

IC Compiler

Design Planning

User Guide

Version F-2011.09-SP2, December 2011

SYNOPSYS®

Copyright Notice and Proprietary Information

Copyright © 2011 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, AEON, AMPS, ARC, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, Chipidea, CHIPit, CODE V, CoMET, Confirma, CoWare, Design Compiler, DesignSphere, DesignWare, Eclypse, Formality, Galaxy Custom Designer, Global Synthesis, HAPS, HapsTrak, HDL Analyst, HSIM, HSPICE, Identify, Leda, LightTools, MAST, MaVeric, METeor, ModelTools, NanoSim, NOVeA, OpenVera, ORA, PathMill, Physical Compiler, PrimeTime, SCOPE, SIVL, SNUG, SolvNet, Sonic Focus, STAR Memory System, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YieldExplorer are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, ASAP, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, EMBED-IT!, Encore, EPIC, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, plus HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Intelli, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Macro-PLUS, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, ORAengineering, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, RippledMixer, Saturn, Scirocco, Scirocco-i, SiWare, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCSI, VHDL Compiler, VMC, and Worksheet Buffer 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.

Copyright Statement for the Command-Line Editing Feature

Copyright © 1992, 1993 The Regents of the University of California. All rights reserved. This code is derived from software contributed to Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright Statement for the Line-Editing Library

Copyright © 1992 Simmule Turner and Rich Salz. All rights reserved.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

1. The authors are not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
4. This notice may not be removed or altered.

Contents

What's New in This Release	xx
About This Guide	xx
Customer Support	xxiii
1. Introduction to Design Planning	
Overview	1-2
Why Plan the Design?	1-3
Using a Hierarchical Methodology	1-4
Hierarchical Methodology for Design Planning	1-4
Design Partitioning	1-6
Hierarchical Floorplanning	1-8
Hierarchical Timing Closure	1-9
Using the GUI to Manage Tasks in Parallel	1-11
2. Creating a Floorplan	
Supported Types of Floorplans	2-2
Channeled Floorplans	2-2
Abutted Floorplans	2-2
Narrow-Channel Floorplans	2-3
Preparing the Design	2-4
Connecting Power and Ground Ports	2-4
Adding Power, Ground, and Corner Cells	2-4
Setting the I/O Pad Constraints	2-5

Setting the Pin Constraints	2-7
Saving the Pin and Pad Constraints	2-7
Reading an Existing Pad and Pin Constraints File	2-8
Reporting the Pad and Pin Constraints	2-8
Removing the Pad and Pin Constraints	2-9
Initializing the Floorplan	2-9
Creating a Rectangular Floorplan	2-9
Creating a Simple Rectilinear Floorplan	2-15
Creating a Complex Rectilinear Floorplan	2-20
Refining the Floorplan	2-21
Adjusting the Floorplan	2-21
Manually Modifying the Floorplan	2-23
Defining the Die Area	2-23
Removing Cell Rows	2-24
Adding Cell Rows	2-24
Defining the Wire Tracks	2-26
Adjusting the I/O Placement	2-27
Saving the Floorplan Information	2-28
Writing the Floorplan File	2-29
Writing the Floorplan to a DEF File	2-30
Floorplan Physical Constraints in Design Compiler	2-31
Extracting Physical Constraints	2-31
Exporting Physical Constraints	2-31
Reading In an Existing Floorplan	2-32
Reading DEF Files	2-32
Reading Floorplan Files	2-33
Copying a Floorplan From Another Design	2-33
3. Automating Die Size Exploration	
MinChip Design Flow	3-2
Preparing the Design for MinChip Technology in IC Compiler	3-4
Using the Estimate Area GUI	3-8
Using the estimate_fp_area Command and Options	3-18
Using MinChip Technology With Multivoltage Designs	3-25

