mouse tool.

Figure A-4 Selection Tool Options on Mouse Tool Options Toolbar



You can select objects by doing any of the following:

- To select an object, select the Smart option, and click the object.
To select additional objects, press the Control key and click the other objects. To remove objects from the selection, press the Shift key and click the objects.
- To select all objects in a rectangular area, select the Smart option or the Rectangle option, and drag the pointer diagonally to define a rectangular box around the objects. By default, the region must completely encompass all the objects you need to select.
 - If you want to select objects that are only partially inside the region, select the rectangle intersect Enable option.
 - If you select the Rectangle option, you can define the region by either dragging the pointer or by clicking two diagonally opposite corners. This allows you to click one corner, and then use the zoom and pan tools to magnify or traverse a large area of the design before clicking the other corner of the region.
- To select objects in a line, select the Line option, and click or drag the pointer to define a straight line over at least one edge of each object you need to select.

You can add objects to the current selection by pressing the Control key when you click or drag the pointer.

You can remove objects from the current selection by pressing the Shift key when you click or drag the pointer.

In a layout view, if you need to select an object at a location where objects overlap, you can preview the objects at the location. Move the pointer over an object to display object information in an InfoTip. The object is highlighted in the preview color, which is white by default but is thinner than selection coloring. To preview multiple objects at the same location sequentially, forward or backward, press F1 or Shift-F1. For more details, see [Previewing Objects in a Layout View](#).

If you need to deselect all selected design objects and timing paths in the current design, choose Select > Clear.

Highlighting Objects

You can interactively highlight objects in the active layout or schematic view by using the Highlight tool. In a layout view, you can set options to highlight the object's net connections and to display information about the object on the Query panel. You can also remove highlighting from objects by using the Highlight tool.

When you highlight an object, the GUI displays the object in the current highlight color, yellow by default, and cross-highlights the object in all other open layout and schematic views.

To enable the Highlight tool in the active view,

1. Click the  button on the Mouse Tools toolbar (or choose View > Mouse Tools > Highlight Tool).

When you enable the Highlight tool in a layout view, the Highlight Tool options appear on the Mouse Tool Options toolbar.

2. (Optional) Set options on the Mouse Tool Options toolbar as needed.
 - To display object information on the Query Object panel when you highlight an object, select the Query On Highlight option.
 - To highlight the nets connected to objects that you highlight, select the Highlight Nets option, and select the options for the types of nets you want to highlight. The choices are Signal, Clock, Power, Ground, Tie High, and Tie Low. By default, Signal and clock are enabled, and the other options are disabled.

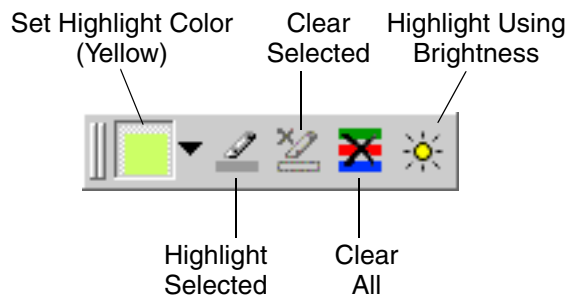
You can use the Highlight toolbar and the Highlight menu to change the highlight color. to add or remove highlighting on selected objects or timing paths.

In a layout view, if you need to highlight an object at a location where objects overlap, you can preview the objects at the location. Move the pointer over an object to display object information in an InfoTip. The object is highlighted in the preview color, which is white by default but is thinner than selection coloring. To preview multiple objects at the same location sequentially, forward or backward, press F1 or Shift-F1. For more details, see [Previewing Objects in a Layout View](#).

You can use highlighting to identify design objects or physical constraints in the active layout view. You can highlight individual objects or objects within a region. You can also toggle (add or remove) highlighting on objects. If you select the "Include nets of chosen objects" option on the Highlight Tool Options panel, flylines appear in the highlight color to show the net connections for the highlighted objects.

The Highlight toolbar, shown in [Figure A-5](#), lets you select the highlight color, add or remove highlighting on selected objects, remove highlighting on all objects, or highlight objects by brightness instead of color. This toolbar can be found in the main, layout, timing analysis, and interactive clock tree synthesis windows; however, the highlight by brightness capability is only available in layout windows.

Figure A-5 Highlight Toolbar



By default, the GUI applies the same color to each object you highlight until you change the color. You can enable an automatic color cycling mechanism.

To enable or disable autocycling,

- Choose Highlight > Auto Cycle Colors.

A check mark next to the Auto Cycle Colors command on the Highlight menu indicates that autocycling is enabled.

When you enable the automatic color cycling mechanism, the GUI automatically cycles through the highlight colors. Each time you apply highlighting to selected objects or paths, the GUI uses the next highlight color in the cycle.

To highlight objects in the active layout view by using the Highlight tool,

1. Make sure the  button is selected on the Mouse Tools toolbar.
2. (Optional) Select a highlight color in the  list on the Highlight tool bar, or click the  button to highlight objects by brightness (non-highlighted objects are dimmed) instead of color.
3. Highlight objects or remove highlighting in any of the following ways:
 - To highlight individual objects, click the objects.
 - To highlight objects in a rectangular area, drag the pointer diagonally to draw a box around the area. The GUI highlights objects that are completely inside the box.

- To remove highlighting from individual objects, Control-click the objects.
- To remove highlighting from objects in a rectangular area, press Shift and drag the pointer diagonally to draw a box around the area. The GUI removes highlighting from objects that are completely inside the box.

4. Repeat step 2 or steps 2 and 3 as needed.

You can select or deselect options on the Highlight Tool Options panel at any time when the Highlight tool is active. If you need to close the panel, move the pointer over the raised bars on one edge of the panel, then right-click and choose Close. The panel is closed automatically when you change to a different mouse tool.

You can also highlight selected timing paths in the active layout or schematic view. When you highlight a timing path, the pins and ports on the path appear in a different color from the unhighlighted objects. Flylines appear in the highlight color to show the connections between the objects on the path.

To highlight timing paths,

1. Select the paths you want to highlight.

2. Click the  button on the Highlight toolbar, or choose Highlight > Selected.

You can remove the highlighting from selected objects or timing paths, remove the current highlight color from objects and paths highlighted with that color, or remove all highlighting in the active layout or schematic view.

To remove highlighting from selected objects and paths,

1. Select the objects and paths from which you want to remove the highlight color.

2. Click the  button on the Highlight toolbar, or choose Highlight > Clear Selected.

To remove the current highlight color from objects and paths,

- Choose Highlight > Clear Current Color.

To remove all highlighting from objects and paths,

- Click the  button on the Highlight toolbar, or choose Highlight > Clear All.

Viewing Object Information

You can use the Query Tool to view design and timing information for objects in a layout or schematic view. When you click an object in the active view, the information appears on the Query panel (the panel opens automatically if it is not already open). You can also set options on the Query panel to automatically or manually copy the information to the session transcript in the console log view.

To enable object queries in the active layout or schematic view,

- Click the  button on the Mouse Tools toolbar (or choose View > Mouse Tools > Query Tool).

The Query panel appears, docked by default to the right side of the window. You should dock the panel or move it to a place on the screen where you can work with both it and the active view at the same time.

In a layout view, if you need to query an object at a location where objects overlap, you can preview the objects at the location. Move the pointer over an object to display object information in an InfoTip. The object is highlighted in the preview color, which is white by default but is thinner than selection coloring. To preview multiple objects at the same location sequentially, forward or backward, press F1 or Shift-F1. For more details, see [Previewing Objects in a Layout View](#).

To display information for an object,

1. Make sure the  button is selected on the Mouse Tools toolbar.
2. Click the object.

The object is highlighted in the query color (white by default), and the information for the object appears on the Query panel. You can select and copy text on the panel, but you cannot edit it.

Note:

Query highlighting has thinner lines than selection highlighting.

To copy the object information into the session transcript in the console log view,

- Select the “Print to Log” option on the Query panel.

When this option is selected, information for objects you click in Query Tool is automatically copied to the transcript.

In a layout view, you can display information about a selected object by pressing the F1 key. The object information appears on the Query panel. If multiple objects are selected, the information for the next object in the selection list appears on the Query panel each time you press F1.

You can also display information about a visible cell, port, or pin in the active layout view by selecting it. When you select an object, the object's name, the x- and y-coordinates for the origin of the object's placement bounding box, and the width and height for a cell) appear in the status bar.

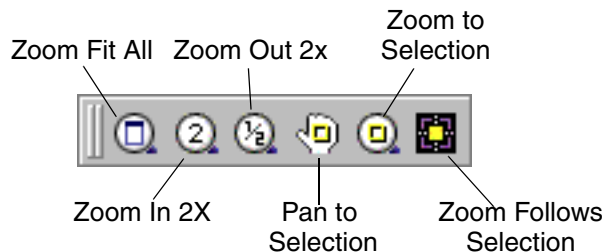
For more details about the Query tool, see the IC Compiler online Help.

Magnifying and Traversing the Design

You can use the zoom and pan tools and the zoom commands to find and display information in different parts of the design.

The View Zoom/Pan toolbar, shown in [Figure A-6](#), lets you instantly zoom or pan in increments defined by the tool. For example, you can zoom to a full view of the design, zoom in or out by a factor of 2, pan or zoom to a selection, and have zoom follow your selections. This toolbar can be found in the main, layout, timing analysis, and interactive clock tree synthesis windows.

Figure A-6 View Zoom/Pan Toolbar



To magnify a view, you can

- To double the magnification, click the  button on the View Zoom/Pan toolbar, or choose View > Zoom > Zoom In.
- Click the  button on the Mouse Tools toolbar and drag the pointer diagonally across a rectangular area to magnify it to fill the view.

To shrink a view, you can

- To shrink the view to half the magnification, click the  button on the View Zoom/Pan toolbar, or choose View > Zoom > Zoom Out.
- Click the  button on the Mouse Tools toolbar and drag the pointer diagonally across a rectangular area to shrink the view by fitting it within the area.

To display the entire design, you can

- To display the entire design, click the  button on the View Zoom/Pan toolbar, or choose View > Zoom > Zoom Fit All.

When only part of the design is visible in the active layout or schematic view or part of the histogram is visible in the active histogram view, you can use the Pan tool or the scroll bars to traverse the view to see different areas of the design or histogram. You can also use the pan commands to recenter the design by displaying selected objects or highlighted objects in the active layout or schematic view.

To traverse a view in any direction, you can

- Click the  button on the Mouse Tools toolbar, and drag the pointer in the direction that you want to move the design within the view.

Zoom Follow Selection

Zoom Fit Highlight

Pan to Selection

Pan to Highlight

Zoom to ... Zoom To dialog ... Coordinates box ... OK or Apply

Zoom Layout View to Current View

Recompute Cell Boundary

You can also enable a mechanism that automatically zooms to an object when it is selected, and you can reuse zoom and pan settings. For more information, see the following sections:

- [Following Selected Objects](#)
- [Reusing Zoom and Pan Settings](#)

Following Selected Objects

You can control whether a layout or schematic view automatically zooms to an object when you select it, thus increasing or decreasing the magnification of the design to fit the object within the view.

You can enable or disable this “follow-selection” mechanism separately for each open view. When you enable follow selection in a view, it remains enabled even when the view is not the active view. Follow selection is disabled by default.

To enable or disable the follow-selection mechanism for the active layout or schematic view,

- Click the  button on the View Zoom/Pan toolbar, or choose View > Zoom > Follow Selection.

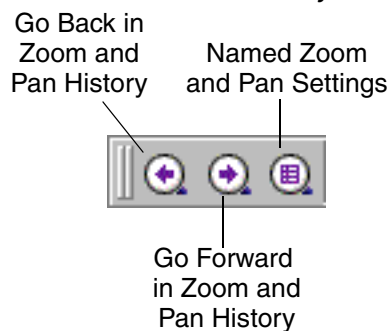
A check mark next to the Follow Selection command on the Zoom menu indicates that the follow-selection mechanism is enabled.

Reusing Zoom and Pan Settings

The IC Compiler GUI maintains a list of zoom and pan settings for each layout or schematic view. Each zoom and pan setting consists of both a zoom (magnification) level and a pan position (visible area). When you change the magnification or traverse the view (by panning or scrolling), the previous zoom and pan setting is added to the list.

The Zoom and Pan History toolbar, shown in [Figure A-7](#), lets you use previously used or specially defined zooms. It also lets you define special zoom settings for future use. This toolbar can be found in the main and layout windows.

Figure A-7 Zoom and Pan History Toolbar



You can

- Reverse the zoom level and pan position to the last zoom and pan setting by stepping backward in the list toward the initial display you encountered when you opened the view
- Reapply the next (previously reversed) zoom level and pan position by stepping forward in the list after stepping backward in the list
- Save the current zoom level and pan position in a list of named zoom and pan settings
- Return the view to a certain zoom level and pan position by selecting a named zoom and pan setting that you saved earlier in the session

For more information, see the “Reusing Zoom and Pan Settings” topic in IC Compiler online Help.

Drawing Rulers

You measure distances in a layout view by drawing rulers. A ruler can be composed of one or more horizontal, vertical, or diagonal segments. By default, the distances from the beginning of the ruler are labeled at the ends of each segment and at every tenth tick mark within a segment.

While you draw a ruler, the GUI displays a ghost image of the ruler in the drag object color (red by default), and displays the coordinates for the current point position on the status bar. The rulers also appear in any other layout views that you have open in the layout window.

Note:

You cannot reverse (by choosing Edit > Undo) ruler drawing operations.

To draw rulers in a layout view,

1. Click the  button on the Mouse Tools toolbar or choose View > Mouse Tools > Ruler Tool.

The Ruler Settings panel appears and the pointer changes shape. By default, cross hair rulers appear attached to the pointer in the layout view.

2. (Optional) Set options as needed on the Mouse Tool Options toolbar.

You can

- Select a direction option
- Display or hide the cross hair rulers attached to the pointer when you start a new ruler, The labels that show the distance from the ruler origin to the endpoint of each ruler segment, or the labels that show the segment length at the midpoint of each ruler segment
- Control whether the distances displayed by the ruler labels are absolute measurements from the design origin, relative measurements from the ruler origin, or relative measurements from each segment origin
- Change the grid snapping option for the ruler segments
- Record the ruler segment lengths or distances in the session transcript (console log view)

3. Click in the layout view where you want to begin the ruler.

A ghost line appears indicating the horizontal or vertical distance between the initial point and the current pointer location.

4. Click where you want to end the ruler segment.

The ruler segment appears.

5. (Optional) To draw another ruler segment, repeat steps 2 through 4.

6. To end the ruler, press the Escape key or right-click and choose End Ruler.

You can also end a ruler by selecting a different mouse tool (click a button on the Mouse Tools toolbar).

7. (Optional) To draw another ruler, repeat steps 2 through 6.

You can remove individual rulers or all rulers in the layout view, but you cannot remove or modify individual ruler segments.

To remove a ruler from the layout view,

1. Move the pointer over the ruler you need to remove.
2. Right-click and choose Remove Nearest Ruler.

To remove all rulers from the layout view,

- Click the  button on the Mouse Tools toolbar (or choose View > Mouse Tools > Clear Rulers).

Alternatively, you can remove all rulers by clicking Clear on the Ruler Settings panel.

Redrawing the Active View

Certain operations (such as zooming or panning) in a layout view cause the GUI to automatically redraw the view. In addition, you can manually redraw the active layout or schematic view at any time.

To redraw the active view,

- Choose View > Refresh.

If necessary, you can recompute the boundary of the entire design, including the die area, to fit within the active layout view.

To recompute the boundary of the entire design,

1. Choose View > Zoom > Recompute Cell Boundary.

The GUI recalculates the cell boundary coordinates.

2. Update the layout view by choosing View > Zoom > Zoom Fit All.

Examining Design Objects

In addition to layout and schematic views, the IC Compiler GUI provides tools and commands that you can use to select and examine design objects and other design data. You can select objects or timing paths, view and edit object properties, explore the design hierarchy, and view logic design and timing path data in design and path schematics.

For information about performing these tasks, see the following sections:

- [Selecting Objects by Name](#)
- [Selecting Timing Paths](#)
- [Viewing and Editing Object Properties](#)
- [Exploring the Design Hierarchy](#)

- [Viewing Design and Path Schematics](#)
- [Viewing Reports](#)

Selecting Objects by Name

You can use the Select By Name dialog box to search for objects of any type by specifying a name or name pattern or by defining a regular expression. Then you can select or highlight some or all of the objects found in the search.

To search for objects, you select an object type and a design scope. You can search

- All objects in the design
- Selected objects
- All or selected objects found in the previous search

You can define and save filters for different object types based on object name patterns and attribute or function values. You can also copy or modify existing filters.

The search results include the total number of objects found and a list of object names with the values of any attributes or functions used to generate the search. When you are satisfied with the search results, you can select the names of objects that you want to select, deselect, or highlight in the active layout view.

To search for and select objects by name or regular expression,

1. Choose Select > By Name to open the Select By Name dialog box.
2. Select an object type and a design scope in the left and right “Search for” lists.

The default object type is Leafiest and the default design scope is “In entire design.”

3. Specify the name or regular expression by doing one of the following:
 - To search for objects by name, select the “Wildcard matching” option (the default), and type an object name or name pattern (with wildcards) in the Name box.
 - To search for objects by using a regular expression, select the “Full regular expressions” option, and either define a search filter or select the name of a previously defined filter in the Filter list.

You can define a search filter with one or more terms. To start a new filter, click the Add Term button and select an object name, an operator, and a value in the table below the Name box. To add a term, select AND or OR in the Next Term list.

If you want to save the filter for future use, click the Save button to open the Save Filter As dialog box, type a filter name, and click OK. The GUI adds the filter name to the Filter list and saves the filter definition in your preferences file. You can delete a saved filter by selecting the filter name in the Filter list and clicking the Delete button.

4. Set other options as needed.
5. Click Search.

The names of the objects found in the search appear in the “Search results” list. Initially, all the names in the list are selected. You can modify the list by removing object names or by performing another search and adding the object names to the list.

6. Select the names of objects in the “Search results” list that you want to use to select, deselect, or highlight in the most recently active layout view.
7. Select, deselect, or highlight the objects by doing one or more of the following:
 - To select objects and replace the current selection, select the object names, select the Replace option and click the Change Selection button.
 - To select objects and add them to the current selection, select the “Add to” option and click the Change Selection button.
 - To deselect objects (remove them from the current selection, select the “Remove from” option and click the Change Selection button.
 - To highlight objects in the most recently active layout view, click the Add to Highlight button.

You can also display the object names in a new list view by clicking the Show in Table View button.

8. To close the Select By Name dialog box, click Close.

Selected objects are displayed in the selection color (white) in graphic views and indicated by reverse video in hierarchy views, their names appear in the selection list, and the number of selected objects appears in the status bar. Highlighted objects appear in the current highlight color.

Selecting Timing Paths

You can select the timing paths with the worst slack in the design or in a path group. You can also select individual timing paths to or from specific inputs, outputs, or registers. When you select paths with the worst slack in the design, you can

- Include specific paths
- Set the maximum number of paths in the design, in a specific path group, or to a single endpoint
- Set other timing path options

You can also specify the selection bus and the selection operation.

To select specific paths or the paths with the worst slack in the design,

1. Choose Select > Paths From/Through/To.

The Select Paths dialog box appears.

2. (Optional) Specify individual timing paths by entering the startpoint, throughpoint, or endpoint names necessary to identify the paths in the From, Through, or To boxes.

Only paths that begin at valid startpoints or end at valid endpoints are included in the selection. If you specify more than one startpoint or endpoint, all paths that start at any of the startpoints or end at any of the endpoints (and meet the other criteria) are included.

3. Set options to control the maximum number and type of paths in the selection.
 - To change the maximum number of paths to an endpoint, enter a value in the “Nworst paths” box.
 - To change the maximum number of paths, enter a value in the “Max paths” box.
 - To specify a path group, select its name in the “Group name” list.
4. Set other options as needed.
5. Click OK or Apply.

When you select a timing path, the pins and ports on the path appear in the selection color (white). Flylines appear in the selection color to show the connections between the objects on the path.

Viewing and Editing Object Properties

You can use the Properties dialog box to view property values for selected designs, design objects, or timing paths. The Properties dialog box lists object properties in a table with two columns for property names and property values and a row for each property. The properties you can view include object names, attribute values, and certain timing and placement values. The list of properties differs depending on the type of object you select.

Some object properties are attributes that you can edit by changing the value (if one is already assigned), applying a value (if one is not already assigned), or removing the attributes. A bold border in the value column indicates an editable property value.

To display object properties,

1. Select one or more objects.
2. Choose Edit > Properties.

The Properties dialog box appears with the number of selected objects displayed above the property list table. The property values for the first selected object appear in the table. If you change the current selection when the Properties dialog box is open, the dialog box changes to display the properties for the newly selected objects.

For some object types, you can control which types of properties appear in the table by selecting an option in the “Attribute group” list. By default, all the properties for an object appear in the table.

- For cells, you can display basic, placement, timing, user, or all properties.
- For other objects, you can display basic, user, or all properties.

You can also display the properties displayed by InfoTips in a layout view.

Note:

Timing attribute values do not appear until you perform an operation that updates timing information, such as generating a timing report or opening a histogram.

When you select multiple objects, the property lists are displayed separately for each object.

You can use the previous and next arrow buttons ( and ) to navigate from one list of object properties to another.

You can select an option to list the property values for all the selected objects together in a single table.

- If you display properties for all objects of the same type, the dialog box shows the values that are identical for all the objects and displays “<Multiple values>” for other values.
- If you try to display properties for all objects of more than one type, the dialog box appears empty.

For more information about using the Properties dialog box, see IC Compiler online Help.

Exploring the Design Hierarchy

Use the hierarchy browser to examine the design hierarchy and to display information about the hierarchical cells and logic blocks in the design. You can select hierarchical cells, leaf cells, or other objects you want to examine in a layout view or schematic view, or with other analysis tools.

If you are not familiar with a design, you can explore the hierarchy to understand its structure and gather information about the design objects (cells, ports, pins, and nets) in a cell or block. You can also use the hierarchy browser to

- Highlight cells or blocks (and their leaf cells) with different colors in both the hierarchy browser and the layout view
- Perform floorplan tasks such as creating plan groups and importing black box cells
- Display design schematics of hierarchical cells

To open the hierarchy browser, do one of the following:

- In the layout window, choose Partition > New Hierarchy Browser View.
- In the main window, choose Hierarchy > New Hierarchy Browser View.

The view window consists of two panes, with an instance tree on the left and an object table on the right. The instance name of the top-level design appears at the top of the instance tree. You can use the split bars between the panes to increase or decrease the relative widths of the instance tree and the object table.

The hierarchy browser indicates cell types (standard cells, macros, ILMs, and so forth) by displaying icons beside the cell names. If you highlight cell hierarchies with colors, the hierarchy browser displays the cell colors on the icons.

You can explore the complete hierarchical structure of the design and observe how many hierarchical blocks are present. To explore the design hierarchy, you can

- Click the expansion button (plus sign) next to an instance name to expand the instance tree, showing the names of the subblocks at the next level in the hierarchy
- Select an instance in the instance tree to display leaf cell information in the object table
Shift-click or Control-click instance names to select combinations of instances. The cell information includes cell instance names, cell types, and cell reference names.
- Select an object type in the list above the object table to display object information for cells, ports and pins, or nets.

The object table displays cell information by default. You can filter the cell names in the object table by cell type: hierarchical, hard or soft macro, plan group, black box, or ILM.

Note:

Although you can view and filter the names of objects inside a soft macro, you cannot select the objects or perform any operations on them.

You can use the arrow keys to navigate the instance tree or the object table and to expand or collapse levels in the instance tree. You can also sort, resize, and filter the columns in the object table.

For more information about the hierarchy browser, see IC Compiler online Help.

Viewing Design and Path Schematics

You can use the following schematic and symbol views to show graphical representations of the logic design:

- Path schematic view

Path schematic views show graphical representations of the logic design. You use these views to visually analyze timing and logic in the optimized design and to gather information that can help you guide later optimizations. Objects you select in a schematic or symbol view are cross-selected in other views. This capability allows you to efficiently analyze the logic and timing aspects of your design.

- Design schematic view

Design schematics display a design as a schematic composed of cell instances, pins, nets, and ports. The schematic shows both leaf-level logic (gates) and subdesigns (blocks). You can move up or down in the design hierarchy to view the schematic for each block in the hierarchy.

- Symbol view

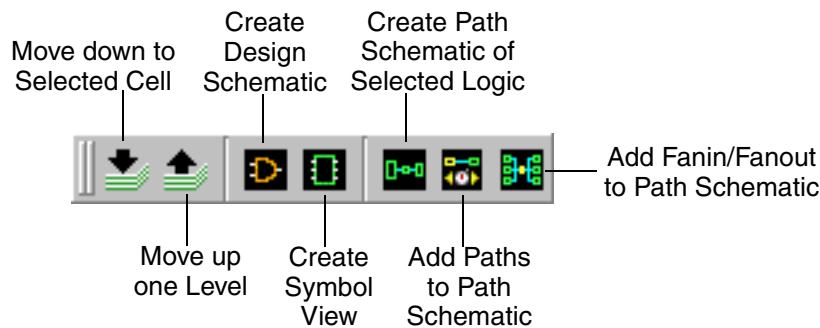
Symbol views display a hierarchical cell or a leaf cell as a black box with input and output ports. To change the view to a design schematic of the cell, double-click inside the box.

You use schematic and symbol views to visually analyze timing and logic in the optimized design and to gather information that can help you guide later optimizations.

Objects you select in a schematic or symbol view are cross-selected in other views. This capability allows you to efficiently analyze the logic and timing aspects of your design.

The Schematics toolbar, shown in [Figure A-8](#), lets you open design schematics (for selected designs), symbol views (for selected cells), and path schematics (for selected timing paths or design logic). When a design schematic is the active view, you can use this toolbar to move down or up a level in the design hierarchy. When a path schematic is the active view, you can use this toolbar to add timing paths, fanin logic, and fanout logic.

Figure A-8 Schematics Toolbar



You can use the View Settings panel to change the following display characteristics for the active schematic view:

- Object name visibility, text colors, and text sizes
- Object annotation visibility, text colors, and text sizes
- Object colors
- Display style for highlighted timing paths

To display the View Settings panel,

- Choose View > Toolbars > View Settings.

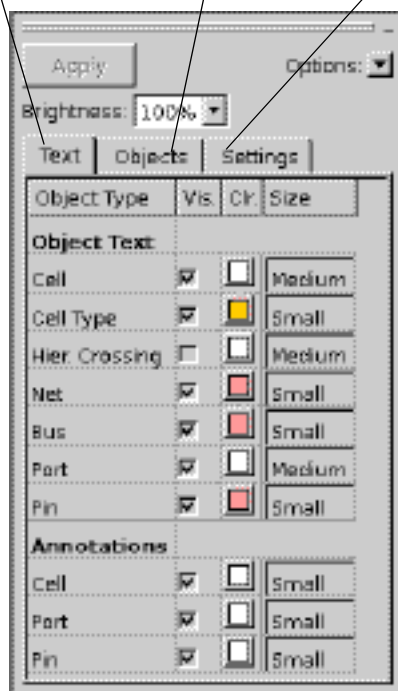
[Figure A-9](#) shows the View Settings panel as it appears when a schematic view is the active view.

Figure A-9 View Settings Panel for Schematic Views

Change visibility,
colors, and text sizes
for object labels
and annotations

Change colors
for different types
of objects

Change settings for
schematic display options



For more information, see the IC Compiler online Help.

Viewing Reports

The GUI displays reports in report views. By default, when you generate a report by choosing a report command on a menu, the GUI displays the report in a new report view.

Note:

When you enter a report command on the command line, the tool displays the report in the console log view and in `icc_shell` but does not open a report view.

You can use report views to

- Save a report in a text file
- Open a report previously saved in a text file

You can also open a blank report view and use it to view reports saved in text files.

To save a report displayed in a report view,

1. Click the  button at the bottom of the report view.

The “Save Report to File” dialog box appears.

2. Navigate to the directory where you want to save the report.
3. Select a file name, or type a file name in the “File name” box.

If you select or enter the name of an existing file, the GUI overwrites the file.

4. Click Save.

To display a report saved in a file,

1. Click the  button at the bottom of the report view.

The Open Report File dialog box appears.

2. Navigate to the directory where the report file is located.
3. Select the file name.
4. Click Open.

To remove the report text from a report view,

- Click the  button.

Visualizing the Physical Design

The layout view displays a very accurate graphic representation of the physical design. You use the layout view to visually examine the floorplan, placement, clock tree, and routing objects and gather information that can help you work on the design. You can open multiple layout views to simultaneously work with different areas of the design.

To open a layout view,

- In the layout window, choose View > New Layout View.

You can use the Overview panel to quickly magnify or traverse the design in the active layout view. If multiple layout views are open, you can change the active view to a different layout view.

Layout data can be densely packed with overlapping objects on different layers. You can control the visibility (display or hide) and selection (enable or disable) of individual object types or layers by setting options on the View Settings panel. You can also customize object or layer properties (color, fill pattern, and so forth) and set other layout and object display options. If you open multiple layout views, you can set different options for each view.

Typically you set layout view properties for each design. For example, the colors you use for individual layers might change depending on the number of layers in the design. If you change settings in the active view and want to use the same settings in another view or during a future session, you can save them in your preferences file. You can also restore previously saved view settings by loading them from your preferences file.

You can visualize the physical layout as explained in the following sections:

- [Navigating Through Layout Views](#)
- [Displaying Grid Lines](#)
- [Setting and Saving View Properties](#)
- [Adjusting the Color Brightness](#)
- [Enhancing Image Accuracy](#)
- [Displaying Cell Orientations](#)
- [Examining ILM Blocks](#)
- [Viewing Design Overlays](#)
- [Saving an Image of the Layout View](#)
- [Using Stroke Activated Commands](#)

Navigating Through Layout Views

In the layout window, the Overview panel shows you what portion of the design is visible in each open layout view.

- The portion of the design displayed in the active layout view is shown as a solid yellow rectangle.
- The portions of the design displayed in other layout views are shown as solid gray rectangles.

You can use the Overview panel to quickly magnify or traverse the design in the active layout view. When multiple layout views are open, you can quickly change the active view to a different layout view.

By default, the Overview panel appears docked below the toolbars at the top left corner of the layout window. If you hide the Overview panel, you can redisplay it by choosing View > Toolbars > Overview.

For more information, see the “Using the Overview Panel” topic in IC Compiler online Help.

Displaying Grid Lines

You can display or hide the lithography grid and the user grid. By default both grids are hidden.

To display or hide the lithography grid,

- Choose View > Show Litho Grid.

To display or hide the user grid,

- Choose View > Show User Grid.

To switch between the default grid spacing and ten times the default grid spacing,

- Choose View > Cycle Grid Spacing.

Setting and Saving View Properties

The View Settings panel provides options you can use to set display properties in the active layout view. You can also save the current settings in your preferences file, or load settings from the preferences file. If you open multiple layout views, you can set different options for each view.

To display or hide the View Settings panel,

- Choose View > Toolbars > View Settings.

A check mark beside the command on the Toolbars menu indicates that the View Settings panel is visible.

You can set options on the View Settings panel to

- Control object or layer visibility and selection
- Change object or layer style properties such as colors or fill patterns

- Set view or object (for cells, ports, pins, or nets) display options

Figure A-10 shows the View Settings panel as it appears when you open a new layout window, and explains what you can do with it.