4. Handling Black Boxes

Black Box Flow	4-2
Reading Netlists Containing Black Boxes	4-2
Importing Black Boxes	4-4
Sizing Black Boxes	4-5
Specifying the Black Box Boundary	4-5
Specifying a Rectangular Black Box Boundary	4-6
Specifying a Rectilinear Black Box Boundary	4-6
Sizing Black Boxes by Content	4-6
Defining the Size by Hard Macro Content	4-6
Defining the Size by Gate Equivalents	4-7
Fixing the Black Box Shape	4-7
Configuring Black Boxes	4-8
Creating a FRAM View	4-8
Constraining Routing Over Black Boxes	4-8
Setting Black Box Status	4-8
Converting Physical Black Boxes to Logical Black Boxes	4-9
Creating Quick Timing Models	4-9
Writing a Quick Timing Model to a .db File	4-12
Loading a Quick Timing Model	4-12
Propagating Timing Information to Soft Macros	4-12
Creating Unique Cell Instances	4-13

5. Using the On-Demand-Loading Flow for Large Designs

Overview of On-Demand-Loading Design Planning Flow.	5-2
Creating On-Demand Netlists	5-4
Displaying Plan Group Information in the GUI	5-5
Using Distributed Processing in the On-Demand-Loading Flow	5-6
Using Multicorner-Multimode in the On-Demand-Loading Flow	5-8
Features Not Supported by the <code>create_on_demand_netlist</code> Command.	5-10
Removing On-Demand Netlists	5-10
Reporting and Querying Information About On-Demand Netlists.	5-10
Cell Placement in the On-Demand-Loading Flow.	5-11

Timing Budgeting Using On-Demand Netlists	5-12
Committing Plan Groups Within the On-Demand-Loading Flow.	5-12
6. Virtual Flat Placement	
Virtual Flat Placement Overview.	6-2
Virtual Flat Placement Command.	6-3
Congestion-Driven Placement	6-6
Timing-Driven Placement	6-6
Hierarchical Gravity	6-6
Exploration Mode	6-7
Simultaneous Placement and Pin Assignment	6-10
Virtual Flat Placement Guidelines	6-10
Virtual Flat Placement Strategy Options.	6-11
Macro-Related Options	6-12
Automatic Grouping of Macros Into Arrays	6-13
Fixing Macros in Place	6-13
Macro Orientation	6-14
Macro Setup Only	6-14
Macro Minimum Distance	6-15
Macros on Edge	6-16
Pin Routing Aware	6-18
Sliver Size	6-18
Snap Macros to User Grid	6-19
Congestion-Related Options	6-19
Net Weighting Options.	6-20
Miscellaneous Options.	6-21
Reset to Default Settings	6-22
Block Constraint File.	6-22
Flip-Chip Driver Placement.	6-23
Automatic Hierarchy Detection	6-23
Hierarchical Gravity	6-24
Honor Multivoltage Cells.	6-24
Spread Spare Cells.	6-25
Virtual In-Place Optimization	6-25
Hard Macro Constraints	6-26
Macro Array Constraints	6-27
Macro Placement Options	6-31

Relative Location Constraints	6-35
Blockages, Margins, and Shielding.....	6-38
Placement Blockages	6-38
Keepout Margins	6-40
Relative Placement Groups	6-42
Placing Relative Placement Groups	6-44
Using the Default Relative Placement Flow	6-44
Using the Relative Placement Cells Flow	6-45
Propagating Relative Placement Groups by Commit and Uncommit	6-46
Committing Relative Placement Groups	6-46
Uncommitting Relative Placement Groups	6-47
Design Flow for Hierarchical Relative Placement Groups	6-47
Placement Evaluation	6-48
Placement Quality of Results Report.....	6-49
Evaluating Macro Placement.....	6-50
Total Wire Length Estimate	6-52
Floorplan Editing.....	6-52
Removing Cell Placement	6-54
Packing Hard Macros Into an Area	6-55
Manually Adjusting the Hard Macros	6-56
Using Constraints With the Core and Die Editing Commands.....	6-57
Voltage Area Planning	6-57
Supported Voltage Area Physical Boundary Scenarios.....	6-61
Updating Voltage Areas in a Design	6-62
Removing Voltage Areas	6-62
Placing Multiply Instantiated Modules	6-62
Identifying Multiply Instantiated Modules.....	6-63
Removing the Multiply Instantiated Module Property	6-63
Identifying Multiply Instantiated Module Plan Groups	6-64
Shaping Multiply Instantiated Module Plan Groups.....	6-65
Placing and Analyzing Multiply Instantiated Module Plan Groups.....	6-65
Copying the Cell Placement of a Plan Group	6-66
Copying Placement Blockages and Boundaries.....	6-67
Restoring the Placement of Cells in Plan Groups	6-67
Flipping Multiply Instantiated Module Plan Groups	6-68
Horizontal Requirements for Plan Group Locations	6-69