Figure A-10 View Settings Panel for Objects in Layout Views

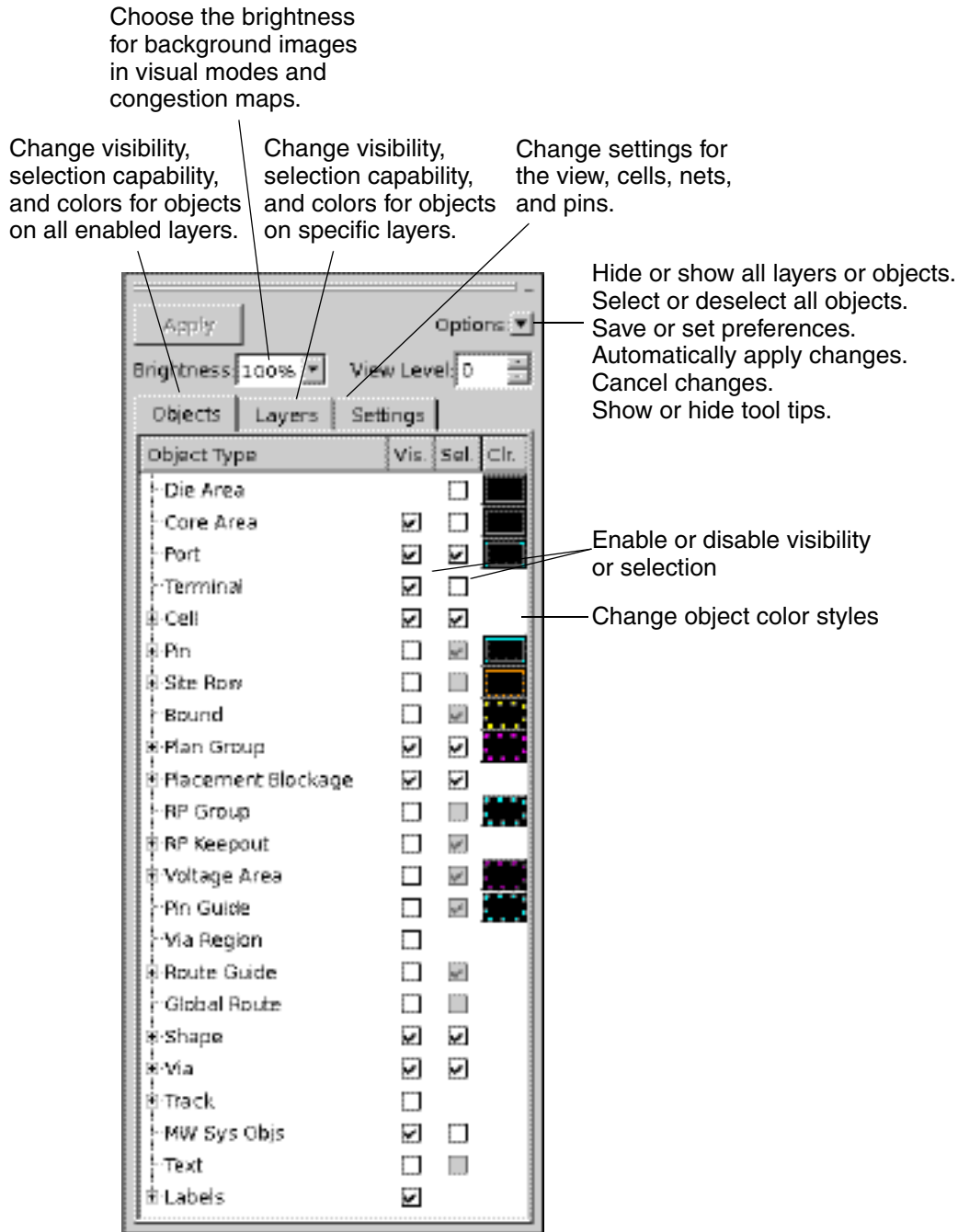
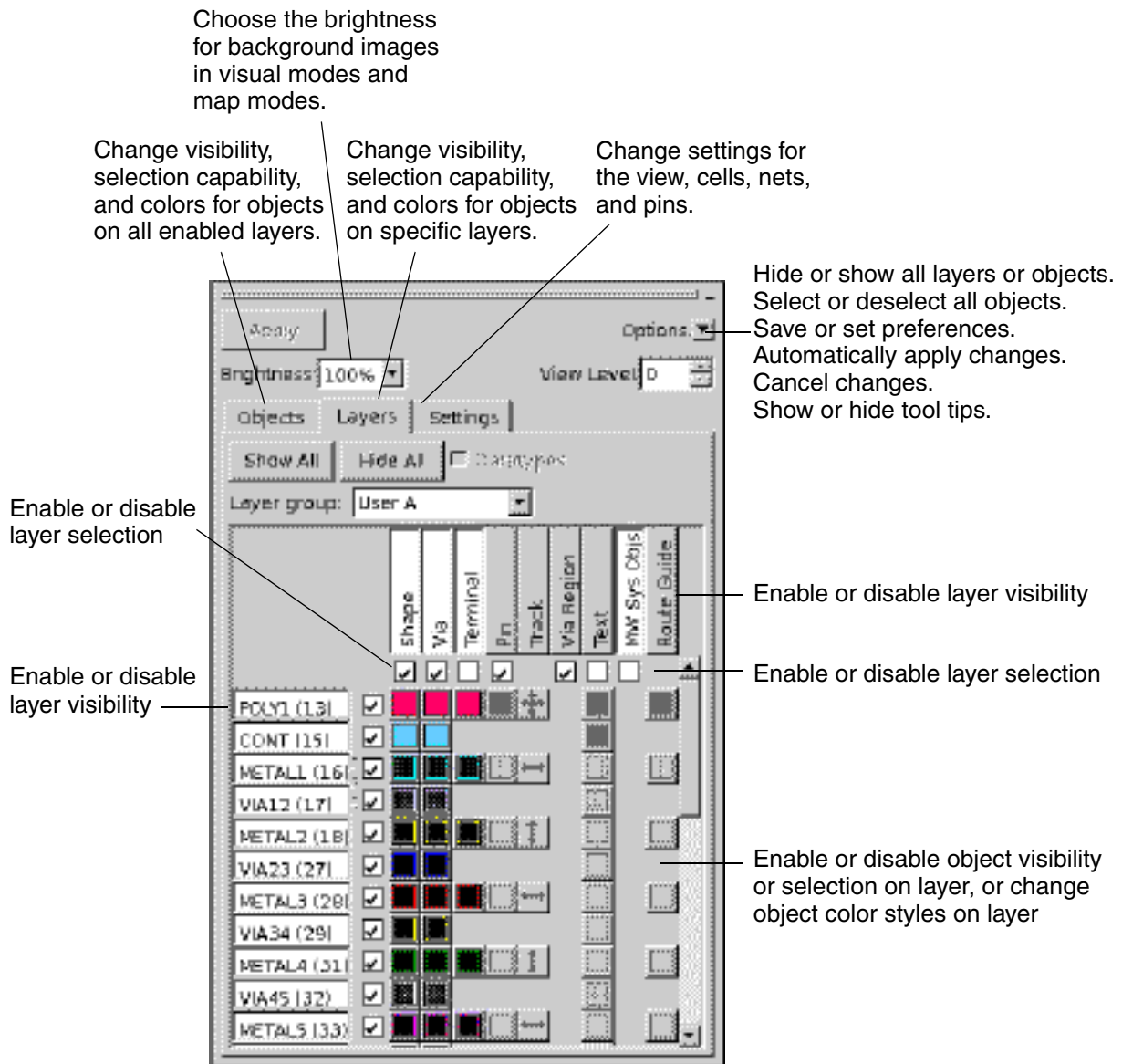


Figure A-11 shows the View Settings panel as it appears when you click the Layers tab to set options for layers or objects on layers.

Figure A-11 View Settings Panel for Layers in Layout Views



To change layout view display properties in the active layout view,

1. Set options as needed on the View Settings panel.
2. Click Apply.

By default, when you change settings on the View Settings panel, you must click Apply before the changes take effect in the active view. If you prefer, you can set the panel to automatically apply your changes as soon as you make them.

To enable or disable the automatic apply mechanism,

- Choose Options > Auto Apply.

A check mark beside the command on the Options menu indicates that the auto apply mechanism is enabled.

Alternatively, when the automatic apply mechanism is not enabled, you can reverse changes that you have not already been applied.

To reverse unapplied changes,

- Choose Options > Cancel.

You can set visibility or selection options or change style properties for

- Object types or subtypes
- All objects on a layer
- Object types on a layer

Object subtypes are categories of objects by property or attribute. For example, when cells are visible, you can display core cells and hide macro cells.

By displaying or hiding particular object types or layers, you can visually inspect just the physical layout data that you are interested in viewing while ignoring unrelated data. By enabling or disabling selection for particular object types or layers, you can control which types of objects are selected when you click or drag the pointer in a layout view.

You can customize how objects appear in the layout view by changing their style properties. Object styles set the appearance of objects in the active layout view. You can set the color, fill pattern, outline style, outline width, or exaggeration value for individual object types or layers.

Note:

The exact properties that you can change depend on an object's graphic representation in the layout view.

Typically you customize object style properties for a design. For example, the colors you use for individual layers might change, depending on the number of layers in the design.

The default display characteristics for design objects in a layout view are set for optimal viewing. These default characteristics work well for most designs. However, you can change the display characteristics for individual object types if you need to customize your layout views for a particular design or working environment.

IC Compiler does not save view settings when you exit the session. If you change layout view settings during a session and want to use the same settings in a future session, you can save them in your preferences file. You can also restore previously saved view settings by loading them from your preferences file.

To save the current settings for the active layout view,

- Choose Options > Preferences > Save to Preferences on the View Settings panel.

To restore the most recently saved layout view settings,

- Choose Options > Preferences > Set from Preferences on the View Settings panel.

For more information on using the View Settings panel, see the IC Compiler online Help.

Adjusting the Color Brightness

When you enable a visual mode or a map mode, the GUI dims the visible objects in the layout view. You can use the Brightness option on the View Settings panel to control the visual contrast between the visual mode or map mode colors and the other visible objects in the layout view. The GUI automatically dims the visible objects by resetting the color brightness to 33 percent.

To adjust the contrast between the color overlay and visible objects in the layout view,

- On the View Settings panel, select a value in the Brightness list.

The choices are 100, 75, 66, 50, 33, 29, and Off.

Enhancing Image Accuracy

You can display a photo-realistic view of the design by enabling TrueVue mode. TrueVue uses anti-aliasing to render small objects; large objects are not affected. By default, TrueVue is deactivated.

Note:

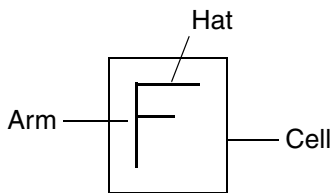
Over slow networks, graphic redraws are slower when TrueVue is enabled.

To enable or disable TrueVue in a layout view,

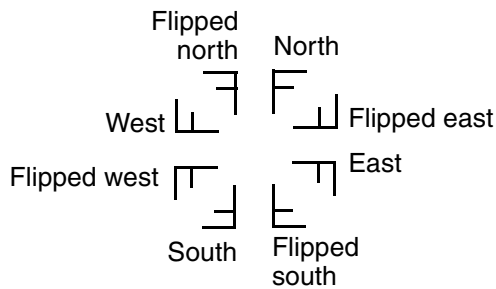
1. On the View Settings panel, click the Settings tab.
2. Click the View tab.
3. Select or deselect the TrueVue option.
4. Click Apply.

Displaying Cell Orientations

IC Compiler represents cell orientation in the layout view as an F. The position of the “hat” of the F indicates the first direction; the “arm” indicates the second direction.



Cells can be oriented in any of the following ways:



To display cell orientations in the active layout view,

1. On the View Settings panel, click the Settings tab.
2. Click the Cells tab.
3. Select or deselect the Show Cell Orientation option.
4. Click Apply.

You can change the orientations of individual cells. For details, see “Changing Cell Orientations” in the IC Compiler Online Help.

Examining ILM Blocks

An ILM is a structural model of a circuit that is modeled as a smaller circuit representing the interface logic of the block. In an ILM block, the gate-level netlist for the block is modeled by another gate-level netlist that contains only the interface logic of the block. ILMs contain cells whose timing affects or is affected by the external environment of a block.

If your design contains ILMs, you should visually check the ILM blocks for correct site locations when you validate the preplacement floorplan. You should also visually check for ILM blocks in highly congested areas when you examine the congestion map.

You can expand ILM blocks to display the objects (cells, ports, pins, nets, and so forth) inside the blocks. When ILM blocks are expanded, you can use layout window analysis and editing tools to analyze and modify the placement of cells within the blocks.

To expand ILM blocks,

1. Select a value greater than 0 in the “View level” option on the View Settings panel.
2. Click Apply.

To collapse (close) ILM blocks,

1. Select 0 in the “View level” option on the View Settings panel.
2. Click Apply.

You can control the visibility and selection of objects within ILM blocks by using the same View Settings panel options you use for other objects in the layout window.

Viewing Design Overlays

You can use the View Settings panel to overlay other designs on the primary design in the active layout view. For example, to view the fill data for a design, you can display the FILL view as an overlay over the CEL view. Similarly, you can view changes in a design by displaying an earlier version as an overlay over the current version.

To add an overlay on the active layout view,

1. On the View Settings panel, click the Settings tab and then the Overlays tab.

The panel lists the names of the current overlay designs and their brightness levels.

2. Click the Add button.

The Add Overlay Design dialog box appears, with a list of the designs that are open and available to be used as layout view overlays.

3. (Optional) If you need to open a design and display it as an overlay, click the Open Design button, select the design name in the Open Design dialog box, and click OK.

Note:

The GUI does not open a new layout window for a design you open to add as an overlay. However, when you open a new layout window (by choosing Window > New Layout Window), you can select any design as the primary design for the window.

4. Select the design name in the Add Overlay Design dialog box, and click OK.

The tool displays the design name on the Overlays tab, sets the brightness level to 50 percent, and displays the overlay on the active layout view.

When you add an overlay to the active layout view, the tool displays the design name on the View Settings panel and sets the brightness level to 50 percent. You can display or hide an overlay by controlling its brightness, but you cannot select, query, or edit objects in an overlay design.

To remove an overlay from the active layout view,

1. On the View Settings panel, click the Settings tab.
2. Click the Overlays tab.
3. Click the Remove button for the design you want to remove.

A message box appears asking if you want to close the design. Do one of the following:

- To keep the design open, deselect the "Close design after removing" option and click OK.
- To close the design, click OK.

If you have unsaved design changes, the Close Design dialog box appears. Set options to save or discard design changes as needed and click OK to close the design, or click Cancel to keep the design open without saving the changes.

Saving an Image of the Layout View

You can capture a picture of the design in a layout view and save it in an image. You can save the image in any of the following formats: BMP, JPEG, PNG, or XPM. You can save an image of entire layout window (the default) or just the layout view.

To save an image of the design in the active layout view,

1. Choose View > Save Screenshot As.

The Save Screenshot As dialog box appears.

2. Navigate to the directory where you want to save the image file and select or type a file name in the “File name” box.

You can save the image in BMP, JPEG, PNG, or XPM format by using the appropriate file name extension. The default format is PNG.

3. To save an image of just the layout view, select the “Grab screenshot of layout view window only” option.

By default the image includes the entire layout window.

4. Click Save.

Alternatively, you can use the `gui_set_setting` command. For example,

```
icc_shell> gui_set_setting \  
-window [gui_get_current_window -types Layout -mru] \  
-setting windowshot -value file_name.jpeg
```

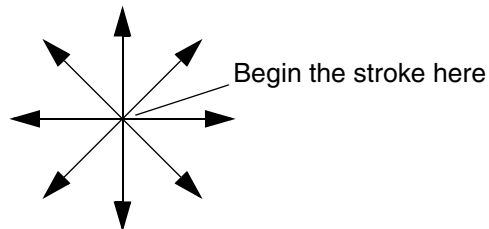
Using Stroke Activated Commands

You can stroke the pointer to activate preset commands, which can include `icc_shell` commands, in the active layout view. By default, the strokes are set up to activate commands that do the following:

- Increase or decrease the magnification of the design
- Pan the design
- Display the entire design

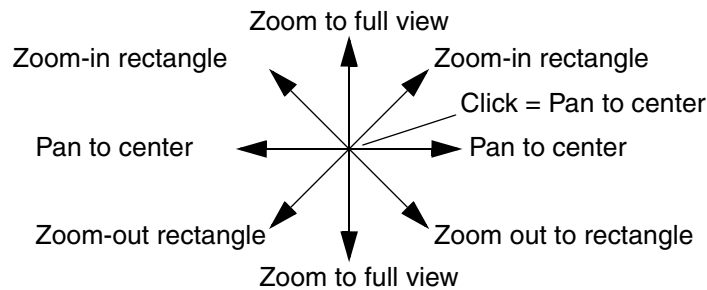
You can also define your own strokes. You can define up to 72 basic strokes and many more multisegmented strokes.

A basic stroke is a single click of the middle mouse button with the pointer in the layout view. A point-to-point stroke (dragging the pointer from point to point in one of the following directions while holding down the middle mouse button).

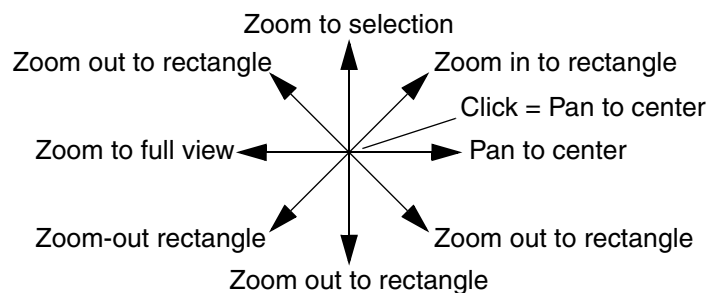


You can activate other commands for each of these strokes by also pressing a modifier key (Shift, Control, or Alt), or a combination of these keys as shown below.

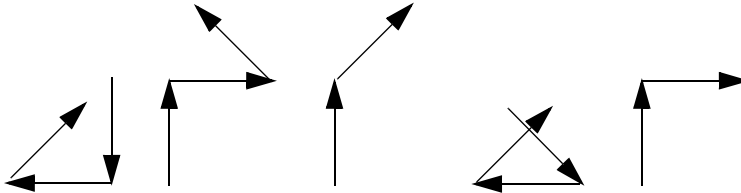
Basic strokes made without pressing Shift, Control, or Alt



Basic strokes made while pressing Control



In addition to the basic and modified stroke commands, you can perform multisegmented strokes. Each segment can go in any one of the eight directions used for the basic stroke commands. The following drawing shows some of the simple multisegmented strokes you can define:



To bind a command to a basic or multisegmented stroke, use the `set_gui_stroke_binding` command. By default, the GUI is set up to use basic strokes instead of multisegmented strokes. You can change this preference by using the `set_gui_stroke_preferences` command. For details about using these commands, see their man pages.

Analyzing the Physical Layout

The layout window provides tools and commands that you can use to visualize and analyze your design at each stage in the design process. These tools and commands are described in the following sections:

- [Using Visual Analysis Modes](#)
- [Exploring Relative Placement Groups](#)
- [Analyzing Signal Integrity](#)
- [Examining Routing and Verification Errors](#)

Using Visual Analysis Modes

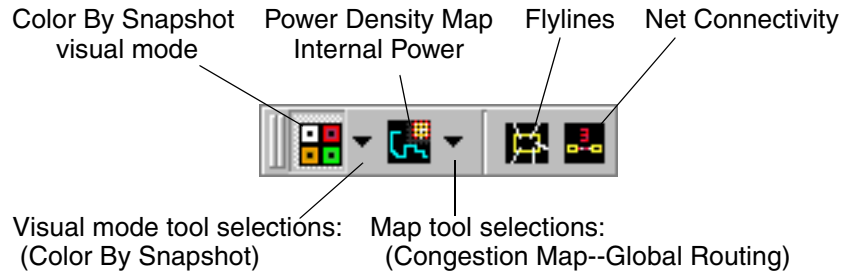
The layout view provides visual analysis modes that help you analyze the physical design by coloring specific types of design or timing data. You can use the Analysis toolbar, shown in [Figure A-12](#), to enable or disable the following visual analysis modes in the active layout view:

- Flylines
- Net connectivity

- Map modes
- Visual modes

The Analysis toolbar is available only in the layout window.

Figure A-12 Analysis Toolbar



In addition, the layout window provides tools that you can use to examine relative placement groups, analyze signal integrity, and examine routing and verification errors.

You can use the GUI to analyze the physical layout as explained in the following sections:

- [Displaying Flylines](#)
- [Displaying Net Connectivity](#)
- [Using Visual Modes](#)
- [Using Map Modes](#)

Displaying Flylines

Flylines let you view the connectivity of selected objects (cells, pins, or ports) in the layout window. You can select an object and display flylines to see the locations of the objects that have net connections to the selected object.

Flylines represent unrouted straight-line pin-to-pin connections. By default, the layout window shows flyline connections from the selected objects to macro cells, I/O ports, and other selected objects.

Note:

Make sure the desired object types (core cells, macro cells, I/O cells, pins, or ports) are visible and can be selected in the layout view (select their visibility and selection options on the View Settings panel).

To view the flylines from selected cells,

1. In the layout window, click the  button on the Analysis toolbar, or choose View > Flylines.

The Flyline Settings panel appears.

2. (Optional) Set options as needed on the Flyline Settings panel.

You can control how the flylines are displayed and change their appearance.

3. Select one or more cells, pins, or ports.

Flylines appear from the selected objects to the cells, pins, ports, or other selected objects with which they are connected. You can

- Zoom in if necessary to see individual flylines
- Pan or scroll to follow a flyline through the design

4. Select or deselect objects as needed to view their flylines.

If you highlight an object while its flylines are displayed, the flylines are also highlighted, and they remain highlighted when the object is no longer selected.

Note:

A flyline is a line that represents a single connection between two objects such as pins or cell instances. If you need to visualize the connectivity for an object such as a plan group that requires multiple connection lines, use the net connectivity display.

For more information, see IC Compiler online Help.

Displaying Net Connectivity

Net connectivity lets you view the connections of selected nets in the layout window. You can select a net and display net connectivity to see the locations of the objects that have connections to the selected net.

Net connectivity displays the net connections to an object as a line with a number representing one or more nets between objects. These objects can be macro cells (including soft macros, hard macros, and black boxes), I/O cells, bounds, or plan groups. Net connections represent unrouted straight-line pin-to-pin connections. By default, the layout window shows net connections from a selected net to macro cells, I/O ports, and other selected objects.

Note:

Make sure the desired net types are visible and can be selected in the layout view (select their visibility and selection options on the View Settings panel).

To view the connections from selected nets,

1. In the layout window, click the  button on the Analysis toolbar, or choose View > Net Connectivity.

The Connectivity Settings panel appears.

2. (Optional) Set options as needed on the Connectivity Settings panel.

You can control how the net connections are displayed and change their appearance.

3. Select one or more cells, pins, or ports.

Flylines appear from the selected objects to the cells, pins, ports, or other selected objects with which they are connected. You can

- Zoom in if necessary to see individual flylines
- Pan or scroll to follow a flyline through the design

4. Select or deselect objects as needed to view their flylines.

If you highlight an object while its net connections are displayed, the connections are also highlighted, and they remain highlighted when the object is no longer selected.

For more information, see IC Compiler online Help.

Using Visual Modes

Visual modes are analysis tools you can use to highlight specific design or timing information with colors in the active layout view. When you enable a visual mode, the GUI dims the visible objects in the layout view by default and opens the Visual Mode panel. The name of the active visual mode appears in the list at the top of the Visual Mode panel.

For floorplan and placement analysis, you can use visual modes to highlight

- Illegal cell locations
- Worst slack through cells
- Cells by hierarchy
- Net connections in an area
- Voltage areas

- Relative placement groups
- Scan chains

For routing analysis, you can use visual modes to highlight

- Worst slack timing paths
- Timing path pins
- Net capacitance

For signal integrity analysis, you can use visual modes to highlight

- Crosstalk victim and aggressor nets
- Static noise
- Switching noise

For clock tree analysis, you can use visual modes to highlight

- Clock trees
- Clock tree timing (latency or transition)

A visual mode groups design objects, timing data, or other information into categories or ranges called bins. The layout view displays each bin in a different color.

- To enable or disable visual mode, choose View > Visual Mode.
- To display a particular visual mode, choose a command from the Visual Mode menu on the Analysis toolbar.
- To change to a different visual mode, select its name in the list on the Visual Mode panel.

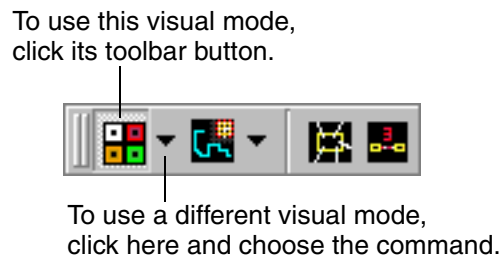
Note:

The first time you enable some visual modes during a session, you must load the data before you can view the color overlay in the layout view.

To use a visual mode,

- Click the current visual mode tool on the Analysis toolbar, or click the Visual Mode Menu button () to list the other visual modes (see [Figure A-13](#)).

Figure A-13 Visual Mode Tools on the Analysis Toolbar



Each visual mode displays a legend of colored bars on the Visual Mode panel. The legend colors correspond to colors in the overlay. You can click the visibility options at the left end of each bar to display or hide individual bins in the layout view.

From left to right, the legend consists of

- A column of visibility options
- A column of labels (names or range values) that identify the bins
- Columns of information about the bins, including (by default)
 - Color and fill pattern (colored boxes)
 - Count (total items in a bin)
 - Exaggeration value in pixels (hidden by default)
- A histogram showing the distribution of objects or values by the relative lengths of the colored bars

You can change the color styles for individual bins, hide or display information columns on the legend, and adjust the color exaggeration.

In visual modes that color design objects, you can select or deselect the objects in each bin. In visual modes that color discrete, unrelated sets of objects or other information, you can reorder the bars in the legend.

Using Map Modes

Map modes are analysis tools you can use to highlight specific floorplan, placement, or routing results with colors in the active layout view. When you enable a map mode, the GUI dims the visible objects in the layout view by default and opens the Map Mode panel. The name of the active map mode appears in the list at the top of the Map Mode panel.

You can generate color maps to display

- Congestion (global routing)
- Power density (leakage, internal, switching, dynamic, or total)
- Power network analysis (electromigration, voltage drop, or resistance)
- Cell placement (cell density or pin density)
- Critical areas (shorts and opens)
- Chemical mechanical polishing (thickness variations or hot spots)

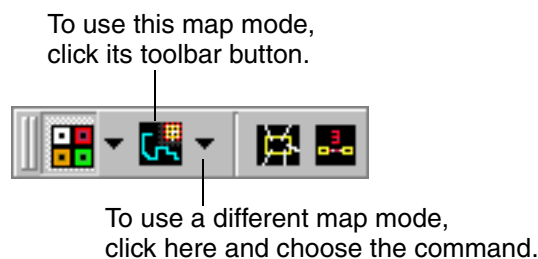
A map mode divides the core area into an array of boxes with colored edges. The box edges are colored and labeled to represent the map data (such as congestion or power density) through the horizontal and vertical planes. Each map color represents a range of values.

- To enable or disable map mode, choose View > Map Mode.
- To display a particular map mode, choose a command from the Map Mode menu on the Analysis toolbar.
- To change to a different map mode, select its name from the list on the Map Mode panel.

To use a map mode,

- Click the current map mode tool on the Analysis toolbar, or click the Map Mode Menu button () to list the other map modes (see [Figure A-14](#)).

Figure A-14 Map Mode Tools on the Analysis Toolbar



Each map mode displays a legend of colored bars on the Map Mode panel. The legend colors correspond to colors in the map. You can click the visibility options at the left end of each bar to display or hide individual bins in the layout view.

From left to right, the legend consists of

- A column of visibility options
- A column of labels (names or range values) that identify the bins

- Columns of information about the bins, including (by default)
 - Color and fill pattern (colored boxes)
 - Count (total items in a bin)
 - Exaggeration value in pixels (hidden by default)
- A histogram showing the distribution of map data by the relative lengths of the colored bars

You can change the color styles for individual bins, hide or display information columns on the legend, and adjust the color exaggeration.

Exploring Relative Placement Groups

A relative placement group can contain leaf cells, hierarchy (other groups), and placement keepouts. A hierarchy level is a relative placement group embedded in another relative placement group, either in the same design (included groups) or in a different design or designs (instantiated groups).

You can use the relative placement hierarchy view to examine the structures of relative placement groups and gather placement information about each group. You can also select the names of instances (leaf cells or embedded groups) that you want to examine in graphic views (such as the layout view) or with other analysis tools.

To open a relative placement hierarchy view,

- Choose Placement > New RP Hierarchy View.

The view window consists of two panes, with expandable level trees (one for each top-level group) in the left pane and instance placement information in the right pane. An instance can be a cell or another relative placement group. When you open a new relative placement hierarchy view, the names of the top-level groups appear in the left pane.

The right pane contains information about the instances (leaf cells or embedded groups) in the selected relative placement group. You can select the type of information display you want to view by clicking a tab at the top of the pane.

- Click Grid to display the instance names in a grid of rows and columns that indicate their relative positions in the site array.
- Click List to display a table that contains instance placement information, including the instance names, row and column numbers, and pin alignment names.

You can explore the complete hierarchical structure of a relative placement group, observe how many hierarchy levels are present and whether they are included or instantiated, and examine the placement grid and cell lists at each level.

For more information about using the relative placement hierarchy view, see IC Compiler online Help.

Analyzing Signal Integrity

The GUI provides tools you can use to analyze the affects of crosstalk by examining the victim and aggressor nets. The two major effects of crosstalk are crosstalk-induced delay and static noise.

- Crosstalk-induced delay occurs when the victim and aggressor nets switch at the same time or nearly the same times, either speeding up or slowing down the victim net transition times.
- Static noise is a glitch on the victim net that occurs when the victim net is in a steady state of either a high or low, and the aggressor net is switching.

You can analyze crosstalk problems on a design after performing global routing, track assignment, or detail routing. The minimum requirement is a global-routed design. You can

- Examine a high-level view of the delta delays for victim nets in the routed design by using the delta delay visual mode
- Examine a high-level view of the accumulated static noise levels for victim nets in the routed design by using the static noise visual mode
- View information about individual victim nets and their aggressor nets in the static noise analysis window or the path inspector window
- Highlight a victim net and its aggressor nets in the layout view by using the crosstalk visual mode

Before analyzing crosstalk to find static noise and crosstalk-induced delay problems, you can set crosstalk-related timing setup options and specify supply voltages. You can set these options by using the Set SI Options dialog box (choose Timing > Set SI Options) or the `set_si_options` command.

For more information about performing crosstalk analysis, see [Chapter 10, “Signal Integrity.”](#)

Examining Routing and Verification Errors

The error browser lets you examine routing design rule errors and verification errors. You select the errors you need to examine by error type or layer, view error information in the error browser, and view error locations in the layout view. You can

- Filter the error list
- Mark errors as fixed and optionally hide fixed errors
- Highlight errors in the layout view
- Display error information on the Query panel and optionally record the information in the session log
- Select errors interactively by using the Error Selection tool
- Save the error information in a file

You can also highlight the nets associated with detail routing errors and display the net nodes associated with open locator errors.

IC Compiler provides several routing and verification processes that check your design for rule violations and save the error data in files called error cells. You can examine detail routing errors, layout versus schematic (LVS) verification errors, DRC errors, signoff DRC errors, and third-party DRC errors. For more information about using these rule checking processes, see [“Verifying the Routed Design” on page 8-35](#).

The error browser loads detail routing errors from memory and other types of errors from the appropriate error cells. The error data is grouped by the error types or layers associated with the errors. The available error types depend on the verification process used to generate the error records. Each error record contains information that includes a description string and bounding box coordinates correlated with the physical design. An error record can also provide references to design objects. This is the data that the error browser displays

To open or close the error browser,

- Choose Verification > Error Browser.

The Error Browser dialog box appears. You can set error browser options by choosing commands on the Options menu at the top of the error browser. To display the Options menu, click the menu button (). If you need help using the error browser, click the Help button.

Note:

The Verification menu is available only in the block implementation task mode. To display the block implementation menus in the layout window, choose File > Task > Block Implementation or File > Task > All Tasks.

The error browser loads detail routing errors by default. If you want to examine other types of errors, you must load the error data from the appropriate error cells:

To load error data in the error browser,

1. Click the Reload button to open the Local Error Cell dialog box.
2. Select the options for the types of errors you want to examine, and deselect the options for the types of errors you do not want to examine.
3. For each option you select except Detail Route, type the error cell file name in the text box, or click the browse button and select the file in the Choose Cell dialog box.
4. Click OK.

The Error Browser dialog box contains three error panes: an error hierarchy tree on top, an error list in the middle, and an error detail pane on the bottom. You can sort the items in the error hierarchy tree or the error list by clicking in a column heading.

- The error hierarchy tree groups errors by type, the default, or by layer to help you find the problems you need to examine.
- The error list displays the ID number, error type, layer name, and a short summary for each error in the list.
- The error detail pane displays information about the errors that you select in the error list.

You can display information for up to 100 errors. If you hold the pointer over the error detail pane, the details for the first selected error appear in an InfoTip.

You can control the visual display of errors in the active layout view by choosing commands on the Options menu above the error hierarchy tree and setting options below the error detail pane.

To examine errors in the error browser,

1. (Optional) To group the errors by layer instead of type, choose Options > Show by Layer.
2. Select an error group in the error hierarchy tree.

To expand an error cell, error type, or layer in the hierarchy, click its expansion button (+). When you select an error group, only the errors in that group appear in the error list.

3. (Optional) Filter the error list until it displays just the errors you want to examine.

To set a filter, click the Filter button and select the filter options in the Filter Error List dialog box. You can display or hide selected errors, errors on specific layers when the errors are grouped by error type, or errors of a specific type when the errors are grouped by layer. For more details, see [“Filtering Errors in the Error List” on page A-50](#).

4. Select one or more errors in the error list.

You can Control-click or drag the pointer to select multiple errors.

The error browser displays information about the selected errors in the error detail pane. You can display information for up to 100 errors. If you hold the pointer over the error detail pane, the details for the first selected error appear in an InfoTip.

5. Examine the errors on the layout view.

By default, when you select an error in the error list, the layout view zooms in to magnify the error shape and colors it in the select color, which is white by default. The error shapes for the other errors in the error list appear in the DRC violation color, which is yellow by default.

You can adjust the layout view error display by setting options in the area below the error detail pane

- To control whether the tool displays all, selected, or no errors in the layout view, select an option in the Show list. The default option is All.
- To control whether the tool zooms in, pans, or neither to display the selected errors, select an option in the Follow list. The default option is Zoom.

To set the scaling factor for the Zoom option, select or type a value in the box beside the Follow list. The default scaling factor is 1.0.

- To control whether the tool dims other objects in the layout view when errors are displayed, select or deselect the Dim option. This option is deselected by default.

To further examine the errors, you can

- Select error shapes in the active layout view by using the Error Selection tool. For details, see [“Selecting Errors in the Layout View” on page A-51](#).
- Display error information on the Query panel by clicking an error shape with the Query tool. For details about using the Query tool, see [“Viewing Object Information” on page A-11](#).
- Highlight the nets for selected detail routing errors by choosing Options > Highlight Nets For Selected Errors. For details, see [“Examining Nets Associated With Detail Routing Errors” on page A-52](#).
- Display the net nodes for selected Open Locator type errors by choosing Options > Display Net Nodes For Selected Open Locators. For details, see [“Examining Net Nodes Associated With Open Locator Errors” on page A-52](#).

6. (Optional) Mark errors as fixed in the error list by selecting the errors and clicking the Fixed button.

You can display or hide fixed errors in the error list by choosing Options > Hide Fixed Errors. The fixed indicator is relevant only for the GUI session. It is not saved in the error file or the design library. For more details, see IC Compiler online Help.

7. Examine other errors in the error list by repeating steps 4 through 6, or examine errors in a different error group by repeating steps 2 through 6.

The error browser is a global tool: errors that you select in the active layout view are cross-selected in any other open layout views. However, only the active layout view changes when the error browser automatically zooms or pans to display the selected errors.

For more information about examining errors with the error browser, see the following sections:

- [Filtering Errors in the Error List](#)
- [Selecting Errors in the Layout View](#)
- [Examining Net Nodes Associated With Open Locator Errors](#)
- [Examining Nets Associated With Detail Routing Errors](#)
- [Examining Errors in an Area](#)
- [Saving the Error List Information in a File](#)

Filtering Errors in the Error List

When you examine errors in the error browser, you can filter the errors in the error list. You can display or hide selected errors, specific types of errors, or errors on specific layers.

To filter selected errors.

1. Select the errors you want to filter.

You can either hide the selected errors or hide all the other errors.

2. Click the Filter button to open the Filter Error List dialog box.
3. Select a filter operation option: Show or Hide.
4. Click OK.

When the errors are grouped by error type, you can filter the error list by layer. When the errors are grouped by layer, you can filter the error list by error type.

To filter the error list by layer or error type,

1. Click the Filter button to open the Filter Error List dialog box.
2. Select a filter operation option: Show or Hide.
3. Select the criteria option: Layers or Error Types.
4. Select the options for the layers or error types that you want to show or hide.
5. Click OK.

You can repeat this procedure to filter the remaining errors in the error list.

To remove the filters, you can

- Click the Clear Filter button in the Error Browser window.
- Click the filter button to open the Filter Error List dialog box, and click Clear Filters.

Selecting Errors in the Layout View

When you examine errors in the error browser, you can select detail route or verification errors in the active layout view by using the Error Selection tool. This tool is available only when the All option, the default, is selected in the Show list. Error selection is local to the error browser and the active layout view and does not affect the global selection in other views.

To enable the Error Selection tool in the error browser,

- Click the  button on the Mouse Tools toolbar, or choose View > Mouse Tools > Selection Tool.

The pointer becomes the Error Selection Tool pointer.

You can select or deselect error shapes in the active layout view.

- To select an error and deselect any other selected errors, click the error shape.
- To add an error to the current selection, Control-click the error shape:
- To remove an error from the current selection, Shift-click the error shape:

You can select or deselect multiple errors at the same time by dragging the pointer to draw a rectangular box around the error shapes.

- To select the errors in a rectangular area and deselect any other selected errors, drag the pointer to draw a rectangular box around the error shapes.

- To add the errors in the area to the current selection, press the Control key while you drag the pointer.
- To remove the errors in the area from the current selection, press the Shift key while you drag the pointer.

Examining Nets Associated With Detail Routing Errors

When you examine detail routing errors in the error browser, you can set an option to highlight the nets in the layout view that are associated with the errors selected in the error list.

To highlight the nets for selected detail routing errors,

1. Select the Detail Route error type in the error hierarchy tree.
2. Select one or more errors in the error list.
3. Choose Options > Highlight Nets for Selected Errors.

A check mark appears beside the command on the Options menu when this option is enabled.

Examining Net Nodes Associated With Open Locator Errors

When you examine open locator errors in the error browser, you can set an option to display the net nodes in the layout view that are associated with the errors selected in the error list.

To display the net nodes for selected open locator errors,

1. Select the Open Locator error type in the error hierarchy tree.
2. Select one or more errors in the error list.
3. Choose Options > Display Net Nodes for Selected Open Locators.

A check mark appears beside the command on the Options menu when this option is enabled.

Examining Errors in an Area

To analyze the relationships among errors due to their physical proximity, you can display just the errors that occur in a specific area of the design.

To display errors in an area of the design,

1. Select the error cell in the error hierarchy tree that contains the errors you want to examine.
2. Make sure the All option is selected in the Show list.
3. Display the area of the design you want to examine by using the Zoom In, Zoom Out, and Pan tools.
4. Enable the Error Selection tool, and then select errors in the area by clicking or dragging the pointer.
5. Filter the error list by clicking the Filter button, selecting the Show and Selected options in the Filter Error List dialog box, and clicking OK.

When you are ready to analyze errors in a different area, clear the error list filter by clicking the Clear Filter button, and repeat steps 3 through 5.

Saving the Error List Information in a File

When you examine errors in the error browser, you can save the error list information in a text file, and you can choose whether to exclude or include hidden errors.

To save the error list information in a file,

1. Choose one of the following commands on the Options menu:
 - To exclude hidden errors, choose Options > Save to File > Omit Hidden Records.
 - To include hidden errors, choose Options > Save to File > Include Hidden Records.

The Save Errors to File dialog box appears.

2. Navigate to the directory where you want to save the file, and type the file name in the "File name" box.
3. Click Save.

Editing the Physical Layout

For information about editing physical design objects and constraints, see the following sections:

- [Editing a Design](#)
- [Creating Objects](#)

Editing a Design

The layout window provides both mouse tools and editing commands that you can use to interactively view the physical design. You can access these tools and commands by using the Edit toolbar or the Edit menu.

The layout editing mouse tools allow you to interactively create, edit, or remove physical objects in a layout view. You can enable these mouse tools by clicking buttons on the Edit toolbar and Advanced Edit toolbar or by choosing commands on the Edit menu. The layout editing commands provide additional capabilities for adjusting the relative alignment and distribution of objects. [Table A-2](#) shows the standard conventions for all of the layout editing mouse tools.

Table A-2 Layout Editing Tool Conventions

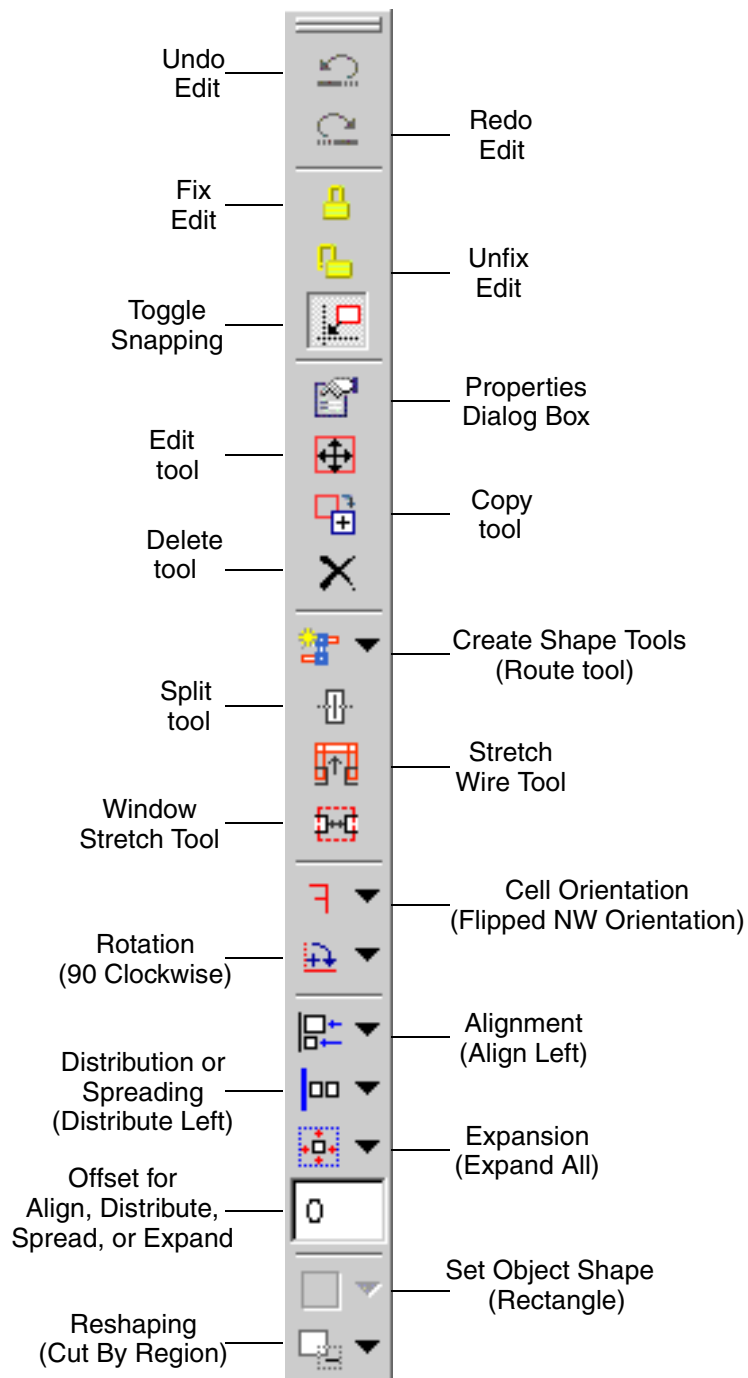
Event	Action
Click or Drag	Add points to perform the operation
Middle-click	Activate stroke menu
Right-click	Display context menu
Escape	Cancel pending input
F1	Cycle forward through overlapping objects
Shift-F1	Cycle backward through overlapping objects

The Edit toolbar, shown in [Figure A-15](#), provides tools and commands that allow you to

- Move, resize, copy, or remove objects
- Move, stretch, and split wires and shape objects
- Orient, rotate, align, distribute, spread, and expand selected objects
- Reshape (cut from or add to) objects
- Route nets, physical buses, and custom wires

For some of these editing functions, you can choose a specific way to perform the function from the list that opens when you click the  button. This toolbar also lets you enable or disable grid snapping and prevent or enable editing changes for selected objects by fixing or unfixing them. You can also use this toolbar to undo or redo recent editing functions.

Figure A-15 Edit Toolbar



[Table A-3](#) explains the global operations that you can use with any layout editing tool or command.

Table A-3 *Global Layout Editing Operations*

To perform this operation	Do this
Reverse the previous edit change	Click  , or choose Edit > Undo. Alternatively, you can use the <code>undo</code> command.
Reapply the subsequent edit change	Click  , or choose Edit > Redo. Alternatively, you can use the <code>redo</code> command.
Lock or unlock (fix or unfix) selected objects to prevent or allow edits	Click a lock tool ( or ) or choose Edit > Constraints > <code>lock_type</code> .
Enable or disable grid snapping	Click  or choose Edit > Snapping.

[Table A-4](#) explains the mouse tools you can use to edit the physical design.

Table A-4 *Editing Functions for Design Objects*

To perform this operation	Use this tool
Move objects	Click  or choose either Edit > Move/Resize or View > Mouse Tools > Edit Tool, and then select the objects and click or drag to the new location. Alternatively, you can use the <code>move_objects</code> command.
Resize objects	Click  or choose either Edit > Move/Resize or View > Mouse Tools > Edit Tool, and then select an object and drag an edge or corner to resize. Alternatively, you can use the <code>resize_objects</code> command.
Copy objects	Click  or choose either Edit > Copy or View > Mouse Tools > Copy Tool, and then select and drag the objects. Alternatively, you can use the <code>copy_objects</code> command.

Table A-4 Editing Functions for Design Objects (Continued)

To perform this operation	Use this tool
Remove objects	Click  or choose Edit > Delete, and then click to delete objects. Alternatively, you can use the <code>remove_objects</code> command.
Route single nets	Click  or choose Edit > Create > Route Tool, select the net, and click or drag to create route segments. You can set options as needed in the Create Route Tool dialog box.
Route one or more nets by adhering to preroute DRC rules	Click  or choose Edit > Create > Custom Wires, select the nets, and click or drag to create wire segments. You can set options as needed in the Create Custom Wires dialog box.
Split routing or user shapes	Click  or choose Edit > Split, and then drag a line or rectangle across the objects where the split is to occur. Alternatively, you can use the <code>split_objects</code> command.
Move and stretch objects while maintaining connectivity	Click  or choose Edit > Window Stretch or View > Mouse Tools > Window Stretch Tool, and then drag the objects. Alternatively, you can use the <code>window_stretch</code> command.
Stretch wires while maintaining connectivity	Click  or choose Edit > Stretch Wire, and then drag the wires. Alternatively, you can use the <code>stretch_wire</code> command.
Route physical buses or multiple nets	Click  or choose Edit > Advanced > Advanced Route Tool, select the buses or nets, and click or drag to create route segments. You can set options as needed in the Advanced Route Tool dialog box.

If you need to magnify or traverse the design or measure distances when an editing tool is active, you can temporarily activate the Zoom In tool, Zoom Out tool, Pan tool, or Ruler tool without deactivating the editing tool. For more details, see [“Selecting a Mouse Tool” on page A-3](#).

When you enable a layout editing tool, you can operate the tool in either preselection or postselection mode.

- In preselection mode, you select the objects you need to edit and then activate the tool.
- In postselection mode, you activate the tool and then select the objects you need to edit.

You can preview an object before selecting it by holding the pointer over its position in the active layout view. Information about the object appears in an InfoTip. If multiple objects occupy the same location, you can cycle through the objects, forward or backward, by pressing F1 or Shift-F1. Click to select the object you need when its name appears in the InfoTip. For details, see [“Previewing Objects in a Layout View” on page A-5](#).

[Table A-5](#) explains other ways you can edit the physical design.

Table A-5 *Additional Editing Functions for Design Objects*

To perform this operation	Do this
Align selected objects to one side or a center	Select an align tool ( ,  ,  ,  ,  , or ) or choose Edit > Align > <i>alignment_type</i> . Alternatively, you can use the <code>align_objects</code> command.
Expand selected objects to objects on one or all sides	Click an expansion tool ( ,  ,  ,  , or ) or choose Edit > Expand > <i>expansion_direction</i> . Alternatively, you can use the <code>expand_objects</code> command.
Orient selected objects to north	Click an orientation tool ( ,  ,  ,  ,  ,  , or ) or choose Edit > Orientation > <i>orientation</i> .
Distribute selected objects from a side	Click a distribution tool ( ,  ,  , or ) or choose Edit > Distribute > <i>side</i> . Alternatively, you can use the <code>distribute_objects</code> command.
Spread selected objects horizontally or vertically	Click a spread tool ( or ) or choose Edit > Spread > <i>direction</i> .

Table A-5 Additional Editing Functions for Design Objects (Continued)

To perform this operation	Do this
Rotate selected objects 90 or 180 degrees	Click a rotation tool ( ,  , or ) or choose Edit > Rotate > <i>rotation</i> . Alternatively, you can use the <code>rotate_objects</code> command.
Flip selected objects on an x- or y-axis	Click a flip tool ( , or ) or choose Edit > Rotate > <i>axis</i> . Alternatively, you can use the <code>flip_objects</code> command.

For more information about layout editing, see the following sections:

- [Fixing and Unfixing Objects for Edits](#)
- [Changing the Way Objects Snap to a Grid](#)
- [Undoing and Redoing Edit Changes](#)

Fixing and Unfixing Objects for Edits

You can fix objects in the active layout view to

- Prevent editing change
- Use as anchors for the distribution or splitting of multiple objects.

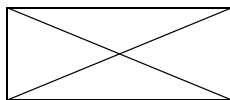
To fix objects,

1. Select the objects to fix.

The objects become highlighted.

2. Click the  button on the Edit toolbar, or choose Edit > Constraints > Fix Edit.

Each fixed object displays an X across it.



Fixed



Not fixed

Note:

When you use the Edit tool to move or resize objects, you can allow the tool to edit fixed objects by selecting the “Ignore fixed” option on the Mouse Tool Options toolbar.

To unfix objects and allow editing changes,

1. Select the objects to unfix.
2. Click the  button on the Edit toolbar, or choose Edit > Constraints > Unfix Edit.

Changing the Way Objects Snap to a Grid

IC Compiler automatically snaps objects to the grids associated with them. When you move, align, resize, expand, distribute, or spread objects, IC Compiler first moves them as you specify and then snaps them to the nearest associated grid. You can change the grids associated with each type of object, and you can turn snapping on or off.

To change the ways objects are snapped,

1. Display the Edit Snapping Settings panel if it is not already open by choosing Edit > Settings or View > Toolbars > Edit Snapping Settings.
2. Enable or disable snapping for all object types by clicking the “Enable snapping” option.
A check mark appears on the option when snapping is enabled.
3. Change the snapping grid for individual object types by selecting a grid type in the lists to the right of the object type names.
 - To snap objects to the lithography grid, select Min.
 - To snap objects to the nearest placement site, select Placement Site.
 - To snap objects to the nearest routing track, select Routing Track.
 - To snap objects to the middle of the nearest routing track, select Middle Routing Track.
 - To snap objects to the user grid, select User Grid.

Undoing and Redoing Edit Changes

You can reverse (undo) or reapply (redo) one or more editing commands. By using the undo function, you can reverse the most recent editing function. If you use this function repeatedly, you can reverse the most recent editing command for each use of this function until it encounters a nonediting command (such as a placement or routing command).

By using the redo function, you can reapply the most recently reversed editing function. If you use this function repeatedly, you can reapply the most recently reversed editing command for each use of this function.

- To undo an edit, click the  button on the Edit toolbar, or choose Edit > Undo.
- To redo an edit that you previously undid, click the  button on the Edit toolbar, or choose Edit > Redo.

Creating Objects

You can use the GUI or `icc_shell` commands to create floorplan objects and shape objects in the physical design.

Table A-6 lists the floorplan objects you can create and the menu commands and the `icc_shell` commands you can use to create them.

Table A-6 Creating Floorplan Objects

Object	Menu command	<code>icc_shell</code> command
Placement blockages	Floorplan > Create Placement Blockage	<code>create_placement_blockage</code>
Bounds	Floorplan > Create Bound	<code>create_bounds</code>
Plan groups	Floorplan > Create Plan Group	<code>create_plan_groups</code>
Route guides	Floorplan > Create > Route Guide	<code>create_route_guide</code>
Routing tracks	Floorplan > Create Track	<code>create_track</code>
Voltage areas	Floorplan > Create Voltage Area	<code>create_voltage_area</code>
Pin guides	Pin Assignment > Create Pin Guide	<code>create_pin_guide</code>

[Table A-7](#) lists the shape objects you can create and the menu commands and the `icc_shell` commands you can use to create them.

Table A-7 Creating Net Shapes and User Shapes

Object	Menu command	icc_shell command
Net shapes	Edit > Create > Net Shape	<code>create_net_shape</code>
User shapes	Edit > Create > User shape	<code>create_user_shape</code>
Vias and via arrays	Edit > Create > Via	<code>create_via</code>
Terminals	Edit > Create > Terminal	<code>create_terminal</code>
Text	Edit > Create > Text	<code>create_text</code>

For more information about creating objects, see IC Compiler online Help.

Analyzing Clock Tree Synthesis Results

You can use tools in the layout window and the interactive clock tree synthesis window to perform both high-level and detailed analyses of the clock tree structures and timing in your design.

- In the layout window, you can use visual modes to view the overall structures or timing distribution of the synthesized clock trees.
- In the interactive clock tree synthesis window, you can view information about all the clocks in the design and detailed information about individual clock trees.

You can select clock tree objects in the interactive clock tree synthesis window to highlight them in the layout view or color them in the clock trees visual mode. For information about layout view visual modes, see [“Using Visual Modes” on page A-41](#).

In both windows, you can use clock tree synthesis menu commands to perform tasks such as viewing clock tree options and setting, changing, or removing clock tree options, exceptions, and references. In addition, the interactive clock tree synthesis window provides tools you can use to

- View clock details and select clocks in the clock browser
- View the distribution of clock tree pin arrival times in a clock tree arrival histogram
- Explore the pin-to-pin fanout structure of a clock tree in the clock tree fanout browser

- Examine the fanin or fanout network for an entire clock tree or selected clock tree structures in clock tree network schematics
- Find, select, and view information about specific clock tree objects (cells, pins, or load nets) in clock tree synthesis object list views

The following sections describe the analysis tools provided in the interactive clock tree synthesis window:

- [Opening an Interactive Clock Tree Synthesis Window](#)
- [Exploring Clock Tree Structures](#)
- [Highlighting the Critical Paths](#)

Opening an Interactive Clock Tree Synthesis Window

To open an interactive clock tree synthesis window,

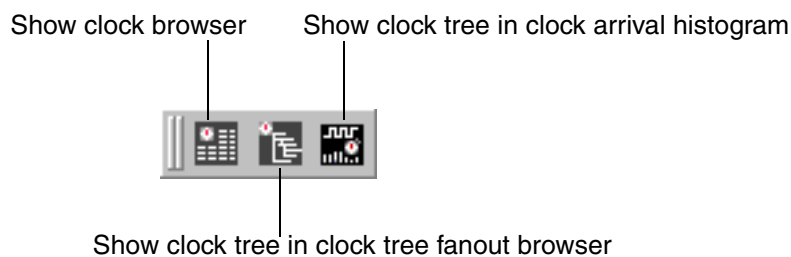
- Choose Window > New Interactive CTS Window.

When you open an interactive clock tree synthesis window, the clock browser view window appears. The clock browser displays a list of the clocks in your design with information about each clock.

The Browsers toolbar, shown in [Figure A-16](#), lets you

- Display the clock browser when it is hidden
- Display a clock arrival histogram.
- Select a clock in the clock browser view and display the clock tree in a clock tree fanout browser

Figure A-16 Browsers Toolbar



The clock browser and the clock tree fanout browser are the starting points for detailed clock tree analysis. You can open one clock browser in an interactive clock tree synthesis window, and you can open one clock tree fanout browser for each clock tree in the design.

You can use tools in the interactive clock tree synthesis window to perform both high-level and detailed analyses of the clock trees in your design.

You can

- Examine clock details in the clock browser
- View the distribution of clock tree pin arrival times in a clock arrival histogram
- Explore the pin-to-pin fanout structure of a clock tree in the clock tree fanout browser

You can navigate through the design or cell hierarchy and view information about the objects in a block or hierarchical cell. You can also select objects that you want to examine in schematics or other views. For more details, see the next section, “[Exploring Clock Tree Structures](#).”

- Examine the fanin or fanout network for selected clock tree structures in clock tree network schematics
- Find and view information about specific clock tree objects in clock tree synthesis object list views

You can select clock tree objects in the clock tree fanout browser for analysis with tools in the layout window.

Exploring Clock Tree Structures

You can use the clock tree fanout browser to display pin-to-pin fanout information about clock tree structures in the physical design. If you are not familiar with the clock trees in your design, you can explore the clock tree structures and gather information about objects (buffers, inverters, and preexisting cells) at each level. You can also select the names of objects you want to examine in graphic views or with other analysis tools.

- Before you perform clock tree synthesis, use the clock tree fanout browser to understand the existing clock tree structure.
- After you perform clock tree synthesis, use the clock tree fanout browser to analyze the resulting clock tree structures and for troubleshooting if you failed to achieve the expected QoR.

You can use the clock tree fanout browser together with clock network fanin and fanout schematics or the clock trees visual mode to determine what constraints or options you need to modify in order to improve the clock tree synthesis results.

To open a clock tree fanout browser,

1. In the interactive clock tree synthesis window, select a clock tree name in the clock browser.
2. Choose View > Show Clock Tree Hierarchy Browser.

The view window consists of two panes, with an expandable level in the left pane and an object information table in the right pane. You can explore the complete structure of a clock tree, observe how many levels are present, and examine the clock tree fanout information at each level.

The object table shows clock tree fanout information for the clock tree objects you select in a level tree. You can display information about child (next level) objects or associated objects by selecting an option in the list above the object table.

For more information about using the clock tree fanout browser, see the IC Compiler online Help.

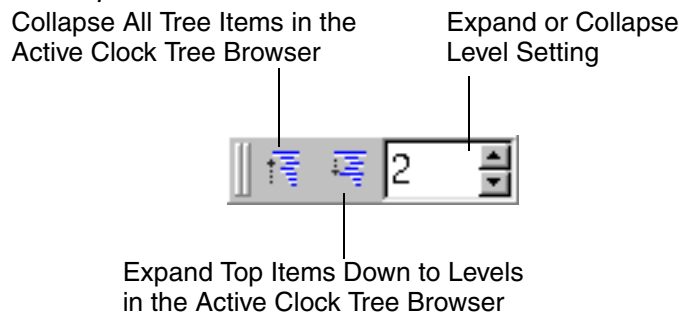
For more information about using the clock tree fanout browser, see the following sections:

- [Expanding or Collapsing Fanout Levels](#)
- [Highlighting a Selected Object](#)
- [Changing the Timing Mode](#)

Expanding or Collapsing Fanout Levels

The Expand Browser toolbar, shown in [Figure A-17](#), lets you expand or collapse fanout levels for the clock tree displayed in the active clock tree fanout browser. You can expand or collapse all levels, or expand a specified number of levels.

Figure A-17 Expand Browser Toolbar

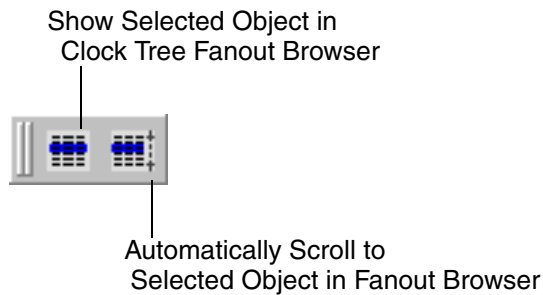


Highlighting a Selected Object

You can locate a specific clock tree object in the fanout browser by selecting the object in another view, such as a layout or schematic view. You can also enable a mechanism in a clock tree fanout browser that automatically displays and highlights an object when you select it in another view.

The Follow Selection toolbar, shown in [Figure A-18](#), lets you find a clock tree object in the clock tree fanout browser by selecting the object in another view.

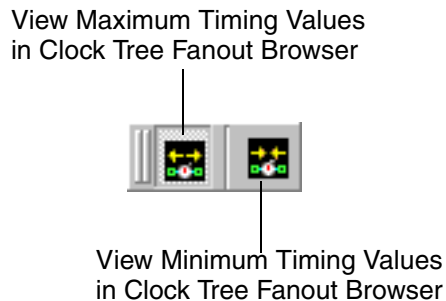
Figure A-18 Follow Selection Toolbar



Changing the Timing Mode

When you view object information in a clock tree fanout browser, you use the Clock Tree Timing Modes toolbar, shown in [Figure A-19](#), to control whether the tool uses maximum or minimum timing values to calculate the timing data.

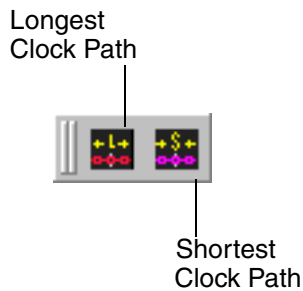
Figure A-19 Clock Tree Timing Modes Toolbar



Highlighting the Critical Paths

You can use the Critical Path toolbar, shown in [Figure A-20](#), to highlight the longest and shortest clock tree fanout paths in the active clock tree fanout browser or clock tree schematic.

Figure A-20 Critical Path Toolbar



Analyzing Timing Paths

The timing analysis window provides a centralized area for performing timing path analysis. The window includes a timing analysis driver, a path inspector, and other analysis tools such as histograms, schematics, path profiles, and timing reports.

You can use tools in the timing analysis window to perform both high-level and detailed analyses of the timing paths in your design. You can

- Examine timing path details in the timing analysis driver
- View histograms that show the distribution of worst path slack, endpoint slack, net capacitance, or other path details in histogram views
- Inspect the clock network and path data elements for a selected path in the path inspector
- View path delay profiles for selected paths in path profile views
- View selected paths, fanin and fanout cones, and other objects in path schematics
- View and save timing reports for selected paths or the paths with the worst slack in the design

The following sections describe the timing analysis tools:

- [Opening a Timing Analysis Window](#)
- [Viewing Timing Path Details](#)

- [Using Timing Analysis Views](#)
- [Inspecting a Timing Path](#)

Opening a Timing Analysis Window

When you open a timing analysis window, you can set the criteria for displaying timing path details in the timing analysis driver. By default, details for the 20 worst paths in each path group are displayed.

To open a timing analysis window,

1. Choose Window > New Timing Analysis Window.

The Select Paths dialog box appears.

2. (Optional) To display path details in the window, select a method for choosing the paths, and select other options as needed.
3. Click OK or Apply.

Note:

You can open the timing analysis window without displaying path details by clicking Cancel in the Select Paths dialog box when it appears.

Viewing Timing Path Details

When you open a timing analysis window, the timing analysis driver view window appears attached to the left side of the window. The timing analysis driver displays detailed information about the timing paths that have the worst slack in the design. You can view the timing path details and select paths for further examination with other analysis tools.

This view consists of a timing path table and a button bar.

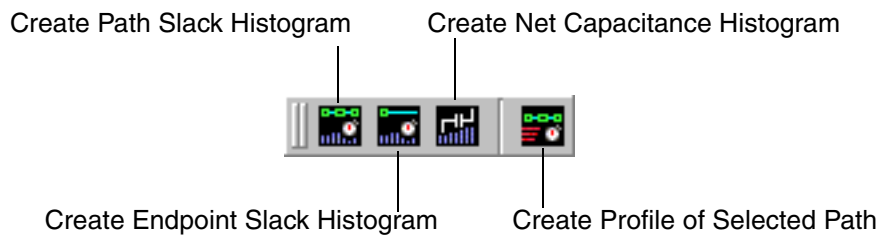
- The table displays path details for a list of timing paths.
- The button bar provides buttons you can use to manage the table contents and access other analysis tools.

The table columns show the startpoint name, endpoint name, and other details about each path. You can sort, filter, and customize the table, and you can save the path details in a text file.

Using Timing Analysis Views

The Timing Views toolbar, which is shown in [Figure A-21](#), lets you generate histograms (path slack, endpoint slack, and net capacitance) and path profiles for timing analysis. The histogram buttons open dialog boxes in which you can set options to customize the histogram. The path profile button is enabled only when a timing path is selected. This toolbar can be found in the timing analysis window.

Figure A-21 Timing Analysis Toolbar



For more information about using timing analysis views, see the following sections:

- [Viewing Histograms](#)
- [Examining Path Profiles](#)

Viewing Histograms

You use histograms to view distributions of timing values such as slack or net capacitance. Histograms are focal points of visual timing analysis that allow you to view the overall timing performance of your design.

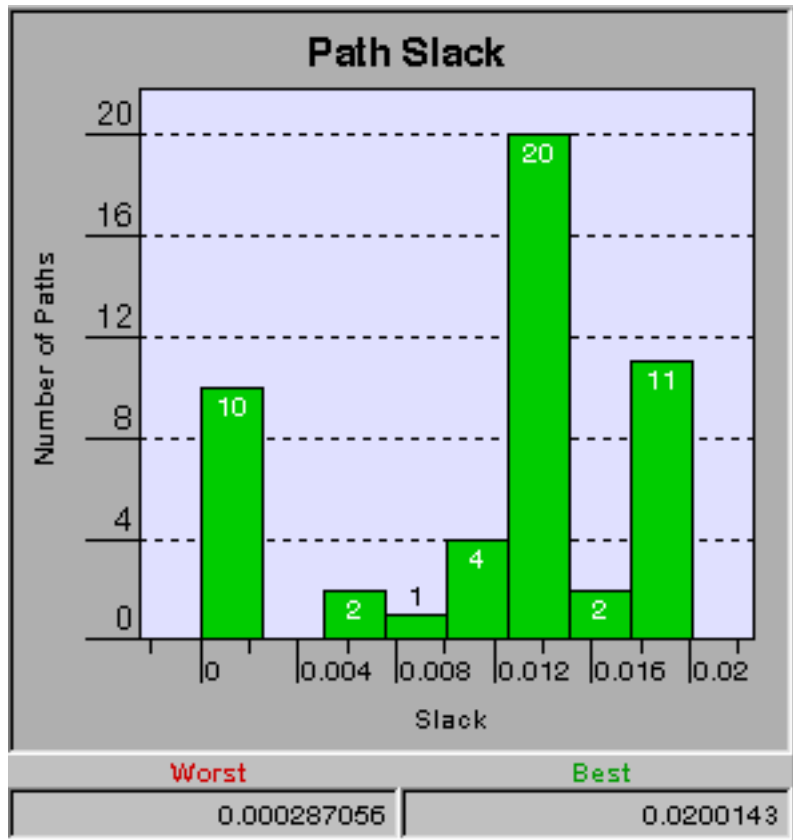
The IC Compiler GUI provides the following types of histograms for timing analysis:

- Endpoint slack histograms provide a high-level overview of the timing quality of your design. Create an endpoint slack histogram to identify endpoints that failed to meet their constraints.
- Path slack histograms provide a high-level overview of the timing quality for selected paths in your design. Create a path slack histogram to identify paths that failed to meet their constraints.
- Net capacitance histograms provide a high-level overview of the capacitance values for selected nets or for all the nets in your design. Create a net capacitance histogram to identify nets that have unacceptably high capacitance values.

In addition, you can create histograms that show the distribution of values for various types of path details (such as slack, endpoint clock skew, arrival time, or required time) listed in the timing analysis driver.

Histogram view windows are split into two panes. By default, the histogram bar graph appears in the left pane and an object table appears in the right pane. Figure A-22 shows an example of the bar graph in a path slack histogram. The bins represent the number of paths (y-axis) versus their slack values (x-axis).

Figure A-22 Example of Path Slack Histogram



The number at the top of the tallest bin indicates the number of paths in the bin. Green bins (on the positive side of 0) contain paths that met their constraints. Red bins (on the negative side of 0) contain paths that failed to meet their constraints. You can

- Hold the pointer over a bin in the bar graph to display its value range and number of objects or paths in an InfoTip below the pointer
- Click a bin to display its contents (object names and slack or capacitance values) in the object table
- Select objects in the object table for further examination with other analysis tools such as path schematics or reports
- Filter the object table, limiting it to certain objects based on a character string or regular expression that you define

Examining Path Profiles

You can use the path profile view to visualize the relative importance of different cells and nets in contributing to the total delay of a timing path.

The path delay profile shows the pin-by-pin distribution of delays on a path—the same information that the timing report provides, except in graphic form. You can view two different types of profiles: hierarchical and flat.

The path profile view is identical to the path profile tool that you can view by clicking the Path Delay tab in a path inspector window.

To view the delay distribution along a timing path,

1. Select one or more timing paths.

2. Click the  button on the Timing Views toolbar, or choose Timing > Path Profile View.

The path profile view window appears. The hierarchical profile appears by default the first time you open this view.

3. Click the tab for the type of path profile you want to view.

- To view a hierarchical profile, click the Hierarchy tab.
- To view a flat, leaf-level profile, click the Flat tab.

The combined delays for each path and the relative delay contribution for each pin appear graphically in bar graphs that represent the percentages of the total path delay. For each timing path, the bar graph shows the percentages of the individual pin delays from startpoint to endpoint. For each pin on a path, the bar graph shows the percentage contribution of the pin delay to the total path delay.

Inspecting a Timing Path

You can select an individual timing path and use the path inspector window to examine detailed information about the clock network and datapath elements in the path. The path inspector provides tools for viewing different aspects of the path, including

- The path status
- Path and clock summaries
- Clock network and path data elements
- Path delay profiles

- Crosstalk victim and aggressor nets
- A path schematic with clock path highlighting

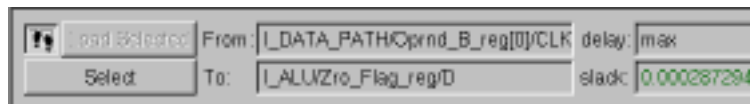
To open a path inspector window,

1. Select the path you want to inspect.
2. Click the Inspector button on the timing analysis driver button bar.

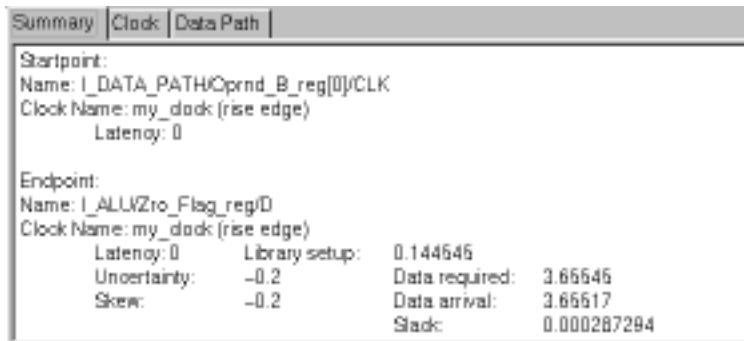
Alternatively, you can select a path and choose View > Show Path Inspector.

The path inspector window displays path information on three panels:

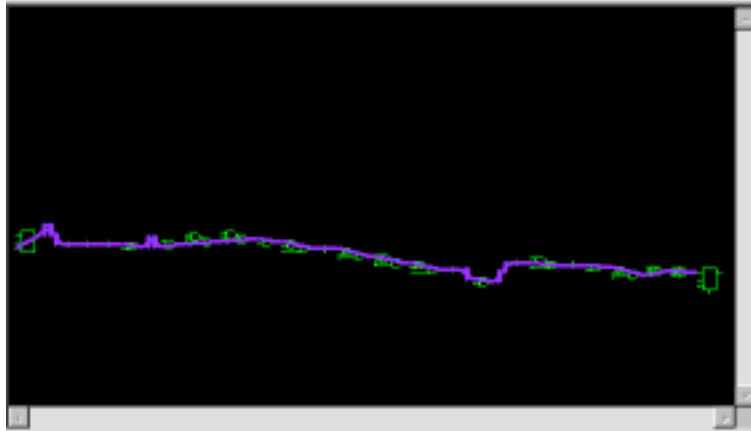
- The Status panel displays the path startpoint, endpoint, delay type, and slack. It also provides controls for automatically or manually loading another path.



- The Tab panel displays path summaries, clock network and path data element tables, path delay profiles, and crosstalk victim and aggressor net lists in a set of tabbed pages. You can copy the summaries and paste them into text files, export the tables to text files, customize the element tables, and generate delay calculation reports for selected cells or nets.