Vertical Requirements for Plan Group Locations	6-69
QoR Analysis of Multiply Instantiated Module Plan Groups	6-69
7. Plan Groups	
Creating Plan Groups	7-2
Removing the Plan Groups	7-4
Adding Padding to Plan Groups	7-4
Removing the Plan Group Padding	7-5
Adding Block Shielding to Plan Groups or Soft Macros	7-6
Removing Module Block Shielding	7-7
Analyzing and Manipulating the Hierarchy	7-8
Opening the Hierarchy Browser.....	7-8
Exploring the Hierarchical Structure	7-10
Manipulating the Hierarchy	7-10
Merging the Hierarchy	7-10
Flattening the Hierarchy	7-17
Automatically Placing and Shaping Objects in a Design Core	7-18
Controlling the Placement and Shaping of Objects.....	7-23
Using Relative Placement Constraints.....	7-25
Plan Group Utilization	7-26
Physical-Only Cells in Plan Groups	7-29
DFT-Aware Design Planning Flow	7-29
Generating and Using SCANDEF	7-31
Performing Plan Group-Aware Scan Chain Optimization	7-31
Using Plan Group-Aware Repartitioning	7-32
Committing Physical Hierarchy Changes After Scan Chain Optimization	7-32
Checking the SCANDEF Data Against the Netlist.....	7-33
8. Performing Power Planning	
Performing Power Network Synthesis on Single-Voltage Designs	8-2
Saving the Design	8-3
Defining Logical Power and Ground Connections	8-3
Applying Power Rail Constraints	8-3
Applying Power Ring Constraints.....	8-5

Creating Virtual Pads	8-5
Synthesizing the Power Network	8-7
Committing the Power Plan	8-8
Performing Power Network Synthesis on Multivoltage Designs	8-9
Saving the Design	8-10
Selecting a Voltage Area and Specifying Synthesis Constraints	8-10
Setting Layer Constraints	8-12
Setting Ring Constraints	8-13
Setting Global Constraints	8-15
Synthesizing the Power Network	8-16
Performing Template-Based Power Planning	8-17
Defining Power Plan Regions	8-17
Creating a Power Plan Region	8-18
Retrieving Power Plan Regions	8-18
Reshaping Power Plan Regions	8-19
Reading and Writing Power Plan Regions	8-21
Creating the Power Ring Template File	8-22
Creating the Side Section	8-22
Creating the Advanced Rules Section	8-23
Creating the Power Ring Strategy	8-24
Reporting and Removing Power Ring Strategies	8-25
Compiling the Power Rings	8-26
Creating the Power Mesh Template File	8-27
Creating the Layer Section	8-27
Creating and Using Parameters in a Template File	8-29
Creating the Advanced Rules Section	8-30
Creating the Power Mesh Strategy	8-35
Reporting and Removing Power Mesh Strategies	8-37
Compiling the Power Mesh	8-37
Template-Based Power Plan Examples	8-39
Creating Rings Around Voltage Areas	8-40
Aligning Rings With Standard Cell Rails	8-42
Creating Rings Around a User-Specified Polygon	8-44
Creating Rings Around Macro Cells	8-45
Creating Corner Bridging	8-45
Creating Partial Power Rings	8-47
Creating Power Rings with Varying Segment Widths	8-48
Creating a Power Plan Strategy for a Multivoltage Design	8-49
Creating Power Straps on Macro Cells	8-51