- The Schematic View panel displays a path schematic of the complete path (datapath, clock launch path, and clock capture path) with data and clock path overlays.



You can add fanin and fanout logic to the path schematic.

The legend below the Schematic View panel provides options you can use to display or hide the paths and buttons you can use to display or hide the overlays.



B

Defining and Analyzing Clocks

An essential part of timing analysis is the accurate specification of clocks and clock effects such as latency (the delay from the clock source) and uncertainty (the amount of skew or variation in the arrival of clock edges). This chapter describes the specification, reporting, and analysis of clocks.

This chapter contains the following sections:

- [Clock Overview](#)
- [Specifying Clock Characteristics](#)
- [Specifying Clock Network Effects](#)
- [Using Multiple Clocks](#)
- [Propagating Clock Paths Through a Non-Unate Clock Network](#)
- [Specifying Clock Sense](#)
- [Defining Data-to-Data Checks](#)
- [Specifying Clock-Gating Setup and Hold Checks](#)

Clock Overview

IC Compiler supports these types of clock information:

Multiple clocks

You can define multiple clocks that have different waveforms and frequencies. Clocks can have real sources in the design (ports and pins) or can be virtual. A virtual clock has no real source in the design itself.

Clock network delay and skew

You specify the delay of the clock network relative to the source (clock latency) and the variation of arrival times of the clock at the destination points in the clock network (clock skew). For multiclock designs, you can specify interclock skew. You can specify an ideal delay of the clock network for analysis before clock tree generation, or IC Compiler can compute the delay of the clock network for analysis after clock tree generation.

Gated clocks

You can analyze a design that has gated clocks. A gated clock is a clock signal under the control of gating logic (other than simple buffering or inversion of a clock signal). IC Compiler performs both setup and hold checks on the gating signal.

Generated clocks

You can analyze a design that has generated clocks. A generated clock is a clock signal generated from another clock signal by a circuit within the design itself, such as a clock divider.

Clock transition times

You can specify the transition times of clock signals at register clock pins. The transition time is the amount of time it takes for the signal to change from one logic state to another.

Specifying Clock Characteristics

You must specify all clocks used in the design. The clock information includes the

- Period and waveform
- Latency (insertion delay)
- Uncertainty (skew)
- Divided and internally generated clocks
- Clock-gating checks
- Fixed transition time for incomplete clock networks

IC Compiler supports analysis of the synchronous portion of a design and analyzes paths between registers or ports. For a design with multiple interacting clocks, IC Compiler determines the phase relationship between the startpoint and endpoint clocks. The clocks can be single-phase, multiple-phase, or multiple-frequency clocks.

Creating a Clock

The `create_clock` command creates a clock at the specified source. A source can be defined at an input port of the design or an internal pin. IC Compiler traces the clock network automatically, so the clock reaches all registers in the transitive fanout of its source.

A clock you create with the `create_clock` command has an ideal waveform that ignores the delay effects of the clock network. After you create the clock, you must describe the clock network to perform accurate timing analysis (see [“Specifying Clock Network Effects” on page B-20](#)).

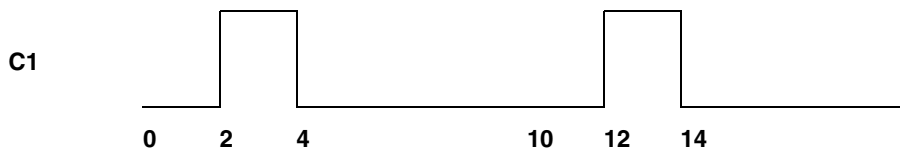
Example

To create a clock on ports C1 and CK2 with a period of 10, a rising edge at 2, and a falling edge at 4, enter

```
icc_shell> create_clock -period 10 -waveform {2 4} \  
           {C1 CK2}
```

The resulting clock is named C1, to correspond to the first source listed. C1 produces the waveform shown in [Figure B-1](#).

Figure B-1 C1 Clock Waveform



IC Compiler supports the analysis of multiple clocks propagated to a single register. You can define multiple clocks on the same port or pin by using the `-add` option of the `create_clock` command.

Creating a Virtual Clock

You can use the `create_clock` command to define virtual clocks for signals that interface with external (off-chip) clocked devices. A virtual clock has no actual source in the current design, but you can use it for setting input or output delays.

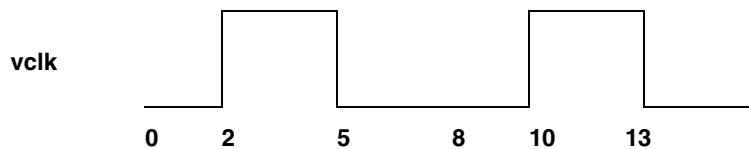
Example

To create a virtual clock named `vclk`, enter

```
icc_shell> create_clock -period 8 -name vclk \  
              -waveform {2 5}
```

The `vclk` clock has the waveform shown in [Figure B-2](#).

Figure B-2 vclk Clock Waveform

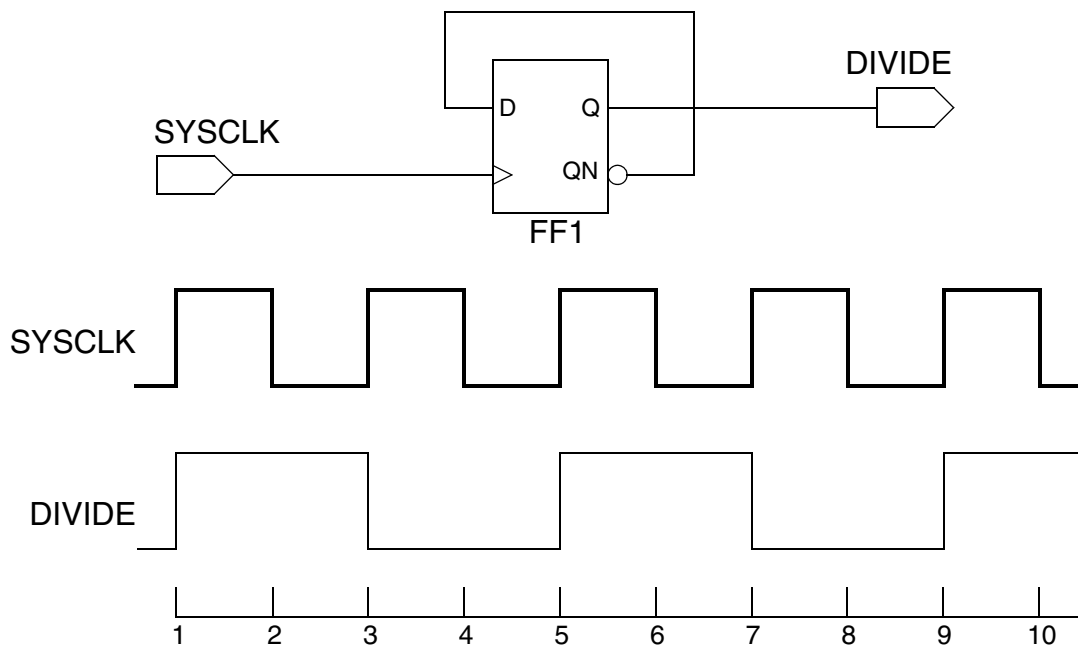


Creating Internally Generated Clocks

A design might include clock dividers or other structures that produce a new clock from a master source clock. IC Compiler represents these structures with a generated clock object.

[Figure B-3](#) shows the waveforms of the master and generated clock for a divide-by-2 clock generator (assuming that the generated clock is ideal and that there is no clock-to-Q delay). IC Compiler does not consider the circuit logic of a clock generator, so you must specify the behavior of a generated clock as a separate clock. However, you can define the relationship between the master clock and the generated clock, so that the generated clock characteristics change automatically when the master clock is changed.

Figure B-3 Divide-by-2 Clock Generator



The `create_generated_clock` command specifies the characteristics of an internally generated clock. By default, using `create_generated_clock` on an existing generated clock object overwrites that clock. Generated clock objects are expanded to real clocks at the time of analysis. The clock expansion happens automatically within `report_timing` or explicitly with `update_timing`.

The generated clock can be created as a frequency-divided clock (`-divide_by` option), frequency-multiplied clock (`-multiply_by` option), edge-derived clock (`-edges` option), or sense-constrained clock (`-invert` or `-preinvert` option).

Creating a Divide-By Generated Clock

There are three methods of creating a divide-by generated clock:

- Use the `-divide_by` option
- Define the clock based on rising edges
- Define the clock based on falling edges

The following sections describe these methods.

Creating a Divide-by Clock by Using -divide_by. To specify a divide-by clock, use the `-divide_by` option of the `create_generated_clock` command and specify the frequency division factor. For example, to create the divide-by-2 generated clock in [Figure B-3](#), specify 2 as the frequency division factor:

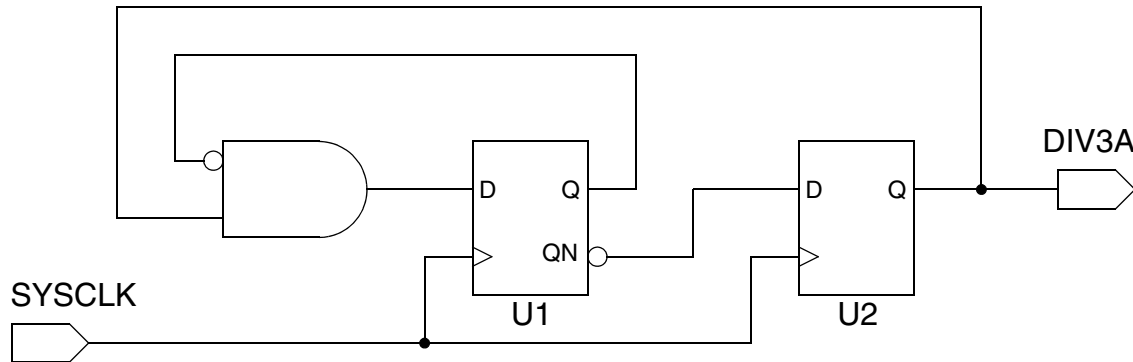
```
icc_shell> create_generated_clock -name DIVIDE \  
          -source [get_ports SYSCLK] -divide_by 2 \  
          [get_pins FF1/Q]
```

Specify a port or a pin (not a clock name) as the master source from which the new clock is generated. Specify a pin as the creation point for the new generated clock.

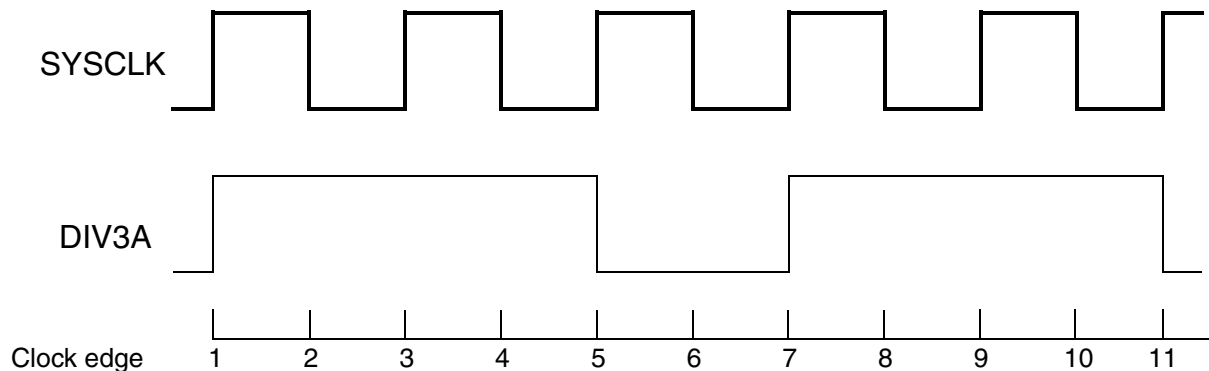
Note that generated clock edges are based on occurrences of rising edges at the master clock source pin (specified with the `-source` option). If you need to create a generated clock based on falling edges at the master clock source pin, use the `-edges` option rather than the `-divide_by` option (see [“Creating a Divide-by Clock Based on Falling Edges” on page B-8](#)).

Creating a Divide-by Clock Based on Rising Edges. You can use `create_generated_clock -edges` to specify the generated clock in terms of edges of the master clock waveform on the master pin. For example, consider the clock generator circuit shown in [Figure B-4](#).

Figure B-4 Divide-by-3 Clock Generator



```
create_clock -period 2 -name SYSCLK [get_ports SYSCLK]
```

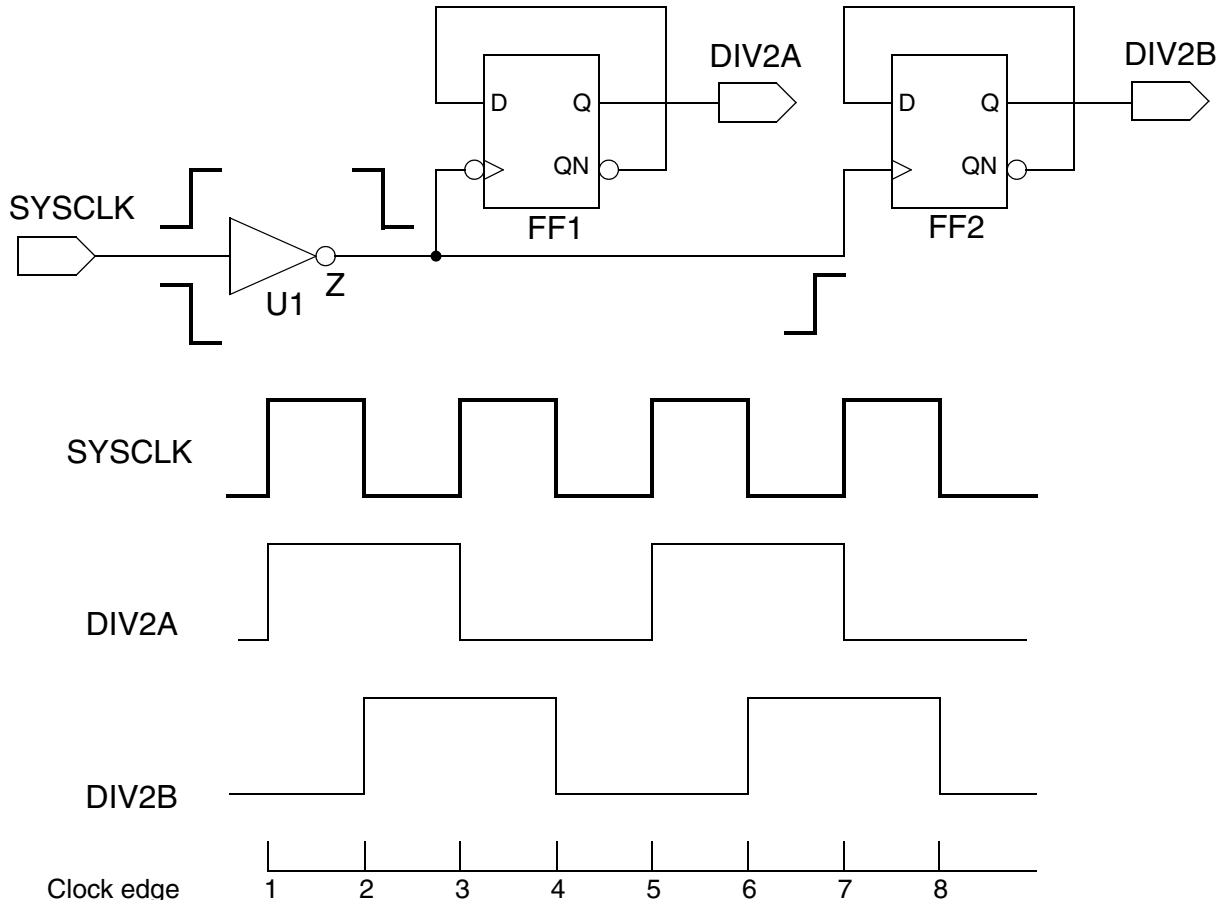


The generated clock signal DIV3A has a period three times longer than that of the master clock, with an asymmetrical waveform. To specify this waveform, you can enter

```
icc_shell> create_generated_clock -edges { 1 5 7 } \
-name DIV3A -source [get_ports SYSCLK] \
[get_pins U2/Q]
```

Creating a Divide-by Clock Based on Falling Edges. If you need to create a generated clock based on falling edges at the master clock pin, use the `create_generated_clock` command with the `-edges` option. The generated clock examples in [Figure B-5](#) demonstrate how to do this.

Figure B-5 Generated Divide-by-2 Clocks Based on Different Edges



The generated clock DIV2A is based on the rising edge of the master clock at the SYSCLK port, so you can specify the generated clock by using either the `-divide_by` option or the `-edges` option.

```
icc_shell> create_generated_clock -name DIV2A \
        -source [get_ports SYSCLK] \
        -divide_by 2 [get_pins FF1/Q]
icc_shell> create_generated_clock -name DIV2B \
        -source [get_ports SYSCLK] \
        -edges { 1 3 5 } [get_pins FF1/Q]
```


The generated clock DIV2B is based on the falling edge of the master clock at the SYSCLK port, so you cannot use the `-divide_by` option. However, you can still use the `-edges` option.

```
icc_shell> create_generated_clock -name DIV2B \  
        -source [get_ports SYSCLK] \  
        -edges { 2 4 6 } [get_pins FF2/Q]
```

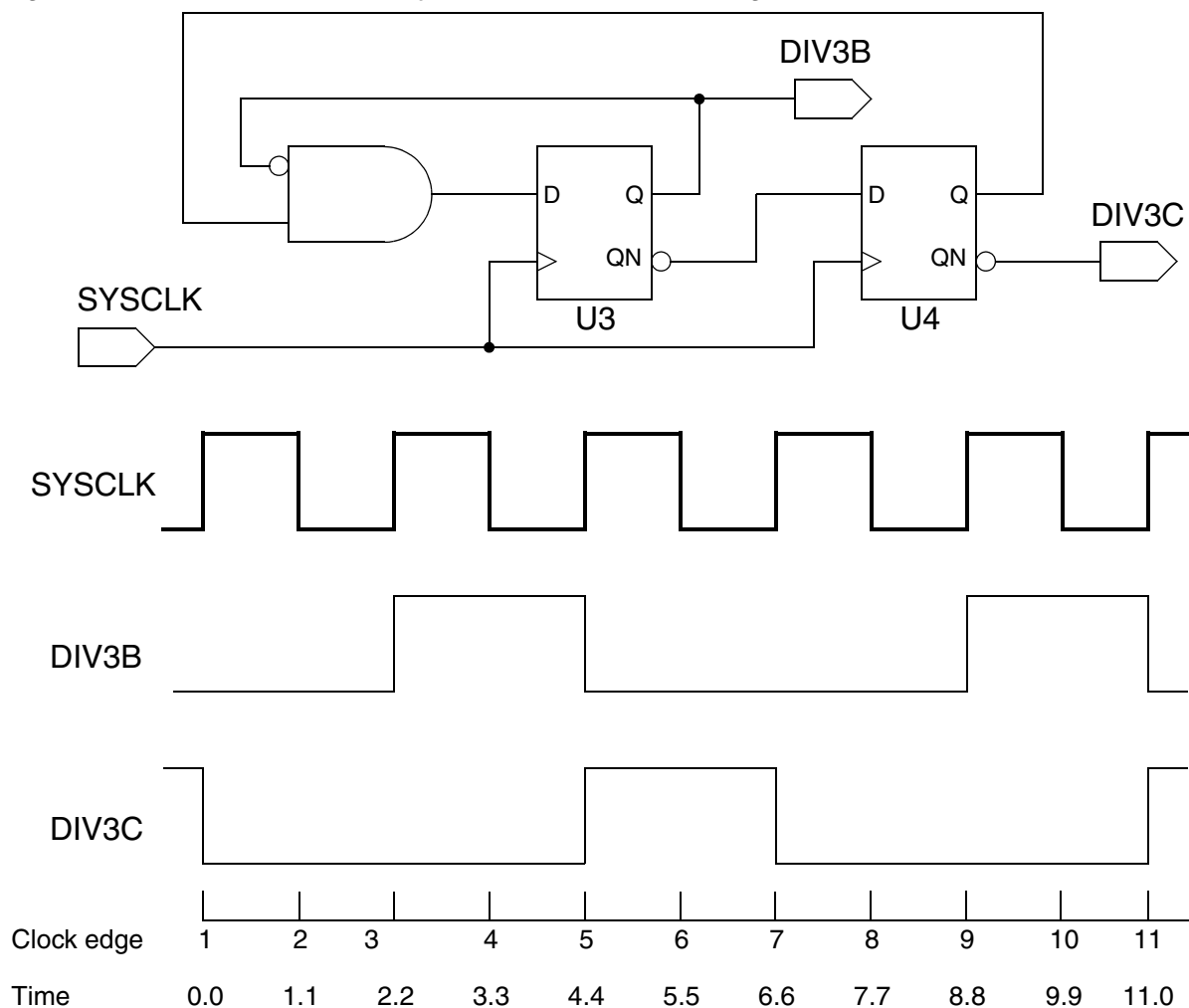
Another way to specify DIV2B is to use a different source pin.

```
icc_shell> create_generated_clock -name DIV2B \  
        -source [get_pins U1/Z] \  
        -edges { 1 3 5 } [get_pins FF2/Q]
```

Shifting the Edges of a Generated Clock

You can shift the edges of a generated clock by a specified amount of time. This shift is not considered clock latency. For example, consider the clock generator circuit shown in [Figure B-6](#).

Figure B-6 Generated Divide-by-3 Clock With Shifted Edges



To specify the master source clock and the two generated clocks, you can use commands such as the following:

```
icc_shell> create_clock -period 2.2 -name CLK \
           [get_ports SYSCLK]

icc_shell> create_generated_clock -edges { 3 5 9 } \
           -name DIV3B -source [get_ports SYSCLK] \
           [get_pins U3/Q]

icc_shell> create_generated_clock -edges { 3 5 9 } \
           -edge_shift { 2.2 2.2 2.2 } \
           -name DIV3C -source [get_ports SYSCLK] \
           [get_pins U4/QN]
```

The `report_clock` command reports the specified clocks as follows:

p - propagated_clock
G - Generated clock

Clock	Period	Waveform	Attrs	Sources

-				
CLK	2.20	{0 1.1}		{SYSCLK}
DIV3B	6.60	{2.2 4.4}	G	{U3/Q}
DIV3C	6.60	{4.4 6.6}	G	{U4/Q}

Generated Clock	Master Source	Generated Source	Waveform Modification

-			
DIV3B	MYCLK	U3/Q	edges(3 5 9)
DIV3C	MYCLK	U4/Q	edges(3 5 9)
			shifts(2.2 2.2 2.2)

Note:

The `-edge_shift` option must be used in conjunction with the `-edges` option.

Restricting Clock Source Latency Calculation To Combinational Paths

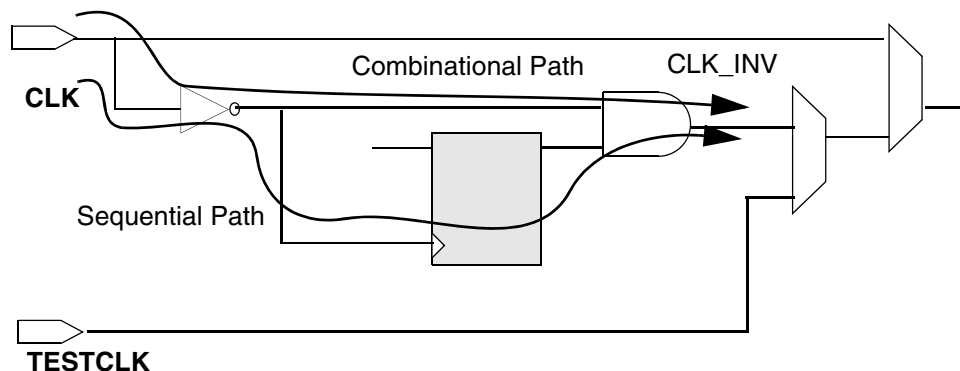
During timing analysis, when IC Compiler calculates clock source latency of a generated clock, it traces both combinational and sequential paths between the source pin of a generated clock and the source pin of its master clock. A sequential path is defined as any path that contains a pin that has one of the following characteristics:

- It is the clock pin of a sequential element
- It is the data pin of a transparent latch
- It is a generated clock definition point (except if it is the final pin in the path)

In some situations, you might want IC Compiler to avoid these sequential paths when it is calculating clock latency. For example, in [Figure B-7](#), the generated clock CLK_INV is specified by using the following command:

```
icc_shell> create_generated_clock -name CLK_INV \
        -source [get_ports CLK] -divide_by 1 \
        get_pins A/Y
```

Figure B-7 Paths in a Generated Clock Source Network



To have IC Compiler avoid sequential paths, use the `-combinational` option with the `create_generated_clock` command. For example,

```
icc_shell> create_generated_clock -name CLK_INV \
      -source [get_ports CLK] -divide_by 1 \
      [get_pins A/Y] -combinational
```

When you specify this option, the generated clock has the same period as the master clock. You can use the `-combinational` option only for those generated clocks that are frequency-divided (`-divide_by 1` option) or frequency-multiplied (`-multiply_by 1`) generated clocks of the master clock. You cannot use this option for edge-derived clocks (`-edges` option).

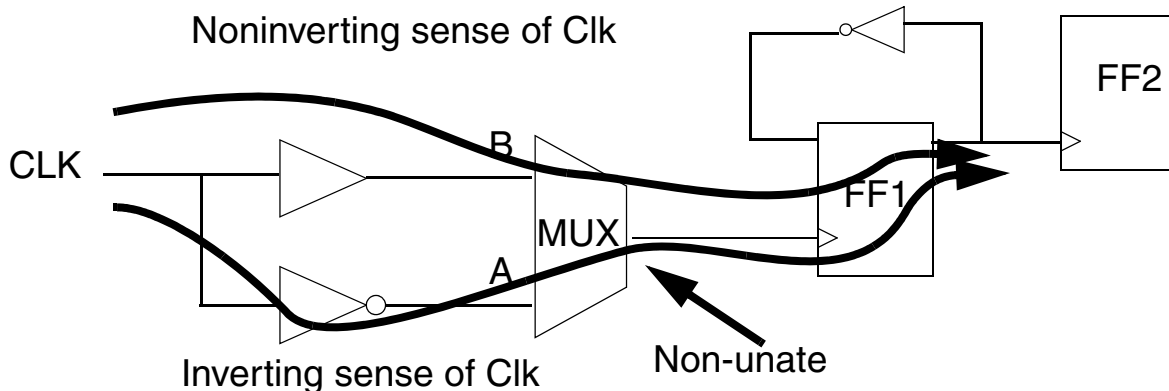
The latency calculation for other generated clocks remains unchanged.

Creating Generated Clocks Based on a Non-Unate Master Clock

A clock signal is not unate if the clock sense is ambiguous as a result of non-unate timing arcs in the clock path. For example, a clock that passes through an XOR gate is not unate, because there are non-unate arcs in the gate.

IC Compiler considers the output of the multiplexer in [Figure B-8](#) to be non-unate, because there are both inverting and noninverting paths through the logic. The noninverting path is the direct path through the multiplexer, and the inverting path is through the inverter. By default, IC Compiler propagates both positive unate clock paths and negative unate clock paths through the non-unate clock network. For more information on propagating non-unate paths, see [“Propagating Clock Paths Through a Non-Unate Clock Network”](#) on page B-36.

Figure B-8 Non-Unate Clock Paths



The clock network starting from the convergent point sees both the positive and negative sense of the original clock. IC Compiler keeps track of clock properties of both senses separately. For example, when the positive sense clock has a waveform of {0 5}, the negative sense clock has a waveform of {5 10}.

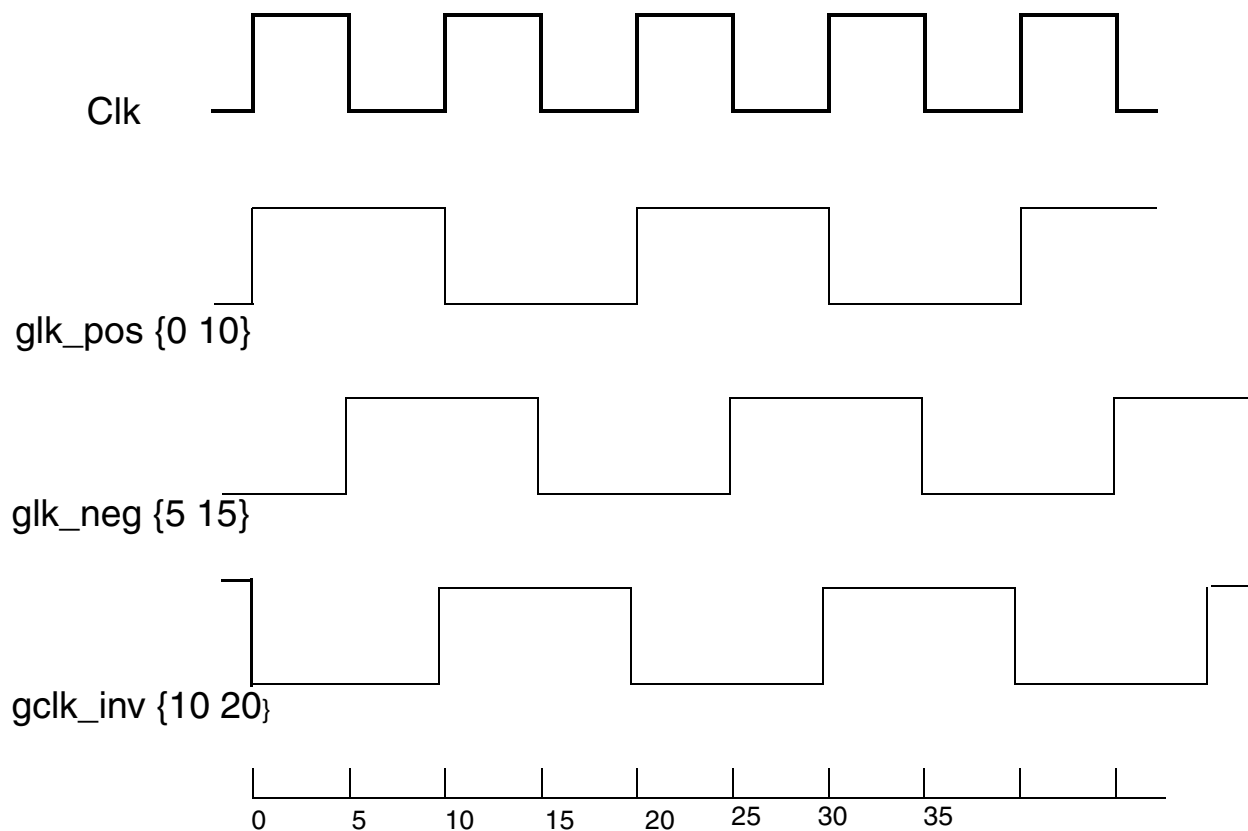
In general path analysis, the particular path is constrained by the most constraining sense of the non-unate clock. When you need to generate a clock, such as a divide-by-2 clock, based on the non-unate master clock, new clocks generated by the `create_generated_clock` command are based on the original clock wave form. To generate a clock based on the expanded wave form of divide-by-2 (inverted clock of the divide-by-2 clock), use the `create_generated_clock -invert` command.

To specify the negative sense before waveform expansion (divide-by-2 clock of the negative unate master clock), you can include the `-preinvert` option with the `create_generated_clock` command. Note that the generated source has to be a pin in the fanout of the non-unate clock in order for the correct waveform expansion to occur. Consider Figure B-8. Example B-1 shows how to use the `-preinvert` and `-invert` options when a generated clock is driven by a non-unate master clock. Figure B-9 shows the resultant waveforms.

Example B-1 Specifying Generated Clocks Based on a Non-Unate Master Clock

```
create_generated_clock -divide_by 2 -source \  
    [get_pins FF1/CLK] -name gclk_pos [get_pins FF1/Q]  
  
create_generated_clock -divide_by 2 -source \  
    [get_pins FF1/CLK] -name gclk_neg \  
    [get_pins FF1/Q] -preinvert  
  
create_generated_clock -divide_by 2 -source  
    [get_pins FF1/CLK] -name glk_inv  
    [get_pins FF1/Q] -invert
```

Figure B-9 Resultant Waveforms



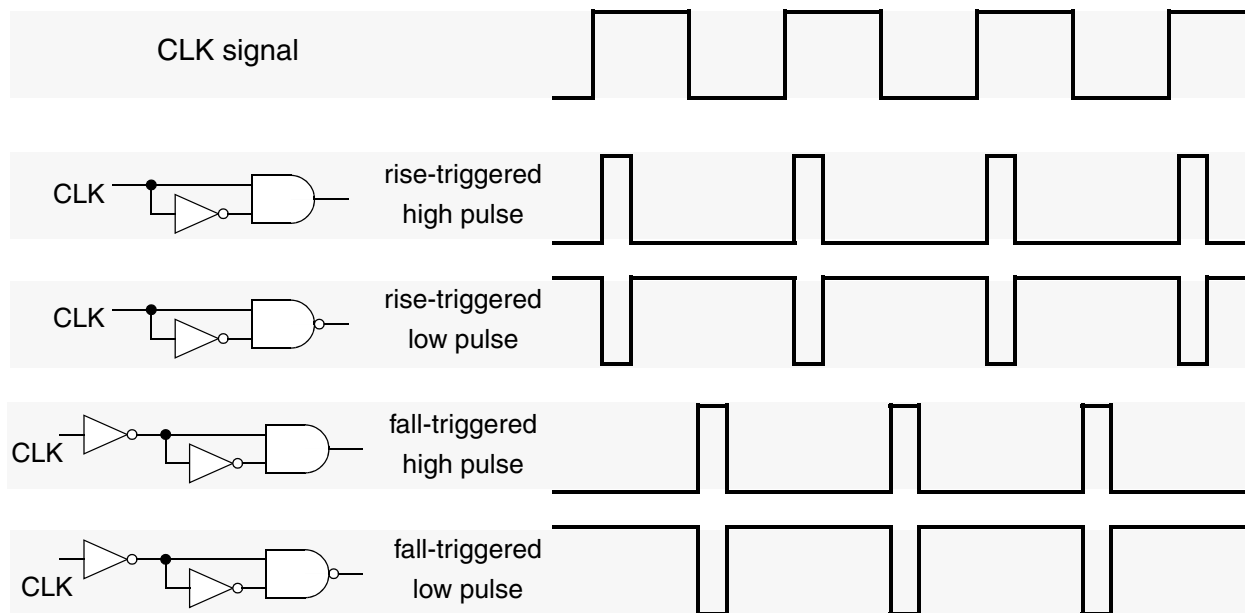
Multiple Clocks at the Source Pin

The `create_generated_clock` command defines a clock that depends on the characteristics of a clock signal reaching a pin in the design, as specified by the `-source` argument. You can define multiple clocks on the same port or pin by using the `-add` option of the `create_generated_clock` command. Because it is possible for the source pin to receive multiple clocks, you might need to specify which clock is used to create the generated clock. To do so, use the `-master_clock` option together with the `-source` option in the `create_generated_clock` command.

Using Pulse Clocks

A pulse clock consists of a sequence of short pulses whose rising and falling edges are both triggered by the same edge of another clock. Pulse clocks are often used to improve performance and reduce power consumption. Figure B-10 shows some examples of pulse clocks.

Figure B-10 Clock Sense Examples



To analyze the timing of a circuit containing pulse clocks, IC Compiler needs information about the timing characteristics of the clock. There are three ways to provide this information:

- Use a pulse generator cell that has been characterized with pulse generator attributes in the .lib description.
- Use the `create_generated_clock` command to describe the pulse timing with respect to the source clock.

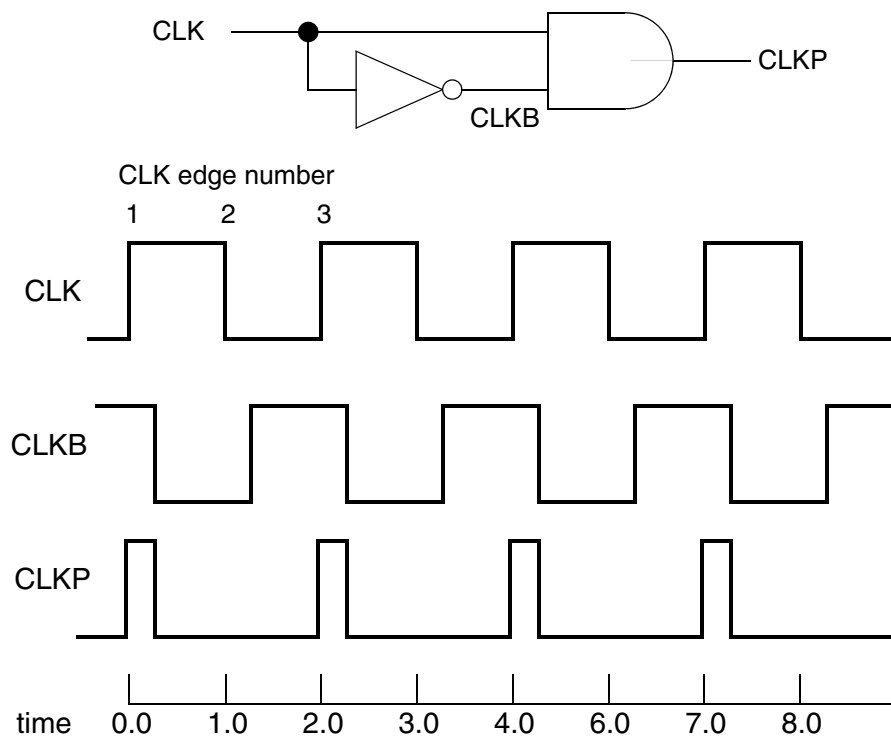
- Use the `set_clock_sense` command to specify the sense of the generated pulses with respect to the source clock.

The best method is to use a pulse generator cell that has been characterized in its .lib library description. In this case, no additional action is necessary in IC Compiler to specify the pulse clock characteristics. For information on specifying the pulse generator characteristics of a library cell, see the Library Compiler documentation.

If characterized pulse generator cells are not available in the library, you must specify the pulse clock characteristics at each pulse generation point in the design, by using either the `create_generated_clock` command or the `set_clock_sense` command. Using the `create_generated_clock` command creates a new clock domain at a pulse generation point. Using the `set_clock_sense` command does not create a new clock domain but merely specifies the sense for an existing clock downstream from the specified point.

For example, consider the pulse clock circuit shown in [Figure B-11](#). Each rising edge of CLK generates a pulse on CLKP.

Figure B-11 Pulse Clock Specified as a Generated Clock



The same edge (edge number 1) of the source triggers both the rising and falling edges of the pulse clock. The pulse width is determined by the delay of the inverter.

To specify the generated pulse clock CLKP as a generated clock, enter the following command:

```
icc_shell> create_generated_clock -name CLKP -source  
CLK -edges {1 1 3} [get_pins and2/z]
```

Specifying the generated clock as a pulse clock by using repeated edge digits (rather than simply specifying the pulse clock edge times) ensures correct checking of delays between the source clock and the pulse clock.

In general, the position of the repeated digit determines whether an active-high or active-low pulse is generated and the edge number that is repeated determines the type of edge in the master clock used to trigger the pulse:

- -edges {1 1 3} — rising edge of source triggers high pulse
- -edges {2 2 4} — falling edge of source triggers high pulse
- -edges {1 3 3} — rising edge of source triggers low pulse
- -edges {2 4 4} — falling edge of source triggers low pulse

Instead of using the `create_generated_clock` command to define a new clock, you can use the `-pulse` option of the `set_clock_sense` command to create a pulse clock:

```
icc_shell> set_clock_sense -pulse \  
rise_triggered_high_pulse and2/z
```

This command tells IC Compiler that the clock at the output of the AND gate is a pulse that rises and falls on the rising edge of the source clock. The pulse clock is not defined as a separate clock domain. Instead, it is just a different sense (rise-triggered high pulse sense) of the source clock downstream from the specified point in the clock network.

In general, the clock sense of a pulse clock can be specified at a location in the design by one of the following forms of syntax:

```
set_clock_sense -pulse rise_triggered_high_pulse object_list  
set_clock_sense -pulse rise_triggered_low_pulse object_list  
set_clock_sense -pulse fall_triggered_high_pulse object_list  
set_clock_sense -pulse fall_triggered_low_pulse object_list
```

The nominal width of the generated pulses is zero, whether you use a pulse generator cell defined in the library, the `create_generated_clock` command, or the `set_clock_sense` command. To determine the actual pulse width, IC Compiler considers the different rise and fall latency values at the pulse generator output pin. For a high pulse:

high pulse width = fall network latency – rise network latency

Similarly, for a low pulse:

low pulse width = rise network latency – fall network latency

You can allow IC Compiler to calculate the propagated latency from the circuit, or you can use the `set_clock_latency` command to specify the latency values (and therefore the pulse width) explicitly. For example, to set an ideal pulse width to 0.5 for high pulses, for all registers downstream from the “and2/z” pin and with an overall latency of 0.6, the commands would be:

```
icc_shell> set_clock_latency -rise 0.6 and2/z
icc_shell> set_clock_latency -fall 1.1 and2/z
```

Reporting Clock Information

The `report_clock` command shows information about clocks and generated clocks in the current design. For generated clock, the report tells you the name of the master clock, the name of the master clock source pin, and the name of the generated clock pin.

The `report_transitive_fanout -clock_tree` command shows the clock networks in your design.

Selecting Clock Objects

The `get_clocks` command selects clocks for a command to use, to ensure, for example, that a command works on the CLK clock and not on the CLK port.

Example 1

To report the attributes of clocks having names starting with PHI1 and a period less than or equal to 5.0, enter

```
icc_shell> report_clock \
           [get_clocks -filter "period <= 5.0" PHI1*]
```

Example 2

To set false paths from all generated clocks having names that start with CLK_DIV to clock CLKB, enter

```
icc_shell> set_false_path \
           -from [get_generated_clocks CLK_DIV*] \
           -to [get_clocks CLKB]
```

Applying Commands to All Clocks

The `all_clocks` command is equivalent to the IC Compiler `get_clocks * command` and returns a collection of clock objects. The actual clock names are not printed.

Example

To disable time borrowing for all clocks, enter

```
icc_shell> set_max_time_borrow 0 [all_clocks]
```

Removing Clock Objects

The `remove_clock` command removes a list of clock objects or clocks from the design. The `remove_generated_clock` command removes generated clocks.

Note:

The `reset_design` command removes clocks as well as other information.

Example 1

To remove all clocks whose names start with CLKB, enter

```
icc_shell> remove_clock [get_clocks CLKB*]
```

Example 2

To remove all clocks in the design, enter

```
icc_shell> remove_clock -all
```

Example 3

To remove the generated clock named CLK_DIV2 (and its corresponding clock object), enter

```
icc_shell> remove_generated_clock CLK_DIV2
```

Specifying Clock Network Effects

After you have performed clock tree synthesis, IC Compiler automatically determines the clock network effects. However, before you run clock tree synthesis, the clocks you create with the `create_clock` command have perfect waveforms that ignore the delay effects of the clock network. You must describe the clock network to perform accurate timing analysis. The main attributes of clock networks in a design are latency and uncertainty.

Latency consists of clock source latency and clock network latency. Clock source latency is the time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point in the design. Clock network latency is the time a clock signal (rise or fall) takes to propagate from the clock definition point in the design to a register clock pin.

Clock uncertainty is the maximum difference between the arrival of clock signals at registers in one clock domain or between domains. This is also called skew. Because clock uncertainty can have an additional margin built in to tighten setup or hold checks, you can specify different uncertainties for setup and hold checks on the same path.

Setting Clock Latency

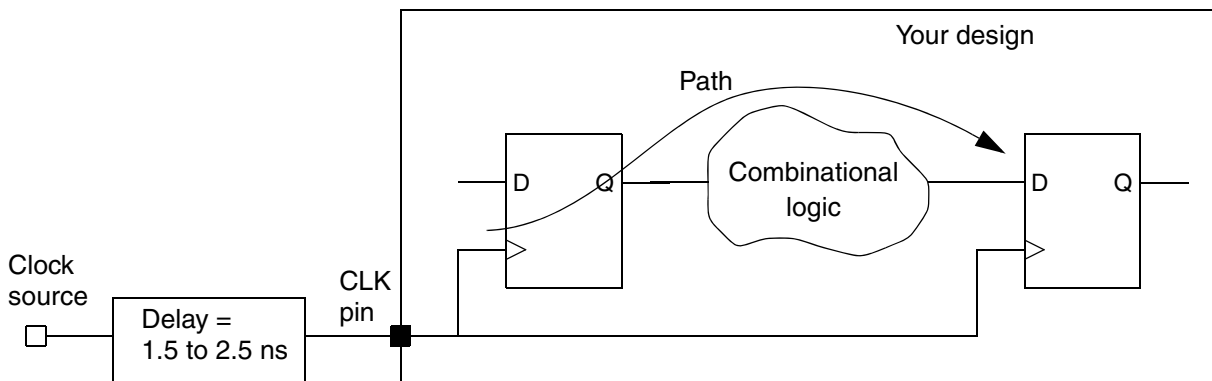
Before running clock tree synthesis, you can estimate the latency of each clock and directly set that estimation with the `set_clock_latency` command.

You can specify clock source latency for ideal or propagated clocks. Source latency is the latency from the ideal waveform to the source pin or port. The latency at a register clock pin is the sum of clock source latency and clock network latency.

For internally generated clocks, IC Compiler can automatically compute the clock source latency, if the master clock of the generated clock has propagated latency and there is no user-specified value for generated clock source latency.

To specify an external uncertainty for source latency, use the `-early` and `-late` options of the `set_clock_latency` command. For example, consider a source latency that can vary from 1.5 to 2.5 ns, as illustrated in [Figure B-12](#).

Figure B-12 External Source Latency



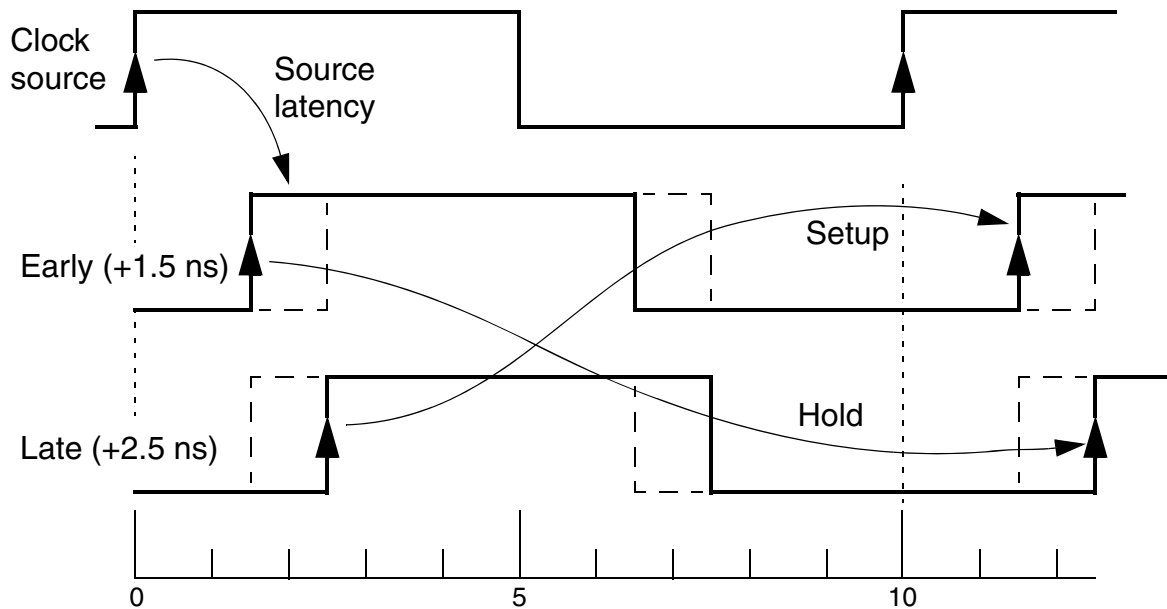
To specify this type of source latency, you can use commands such as the following:

```
icc_shell> create_clock -period 10 [get_ports CLK]
icc_shell> set_clock_latency 1.5 -source -early \
[get_clocks CLK]
icc_shell> set_clock_latency 2.5 -source -late \
[get_clocks CLK]
```

IC Compiler uses the more conservative source latency value (either early or late) for each startpoint and endpoint clocked by that clock. For setup analysis, it uses the late value for each startpoint and the early value for each endpoint. For hold analysis, it uses the early value for each startpoint and the late value for each endpoint.

Figure B-13 shows the early and late timing waveforms and the clock edges used for setup and hold analysis if the startpoint and endpoint are clocked by the same clock.

Figure B-13 Early/Late Source Latency Waveforms



The following examples demonstrate how to set different source latency values for rising and falling edges.

Example 1

To set the expected rise latency to 1.2 and the fall latency to 0.9 for CLK, enter

```
icc_shell> set_clock_latency -rise 1.2 [get_clocks CLK]
icc_shell> set_clock_latency -fall 0.9 [get_clocks CLK]
```

Example 2

To specify an early rise and fall source latency of 0.8 and a late rise and fall source latency of 0.9 for CLK1, enter

```
icc_shell> set_clock_latency 0.8 -source \
    -early [get_clocks CLK1]
icc_shell> set_clock_latency 0.9 -source \
    -late [get_clocks CLK1]
```

The `remove_clock_latency` command removes user-specified clock network or source clock latency information from specified objects.

Setting Clock Uncertainty

You can model the expected uncertainty (skew) for a prelayout design with setup or hold and rise or fall uncertainty values. IC Compiler subtracts a setup uncertainty value from the data required time when it checks setup time (maximum paths), and it adds a hold uncertainty value to the data required time when it checks the hold time (minimum paths). If you specify a single uncertainty value, IC Compiler uses it for both setup checks and hold checks.

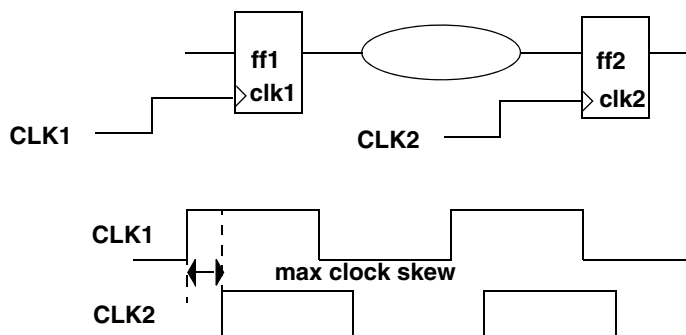
You can specify simple clock uncertainty or interclock uncertainty. Simple clock uncertainty means that all paths to a specific endpoint will have the same setup uncertainty and the same hold uncertainty (regardless of their source clock). Interclock uncertainty is more specific and flexible, supporting different uncertainties between clock domains.

To specify the rising and falling clock edges, you can use the `-rise_from`, `-fall_from`, `-rise_to`, and `-fall_to` options. For example, you can specify uncertainty from the rising edge of CLK1 to the rising edge of CLK2 and a different uncertainty from the falling edge of CLK1 to the falling edge of CLK2. See [“Example 6” on page B-25](#).

The `set_clock_uncertainty` command captures the actual or predicted clock uncertainty.

Interclock uncertainty is relevant when the source and destination registers are clocked by different clocks. You can define uncertainty similarly between two clock pins driven from the same clock, or you can define it as an interclock uncertainty between two registers with different clocks, as shown in [Figure B-14](#).

Figure B-14 Example of Interclock Uncertainty



Example 1

When a path has both simple clock uncertainty and interclock uncertainty, the interclock uncertainty value is used. For example

```
icc_shell> set_clock_uncertainty 5 [get_clocks CLKA]
icc_shell> set_clock_uncertainty 2 \
    -from [get_clocks CLKB] -to [get_clocks CLKA]
```

When the path is from CLKB to CLKA, the interclock uncertainty value 2 is used.

Example 2

The following commands specify interclock uncertainty for all possible interactions of clock domains. If you have paths from CLKA to CLKB, and CLKB to CLKA, you must specify the uncertainty for both directions, even if the value is the same. For example,

```
icc_shell> set_clock_uncertainty 2 \
    -from [get_clocks CLKA] -to [get_clocks CLKB]
icc_shell> set_clock_uncertainty 2 \
    -from [get_clocks CLKB] -to [get_clocks CLKA]
```

When there is no applicable interclock uncertainty for a path, simple uncertainty is used. For example, if the path goes from CLKA to CLKB and you specified the following:

```
icc_shell> set_clock_uncertainty 5 [get_clocks CLKA]
icc_shell> set_clock_uncertainty 2 \
    -from [get_clocks CLKB] -to [get_clocks CLKA]
```

the simple uncertainty of 5 is used.

Example 3

To set simple clock uncertainty (setup and hold) for all paths leading to endpoints clocked by U1/FF*/CP, enter

```
icc_shell> set_clock_uncertainty 0.45 \
    [get_pins U1/FF*/CP]
```

Example 4

To set a simple setup uncertainty of 0.21 and a hold uncertainty of 0.33 for all paths leading to endpoints clocked by CLK1, enter

```
icc_shell> set_clock_uncertainty -setup 0.21 \
    [get_clocks CLK1]
icc_shell> set_clock_uncertainty -hold 0.33 \
    [get_clocks CLK1]
```


Example 5

To specify an interclock uncertainty of 0.54 for paths between the CLK1 and CLK2 registers, enter

```
icc_shell> set_clock_uncertainty 0.54 \  
           -from [get_clocks CLK1] -to [get_clocks CLK2]
```

Example 6

To specify an interclock uncertainty of 0.66 from the rising edge of CLK1 to the falling edge of CLK2, enter

```
icc_shell> set_clock_uncertainty 0.66 \  
           -rise_from CLK1 -fall_to CLK2
```

Limitation When Specifying Negative Uncertainty

Negative uncertainty values are supported for constraining designs with complex clock relationships. However, setting the uncertainty value to a negative number can lead to optimistic timing analysis and should be used with extreme care.

Removing Clock Uncertainty

The `remove_clock_uncertainty` command removes uncertainty information from clocks, ports, pins, or cells or between specified clocks.

Estimating Clock Pin Transition Time

Transition times are typically computed for the ports, cells, and nets in the clock network. If the clock network is not complete, the result can be inaccurate transition times on register clock pins. For example, a single buffer might be driving 10,000 register clock pins, because the clock tree has not been constructed. This can cause a long transition time on the register clock pins affecting clock-to-output delays, as well as setup and hold delay calculation.

You can estimate the clock transition time for an entire clock network by using the `set_clock_transition` command.

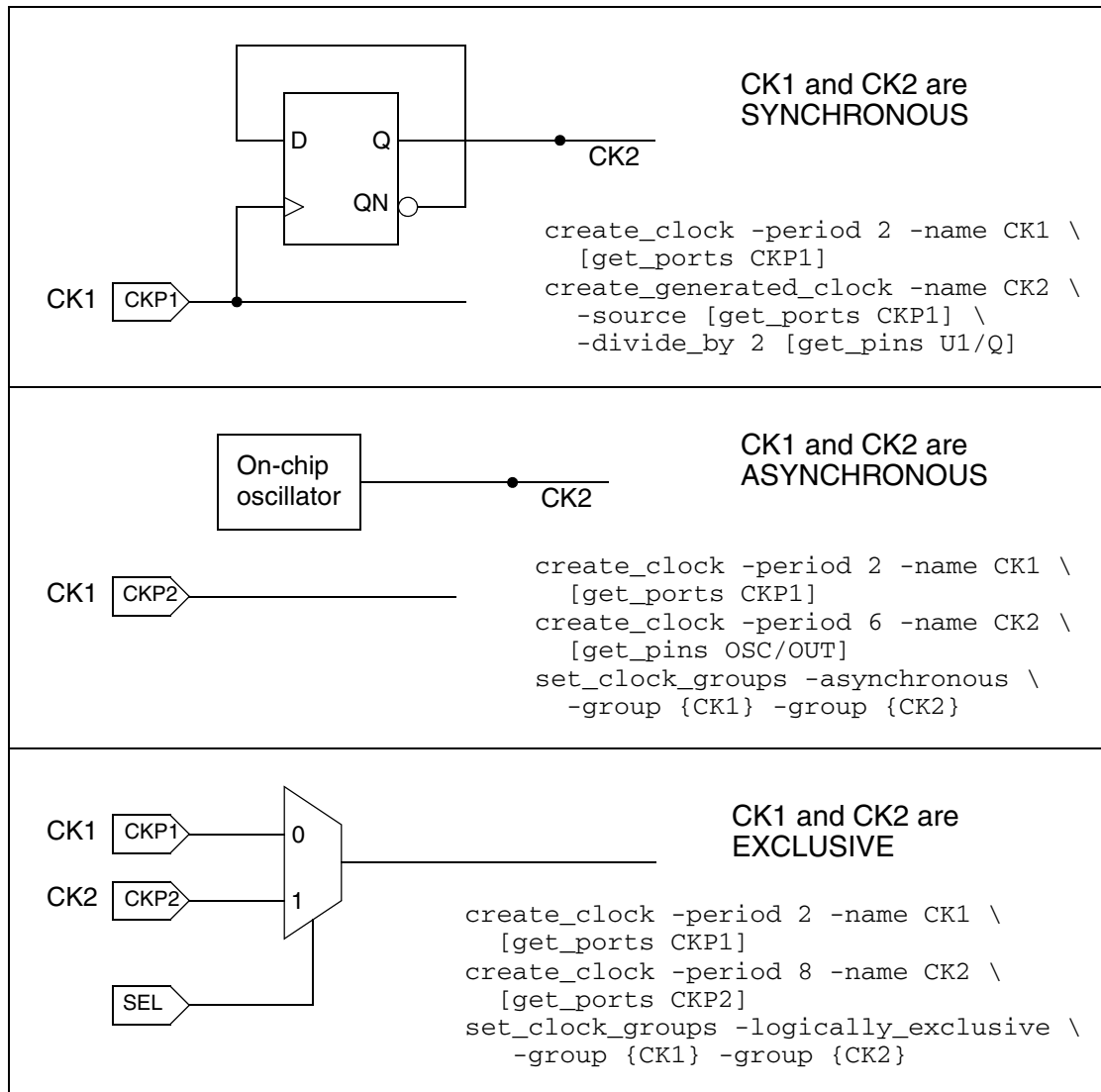
Removing Estimated Clock Pin Transition Time

The `remove_clock_transition` command removes the estimated clock pin transition time.

Using Multiple Clocks

When multiple clocks are defined for a design, the relationships between the clock domains depend on how the clocks are generated and how they are used in the design. The relationship between two clocks can be synchronous, asynchronous, or exclusive, as shown by the examples in [Figure B-15](#).

Figure B-15 Synchronous, Asynchronous, and Exclusive Clocks



For IC Compiler to analyze paths between different clock domains correctly, you might need to specify false paths between clocks, exclude one or more clocks from consideration during the analysis, or specify the nature of the relationships between different clocks.

Synchronous Clocks

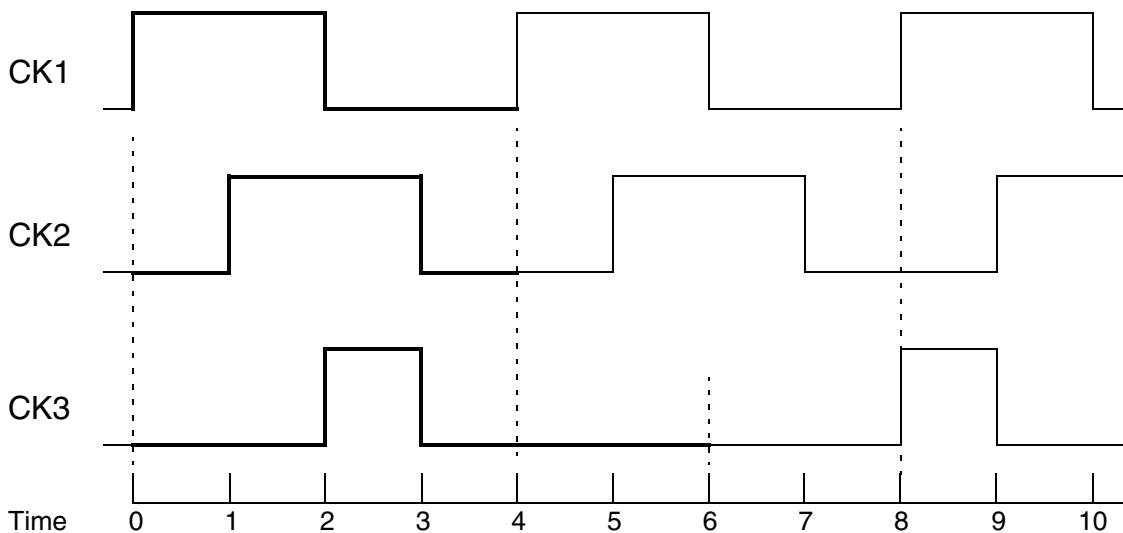
Two clocks are synchronous with respect to each other if they share a common source and have a fixed phase relationship. Unless you specify otherwise, IC Compiler assumes that two clocks are synchronous if there is any path with data launched by one clock and captured by the other clock. The clock waveforms are synchronized at time zero, as defined by the `create_clock` command.

For example, consider the following `create_clock` commands:

```
icc_shell> create_clock -period 4 -name CK1 \  
            -waveform {0 2}  
icc_shell> create_clock -period 4 -name CK2 \  
            -waveform {1 3}  
icc_shell> create_clock -period 6 -name CK3 \  
            -waveform {2 3}
```

IC Compiler creates the clocks as specified in the commands, with the waveforms synchronized as shown in [Figure B-16](#). IC Compiler adjusts the timing relationships further for any specified or calculated latency or uncertainty.

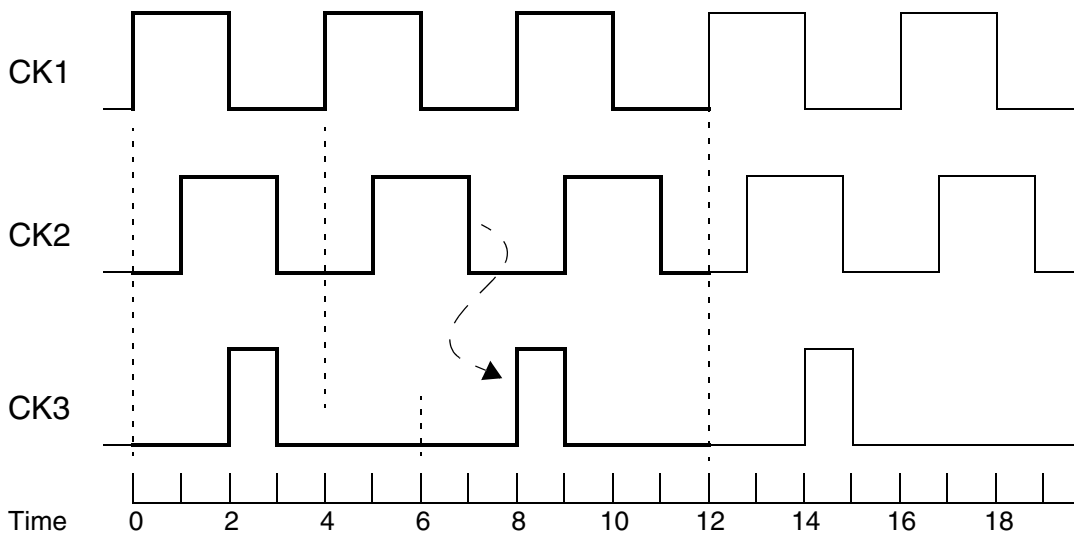
Figure B-16 Synchronous Clock Waveforms



In a design that uses these three clocks, there might be paths launched by one clock and captured by another clock. When such paths exist, IC Compiler internally “expands” the clocks to the least common multiple of all synchronous clock periods to test all possible timing relationships between different clock edges, thus creating longer-period clocks with multiple rising and falling edges.

For example, the three clocks in the foregoing example have periods of 4, 4, and 6. The least common multiple of these periods, called the base period, is 12. To analyze the paths that cross the clock domains, IC Compiler internally expands the clocks by repeating them over the base period. The resulting clock waveforms are shown in [Figure B-17](#). Each expanded clock has a period of 12.

Figure B-17 Expanded Clock Waveforms



IC Compiler checks timing paths between all edges in the expanded clocks. For example, the most restrictive setup check between a falling edge of CK2 and a rising edge of CK3 is from time = 7 to time = 8, as shown by the dashed arrow in [Figure B-17](#).

You should define multiple clocks in a manner consistent with the way they actually operate in the design. To declare clocks that are not synchronous, you can use case analysis or commands such as `set_clock_groups` or `set_false_path`.

Asynchronous Clocks

Two clocks are asynchronous if they do not communicate with each other in the design. For example, a free-running, on-chip oscillator is asynchronous with respect to a system clock signal coming into the chip from the outside. Clock edges in the two clock domains can occur at any time with respect to each other.

To declare an asynchronous relationship between two clocks, use the `set_clock_groups -asynchronous` command.

If two clocks are asynchronous, IC Compiler does not check the timing paths launched by one clock and captured by the other clock, which is like declaring a false path between the two clocks. In addition, if you are doing crosstalk analysis, IC Compiler assigns infinite arrival windows to the nets in aggressor-victim relationships between the two clock domains.

With the `-asynchronous` option, `-allow_paths` can be used to restore analysis of the timing paths between clock groups while still using infinite alignment windows for crosstalk analysis.

For example, the following command defines clocks CLK1 and CLK2 to be asynchronous for purposes of crosstalk analysis (infinite arrival windows), without affecting normal logical timing checks between the two clocks:

```
icc_shell> set_clock_groups -asynchronous -allow_paths \  
           -group {CLK1} -group {CLK2}
```

You can restrict checking of some or all clock-to-clock paths by using the `set_false_path` command in conjunction with the `set_clock_groups` command.

Exclusive Clocks

IC Compiler supports two types of exclusive clocks:

- Logically exclusive clocks

Two clocks that are logically exclusive have no logical timing paths between them. For example, a circuit might multiplex two different clock signals onto a clock line, one a fast clock for normal operation and the other a slow clock for low-power operation. Only one of the two clocks is enabled at any given time, so there is no interaction between the two clocks.

IC Compiler does not check the logical timing between the clocks but still checks for possible crosstalk interaction between them.

- Physically exclusive clocks

Two clocks that are physically exclusive of each other have no logical timing paths between them and, furthermore, are considered physically isolated from each other.

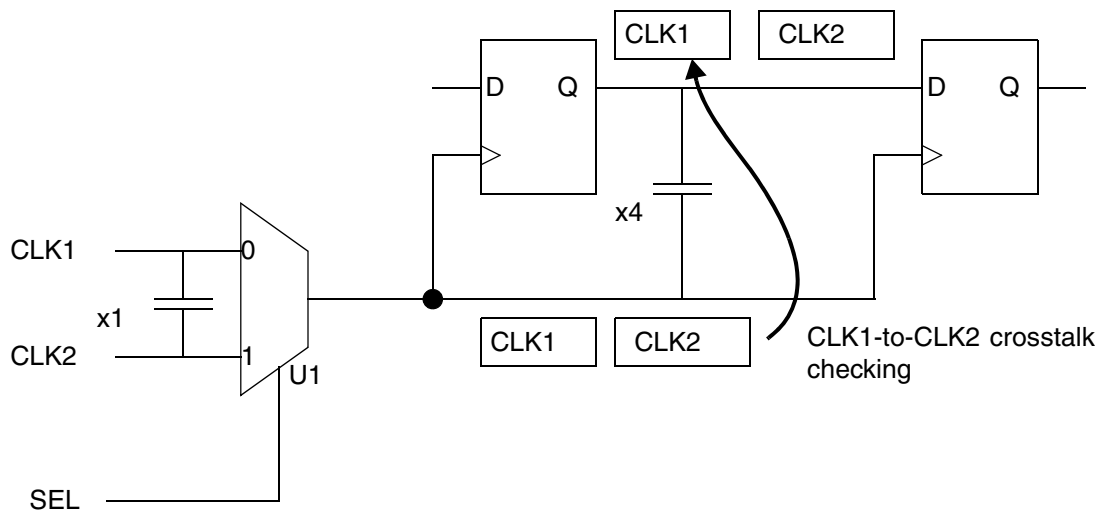
IC Compiler does not check the logical timing between the clocks and assumes no possible crosstalk interaction between them.

Logically Exclusive Clocks

To prevent IC Compiler from spending time checking the interaction between logically exclusive clocks, you can declare a false path between the clocks, use the `set_clock_groups -logically_exclusive` command to declare the clocks to be logically exclusive, or you can use case analysis to disable the clock that you do not want to include in the current analysis.

For example, consider the circuit shown in Figure B-18. Only one of the two input clocks is enabled at any given time. However, by default, IC Compiler considers the crosstalk across capacitor x4, between the overlapping arrival windows of CLK1 and CLK2.

Figure B-18 Circuit With Multiplexed Clocks



The most accurate way to handle this situation is to use case analysis, setting the MUX control signal first to 0 and then to 1. This method ensures that there is no interaction between the clocks and correctly handles all crosstalk situations. However, it requires two analysis runs. For a design with many clocks, it might not be practical to analyze every possible combination of enabled clocks.

To analyze both conditions at the same time, you can define the clocks to be logically exclusive:

```
icc_shell> set_clock_groups -logically_exclusive \  
           -group {CLK1} -group {CLK2}
```

The `-logically_exclusive` option causes IC Compiler to suppress any logical (timing path) checking between CLK1 and CLK2, similar to setting a false path constraint between the clocks. However, IC Compiler still computes crosstalk delta delays across coupling capacitor x4 between the two clocks, which is pessimistic if the two clocks are not simultaneously present on the nets.