Creating a Power Grid in an Abutted Voltage Area	8-52
Creating Extension Finger Connections	8-54
Creating Pad Ring Connections	8-56
Inserting Straps into the Channel Area	8-56
Align Power Straps with Power Switches	8-58
Nonuniform Power Straps	8-60
Creating Power-Switch Arrays and Rings for Multithreshold-CMOS Designs	8-62
Power Switch Overview	8-62
Selecting the Proper Power-Switch Strategy	8-64
Creating a Power-Switch Array	8-65
Creating a Power-Switch Ring	8-67
Exploring Different Switch Configurations	8-70
Optimizing Power Switches	8-71
Connecting Power Switches	8-71
Analyzing the Power Network	8-72
Performing Power Network Analysis	8-72
Creating Connectivity Views	8-74
Performing Power Analysis With Switching Activity Information	8-74
Annotating Switching Activity	8-75
Performing Power Network Analysis for Each Cell Instance	8-76
Viewing the Analysis Results	8-76
Displaying the Voltage Drop Map	8-77
Displaying the Resistance Map	8-78
Displaying the Electromigration Map	8-79
Displaying the Instance Voltage Drop Map	8-79
Displaying the Instance Power Map	8-79
Displaying the Instance Power Density Map	8-79
9. Performing Clock Planning	
Setting Clock Planning Options	9-2
Reporting Clock Planning Options	9-3
Removing Clock Planning Options	9-3
Performing Clock Planning Operations	9-3
Generating Clock Tree Reports	9-4
Generating Clock Network and Source Latency for Each Clock Pin of Each Plan Group	9-5

Creating a Virtual Clock for I/O Paths	9-6
Using Multivoltage Designs in Clock Planning	9-7
Interpreting Level-Shifter Cells as Anchor Cells During Clock Planning	9-7
Performing Plan-Group-Aware Clock Tree Synthesis	9-8
Checking for Electromigration After Clock Tree Synthesis	9-8
Power Tiles	9-10
Power Budget Calculation	9-11
Extra Power Straps on Higher Layers	9-12
Supporting Abutted Floorplans in Hierarchical Clock Planning	9-13
10. Performing In-Place Timing Optimization	
Running In-Place Timing Optimization	10-2
Running Trace Mode In-Place Optimization	10-5
Reporting Trace Mode Status	10-6
Removing the Trace Mode	10-6
Using In-Place Optimization With Multivoltage Designs	10-7
Performing In-Place Optimizations Based on Pin Locations	10-7
Virtual In-Place Optimization	10-7
11. Performing Routing-Based Pin Assignment	
Setting Pin Assignment Constraints	11-2
Specifying Global Pin Assignment Constraints	11-2
Specifying Constraints for Special Conditions	11-3
Reserving Narrow Channels for Special Nets	11-3
Keeping Bus Bits Together	11-4
Reserving Space for Clock Pins	11-4
Specifying Pin Physical Design Constraints	11-4
Ignoring Existing Pin Physical Constraints	11-5
Reporting Pin Assignment Constraints	11-6
Performing Pin Assignment on Soft Macros and Plan Groups	11-7
Performing Plan-Group-Aware Global Routing	11-8
Improving the Quality of the Global Routing for Pin Assignment	11-9
Improving the Performance of the Global Router	11-10

Placing Soft Macro and Plan Group Pins	11-10
Performing Block-Level Pin Placement	11-12
Committing the Hierarchy	11-12
Adjusting the Position of Placed Pins	11-13
Aligning Soft Macro Pins	11-13
Ordering Pins Alphabetically	11-14
Removing Soft Macro Pin Overlaps	11-14
Adding Feedthrough Pins	11-15
Excluding Feedthroughs on Clock Nets	11-15
Specifying Net and Feedthrough Topology	11-16
Reporting Feedthrough Topology	11-19
Previewing Feedthrough Pins	11-20
Creating Feedthrough Pins	11-21
Performing Pin and Feedthrough Analysis	11-21
Removing Feedthrough Ports, Nets, and Buffering	11-22
Adding Feedthrough Pins on Multivoltage Designs	11-23
Setting Voltage Area Feedthrough Constraints	11-23
Creating Voltage Area Feedthroughs	11-23
Creating Feedthroughs Only on Voltage Areas	11-24
Creating Feedthroughs on Voltage Areas Coincident with Plan Groups	11-24
Creating Feedthroughs on Voltage Areas Not Coincident with Plan Groups	11-25
Reporting Voltage Area Constraints	11-25
Removing Voltage Area Constraints	11-25
Performing Pin Placement With Multiply Instantiated Modules	11-26
Setting Pin Assignment Constraints for Multiply Instantiated Modules	11-26
Performing Plan-Group-Aware Routing for Multiply Instantiated Modules	11-26
Selecting a Master Instance	11-27
Committing a Master Multiply Instantiated Module Plan Group	11-27
Using Pin Guides for Pin Placement	11-27
Creating Pin Guides	11-27
Using Pin Guides for Block-Level Off-Edge Pin Placement	11-29
Creating Pin Guides for Feedthroughs	11-30
Reporting Pin Guide Settings	11-31