You can specify multiple clocks in each group. For example, to declare clocks CK1 and CK2 to be exclusive with respect to CK3 and CK4, enter

```
icc_shell> set_clock_groups -logically_exclusive \  
           -group {CK1 CK2} -group {CK3 CK4}
```

This causes IC Compiler to ignore any path that starts in one group and ends in the other group.

If you specify more than two groups, each group is exclusive with respect to the other specified groups. For example,

```
icc_shell> set_clock_groups -logically_exclusive \  
           -group {CK1 CK2} -group {CK3 CK4} -group {CK5}
```

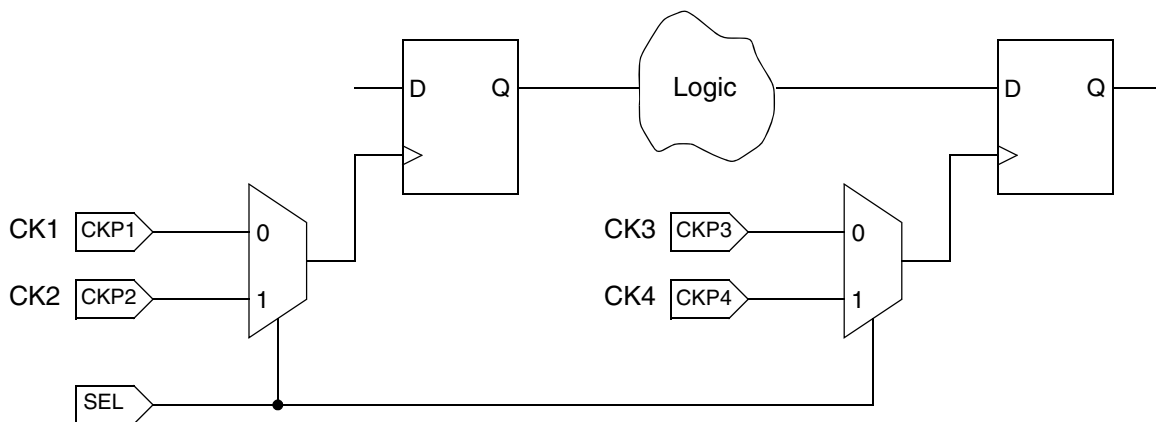
If you specify just one group, that group is exclusive with respect to all other clocks in the design. For example,

```
icc_shell> set_clock_groups -logically_exclusive \  
           -group {CK1 CK2}
```

The following example demonstrates some of the ways to specify logically exclusive clocks.

Example: Four Clocks and One Selection Signal. Consider the circuit shown in [Figure B-19](#). There are four clocks, CK1 through CK4. By default, IC Compiler analyzes the interactions between all combinations of clocks. However, the logic enables only two clocks at a time, either CK1 and CK3 or CK2 and CK4.

Figure B-19 Four Clocks and One Selection Signal



One way to prevent checking between unrelated clocks is to set a false path between the clocks. For example:

```
icc_shell> set_false_path -from CK1 -to CK2
icc_shell> set_false_path -from CK2 -to CK1
icc_shell> set_false_path -from CK3 -to CK4
icc_shell> set_false_path -from CK4 -to CK3
icc_shell> set_false_path -from CK1 -to CK4
icc_shell> set_false_path -from CK4 -to CK1
icc_shell> set_false_path -from CK2 -to CK3
icc_shell> set_false_path -from CK3 -to CK2
```

In that case, IC Compiler tests all of the valid combinations of enabled clocks in a single run, while ignoring the invalid combinations.

Another way is to use case analysis and set a logic value, either 0 or 1, on the SEL input, which checks the timing for a particular case of SEL=0 or SEL=1. For example,

```
icc_shell> set_case_analysis 0 [get_ports SEL]
```

With SEL=0, only CK1 and CK3 are active; CK2 and CK4 are ignored. If you want to analyze both cases, two analysis runs are necessary: one with SEL=0 and another with SEL=1.

Another method to accomplish the same effect is to use the `set_disable_timing` command. For example, to disable checking of all paths leading from the CKP2 and CKP4 clock input pins of the design, enter the following command:

```
icc_shell> set_disable_timing [get_ports {CKP2 CKP4}]
```


Still another way is to specify which clocks can be active together at the same time and which clocks are currently active. For example,

```
icc_shell> set_clock_groups -logically_exclusive -name E1 \  
          -group {CK1 CK3} -group {CK2 CK4}
```

The `set_clock_groups` command defines groups of clocks that are exclusive with respect to each other. IC Compiler will not check paths that start from a clock in one group and end at a clock in another group. If you specify just one group, that group is considered exclusive with respect to all other clocks in the design.

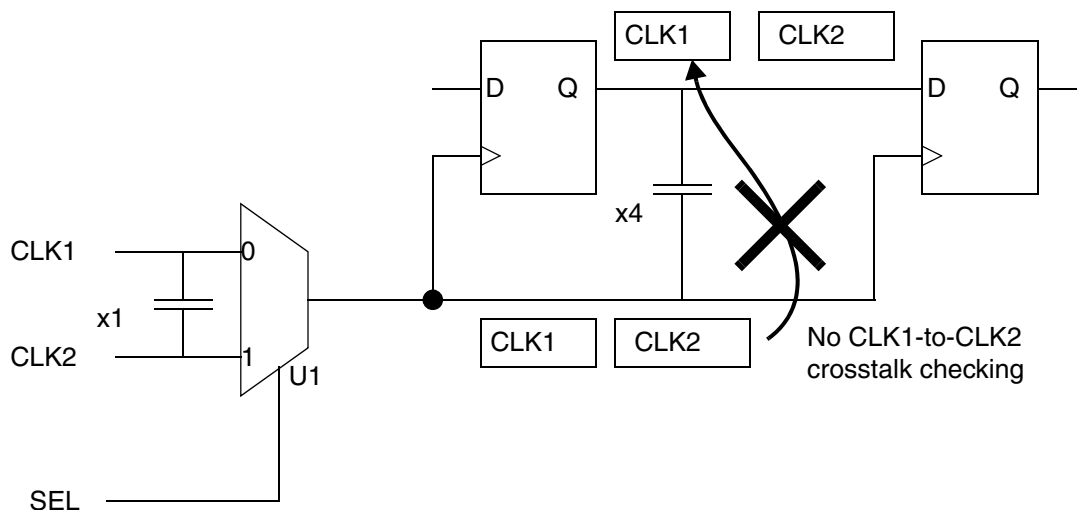
Physically Exclusive Clocks

To prevent IC Compiler from performing crosstalk analysis between two clocks, you can use the `set_clock_groups -physically_exclusive` command to declare the clocks to be physically exclusive.

For example, if you define the clocks shown in [Figure B-18](#) to be physically exclusive, the result is as shown in [Figure B-20](#).

```
icc_shell> set_clock_groups -physically_exclusive \  
          -group {CLK1} -group {CLK2}
```

Figure B-20 Circuit with Multiplexed Clocks



To correctly handle both cross-coupling capacitors, instead of declaring the original clocks to be exclusive, you can define two generated clocks at the output of the MUX and define them to be physically exclusive.

In such a case, IC Compiler computes delta delays between CLK1 and CLK2 across x1 before the MUX, but not between gCLK1 and gCLK2 across x4 or elsewhere in the generated clock tree, as shown in [Figure B-21](#).

The definition of physically exclusive clock groups removes pessimism by eliminating crosstalk analysis where no crosstalk should exist. The pessimism removal applies to both crosstalk timing analysis and crosstalk noise analysis. It is your responsibility to correctly define the clock groups. IC Compiler does not verify that a particular setting makes sense for the design.

IC Compiler detects only group-to-group conflicts, not implied conflicts. For example, if you declare CLK1 and CLK2 to be asynchronous with each other, they are both still synchronous with CLK3 by default. This is an implied three-way conflict that IC Compiler does not detect. You are responsible for resolving such conflicts by correct use of the `set_clock_groups` command.

Working With Clock Groups

This section describes the following aspects of clock groups:

- Comparing clock groups and false paths

A clock group definition has higher priority than a `set_false_path` timing exception. Furthermore, the `reset_path` command does not cancel clock group relationships set with the `set_clock_groups` command.

Avoid setting false paths between clock domains that have been declared to be asynchronous or exclusive, because doing so is redundant. The `set_clock_groups` command invalidates all false paths set between the asynchronous or exclusive clock domains.

- Assigning a name to a clock group

You can optionally assign a name to a clock group, which makes it easier to later remove that clock group.

```
icc_shell> set_clock_groups -logically_exclusive \
               -name EX1 -group {CK1 CK2} -group {CK3 CK4}
```

- Reporting clock groups

The clock group definitions are not reported by the `report_exceptions` command. To find out about clock groups that have been set, use the `report_clock -groups` command.

```
icc_shell> report_clock -groups
```

- Removing clock groups

Use the `remove_clock_groups` command to remove a clock group.

```
icc_shell> remove_clock_groups -logically_exclusive \
          -name EX1
```

Use the `-all` option to remove all clock groups made with the `set_clock_groups` command.

```
icc_shell> remove_clock_groups -logically_exclusive -all
```

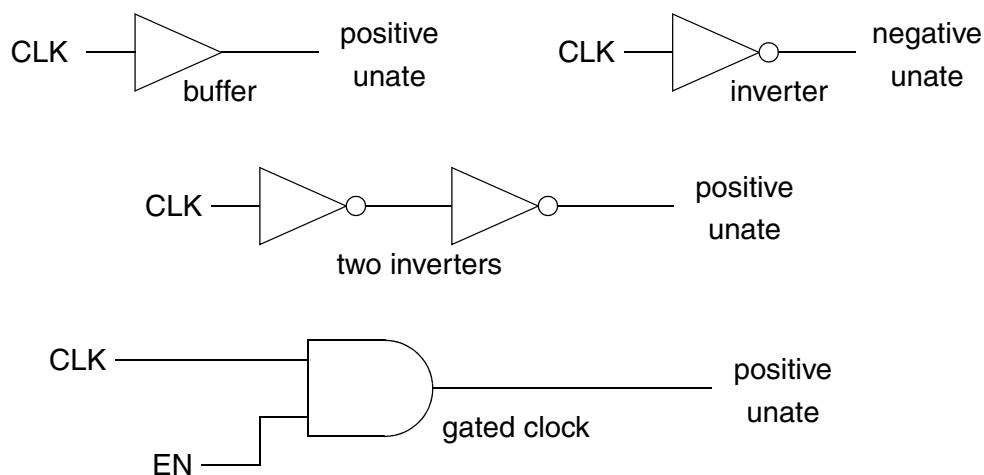
Propagating Clock Paths Through a Non-Unate Clock Network

IC Compiler keeps track of inverters and buffers in clock trees. It recognizes the positive or negative sense of the clock signal arriving at each register clock pin. No specific action is necessary to tell IC Compiler the sense of a clock tree that has only buffers and inverters. In this case, the clock signal arriving at the register clock pin is said to be “unate.”

A clock signal is “positive unate” if a rising edge at the clock source can only cause a rising edge at the register clock pin and a falling edge at the clock source can only cause a falling edge at the register clock pin.

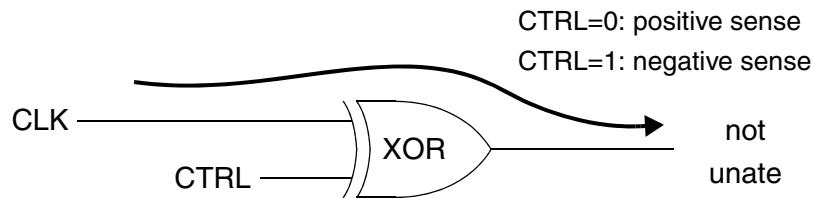
Similarly, a clock signal is “negative unate” if a rising edge at the clock source can only cause a falling edge at the register clock pin and a falling edge at the clock source can only cause a rising edge at the register clock pin. In other words, the clock signal is inverted. See [Figure B-22](#).

Figure B-22 Unate Clock Signals



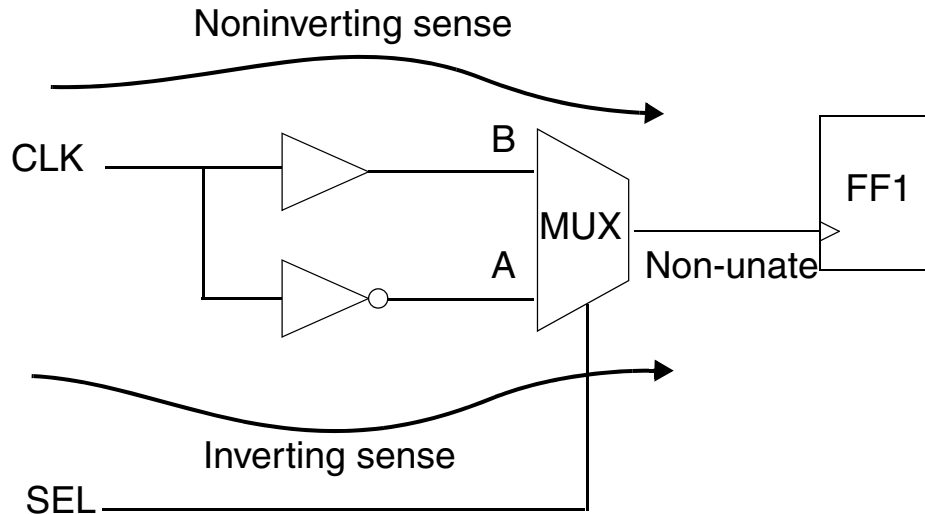
A clock signal is not unate if the clock sense is ambiguous, as a result of non-unate timing arcs in the clock path. For example, a clock that passes through an XOR gate is not unate, because there are non-unate arcs in the gate. The clock sense might be either positive or negative, depending on the state of the other input to the XOR gate, as shown in Figure B-23.

Figure B-23 Non-Unate Clock Signals



IC Compiler considers the output of the multiplexer in Figure B-24 to be non-unate, because there are both inverting and noninverting paths through the logic. The noninverting path is the direct path through the multiplexer, and the inverting path is through the inverter. By default, IC Compiler propagates both positive unate clock paths and negative unate clock paths through the non-unate clock network, as shown in Figure B-24.

Figure B-24 Non-Unate Clock Paths

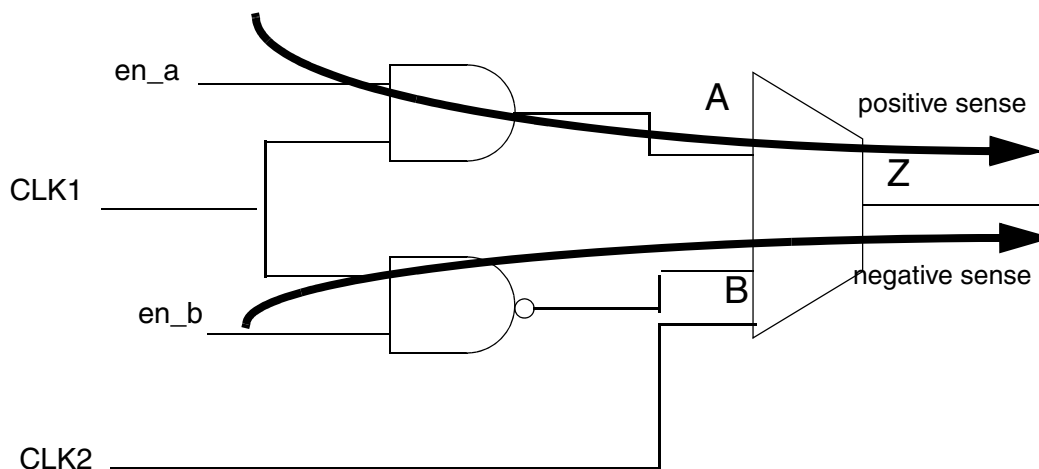


The `create_generated_clock` command supports the `-invert` and `-preinvert` options, which enable you to specify generated clocks based on a non-unate master clock. For example, to generate a clock based on the expanded waveform of a divide_by 2 (inverted clock of the divide_by 2 clock), use the `-invert` option. To specify the negative sense before waveform expansion (divide_by 2 clock of the negative unate master clock), use the `-preinvert` option. For more information, see “Creating Generated Clocks Based on a Non-Unate Master Clock” on page B-12.

Specifying Clock Sense

The `set_clock_sense` command allows you to restrict the unateness at a pin to positive unate, negative unate, or pulse type with respect to the clock source. The command allows you to specify the clock sense when multiple senses of the same clock have merged and it is ambiguous which sense the tool should use, as shown in [Figure B-25](#).

Figure B-25 Non-Unate Clock Signals



You can specify that IC Compiler propagate only the positive unate paths through the output pin of the multiplexer with respect to the original clock source, by entering the following command:

```
icc_shell> set_clock_sense -positive [get_pins MUX/Z]
```

From pin MUX/Z onward, IC Compiler keeps track of the sense of the signal through any subsequent buffers or inverters.

The positive unate setting applies to any clock that passes through the specified pin. If multiple clocks can reach that pin, you can restrict the setting to certain clocks. For example, to restrict the positive unate setting to clock CLK1 in [Figure B-25](#), enter

```
icc_shell> set_clock_sense -positive \  
             -clocks [get_clocks CLK1] [get_pins MUX/Z]
```

To stop clock propagation forward from a specified pin, you can use the `-stop_propagation` option of the `set_clock_sense` command. For example, to stop clock propagation from pin A of the MUX in [Figure B-25](#), enter

```
icc_shell> set_clock_sense -stop_propagation \  
           [get_pins MUX/A]
```

You cannot use the `stop_propagation` option with the `-positive`, `-negative`, or `-pulse` options.

The `set_clock_sense` command has an effect on only the non-unate part of the network. If you apply the command to a pin in the unate part of the network and the requested sense is different from the actual sense, the tool ignores the command and generates an error message. To remove clock sense, use the `remove_clock_sense` command.

Defining Data-to-Data Checks

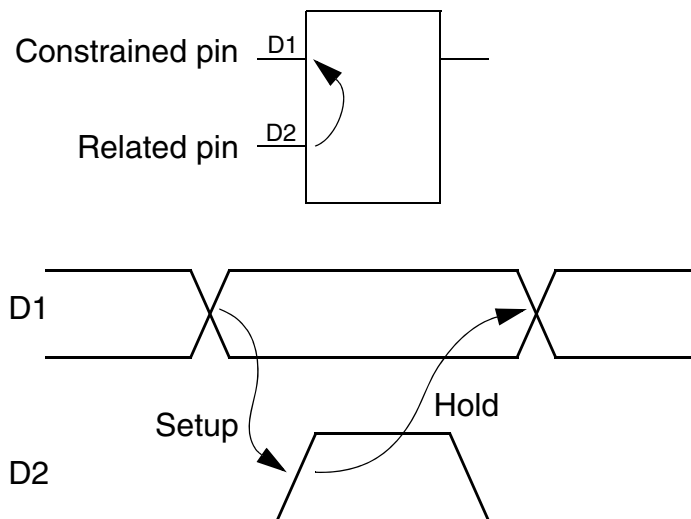
IC Compiler can perform setup and hold checking between two data signals, neither of which is defined to be a clock, at any two pins in the design. A timing constraint between two data (nonclock) signals is called a nonsequential constraint. This feature can be useful for checking the following types of timing constraints:

- Constraints on handshaking interface logic
- Constraints on asynchronous or self-timed circuit interfaces
- Constraints on signals with unusual clock waveforms that cannot be easily specified with the `create_clock` command
- Constraints on skew between bus lines
- Recovery and removal constraints between asynchronous preset and clear input pins

You can define such checks in IC Compiler by using the `set_data_check` command, or you can define them for a library cell by setting nonsequential constraints for the cell in Library Compiler. You should use data checks only in situations such as those described above. Data checks should not be considered a replacement for ordinary sequential checking.

[Figure B-26](#) shows a simple example of a cell that has a nonsequential constraint. The cell has two data inputs, D1 and D2. The rising edge of D2 is the active edge that might be used to latch data at D1. Pin D1 is called the constrained pin, and pin D2 is called the related pin. In a sequential setup or hold check, pin D2 would be considered the clock pin. However, for any of a number of reasons, it might be desirable to consider the signal at D2 a data signal and not define it to be a clock.

Figure B-26 Simple Data Check Example



In this example, the signal at D1 must be stable for a certain setup time before the active edge. It must also be stable for a certain hold time after the active edge. If these nonsequential constraints are not already defined for the library cell, you can define them in IC Compiler.

Specifying Data-to-Data Checks

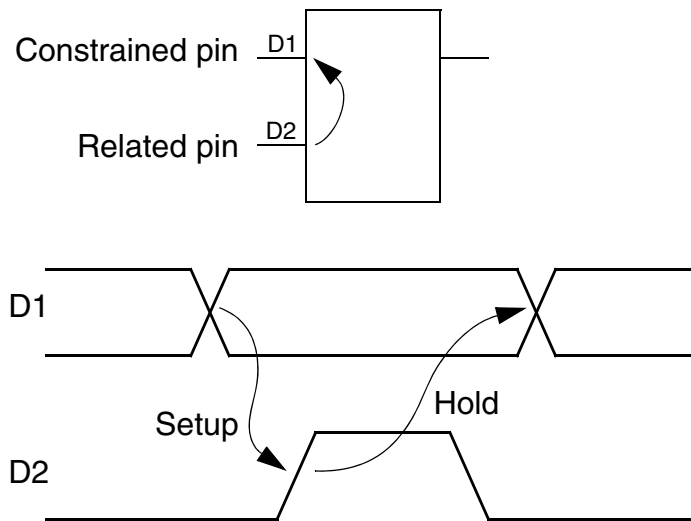
You use the `set_data_check` command to specify that data-to-data checks be performed between two data pins. Use options to the command as follows:

- To specify a pin or port in the current design as the related pin, use the `-from`, `-rise_from`, or `-fall_from` option. The `-rise_from` and `-fall_from` options apply to rising and falling delays.
- To specify a pin or port in the current design as the constrained pin, use the `-to`, `-rise_to`, or `-fall_to` option. The `-rise_to` or `-fall_to` options apply to rising or falling delays.
- To specify that the data check value be for setup only or hold only, use the `-setup` option or `-hold` option. If you do not specify either option, the value applies to both setup and hold.
- To specify the name of a single clock that launches the signal for the related pin, use the `-clock` option.

Example 1

In [Figure B-27](#), the signal at D1 must be stable for a certain setup time before the active edge. It must also be stable for a certain hold time after the active edge.

Figure B-27 Simple Data Check Example



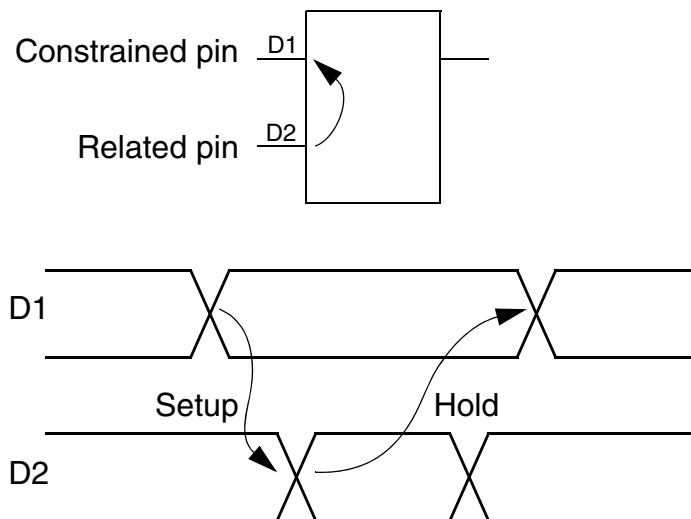
To define these data-to-data checks, use commands similar to the following:

```
icc_shell> set_data_check -rise_from D2 -to D1 -setup 3.5
icc_shell> set_data_check -rise_from D2 -to D1 -hold 6.0
```

Example 2

In [Figure B-28](#), the data checks apply to both rising and falling edges on the related pin. Use the `-from` option instead of the `-rise_from` or `-fall_from` option.

Figure B-28 Data Checks on Rising and Falling Edges



To define these data checks, use the following commands:

```
icc_shell> set_data_check -from D2 -to D1 -setup 3.5
icc_shell> set_data_check -from D2 -to D1 -hold 6.0
```

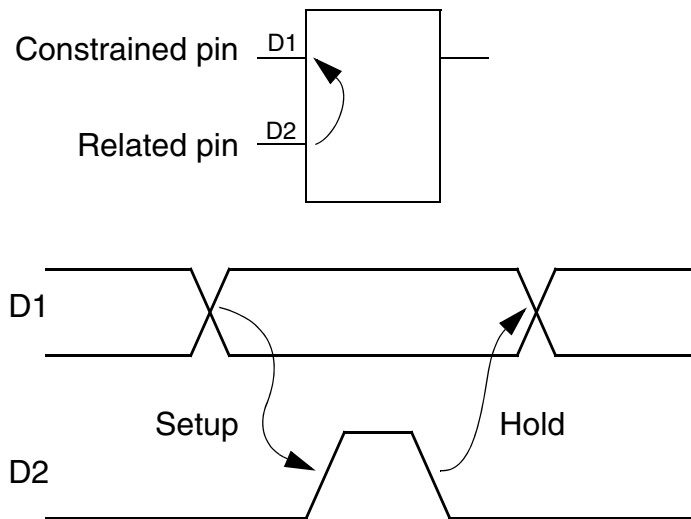
Example 3

You can define a no-change data check by specifying only a setup check from the rising edge and a hold check from the falling edge:

```
icc_shell> set_data_check -rise_from D2 -to D1 -setup 3.5
icc_shell> set_data_check -fall_from D2 -to D1 -hold 3.0
```

IC Compiler interprets this as a no-change check on a positive-going pulse. The resulting timing check is illustrated in [Figure B-29](#).

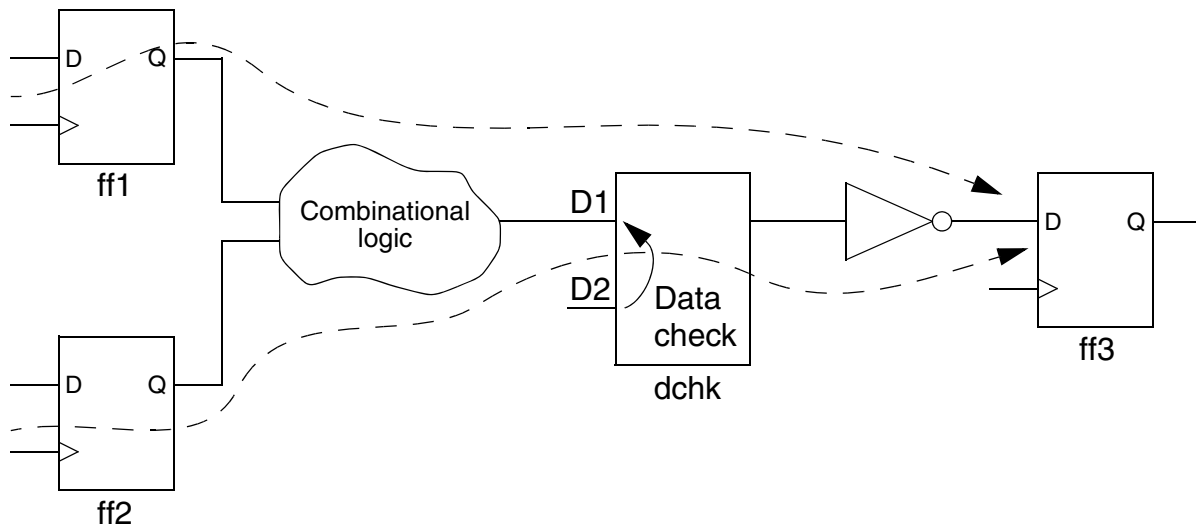
Figure B-29 No-Change Data Check



Generating Timing Reports for Data Checks

Data checks are nonsequential, so they do not break timing paths. For example, in [Figure B-30](#), the data check between D1 and D2 does not interrupt the timing paths shown by the dashed-line arrows. If you were to define the signal at D2 to be a clock, the check would be sequential and the paths would be terminated at D1.

Figure B-30 Timing Paths Not Broken by Data Checks



You can specify a data check pin as a path endpoint for the `report_timing` command. In such a case, IC Compiler reports the data checks that apply to the pin. For example, for the circuit shown in [Figure B-30](#), `report_timing -to dchk/D1` generates a data check report, whereas `report_timing -through dchk/D1` generates a timing report on ordinary paths that pass through the specified pin and paths that end at the data check pin.

Removing Data Checks

To remove data checks set with the `set_data_check` command, use the `remove_data_check` command. For more information, see the man page for the command.

Data Checks and Clock Domains

In a data check, signals arriving at a constrained pin or related pin can come from different clock domains. IC Compiler checks the signal paths separately and puts them into different clock groups, just as in ordinary sequential checks.

If the related pin has signals from multiple clock domains, you might want to specify which clock domain to analyze at that pin for the data check. To do this, either use the `-clock clock_name` option of the `set_data_check` command or disable all clocks other than the clock of interest.

Library-Based Data Checks

You can use the `timing_enable_non_sequential_checks` variable to specify that IC Compiler perform data checking for any cell that has nonsequential timing constraints defined in the library cell (as long as the signal at the related pin is not defined to be a clock). If the signal is defined to be a clock, the tool ignores the library-defined nonsequential timing constraints. The variable is off by default.

Note:

This behavior is different from that in PrimeTime, which honors nonsequential arc checks automatically.

In Library Compiler, you define nonsequential constraints on a cell by specifying a related pin and assigning the following `timing_type` attributes to the constrained pin:

```
non_seq_setup_rising
non_seq_setup_falling
non_seq_hold_rising
non_seq_hold_falling
```

For more information on defining nonsequential constraints in Library Compiler, see the Library Compiler documentation.

Defining nonsequential constraints in the library cell results in a more accurate analysis than using the `set_data_check` command, because the setup and hold times can be made sensitive to the slew of the constrained pin and the related pin. The `set_data_check` command is not sensitive to slew.

To specify which clock domain to use at the related pin for data checks defined in library cells, you can use the `-clock clock_name` option of the `set_data_check` command.

The `remove_data_check` command does not remove data checks defined in library cells.

Specifying Clock-Gating Setup and Hold Checks

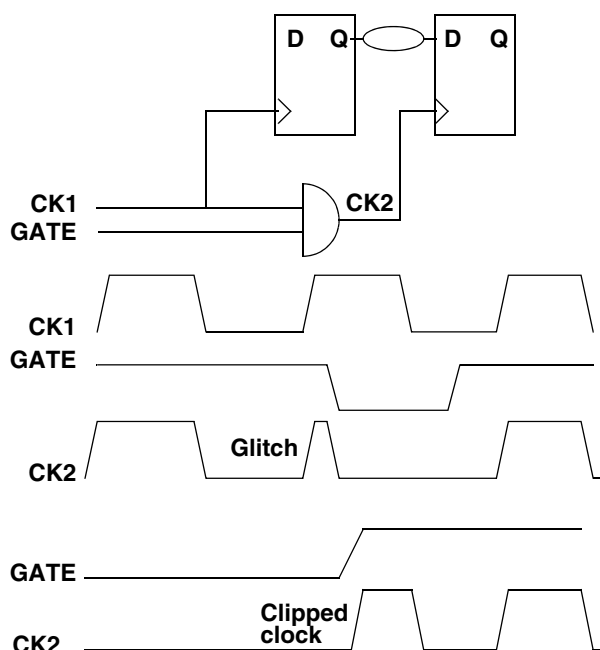
A gated clock signal occurs when the clock network contains logic other than inverters or buffers. For example, if a clock signal acts as one input to a logical AND function and a control signal acts as the other input, the output is a gated clock.

IC Compiler automatically checks for setup and hold violations on gating inputs, to ensure that the clock signal is not interrupted or clipped by the gate. This checking is done only for combinational gates where one signal is a clock that can be propagated through the gate and the gating signal is not a clock.

The clock-gating setup check ensures that the controlling data signal enables the gate before the clock becomes active. The arrival time of the leading edge of the clock signal is checked against both edges of any data signal feeding the data pins, to prevent a glitch at the leading edge of the clock pulse or a clipped clock pulse.

Clock-gating setup violations are shown in [Figure B-31](#).

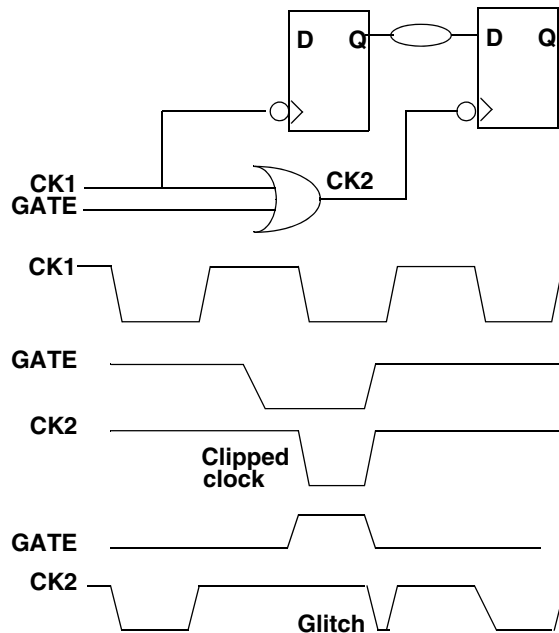
Figure B-31 Clock-Gating Setup Violations



The clock-gating hold check ensures that the controlling data signal remains stable while the clock is active. The arrival time of the trailing edge of the clock pin is checked against both edges of any data signal feeding the data pins using the hold equation. A clock-gating hold violation causes either a glitch at the trailing edge of the clock pulse or a clipped clock pulse. Note that for clock gating checks, it is required that the gating check must reach a clock pin in order to be successful. These clock pins include reference pins of sequential timing constraints, from-pins of sequential arcs, and so on.

Clock-gating hold violations are shown in [Figure B-32](#).

Figure B-32 Clock-Gating Hold Violations



By default, IC Compiler checks setup and hold times for gated clocks. A value of 0.0 is set as the setup and hold time (unless the library cell for the gate has gating setup or hold timing arcs). You can specify a nonzero setup or hold value with the `set_clock_gating_check` command.

Example 1

To specify a setup requirement of 0.2 and a hold requirement of 0.4 on all gates in the clock network of CLK1, enter

```
icc_shell> set_clock_gating_check \
           -setup 0.2 -hold 0.4 CLK1
```

Example 2

To specify a setup requirement of 0.5 on gate and1, enter

```
icc_shell> set_clock_gating_check -setup 0.5 and1
```

The `report_constraint` command performs clock-gating setup and hold checks. Use it to identify any clock-gating violations.

To remove clock-gating checks set with the `set_clock_gating_check` command, use the `remove_clock_gating_check` command.

Disabling or Restoring Clock-Gating Checks

You can specify any pin or cell in the current design or subdesigns to disable or restore clock-gating checks.

To get a report on the clock-gating checks that have been applied to cells or pins, use this command:

```
icc_shell> report_clock_gating_check objects
```

To disable clock-gating checks on cells or pins, use this command:

```
icc_shell> set_disable_clock_gating_check objects
```

To restore clock-gating checks on cells or pins, use this command:

```
icc_shell> remove_disable_clock_gating_check objects
```

If objects are not specified, all clock-gating checks are disabled.

Example 1

To restore disabled clock-gating checks for the object and1, enter

```
icc_shell> remove_disable_clock_gating_check and1
```

Example 2

To disable clock-gating checks on the specified cell or pin, enter

```
icc_shell> set_disable_clock_gating_check and2/Z
```


Index

A

- abbreviations in scripts 2-41
- add_end_cap command 9-34
- add_tap_cell_array command 9-3
- Advanced Route tool 8-51, A-58
- alias command 2-43
- align_objects command A-59
- aligning objects A-59
- alignment tools A-59
- all_active_scenarios command 14-12
- all_clocks command B-19
- all_rp_groups command 12-43
- all_rp_hierarchicals command 12-43
- all_rp_inclusions command 12-43
- all_rp_instantiations command 12-44
- all_rp_references command 12-44
- all_scenarios command 14-11
- always-on
 - power wells, multivoltage designs 13-30
 - synthesis
 - exclusive move bounds 13-30
 - multivoltage designs 13-49
- always-on attribute 13-40
- ambiguous commands 2-41
- AMD64 platform 1-2
- Analysis toolbar A-38

- analysis tools, layout
 - flylines A-38, A-39
 - map modes A-38, A-43
 - net connectivity A-38, A-40
 - visual modes A-38, A-41
- Application Preferences dialog box 2-31
- area utilization 4-24
- asynchronous clocks B-29
- attributes
 - always-on 13-40
 - connected_to_iso_cell 13-29
 - direct_power_rail_tie 13-46
 - dont_touch 13-32, 13-35
 - ignore, removing 12-46
 - is_isolation_cell 13-31, 13-37, 13-38
 - is_level_shifter 13-31, 13-37, 13-38
 - isolation_cell_enable_pin 13-31, 13-37
 - level_shifter_enable_pin 13-31, 13-37
 - relative placement, writing to disk 12-45
 - size_only 13-35, 13-36, 13-37
- auto_restore_mw_cel_lib_setup variable 3-4, 3-27

B

- balance_inter_clock_delay command 7-70
- blockages
 - creating placement blockages 4-5

- blockages, creating placement blockages A-62
- boundary cell, clock tree synthesis 7-32
- bounds, creating objects A-62
- Browsers toolbar 7-84, A-64
- buffer trees
 - creating 4-15
 - removing 4-15
 - reporting 4-15
- buffering high-fanout nets 4-14
- buffer-type level shifters
 - inserting 13-34
 - insertion threshold 13-33
 - placing 13-37
- bus naming style, specifying 3-11

C

- Calibre interface 8-44
- capacitance, net capacitance histogram A-70
- cell orientations A-59
 - displaying or hiding A-33
- cell placement statistics 8-35
- Cells List toolbar 2-33
- change_macro_view command 11-6, 11-23, 11-33, 11-40
- check_clock_tree command 7-57
- check_design command 13-52
- check_isolation_cells command 13-37, 13-52
- check_level_shifters command 13-48
- check_mv_design command 13-8, 13-32, 13-52
- check_physical_design command 3-25
- check_routeability command 8-2
- check_target_library_subset command 13-52
- check_timing command 3-29
- check_tlu_plus_files command 3-27
- chemical mechanical polishing thickness
 - displaying maps 9-44
 - loading data 9-44
- chip finishing

- adding end caps 9-34
- adding tap cell arrays 9-3
- chemical mechanical polishing thickness
 - displaying maps 9-44
 - loading data 9-44
- displaying critical area heat maps 9-28
- filling empty tracks with metal fill 9-37
- filling notches and gaps 9-42
- inserting pad fillers 9-36
- inserting redundant vias 9-30
- inserting standard cell fillers 9-31
- inserting tap cells 9-4
- inserting well fillers 9-34
- performing wire spreading 9-28
- reporting critical areas 9-26

- Clear Rulers A-3

- clear rulers A-15

- clock

- asynchronous B-26, B-29
- characteristics, specifying B-2
- clock divider B-4
- convergent 7-4
- creating B-3
- exclusive B-26
- expansion B-28
- gated clock setup/hold checking B-45
- generated clock B-4
- multiple clock relationships B-26
- off-chip B-4
- overlapping 7-4
- removing B-19
- selecting all clocks B-19
- source latency B-20
- synchronous B-26, B-27
- transition time B-25
- uncertainty B-23
- virtual B-4

- clock arrival histograms 7-86

- clock network element tables, path inspector window A-73

- clock network, non-unate B-12

- propagating clock paths B-12, B-37
- clock tree
 - design rule constraints priority 7-25
 - exception, float pins 7-44
 - removing 7-46
 - reporting 7-79
 - verifying 7-57
 - viewing 7-81
- clock tree arrival time histogram, displaying 7-86
- clock tree clustering, on-chip-variation-aware 7-33
- clock tree design rule constraint
 - maximum capacitance 7-24
 - maximum fanout 7-24
 - maximum transition time 7-24
- clock tree exceptions
 - defining 7-9
 - don't buffer net 7-17
 - don't size cell 7-17
 - don't touch subtree 7-16, 7-46
 - exclude pin 7-7, 7-12
 - float pin 7-13
 - nonstop pin 7-11
 - priority 7-10
 - removing 7-10
 - reporting 7-11
 - stop pin 7-7, 7-15
- clock tree fanout browser A-65
- clock tree hierarchy browser
 - defined 7-85
 - displaying 7-86
- clock tree optimization
 - ECO routing 7-69
 - optimization capabilities 7-68
 - postroute, default 7-69
 - preroute, default 7-69
 - selecting 7-69
 - postroute 7-69
 - preroute 7-69
 - timing analysis, router for 7-68
 - voltage area support 13-46
- clock tree reference
 - defined 7-3
 - resetting 7-19
 - specifying 7-18
- clock tree sinks
 - default 7-7
 - defining 7-7, 7-15
 - optimizing 7-61
- clock tree synthesis
 - boundary cell, defined 7-32
 - design rule constraints, viewing 7-25
 - don't touch subtree
 - defined 7-16, 7-46
 - logic-level balancing 7-35
 - exclude pin
 - defined 7-7, 7-12
 - implicit 7-7
 - float pin
 - defined 7-13
 - examples 7-44
 - modeling hard macros 7-44
 - ILM, logic-level balancing 7-35
 - logic-level balancing
 - defined 7-34
 - don't touch subtree 7-35
 - enabling 7-35
 - hard macro 7-36
 - ILM 7-35
 - maximum capacitance 7-24
 - maximum fanout 7-24
 - maximum level count, setting 7-26
 - maximum transition time 7-24
 - minimum insertion delay, defined 7-26
 - multicorner-multimode 7-56
 - nonstop pin
 - defined 7-11
 - implicit 7-7
 - power-aware placement 6-2
 - prerequisites
 - design 7-2
 - library 7-3
 - skew, defined 7-26

- stop pin
 - defined 7-7, 7-15
 - implicit 7-7
- timing goals 7-26
- voltage area support 13-46
- clock tree synthesis options
 - defining 7-20
 - reporting 7-23
 - resetting 7-23
- clock tree synthesis scenario, multicorn-
multimode 14-17
- clock_opt command 7-58, 13-44, 14-16
- close_mw_lib command 3-16
- closing the GUI 2-8
- collection_result_display_limit variable 12-44
- command
 - abbreviation in scripts 2-41
 - console, main window 2-23
 - displaying options 2-5
 - execution at startup 2-3
 - history 2-26
 - listing for session 2-26
 - listing in GUI 2-28
 - menu, choosing task mode in layout window
2-30
 - output, redirecting 2-43
 - saving session transcript 2-26
- command line 2-3
 - copying commands to command line 2-5
 - entering commands 2-4, 2-41
 - icc_shell 2-4
 - shell 2-4, 2-41
 - X-term window 2-4, 2-41
- command scripts, Tcl (tool command
language) 2-8
- commands
 - add_end_cap 9-34
 - add_tap_cell_array 9-3
 - alias 2-43
 - align_objects A-59
 - all_active_scenarios 14-12
 - all_clocks B-19
 - all_rp_groups 12-43
 - all_rp_hierarchicals 12-43
 - all_rp_inclusions 12-43
 - all_rp_instantiations 12-44
 - all_rp_references 12-44
 - all_scenarios 14-11
 - ambiguous 2-41
 - balance_inter_clock_delay 7-70
 - change_macro_view 11-6, 11-23, 11-33,
11-40
 - check_clock_tree 7-57
 - check_design 13-52
 - check_isolation_cells 13-37, 13-52
 - check_level_shifters 13-48
 - check_mv_design 13-8, 13-32, 13-52
 - check_physical_design 3-25, 13-51
 - check_routeability 8-2
 - check_target_library_subset 13-52
 - check_timing 3-29
 - check_tlu_plus_file 3-27
 - choosing in menus 2-5
 - clock_opt 7-58, 13-44, 14-16
 - close_mw_lib 3-16
 - compare_delay_calculation 3-32
 - compile 13-5, 13-32, 13-37
 - compile_clock_tree 7-66
 - compile_ultra 13-5, 13-32, 13-37
 - connect_power_domain 13-53
 - connect_power_net_info 13-53
 - connect_supply_command 13-17
 - convert_fill_polygons 9-37
 - convert_wire_ends 8-35
 - copy_floorplan 3-24
 - copy_mw_cel 3-21
 - copy_object A-57
 - copying session log 2-25
 - create_auto_shield 8-33
 - create_bounds 4-9, A-62
 - create_bounds, exclusive 13-30
 - create_buffer_tree 4-15
 - create_clock B-3

create_generated_clock B-5
 create_ilm 11-7, 11-17, 11-19, 11-40, 13-47
 create_macro_fram 11-6, 11-23, 11-33, 11-40
 create_mw_lib 3-10
 create_net_shape A-63
 create_physical_bus 8-47, 8-50
 create_physical_buses_from_patterns 8-48
 create_pin_guide A-62
 create_placement 13-38
 create_placement_blockage 11-23, A-62
 create_plan_groups command A-62
 create_power_domain 13-3, 13-13, 13-53
 create_power_net_info 13-53
 create_rail_setup 8-59
 create_route_guide 8-3, 11-23, A-62
 create_scenario 14-2, 14-11
 create_terminal A-63
 create_text A-63
 create_track A-62
 create_user_shape A-63
 create_via A-63
 create_voltage_area 13-4, 13-22, 13-24, 13-26, A-62
 current_mw_cel 2-33
 current_scenario 14-11
 define_routing_rule 7-27, 8-6
 derive_pg_connection 13-53
 disconnect_power_net_info 13-53
 distribute_objects A-59
 expand_objects A-59
 extract_rc 4-20, 8-57, 11-25, 14-15
 extract_rp_group 12-53
 flip_objects A-60
 Flylines A-38
 Follow Selection A-14
 get_always_on_logic 13-49
 get_bounds 4-9
 get_clocks B-18
 get_cts_scenario 7-56, 14-17
 get_die_area 11-23, 11-40
 get_dominant_scenarios 14-15
 get_ilm_objects 11-19, 11-40
 get_ilms 11-19, 11-40
 get_license 2-49
 get_magnet_cells 4-13
 get_placement_blockages 4-6, 4-7
 get_route_guides 8-4
 get_selection 2-24
 get_voltage_area 13-25
 Go Back in Zoom and Pan History A-14
 Go Forward in Zoom and Pan History A-14
 gui_show_form 2-12
 gui_show_man_page 2-11
 gui_start 2-7
 gui_start command 2-3
 gui_stop 2-8
 help 2-11
 history 2-26, 2-43
 history view 2-27
 icc_shell 2-4
 import_designs 3-22
 insert_isolation_cell 13-5, 13-37
 insert_level_shifters 13-5, 13-32, 13-34
 insert_metal_filler 9-39
 insert_ng_filler 9-42
 insert_pad_filler 9-36
 insert_redundant_vias 9-30
 insert_stdcell_filler 9-32, 13-47
 insert_tap_cells_by_rules 9-5
 insert_well_filler 9-34
 interrupting 2-7
 keyboard shortcuts in GUI 2-14
 legalize_placement 13-38
 license_users 2-49
 link 3-22
 magnet_placement 4-12
 Map Mode A-38
 move_object A-57
 Named Zoom and Pan Settings A-14
 Net Connectivity A-38
 Open Design 2-31
 optimize_clock_tree 7-68
 optimize_pre_cts_power 7-51

optimize_wire_via 8-32	remove_level_shifters 13-32
order_rp_groups 12-54	remove_license 2-51
Pan to Selection A-12	remove_net_routing_layer_constraints 4-21
place_opt 4-16, 11-33, 13-4, 13-6, 13-38, 13-44, 13-45, 14-2, 14-13, 14-16	remove_objects A-58
process_particle_probability_file 9-27	remove_placement_blockage 4-7
propagate_constraints 13-47	remove_power_domain 13-53
psynopt 4-18, 13-4, 13-45, 14-2, 14-13	remove_power_net_info 13-53
psynopt, on_route option 14-13	remove_preferred_routing_direction 8-6
read_cmp_thickness 9-44	remove_route_by_type 8-11
read_def 3-23	remove_route_guides 8-5
read_drc_error_file 8-44	remove_rp_group_options 12-46
read_floorplan 3-23	remove_rp_groups 12-46
read_parasitics 3-28	remove_sdc 3-29
read_saif 6-3	remove_stdcell_filler
read_sdc 3-29	-pad 9-37
read_sdf 3-28	-stdcell 9-33
read_verilog 3-21	-tap 9-6
recalling 2-43	remove_target_library_subset 13-8
redirect 2-44	remove_voltage_area 13-30
redo A-57	remove_well_filler 9-36
refine_placement 4-42	report_all_spacing_rules 4-11
remove_all_spacing_rules 4-11	report_annotated_delay 11-9
remove_annotations 3-28	report_area 11-19
remove_bounds 4-9	report_bounds 4-9
remove_buffer_tree 4-15	report_buffer_tree 4-15
remove_clock B-19	report_cell 13-52
remove_clock_groups B-36	report_clock_timing 7-81
remove_clock_tree 7-46	report_clock_tree 7-79
remove_clock_tree_exceptions	report_constraint 7-96, 14-23
-dont_buffer_nets 7-17	report_critical_area 9-26
-dont_size_cells 7-17	report_delay_calculation 4-29, 13-52
-dont_touch_subtrees 7-16	report_design 8-35, 11-19
-exclude_pins 7-12	report_error_coordinates 8-2, 8-44
-float_pins 7-15	report_extraction_options 14-14
-non_stop_pins 7-11	report_hierarchy 13-52
-stop_pins 7-16	report_keepout_margin 4-5
remove_data_check B-44	report_net_routing_layer_constraints 4-21
remove_filler_with_violation 9-33	report_noise_calculation 10-12
remove_from_rp_group 12-51	report_operating_conditions 13-52
remove_ideal_network 3-29	report_physical_signoff 8-37
remove_keepout_margin 4-5	report_placement_blockages 4-7
	report_placement_utilization 4-24

- report_power 4-31
- report_power_domain 13-53
- report_power_guide 13-30
- report_power_net_info 13-53
- report_power_options 6-2
- report_preferred_direction 8-5
- report_qor 14-21
- report_route_opt_strategy 8-23
- report_route_options 8-11
- report_routing_rules 7-27
- report_scan_chain 5-11
- report_signal_em 10-18
- report_target_library_subset 13-8, 13-52
- report_threshold_voltage_group 6-4
- report_timing 4-27, 7-96, 13-52
- report_timing -scenario 14-21
- report_timing_derate 3-31
- report_tlu_plus_files 3-27, 14-24
- report_units 3-16, 10-16, 10-17
- report_voltage_area 13-30, 13-52
- reset_clock_tree_references 7-19
- reset_design 3-29
- reset_timing_derate 3-31
- resize_object A-57
- reusing 2-27
- rotate_objects A-60
- route_detail 8-31
- route_global 8-30
- route_group 8-19
- route_opt 13-44, 14-2, 14-13, 14-14, 14-15, 14-16
- route_search_repair 8-32
- route_spreadwires 9-28, 9-29
- route_track 8-31
- route_zrt_eco 8-33
- rp_group_inclusions 12-44
- rp_group_instantiations 12-44
- rp_group_references 12-44
- run_router 13-45
- Save Design 2-31
- save_mw_cel 3-33
- save_mw_cel -scenario 14-13
- saving transcript of session commands 2-27
- set_active_scenarios 14-12, 14-15
- set_ahfs_options 4-14
- set_always_on_strategy 13-38
- set_attribute 13-39, 13-40
- set_clock_gating_check B-47
- set_clock_groups B-29
- set_clock_latency B-21
- set_clock_transition B-25
- set_clock_tree_exceptions
 - dont_buffer_nets 7-17
 - dont_size_cells 7-17
 - dont_touch_subtrees 7-16
 - exclude_pins 7-12
- float pin options 7-13, 7-44
- non_stop_pins 7-11
- stop_pins 7-15
- set_clock_tree_references 7-18
- set_clock_uncertainty B-23
- set_congestion_options 4-7
- set_cts_scenario 7-56, 14-17
- set_data_check B-40
- set_delay_calculation 3-32
- set_delay_estimation_options 4-22, 4-23
- set_disable_clock_gating_check B-48
- set_distributed_route 8-15
- set_extraction_options 8-58, 9-38, 14-14
- set_fix_hold 14-16
- set_fix_hold_options 14-16
- set_gui_stroke_bindings A-38
- set_gui_stroke_preferences A-38
- set_ignored_layers 8-8
- set_inter_clock_delay_options 7-53
- set_keepout_margin 4-4, 11-24
- set_left_right_filler rule 9-32
- set_level_shifter_strategy 13-32, 13-34
- set_level_shifter_threshold 13-32, 13-33, 13-34
- set_lib_cell_spacing_label 4-10, 4-11
- set_max_leakage_power 14-4
- set_min_library 3-3, 3-30, 14-9
- set_mw_lib_reference 3-17

- set_mw_technology_file 3-16
- set_net_aggressors 8-10
- set_net_routing_layer_constraints 4-21
- set_operating_conditions 3-30, 13-36, 13-47, 14-2, 14-6, 14-7
- set_optimize_dft_options 5-10
- set_physical_signoff 8-37
- set_physical_signoff_options 9-41
- set_power_guide 13-30
- set_power_net_to_voltage_area 13-47
- set_preferred_routing_direction 8-5
- set_route_opt_strategy 8-22
- set_route_options 8-11
- set_route_type 8-11
- set_rp_group_options 12-48
- set_scope 13-13
- set_si_options 3-33, 13-48, 14-15
- set_spacing_label_rule 4-10, 4-11
- set_switching_activity 6-3
- set_target_library_subset 13-6, 13-8
- set_timing_derate 3-31
- set_tlu_plus_files 3-27, 9-38, 14-4, 14-12
- signoff_drc 8-36
- signoff_metal_fill 9-41
- split_objects A-58
- stretch_wire A-58
- stroke A-36
- stroke activated commands A-36
- terminating 2-7
- undo A-57
- uniquify_fp_mw_cel 3-21
- unset_power_guide 13-30
- update_bounds 4-9
- update_clock_latency 7-70
- update_voltage_area 13-25, 13-26
- verify_drc 8-40
- verify_route 8-43
- Visual Mode A-38
- window_stretch A-58
- write_def 3-33
- write_mw_lib_files 3-17
- write_parasitics 3-34, 8-57, 14-13
- write_rp_groups 12-45
- write_sdc 3-34
- write_stream 3-34
- Zoom Fit All A-12
- Zoom Follows Selection A-12
- Zoom In 2x A-12
- Zoom Out 2x A-12
- Zoom to Selection A-12
- compare_delay_calculation command 3-32
- compile command 13-5, 13-32, 13-37
- compile_clock_tree command 7-66
- compile_ultra command 13-5, 13-32, 13-37
- compression, relative placement 12-28
- concurrent analysis and optimization, multicorner-multimode 14-2
- congestion map, layer-based 8-30
- congestion map, defined 8-30
- congestion options 4-7
- connect_power_domain command 13-53
- connect_power_net_info command 13-53
- connected_to_iso_cell attribute 13-29
- Connectivity Settings panel A-40
- console
 - command line 2-24
 - entering commands 2-24
 - history view 2-26
 - log view 2-25
 - main window 2-23
 - saving log messages 2-25
 - searching the transcript 2-25
 - window 2-24
- constrained pin (for data check) B-39
- constraints
 - reading 3-29
 - report 7-96
 - saving 3-34
 - timing 3-28
- convergent clock path, defined 7-4
- convert_fill_polygons command 9-37
- convert_wire_ends command 8-35

- Copy tool A-57
- copy_floorplan command 3-24
- copy_mw_cel command 3-21
- copy_object command A-57
- copying
 - commands 2-5
 - objects A-57
 - session log 2-25
- coupling capacitance
 - extracting 8-57
 - filtering, multicorner-multimode 14-15
 - scaling 8-58
- create object tools A-62
- create_auto_shield command 8-33
- create_bounds command 4-9, A-62
 - exclusive 13-30
- create_buffer_tree command 4-15
- create_clock command B-3
- create_generated_clock command B-5
- create_ilm command 11-7, 11-17, 11-19, 11-40, 13-47
- create_macro_fram command 11-6, 11-23, 11-33, 11-40
- create_mw_lib command 3-10
- create_net_shape command A-63
- create_physical_bus command 8-47, 8-50
- create_physical_buses_from_patterns command 8-48
- create_pin_guide command A-62
- create_placement command 13-38
- create_placement_blockage command 11-23, A-62
- create_plan_groups command A-62
- create_power_domain command 13-3, 13-53
- create_power_net_info command 13-53
- create_rail_setup command 8-59
- create_route_guide command 8-3, 11-23, A-62
- create_scenario command 14-2, 14-11
- create_terminal command A-63
- create_text command A-63
- create_track command A-62
- create_user_shape command A-63
- create_via command A-63
- create_voltage_area command 13-4, 13-22, 13-24, 13-26, A-62
- creating objects A-62
- critical area
 - displaying heat maps 9-28
 - encrypting and decrypting the particle probability function 9-27
 - reporting 9-26
- crosstalk
 - analysis
 - delta delay visual mode 10-11
 - noise reports 10-11
 - preparation 10-8
 - static noise visual mode 10-11
 - timing report 10-11
 - defined 10-2
 - net lists, path inspector window A-73
 - optimization
 - about 10-7
 - postroute 10-7
 - signoff optimization 10-12
 - prevention
 - clock shielding 10-5
 - clock tree synthesis 10-5
 - placement 10-5
 - track assignment 10-6
 - reduction engines 10-6
 - timing impacts 10-10
- crosstalk prevention, net aggressors 8-10
- cts_do_characterization variable 7-78
- cts_enable_rc_constraints variable 7-39
- cts_fix_clock_tree_sinks variable 7-61
- cts_move_clock_gate variable 7-61
- cts_rc_relax_factor variable 7-39
- cts_traverse_dont_touch_subtrees variable 7-16
- cts_use_debug_mode variable 7-78

- cts_use_lib_max_fanout variable 7-25
- cts_use_sdc_max_fanout variable 7-25
- current design, setting 2-33
- current_mw_cel command 2-33
- current_scenario command 14-11
- Custom Wire tool A-58

D

- data checks
 - clock domains B-44
 - defining B-39
 - nochange check B-42
 - removing B-44
 - set_data_check command B-40
 - timing reports B-43
- database, Milkyway
 - about 3-2
 - saving a design 3-33
- datapath, physical, defined 12-1
- DEF file
 - reading 3-23
 - writing 3-33
- define_routing_rule command 7-27, 8-6
- delay calculation algorithm
 - Arnoldi 3-32
 - clock Arnoldi 3-32
 - Elmore 3-32
 - postroute
 - default 3-32
 - setting 3-32
- Delete tool A-58
- deleting objects A-58
- densely-packed objects A-5
- derating factors for timing library 3-31
- derive_pg_connection command 13-53
- design for manufacturing (DFM)
 - chemical mechanical polishing thickness
 - displaying maps 9-44
 - loading data 9-44
 - critical area heat map displaying 9-28
 - critical area reporting 9-26
 - metal density filling 9-37
 - via doubling 9-30
 - wire spreading 9-28
- design overlays, layout view A-34
- design rules, routing
 - metal density 9-37
 - notch and gap filling 9-43
- design task mode, layout window 2-30
- designs
 - linking 3-22
 - opening 2-31
 - saving 2-31
 - schematics A-23
- detail routing 8-31
- dialog boxes
 - finding 2-12
- direct_power_rail_tie attribute 13-46
- disconnect_power_net_info command 13-53
- DISPLAY environment variable
 - checking before startup 2-2
 - overriding 2-3
- displaying command options 2-5
- distribute_objects command A-59
- distributed routing 8-15
- distributing objects A-59
- distribution commands A-59
- divide-by clock B-4
- dont_touch attribute 13-32, 13-35
- draw rulers A-15
- dynamic power optimization 6-5
 - switching activity information 6-3

E

- ECO routing
 - during clock tree optimization 7-69
 - running 8-33
- Edit Snapping Settings panel A-61
- Edit tool

- moving objects A-57
- resizing objects A-57
- Edit toolbar A-55
- editing object properties A-21
- edits
 - redo A-62
 - undo A-62
- electromigration
 - loading net switching information for analysis 10-16
 - performing signal analysis 10-18
 - preparing data for analysis 10-15
 - signal analysis repair file 10-21
- emulation metal fill extraction 9-38
- enable-type level shifters
 - inserting 13-37
 - placing 13-37
- end cap
 - adding 9-34
 - defined 9-34
- endpoint slack histogram
 - histogram views A-70
 - Timing Views toolbar A-70
- enhanced image accuracy A-32
- environment variable, DISPLAY 2-2
- environment variables
 - SNPS_MAX_QUEUEETIME 2-50
 - SNPS_MAX_WAITTIME 2-50
 - SNPSLMD_QUEUE 2-50
- error browser, viewing routing and verification errors A-47
- estimating via resistance 4-23
- exceptions for clock tree synthesis
 - don't buffer net 7-17
 - don't size cell 7-17
 - don't touch subtree 7-16, 7-46
 - exclude pins 7-7, 7-12
 - float pin 7-13, 7-44
 - nonstop pins 7-11
 - stop pins 7-7, 7-15
- exclusive clocks

- logical B-29
- physical B-30
- exclusive move bounds, always-on synthesis 13-30
- executing
 - command at startup 2-3
 - script at startup 2-3
- exiting IC Compiler 2-10
- Expand Browser toolbar 7-84, A-66
- expand_objects command A-59
- expanding
 - clocks B-28
 - objects A-59
- expansion commands A-59
- exporting
 - GDSII 3-34
 - Oasis 3-34
 - parasitic data 3-34
- extract_rc command 4-20, 8-57, 11-25, 14-15
- extract_rp_group command 12-53
- extraction
 - postroute 8-57
 - real and emulation metal fill 9-38

F

- File toolbar 2-31
- files, bytecode-compiled 2-48
- FILL view 9-42
- fill, metal 9-37, 9-39, 9-41
- filler cell
 - inserting pad fillers 9-36
 - inserting standard cell fillers 9-31
 - inserting well fillers 9-34
 - removing pad fillers 9-37
 - removing standard cell fillers 9-33
 - removing standard cell fillers that have routing design rule violations 9-33
 - removing tap cells 9-6
 - removing well fillers 9-36
- fix objects A-60

- fix objects for edit A-57
- flip_objects command A-60
- flipping object commands A-60
- float pin
 - defined 7-44
 - setting 7-44
- floorplan file, reading 3-23
- Flyline Settings panel A-39
- flylines
 - Analysis toolbar A-38
 - displaying A-39
 - Flyline Settings panel A-39
- Flylines command A-38
- Follow Selection command A-14
- following selected objects in graphic views
 - A-14
- functional ECO methods, using 15-2

G

- gated clock, setup/hold checking B-45
- GDSII format 3-34
- generated clocks B-4
 - as clock tree roots 7-4
 - restricting clock source latency calculation
 - B-11
- get_always_on_logic command 13-49
- get_bounds command 4-9
- get_clocks command B-18
- get_cts_scenario command 7-56, 14-17
- get_die_area command 11-23, 11-40
- get_dominant_scenarios command 14-15
- get_ilm_objects command 11-19, 11-40
- get_ilms command 11-19, 11-40
- get_license command 2-49
- get_magnet_cells command 4-13
- get_placement_blockages command 4-6, 4-7
- get_route_guides command 8-4
- get_selection command 2-24
- get_voltage_area command 13-25
- global routing
 - defined 8-29
 - running 8-30
 - voltage area support 13-46
- global skew, defined 7-26
- Go Back in Zoom and Pan History command
 - A-14
- Go Forward in Zoom and Pan History
 - command A-14
- Graphical User Interface (GUI) 1-4
- grid snapping A-57
- grids
 - displaying or hiding A-28
 - litho grid A-28
 - user grid A-28
- groups
 - creating plan groups A-62
 - relative placement, defined 12-9
- guard bands, voltage area 13-26
- GUI 1-4
 - closing 2-8
 - displaying on named terminal 2-3
 - help starting 2-3
 - hiding 2-8
 - opening 2-7
 - starting while executing command 2-3
 - starting while executing script 2-3
- GUI preferences
 - saving 2-31
 - setting 2-31
- GUI windows 2-18
 - menu bar 2-19
 - status bar 2-20
 - toolbars 2-19
 - view windows 2-21
- gui_show_form command 2-12
- gui_show_man_page command 2-11
- gui_start command 2-3, 2-7
- gui_stop command 2-8
- guides

- creating pin guides A-62
- creating route guides A-62

H

- help command 2-11
- Help system 2-11
 - accessing 2-15
 - choosing browser 2-17
 - configuring Web browser for 2-17
 - searching for text in 2-16
- Hercules
 - design rule checking
 - distributed 8-38
 - signoff_drc command 8-36
 - verify_drc 8-40
 - environment, setting up 8-37
 - layer mapping file 8-37
- Hercules metal fill 9-41
- hiding the GUI 2-8
- hierarchy browser
 - exploring design hierarchy A-22
 - logic hierarchy view A-22
- high-fanout nets, buffering 4-14
- high-fanout synthesis
 - during place_opt 4-14
 - standalone 4-15
 - using compile_clock_tree 7-66
 - voltage area support 13-45
- high-level design flow, multivoltage designs 13-6
- Highlight tool A-8
- Highlight toolbar A-9
- highlighting
 - critical path A-10
 - selected objects A-10
 - timing paths A-10
- histograms A-70
 - clock arrival histograms 7-86
 - endpoint slack histogram A-70
 - net capacitance histogram A-70

- path slack histogram A-70

- history
 - command log 2-27
 - view 2-26
- history command 2-43
- hotkeys report, displaying 2-14

I

- IC Compiler 2-3
 - exiting 2-10
 - Help 2-11
 - icc_shell commands 2-4
 - starting 2-2, 2-3
- IC Compiler Help 2-11
- IC Compiler packages
 - IC Compiler 1-2
 - IC Compiler-DP 1-2
 - IC Compiler-PC 1-2
 - IC Compiler-XP 1-2
- icc_shell command 2-3
 - displaying options 2-5
 - listing startup options 2-3
 - starting IC Compiler 2-2
 - using 2-4, 2-41
- icc_shell command-line interface 1-4
- icc_shell> prompt 2-3, 2-4
- ignore attribute, removing 12-46
- ILM (interface logic model) 11-1
- ilm_allow_all_rotations variable 11-30
- ilm_disable_ilm_checks variable 11-10
- image accuracy A-32
- import_designs command 3-22
- importing, Verilog 3-21
- incremental placement 4-18
- information
 - displaying for objects in graphic views A-11
 - displaying InfoTips in layout views A-5
- InfoTips A-5
- insert_isolation_cell command 13-5, 13-37

- insert_level_shifters command 13-5, 13-32, 13-34
- insert_metal_filler command 9-39
- insert_ng_filler command 9-42
- insert_pad_filler command 9-36
- insert_redundant_vias command 9-30
- insert_stdcell_filler command 9-32, 13-47
- insert_tap_cells_by_rules command 9-5
- insert_well_filler command 9-34
- inserting enable-type level shifters, multivoltage designs 13-37
- inserting isolation cells, multivoltage designs 13-37
- insertion threshold, buffer-type level shifters 13-33
- inspecting timing paths A-72
- interactive clock tree synthesis 7-93
- interactive clock tree synthesis window, displaying 7-84
- intercell spacing rules, defining 4-10, 4-11
- interclock delay balancing
 - clock group, specifying 7-53
 - running 7-70
 - timing analysis, router for 7-70
- interclock skew, balancing 7-52
- interface logic model 11-33
 - area information, reporting 11-20
 - automatic checks 11-10
 - benefits for using 11-3
 - clock_opt command 11-34
 - command summary, list 11-40
 - considerations for using 11-4
 - creating 11-17
 - defined 11-1
 - delay estimation, propagating 11-25
 - description 11-2
 - design information
 - differences depending on current design 11-20
 - reporting 11-20
 - used with model 11-16

- examining in GUI A-34
- fanin and fanout of chip-level networks, controlling 11-11
- flow for creating 11-7
- FRAM view 11-4, 11-6, 11-16, 11-23, 11-24, 11-33, 11-40
- guidelines for creating 11-9
- ILM view 11-4, 11-9, 11-14, 11-31
- information about, reporting 11-19
- instantiating and using at the top level 11-31
- keepout margins, creating 11-24
- keepouts
 - creating and propagating 11-22
 - creating manually 11-23
 - creating manually, using get_die_area command 11-23
 - default 11-24
 - placement blockage 11-22
 - route guide 11-22
- logic included in, controlling 11-10
- logic-level balancing in clock tree synthesis 7-35
- mirrored 11-14
- multicorner-multimode flows 14-17
- number of latch levels, controlling 11-12
- on-route capability 11-36
- overview 11-2
- propagation of information to the top-level design 11-22
- reporting information 11-19
- rotated 11-14
- rotated and mirrored models, handling 11-27
- route_opt command 11-36
- side-load cells
 - controlling 11-13
- signal integrity 11-3, 11-36
- smallest model, getting 11-17
- storing parasitic delay information 11-25
- usage at top level 11-32
- viewing in GUI 11-33
- voltage area support 13-47
- interfaces
 - GUI 1-4

- icc_shell 1-4
- interrupting commands 2-7
- is_isolation_cell attribute 13-31, 13-37, 13-38
- is_level_shifter attribute 13-31, 13-37, 13-38
- isolation cells
 - and level shifters, multivoltage designs 13-5
 - handling in multivoltage designs 13-31
 - inserting 13-37
 - requirements, multivoltage designs 13-31
- isolation_cell_enable_pin attribute 13-31, 13-37

K

- keepout margins
 - cell-specific, setting 4-4
- keepouts
 - placement blockage 11-22
 - route guide 11-22
- keyboard shortcuts 2-14
- k-factors
 - multivoltage designs 13-6
 - unsupported in multicorner-multimode 14-8