Checking for Off-Edge Pin Placement	11-32
Checking Pin Assignment and Pin Alignment	11-32
Checking the Pin Assignment	11-32
Checking the Pin Alignment.....	11-33
12. Performing Timing Budgeting	
Timing Budgeting Prerequisites	12-2
Performing Pre-Budget Timing Analysis.....	12-3
Running the Timing Budgeter.....	12-6
Performing Fast Time to Budget Analysis	12-8
Checking the QoR of the Timing Budgets	12-9
Generating a Quick Timing Model From a CEL View	12-10
Generating a Quick Timing Model for the Partitions of Large Designs	12-10
Quick Timing Model Command Summary.....	12-11
Performing Timing Budgeting On Plan Groups.....	12-12
Budgeting With Multiply Instantiated Modules	12-14
Performing Budgeting on Multiple Scenarios	12-25
Performing Distributed Timing Budgeting for Multiple Scenarios	12-26
Performing Hierarchical Signal Integrity Budgeting.....	12-27
Block-Level Hierarchical Signal Integrity Flow	12-27
Top-Level Hierarchical Signal Integrity Flow	12-28
Performing Post-Budget Timing Analysis	12-28
Performing Clock Latency Budgeting	12-31
Creating Budgets to Reflect Real Clock Latencies	12-31
13. Committing the Physical Hierarchy	
Converting Plan Groups to Soft Macros	13-2
Splitting the Soft Macros	13-3
Pushing Physical Objects Down to the Soft Macro Level	13-4
Using the Margin Option for Pushing Down Objects	13-6
Pushing Down Routing in Multiply Instantiated Modules.....	13-6
Pushing Physical Objects Up to the Top Level	13-7

Handling 45-Degree Redistribution Layer Routing	13-8
Committing the Hierarchy of Plan Groups With Power Domains	13-8
Converting Soft Macros to Plan Groups	13-9

Appendix A. Using the Flip-Chip Flow

Flip-Chip Implementation Flows	A-3
Preparing the Library	A-4
I/O Drivers	A-4
Bump Cells	A-4
Creating an Initial Floorplan	A-6
Using a Die-Driven Flip-Chip Flow	A-6
Instantiating Bump Cells and Designating Matching Types	A-7
Creating a Pattern of Bump Cells in a Ring Configuration	A-7
Creating an Array of Bump Cells	A-9
Designating Matching Types for Bump Patterns	A-11
Placing Flip-Chip I/O Drivers	A-12
Defining the Location for Flip-Chip Drivers	A-12
Placing Flip-Chip Drivers	A-14
Designating a Matching Type for Flip-Chip Drivers	A-15
Assigning Nets From Bumps to I/O Drivers	A-15
Merging Multiple Flip-Chip Nets	A-17
Reporting Crossover Nets	A-17
Using a Package-Driven Flip-Chip Flow	A-18
Importing Bump Locations	A-19
Saving Bump Cell Information	A-21
Defining Flip-Chip Driver Locations	A-21
Designating the Driver Matching Type	A-22
Defining the Flip-Chip Placement Grid	A-22
Setting Options for Flip-Chip Driver Placement	A-22
Placing Flip-Chip Drivers	A-23
Routing Flip-Chip Bumps	A-23
Writing Flip-Chip Nets to a File	A-24
Setting Flip-Chip Routing Options	A-25
Creating Stacked Vias on Pad Pins	A-26
Routing Flip-Chip Nets	A-26

Analyzing Flip-Chip Routing Results	A-27
Resolving Flip-Chip Net Routing Congestion	A-28
Performing Incremental Routing and Route Optimization	A-30
Using Flip-Chip Structures in Cover Macros.	A-31
Summary of the Flip-Chip Commands	A-31

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:

<https://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 design planning flow; the companion volume, *IC Compiler Implementation User Guide*, describes the implementation and integration 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 the documentation on SolvNet at the following address:

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

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

- Design Compiler
- Milkyway Environment

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code> .
Courier bold	Indicates user input—text you type verbatim—in examples, such as <code>prompt> write_file top</code>
[]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>low medium high</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 a 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, and technical support.