L

- latency, source B-20
- layer constraints
 - defining 4-21
- layout editing 8-45, 8-47, 8-48, 8-49, 8-51
- layout editing tools 8-45, 8-51, A-55
 - Create Custom Wire tool 8-54
- layout tools, analysis A-38
- layout view A-2, A-26
 - refreshing A-17
 - save screenshot of A-36
- layout window 2-29
- legalize_placement command 13-38
- level shifter commands, multivoltage designs 13-32

- level shifters
 - and isolation cells, multivoltage designs 13-5
 - handling in multivoltage designs 13-31
 - placing 13-37
 - requirements, multivoltage designs 13-31
 - threshold, multivoltage designs 13-33
- level_shifter_enable_pin attribute 13-31, 13-37
- library data, preparing 1-6
- library requirements,
 - multivoltage designs 13-6
- license queuing
 - enabling 2-50
 - environment variables 2-50
 - SNPS_MAX_QUEUE TIME 2-50
 - SNPS_MAX_WAIT TIME 2-50
 - SNPSLMD_QUEUE 2-50
- license_users command 2-49
- licenses
 - checking out 2-49
 - enabling queuing 2-50
 - listing 2-49
 - releasing 2-51
 - using 2-48
 - working with 2-48
- licenses, getting or releasing in GUI 2-51
- link command 3-22
- linking the design 3-22
- Linux platforms 1-2
- Linux shell 2-3
- listing commands 2-28
- listing startup options 2-3
- litho grid, displaying or hiding A-28
- locking objects for edit A-57
- log view 2-25
- logic hierarchy view A-22
- logic libraries 1-6
- logically exclusive clocks B-29
- low-effort flow 1-8

M

macros

- voltage area 13-29

magnet placement 4-12

magnet_placement command 4-12

main library, defined 3-3

main window 2-3, 2-22

man page viewer 2-14

map file for TLUPlus 3-26

Map Mode command A-38

Map Mode panel 9-28, 9-44, 9-45, A-43

map modes

- Analysis toolbar A-38

- displaying A-43

- Map Mode panel A-43

maximum net length optimization, voltage area support 13-45

maximum timing information 3-3, 3-30

mcmm_enable_high_capacity_flow variable 14-16

mcmm_high_capacity_effort_level variable 14-16

menu bar 2-19

menu commands

- choosing 2-5

- finding 2-12

menu task mode, layout window 2-30

menus

- choosing task mode in layout window 2-30

- keyboard shortcuts 2-14

metal density filling

- inserting 9-37

metal fill 9-37, 9-39, 9-41

metal fill extraction, real and emulation 9-38

methodology 1-5

Milkyway database

- about 3-2

- saving a design 3-33

Milkyway design

- copying 3-21

Milkyway design library

- changing settings 3-16

- closing 3-16

- creating 3-10

- defined 3-9

- reporting

 - Milkyway reference libraries 3-9, 3-16

 - units 3-16, 10-16, 10-17

Milkyway format, saving in 1-7

Milkyway reference library

- changing 3-17

- defined 3-9

- specifying 3-11

minimum insertion delay, defined 7-26

minimum timing information 3-3, 3-30

model, interface logic, defined 11-1

Mouse Tool Options toolbar A-5

mouse tools

- Advanced Route tool A-58

- Copy tool A-57

- Custom Wire tool A-58

- Delete tool A-58

- Edit tool (move objects) A-57

- Edit tool (resize objects) A-57

- Highlight tool A-8

- Mouse Tools toolbar A-3

- Pan tool A-3

- Query tool A-3, A-11

- Route tool A-58

- Ruler tool A-3, A-15

- Selection Tool A-3

- Selection tool A-6

- Split tool A-58

- Stretch Wire tool A-58

- Window Stretch tool A-58

- Zoom In tool A-3

- Zoom Out tool A-3

Mouse Tools toolbar A-3

move bound

- adding cells to 4-9

- creating 4-9

- defined 4-8
- deleting 4-9
- exclusive 4-8
- hard 4-8
- removing cells from 4-9
- reporting 4-9
- soft 4-8
- move_object command A-57
- moving and stretching objects A-58
- moving objects A-57
- multicorner-multimode
 - all_active_scenarios command 14-12
 - all_scenarios command 14-11
 - basic concepts 14-2
 - basic flow 14-3
 - clock tree synthesis 7-56
 - clock tree synthesis scenario 14-17
 - commands 14-10
 - concurrent analysis and optimization 14-2
 - create_scenario command 14-2, 14-11
 - current_scenario command 14-11
 - k-factors, unsupported 14-8
 - multicorner library setup 14-5
 - optimization 14-13
 - CC filtering 14-15
 - RC scaling 14-14
 - preroute hold fixing 14-16
 - report_constraint command 14-23
 - report_qor command 14-21
 - report_timing -scenario command 14-21
 - report_tlu_plus_files command 14-24
 - save_mw_cel command
 - scenario 14-13
 - scenario definition 14-2
 - scenario management, commands 14-10
 - scenario reduction 14-15
 - script example 14-24
 - set_active_scenarios command 14-12
 - set_max_leakage_power command 14-4
 - set_min_library command 14-9
 - set_tlu_plus_files command 14-4, 14-12
 - setup considerations 14-4
 - using interface logic models (ILMs) 14-17
 - voltage area support 13-49
 - write_parasitics command 14-13
- multiple instances, support for 3-21
- multivoltage designs
 - always-on power wells 13-30
 - always-on synthesis 13-49
 - creating voltage areas 13-24
 - handling isolation cells 13-31
 - handling level shifters 13-31
 - high-level design flow 13-6
 - identifying characteristics 13-7
 - inserting enable-type level shifters 13-37
 - inserting isolation cells 13-37
 - isolation cell requirements 13-31
 - isolation cells 13-31
 - k-factors 13-6
 - level shifter commands 13-32
 - level shifter placement 13-37
 - level shifter requirements 13-31
 - level shifter threshold 13-33
 - level shifters 13-31
 - level shifters and isolation cells 13-5
 - library requirements 13-6
 - nested voltage areas 13-28
 - optimization script 13-54, 13-55, 13-56
 - physical synthesis 13-6
 - placing and optimizing 13-49
 - power domains 13-2
 - power domains and voltage areas 13-4
 - power domains, using 13-52
 - reporting 13-51
 - target library subsetting 13-8
 - using voltage areas 13-22
 - voltage areas 13-4
- multivoltage specific checks 13-51

N

- Named Zoom and Pan Settings command A-14

- nested voltage areas, multivoltage designs 13-28
- net aggressors 8-10
- net capacitance histogram
 - histogram views A-70
 - Timing Views toolbar A-70
- net connectivity
 - Analysis toolbar A-38
 - Connectivity Settings panel A-40
 - displaying A-40
- Net Connectivity command A-38
- net layer constraints, specifying 8-8
- net shapes, creating objects A-63
- net shielding 8-33
- net switching information, loading 10-16
- NLDM (nonlinear delay model) libraries 13-54
- nondefault routing rules, setting 8-6
- nonlinear delay model libraries 13-54
- nonsequential constraints B-39
- non-unate clock signal B-12
- notch and gap filling 9-42

O

- Oasis format 3-34
- object alignment tools A-59
- object distribution commands A-59
- object expansion commands A-59
- object flipping commands A-60
- object information
 - InfoTips A-5
 - Query tool A-11
- object properties A-21
- object rotation commands A-60
- object snapping options for layout editing A-61
- object splitting tools A-58
- object spreading commands A-59
- objects
 - create object tools A-62
 - fix or unfix for editing A-60

- select by name A-18
- selecting in graphic views A-6
- on-chip-variation, clock tree clustering 7-33
- online Help 2-11
 - accessing 2-15
 - choosing browser 2-17
 - configuring Web browser for 2-17
 - searching for text in 2-16
- Open Design command 2-31
- opening designs 2-31
- opening the GUI 2-3, 2-7
- operating condition analysis modes 3-30
 - best-case and worst-case 3-30
 - on-chip variation 3-30
- optimization
 - multicorner-multimode 14-13
- optimization script, multivoltage designs 13-54, 13-55, 13-56
- optimize_clock_tree command 7-68
- optimize_pre_cts_power command 7-51
- optimize_wire_via command 8-32
- order_rp_groups command 12-54
- orientations A-59
- orientations, displaying or hiding for cells A-33
- orienting cells A-59
- overlapping clock path, defined 7-4
- overlapping objects A-5
- overlays, layout view A-34
- overriding DISPLAY value 2-3
- over-the-block resources, placement and routing 11-4, 11-22, 11-33, 11-40
- overview of interface logic models 11-2
- Overview panel A-27

P

- packages, IC Compiler 1-2
- pad filler
 - inserting 9-36
 - removing 9-37

- Pan to Selection command A-12
- Pan tool A-3
- panels
 - Connectivity Settings panel A-40
 - Edit Snapping Settings panel A-61
 - Flyline Settings A-39
 - Map Mode panel A-43
 - Overview panel A-27
 - Query panel A-11
 - timing analysis driver A-69
 - View Settings panel, layout views A-28
 - View Settings panel, schematic views A-24
 - Visual Mode panel A-41
- path data element table, path inspector window A-73
- path inspector
 - panels A-72
 - path schematic A-74
 - viewing path profiles A-72
- path inspector windows A-72
- path profile views
 - timing analysis A-72
 - Timing Views toolbar A-70
- path profiles
 - in path inspector A-72
 - path inspector window A-73
- path schematics A-23
 - in path inspector A-72
- path slack histogram
 - histogram views A-70
 - Timing Views toolbar A-70
- path specifications for TLUPlus 3-27
- path summaries, path inspector window A-73
- paths
 - select A-20
- photo-realistic view of design A-32
- phyopt_rp_enable_orient_opt variable 12-15
- physical bus
 - creating
 - automatic 8-48
 - manual 8-47
 - modifying 8-49
 - routing 8-51
 - physical bus, modifying 8-49
 - physical bus, routing 8-51
 - physical data
 - copying 3-24
 - validating 3-25
 - physical datapath, defined 12-1
 - physical optimization, voltage area support 13-46
 - physical signoff options
 - reporting 8-37
 - setting 8-37
 - physical synthesis
 - creating voltage areas 13-22
 - multivoltage designs 13-6
 - placing and optimizing designs 13-49
 - voltage-area-aware capabilities 13-44
- physically exclusive clocks B-30
- physopt_check_site_overlap variable 13-23
- physopt_enable_via_res_support variable 4-23
- physopt_rp_enable_orient_opt variable 12-14
- physopt_use_bit_stacked_placement variable 12-29
- pin guides
 - creating objects A-62
- place_opt command 4-16, 11-33, 13-4, 13-6, 13-38, 13-44, 13-45, 14-2, 14-13, 14-16
- placement
 - incremental 4-18
 - magnet 4-12
 - refining 4-42
 - statistics 8-35
- placement and optimization 1-6
 - customizing 4-16
 - multivoltage 13-49
 - physical synthesis 13-49
- placement area utilization 4-24
- placement blockages
 - creating 4-5

- creating objects A-62
- keepout 11-22
- plan groups
 - creating objects A-62
- platforms 1-2
- postroute extraction 8-57
- power and ground net associations, voltage
 - area support 13-47
- power domains
 - definition 13-2
 - multivoltage designs 13-2, 13-52
 - voltage areas 13-4
- power optimization
 - dynamic 6-5
 - power-aware placement 6-2
 - reporting options 6-2
- power statistics 4-31
- power-aware placement 6-2
 - switching activity information 6-3
- preferences
 - saving 2-31
 - setting 2-31
- preferred routing direction
 - reporting 8-5
 - setting 8-5
- prerequisites, routing 8-2
- preroute hold fixing, multicorners-multimode 14-16
- preroute RC estimation 4-20
- preroute scaling factors, setting 4-22, 4-23
- prerouting special nets 8-19
- preview selection A-5
- previewing commands in dialog boxes 2-28
- PrimeRail interface 8-59
- process scaling factors 8-58
- process_particle_probability_file command 9-27
- propagate_constraints command 13-47
- properties, viewing and editing for selected object A-21

- psynopt command 4-18, 13-4, 13-45, 14-2, 14-13
 - on_route option 14-13

Q

- Query panel A-11
- Query tool A-3, A-11
- querying objects A-11
- queuing licenses 2-50
- quick flow 1-8
- quitting IC Compiler 2-10

R

- RC calculations, preparing 3-26
- RC estimation 4-20
 - coefficients, specifying 4-22
 - scaling factors, setting 4-22
 - via resistance 4-23
- RC scaling, multicorners-multimode 14-14
- read_cmp_thickness command 9-44
- read_def command 3-23
- read_drc_error_file command 8-44
- read_floorplan command 3-23
- read_parasitics command 3-28
- read_saif command 6-3
- read_sdc command 3-29
- read_sdf command 3-28
- read_verilog command 3-21
- reading
 - DEF file 3-23
 - Verilog 3-21
- real metal fill extraction 9-38
- reapply edits A-57
- reapplying edits in layout view A-62
- recalling commands 2-43
- redirect command 2-44
- redirect operator (>) 2-44

- redirect operator (>>) 2-44
- redirecting command output 2-43
- redo command A-57
- redo edits A-55, A-62
- redrawing the active view A-17
- reference control file
 - syntax 3-11
 - writing 3-17
- reference, clock tree
 - specifying 7-18
- refine_placement command 4-42
- refreshing the active view A-17
- related pin (for data check) B-39
- relative coordinates A-5
- relative placement
 - aligning by pins 12-15
 - anchoring at a specified location 12-11
 - cell placement using bounds 12-38
 - changing specific items within a group 12-52
 - commands 12-55
 - compression 12-28
 - considerations for using 12-8
 - flow 12-5
 - group
 - adding items to 12-13
 - adding or removing items from in GUI A-45
 - creating 12-9
 - defined 12-9
 - extracting 12-53
 - ordering extracted 12-54
 - renaming 12-51
- hierarchical group
 - adding 12-21
 - exploring in GUI A-45
 - for instantiation 12-24
 - included, defined 12-21
 - instantiated, defined 12-21
 - syntax to include 12-22
 - syntax to instantiate 12-24
- hierarchy view A-45
 - in a design containing obstructions 12-35, 12-36
- keepout
 - adding 12-32
 - adding or removing in GUI A-45
 - syntax to add to group 12-33
- leaf cell
 - adding or removing in GUI A-45
 - specifying orientation 12-14
 - syntax to add to group 12-14
- limitations 12-4
- methodology 12-6
- modifying a group 12-48
- modifying information 12-47
- orientation optimization
 - defined 12-14
 - directing 12-14
- positions for data 12-9
- preserving during optimization 12-35
- relative placement information, ignoring 12-46
- removing 12-46
- removing items from a group 12-51
- required inputs 12-8
- sample script 12-7
- straddling 12-26
- ungrouping 12-29
- uniquifying 12-31
- voltage area support 13-48
- writing information to script file 12-45
- relative placement groups
 - exploring in GUI A-45
 - relative placement hierarchy view A-45
 - removing in GUI A-45
 - removing items from in GUI A-45
- relative placement hierarchy view A-45
- reliability analysis 8-59
- remove_all_spacing_rules command 4-11
- remove_annotations command 3-28
- remove_bounds command 4-9
- remove_buffer_tree command 4-15
- remove_clock command B-19

remove_clock_groups command B-36
 remove_clock_tree command 7-46
 remove_clock_tree_exceptions command
 -dont_buffer_nets 7-17
 -dont_size_cells 7-17
 -dont_touch_subtrees 7-16
 -exclude_pins 7-12
 -float_pins 7-15
 -non_stop_pins 7-11
 -stop_pins 7-16
 remove_data_check command B-44
 remove_filler_with_violation command 9-33
 remove_from_rp_group command 12-51
 remove_ideal_network command 3-29
 remove_keepout_margin command 4-5
 remove_level_shifters command 13-32
 remove_license command 2-51
 remove_net_routing_layer_constraints
 command 4-21
 remove_objects command A-58
 remove_placement_blockage command 4-7
 remove_power_domain command 13-53
 remove_power_net_info command 13-53
 remove_preferred_routing_direction command
 8-6
 remove_route_by_type command 8-11
 remove_route_guides command 8-5
 remove_rp_group_options command 12-46
 remove_rp_groups command 12-46
 remove_sdc command 3-29
 remove_stdcell_filler command
 -pad 9-37
 -stdcell 9-33
 -tap 9-6
 remove_target_library_subset command 13-8
 remove_voltage_area command 13-30
 remove_well_filler command 9-36
 removing objects A-58
 report views A-25
 report, clock tree
 generating 7-79
 types 7-79
 report_all_spacing_rules command 4-11
 report_annotated_delay command 11-9
 report_area command 11-19
 report_bounds command 4-9
 report_buffer_tree command 4-15
 report_cell command 13-52
 report_clock_timing command 7-81
 report_clock_tree command 7-79
 report_constraint command 7-96, 14-23
 report_critical_area command 9-26
 report_delay_calculation command 4-29,
 13-52
 report_design command 8-35, 11-19
 report_error_coordinates command 8-2, 8-44
 report_extraction_options command 14-14
 report_hierarchy command 13-52
 report_keepout_margin command 4-5
 report_net_routing_layer_constraints
 command 4-21
 report_noise_calculation command 10-12
 report_operating_conditions command 13-52
 report_physical_signoff command 8-37
 report_placement_blockages command 4-7
 report_placement_utilization command 4-24
 report_power command 4-31
 report_power_domain command 13-53
 report_power_guide command 13-30
 report_power_net_info command 13-53
 report_power_options command 6-2
 report_preferred_direction command 8-5
 report_qor command 14-21
 report_route_opt_strategy command 8-23
 report_route_options command 8-11
 report_routing_rules command 7-27
 report_scan_chain command 5-11
 report_signal_em command 10-18

- report_target_library_subset command 13-8, 13-52
- report_thresold_voltage_group command 6-4
- report_timing command 4-27, 7-96, 13-52
- report_timing -scenario command 14-21
- report_timing_derate command 3-31
- report_tlu_plus_files command 3-27, 14-24
- report_units command 3-16, 10-16, 10-17
- report_voltage_area command 13-30, 13-52
- reporting
 - cell placement statistics 8-35
 - multivoltage designs 13-51
 - routing and optimization strategy 8-23
 - routing options 8-11
 - routing statistics 8-35
 - threshold voltage 6-4
- reports
 - saving A-25
 - viewing A-25
- reset_clock_tree_references command 7-19
- reset_design command 3-29
- reset_timing_derate command 3-31
- resize_object command A-57
- resizing objects A-57
- reusing commands 2-27
- reverse edits A-57
- reversing edits in layout view A-62
- rotate object commands A-60
- rotate_objects command A-60
- rotating objects A-60
- routability, checking 8-2
- route guide
 - creating 8-3
 - creating objects A-62
 - finding 8-4
 - keepout 11-22
 - removing 8-5
- Route tool 8-45, A-58
- route_detail command 8-31
- route_global command 8-30
- route_group command 8-19
- route_opt command 13-44, 14-2, 14-13, 14-14, 14-15, 14-16
- route_search_repair command 8-32
- route_spreadwires command 9-28, 9-29
- route_track command 8-31
- route_zrt_eco command 8-33
- routing
 - detail 8-31
 - ECO 8-33
 - global 8-29
 - interactive
 - multiple nets 8-51
 - physical bus 8-51
 - single net 8-45
 - postroute extraction 8-57
 - prerequisites 8-2
 - running individual steps 8-29
 - special nets 8-19
 - track assignment 8-31
 - verifying 8-35
- routing and optimization strategy
 - reporting 8-23
 - setting 8-22
- routing direction, setting preferred 8-5
- routing layers, clock tree
 - default 7-31
 - setting 7-31
- routing options, setting 8-11
- routing rule
 - clock tree
 - default 7-27
 - nondefault 7-27
 - defining 7-27
 - nondefault, defined 7-27
 - reporting 7-27
- routing rules (nondefault), setting 8-6
- routing statistics 8-35
- routing tracks, creating objects A-62
- routing type
 - removing 8-11

- setting 8-11
- rp_group_inclusions command 12-44
- rp_group_instantiations command 12-44
- rp_group_references command 12-44
- Ruler tool A-3, A-15
- run_router command 13-45

S

- SAIF file 6-3
- Save Design 2-31
- save_mw_cel command 3-33
- save_mw_cel -scenario command 14-13
- save_mw_cel_lib_setup variable 3-3, 3-27
- saving
 - design using Milkyway 3-33
 - designs 1-7, 2-31
 - log messages 2-25
 - preferences 2-31
 - reports A-25
 - transcript of session commands 2-25, 2-27
- SBPF
 - reading 3-28
 - writing 3-34
- scaling factors
 - process 8-58
- scaling factors, preroute, setting 4-22, 4-23
- scan chains
 - partitioning
 - default 5-9
 - horizontal 5-9
 - vertical 5-10
 - reporting 5-11
 - viewing 5-12
- SCANDEF file, defined 5-4
- scenario definition
 - clock tree synthesis 7-56
 - multicorner-multimode 14-2
- scenario reduction, multicorner-multimode 14-15
- Schematic View panel, path inspector window A-74
- schematic views
 - design schematics A-23
 - path schematics A-23
 - refreshing A-17
 - symbol view A-24
- Schematics toolbar A-24
- screenshot of layout view A-36
- scripts 2-8
 - command abbreviations in 2-41
 - executing at startup 2-3
- SDC
 - reading 3-29
 - writing 3-34
- SDF files, reading 3-28
- search and repair, filling notches and gaps during 9-43
- search_path variable 3-27
- searching the transcript 2-25
- select by name dialog box A-18
- select timing paths A-20
- selected objects, following in graphic views A-14
- selecting objects in graphic views A-6
- selection preview A-5
- Selection Tool A-3
- Selection tool A-6
- session log 2-25
- set_active_scenarios command 14-12, 14-15
- set_ahfs_options command 4-14
- set_always_on_strategy command 13-38
- set_attribute command 13-39, 13-40
- set_clock_gating_check command B-47
- set_clock_groups command B-29
- set_clock_latency command B-21
- set_clock_transition command B-25
- set_clock_tree_exceptions command
 - dont_buffer_nets 7-17
 - dont_size_cells 7-17

- dont_touch_subtrees 7-16
- exclude_pins 7-12
- float pin options 7-13, 7-44
- non_stop_pins 7-11
- stop_pins 7-15
- set_clock_tree_options command
 - layer_list 7-31
 - max_buffer_levels 7-26
 - max_capacitance 7-24
 - max_fanout 7-24
 - max_transition 7-24
 - routing_rule 7-27
 - target_early_delay 7-26
 - target_skew 7-26
 - use_default_routing_for_sinks 7-28
- set_clock_tree_references command 7-18
- set_clock_uncertainty command B-23
- set_congestion_options command 4-7
- set_cts_scenario command 7-56, 14-17
- set_data_check command B-40
- set_delay_calculation command 3-32
- set_delay_estimation_options command 4-22, 4-23
- set_disable_clock_gating_check command B-48
- set_distributed_route command 8-15
- set_extraction_options command 8-58, 9-38, 14-14
- set_fix_hold command 14-16
- set_fix_hold_options command 14-16
- set_gui_stroke_bindings command A-38
- set_gui_stroke_preferences command A-38
- set_ignored_layers command 8-8
- set_inter_clock_delay_options command 7-53
- set_keepout_margin command 4-4, 11-24
- set_left_right_filler_rule command 9-32
- set_level_shifter_strategy command 13-32, 13-34
- set_level_shifter_threshold command 13-32, 13-33, 13-34
- set_lib_cell_spacing_label command 4-10, 4-11
- set_max_leakage_power command 14-4
- set_min_library command 3-3, 3-30, 14-9
- set_mw_lib_reference command 3-17
- set_mw_technology_file command 3-16
- set_net_aggressors command 8-10
- set_net_routing_layer_constraints command 4-21
- set_operating_conditions command 3-30, 13-36, 13-47
- set_operating_conditions command 14-2, 14-6, 14-7
- set_optimize_dft_options command 5-10
- set_physical_signoff command 8-37
- set_physical_signoff_options command 9-41
- set_power_guide command 13-30
- set_power_net_to_voltage_area command 13-47
- set_preferred_routing_direction command 8-5
- set_route_opt_strategy command 8-22
- set_route_options command 8-11
- set_route_type command 8-11
- set_rp_group_options command 12-48
- set_si_options command 3-33, 13-48, 14-15
- set_spacing_label_rule command 4-10, 4-11
- set_switching_activity command 6-3
- set_target_library_subset command 13-6, 13-8
- set_timing_derate command 3-31
- set_tlu_plus_files command 3-27, 9-38, 14-4, 14-12
- setting preferences 2-31
- setting routing types 8-11
- setting the current design 2-33
- shapes
 - creating net shapes A-63
 - creating user shapes A-63
- shell command line
 - copying commands to command line 2-5

- entering commands 2-4, 2-41
- shell commands 2-4
- shell interface 1-4
- shielding nets 8-33
- shortcut keys 2-14
- signal electromigration
 - analysis repair file 10-21
 - performing analysis 10-18
 - preparing data for analysis 10-15
- signal integrity
 - defined 10-1, 10-2
 - options
 - commands affected 10-3
 - setting 10-3
- signal integrity flow
 - ILMs 11-3, 11-6, 11-36
 - voltage area support 13-48
- signoff_drc command 8-36
- signoff_metal_fill command 9-41
- single net, routing 8-45
- size_only attribute 13-35, 13-36, 13-37
- skew
 - balancing between clocks 7-52
 - global 7-26
- snapping options for layout editing A-61
- snapping, enable or disable A-57
- source latency B-20
- sparc64 platform 1-2
- special nets, prerouting 8-19
- SPEF (Standard Parasitic Exchange Format)
 - files
 - reading 3-28
 - writing 3-34
- Split tool A-58
- split_objects command A-58
- spreading object commands A-59
- standard cell filler
 - inserting 9-31
 - removing 9-33

- removing fillers that have routing rule
 - violations 9-33
- rules, defining 9-32
- standard cell row, adding end caps 9-34
- standard-cell filler cell insertion, voltage area
 - support 13-47
- starting IC Compiler 2-2
 - executing commands 2-3
 - executing scripts 2-3
- starting icc_shell 2-2
- starting the GUI 2-2
 - listing startup options 2-3
- startup options, -help list 2-3
- static noise
 - defined 10-2
 - reporting 10-11
 - violations 10-11
 - visual mode 10-11
- status bar 2-20, A-5
- Status panel, path inspector window A-73
- straddling, in relative placement, defined 12-26
- Stretch Wire tool A-58
- stretch_wire command A-58
- stretching and moving objects A-58
- stretching wires A-58
- stroke activated commands A-36
- stroke commands A-36
- supported platforms 1-2
- Suse platforms 1-2
- switching activity
 - file 6-3
 - power-aware placement 6-2
- symbol views A-24
- synchronous clocks B-27
- SYNOPSIS variable 2-2

T

- Tab panel, path inspector window A-73
- tap cell

- inserting 9-4
 - removing 9-6
- tap cell array
 - adding 9-3
- target library subsetting, multivoltage designs 13-8
- target_library variable 13-8
- task mode, layout window menus 2-30
- Tcl (tool command language)
 - scripts 2-8
 - used by IC Compiler 1-4
- TclPro
 - bytecode-compiled files 2-48
 - limitations 2-48
 - procomp 2-48
- technology (.tf) file
 - changing 3-16
 - defined 3-9
 - metal density rules 9-37
 - specifying 3-10
 - writing 3-17
- technology files
 - editing 3-16
- terminals, creating objects A-63
- terminating commands 2-7
- text, creating objects A-63
- threshold voltage 6-4
- tie cell insertion 4-12
- timing analysis
 - histograms A-70
 - path inspector windows A-72
 - preparing 3-26
 - timing analysis driver A-69
 - timing analysis window A-68
 - Timing Views toolbar A-70
- timing analysis driver A-69
- timing constraints
 - removing 3-29
 - setting 3-29
- timing goals, clock tree 7-26
- timing information
 - maximum 3-3, 3-30
 - minimum 3-3, 3-30
- timing libraries 1-6
 - derating factors 3-31
- timing paths
 - analyzing A-68
 - endpoint slack histogram A-70
 - path slack histogram A-70
 - selecting A-20
 - timing analysis driver A-69
 - viewing in path inspector A-72
- timing report 4-27, 7-96
 - crosstalk analysis 10-11
- Timing Views toolbar A-70
- TLUPlus
 - map file 3-26
 - models 3-26
 - reporting files 3-27
 - specifying paths 3-27
 - validating files 3-27
- toggle rate 6-2
- toggle snapping A-57
- tool command language (Tcl)
 - scripts 2-8
 - used by IC Compiler 1-4
- tool command language (Tcl) mode 1-4
- toolbars 2-19
 - Analysis toolbar A-38
 - Browsers 7-84, A-64
 - Cells List 2-33
 - console 2-24
 - Edit toolbar A-55
 - Expand Browser 7-84, A-66
 - File toolbar 2-31
 - Highlight toolbar A-9
 - Mouse Tool options toolbar A-5
 - Mouse Tools toolbar A-3
 - Schematics A-24
 - Timing Views A-70
 - View Zoom/Pan toolbar A-12
 - Zoom and Pan History toolbar A-14

tools A-58

- Copy tool A-57
- creating objects A-62
- Delete tool A-58
- Edit tool (move objects) A-57
- Edit tool (resize objects) A-57
- Highlight tool A-8
- layout editing A-55
- left mouse button tools A-3
- Pan tool A-3
- Query tool A-3, A-11
- Route tool A-58
- Ruler tool A-3, A-15
- Selection Tool A-3
- Selection tool A-6
- Split tool A-58
- Stretch Wire A-58
- Window Stretch tool A-58
- Zoom In tool A-3
- Zoom Out tool A-3

top-level usage 11-32

top-level windows 2-18

track assignment 8-31

transcript of session commands, saving 2-25

transition time B-25

TrueVue mode A-32

U

uncertainty, clock B-23

undo command A-57

undo edits A-55, A-62

unfix objects A-60

unfix objects for edit A-57

Unified Power Format (UPF) 13-9

uniquify_fp_mw_cel command 3-21

units

- default 3-3

- saving with design 3-33

- saving with SDC 3-34

UNIX

- DISPLAY environment variable 2-2

- shell 2-3

- SYNOPSIS variable 2-2

UNIX operator

- append (>>) 2-44

- redirect (>) 2-44

unlocking objects for edit A-57

unset_power_guide command 13-30

update_bounds command 4-9

update_clock_latency command 7-70

update_voltage_area command 13-25, 13-26

user grid, displaying or hiding A-28

user interfaces 1-4

user shapes, creating objects A-63

using aliases 2-43

utilization 4-24

V

variables

- auto_restore_mw_cel_lib_setup 3-4, 3-27

- collection_result_display_limit 12-44

- cts_do_characterization 7-78

- cts_dont_touch_subtrees 7-16

- cts_enable_rc_constraints 7-39

- cts_fix_clock_tree_sinks 7-61

- cts_move_clock_gate 7-61

- cts_rc_relax_factor 7-39

- cts_use_debug_mode 7-78

- cts_use_lib_max_fanout 7-25

- cts_use_sdc_max_fanout 7-25

- DISPLAY 2-2

- ilm_allow_all_rotations 11-30

- ilm_disable_ilm_checks 11-10

- mcm ENABLE_high_capacity_flow 14-16

- mcm ENABLE_high_capacity_effort_level 14-16

- physopt_check_site_overlap 13-23

- physopt_enable_via_res_support 4-23

- physopt_rp_enable_orient_opt 12-14, 12-15

- physopt_use_bit_stacked_placement 12-29

- save_mw_cel_lib_setup 3-3, 3-27
- search_path 3-27
- SNPS_MAX_QUEUEUTIME 2-50
- SNPS_MAX_WAITTIME 2-50
- SNPSLMD_QUEUE 2-50
- SYNOPSIS 2-2
- target_library 13-8
- verify_drc command 8-40
- verify_route command 8-43
- Verilog
 - reading 3-21
 - writing 3-33
- via arrays, creating A-63
- via resistance
 - estimating 4-23
 - scaling 4-23
- vias
 - creating A-63
 - inserting redundant 9-30
- View Settings panel
 - design overlays on layout view A-34
 - layer properties in layout view A-30
 - object properties in layout view A-29
 - object properties in schematic view A-24
 - save or load settings A-31
 - using A-28
- view windows 2-21
- View Zoom/Pan toolbar A-12
- view, FILL 9-42
- viewing
 - man pages 2-14
 - object properties A-21
 - reports A-25
 - scan chains 5-12
- views
 - clock tree fanout browser A-65
 - design schematics A-23
 - histograms A-70
 - layout A-2, A-26
 - path inspector window A-72
 - path profiles A-72
 - path schematics A-23
 - symbol A-24
- virtual clock B-4
- virtual hierarchy routing, voltage area support 13-45
- Visual Mode command A-38
- Visual Mode panel A-41
- visual modes
 - Analysis toolbar A-38
 - delta delay 10-11
 - displaying A-41
 - static noise 10-11
 - Visual Mode panel A-41
- voltage area aware capabilities
 - maximum net length optimization 13-45
 - virtual hierarchy routing 13-45
- voltage areas
 - creating 13-22, 13-24
 - creating for multivoltage designs 13-24
 - creating objects A-62
 - guard bands, using 13-26
 - macro cells 13-29
 - multivoltage designs 13-4, 13-22
 - power domains 13-4
 - power domains, associating 13-24
 - removing 13-30
 - reporting 13-30
 - updating 13-25
- voltage-area-aware capabilities 13-44
 - automatic high-fanout synthesis 13-45
 - clock tree synthesis and optimization 13-46
 - global routing 13-46
 - maximum net length optimization 13-45
 - multicorner-multivoltage 13-49
 - physical optimization 13-46
 - power and ground nets, associating 13-47
 - signal integrity flow 13-48
 - standard-cell filler cell insertion 13-47
 - using interface logic models 13-47
 - using relative placement 13-48
 - virtual hierarchy routing 13-45

voltage-area-based optimization 13-46
voltage-area-based optimization 13-46

W

well filler
 inserting 9-34
 removing 9-36
Window Stretch tool A-58
window_stretch command A-58
windows
 layout 2-29
 main 2-3, 2-22
 menu bar 2-19
 status bar 2-20
 timing analysis A-68
 toolbars 2-19
 top-level 2-18
 view windows 2-21
 working environment 2-18
wire spreading 9-28
wires
 creating with layout editor 8-54
 stretching A-58
working environment 2-18
write_def command 3-33

write_mw_lib_files command 3-17
write_parasitics command 3-34, 8-57, 14-13
write_rp_groups command 12-45
write_sdc command 3-34
write_stream command 3-34
writing
 DEF file 3-33
 Verilog 3-33

Y

yield, optimizing
 displaying critical area heat maps 9-28
 performing wire spreading 9-28
 reporting critical areas 9-26

Z

Zoom and Pan History toolbar A-14
Zoom Fit All command A-12
Zoom Follows Selection command A-12
Zoom In 2x command A-12
Zoom In tool A-3
Zoom Out 2x command A-12
Zoom Out tool A-3
Zoom to Selection command A-12