To access SolvNet, go to the following address:

<https://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.

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 support case to your local support center online by signing in to SolvNet at <https://solvnet.synopsys.com>, clicking Support, and then clicking “Open A Support Case.”
- 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/GlobalSupportCenters/Pages>
- Telephone your local support center.
 - Call (800) 245-8005 from within North America.
 - Find other local support center telephone numbers at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages>

1

Introduction to Design Planning

You can use design planning and chip-level analysis capabilities to perform a feasibility analysis on designs of various sizes and complexities. IC Compiler supports both flat and hierarchical design planning to enable fast exploration of the design to reduce die size and to implement a final, optimized, and detailed floorplan.

This chapter includes the following sections:

- [Overview](#)
- [Why Plan the Design?](#)
- [Using a Hierarchical Methodology](#)
- [Using the GUI to Manage Tasks in Parallel](#)

Overview

Design planning in IC Compiler provides basic floorplanning and prototyping capabilities for flat and hierarchical designs, such as:

- Incomplete or problematic netlist handling
- Automatic die size exploration
- Black box module and cell support
- Fast placement of macros and standard cells
- Packing macros into arrays
- Plan group creation and shaping
- In-place optimization
- Prototype global routing analysis
- Hierarchical clock planning
- Pin assignment on soft macros and plan groups
- Timing budgeting
- Hierarchy conversion
- Pin assignment and refinement

Power network synthesis, applied during the feasibility phase of design planning, provides automatic synthesis of local power structures within voltage areas. Power network analysis performs voltage drop and electromigration analysis. Power network analysis can be applied to partial, complete, or power structures created by power network synthesis. Both power network synthesis and power network analysis can be used for single or multisupply designs. You can also perform low-power planning for multithreshold-CMOS designs.

MinChip technology in IC Compiler automates the exploration and identification the potential die configurations to determine the smallest routable die size for your design while maintaining the relative placement of hard macros and I/O cells. Optionally, the flow uses power network synthesis for power routing that meets voltage drop requirements and routing estimation technology to access routing feasibility.

Why Plan the Design?

Design planning is an integral part of the RTL to GDSII design process. During design planning, you assess the feasibility of different implementation strategies early in the design flow for both flat and hierarchical designs. For large designs, design planning is critical because large designs are more likely to have long interblock paths whose delays can make timing closure difficult, leading to time consuming and unpredictable tapeout schedules. As chips migrate to smaller and smaller technology nodes, design size and design complexity makes the designs more vulnerable to potential timing and power problems and therefore, more in need of careful planning early in the design flow.

In the design planning context, floorplanning is the process of sizing and placing cells and blocks in a way that makes later physical design steps more effective. Floorplanning in a hierarchical flow also provides a basis for estimating the timing of the top-level interconnect for verification. Floorplanning allocates timing budgets to each block based on the top-level timing estimation. Floorplanning can also be an iterative process that reshapes and replaces blocks, reallocates the timing budgets, and rechecks the top-level timing until an optimal floorplan is reached.

An effective floorplan helps ensure timing closure in many ways: by placing blocks so that critical paths are short, by preventing routing congestion that can lead to longer paths, by eliminating the need for routing over noise-sensitive blocks, and so on. The challenge is to create a floorplan with good area efficiency, to conserve silicon real estate and to leave sufficient area for routing.

Similarly, design planning supports power planning, including low-power design techniques that can be used in multivoltage designs and multithreshold-CMOS power switching. Power network synthesis and power network analysis enable designers to create power structures that meet voltage drop and electromigration specifications while minimizing the consumption of routing resources. The placement engine recognizes power domains and keeps together the cells of the same power domain. After you define voltage areas, power network synthesis creates power meshes for all voltage areas simultaneously.

Floorplanning also helps to reduce IR drop and electromigration problems through strategies such as placing blocks with the highest power consumption close to the periphery of the chip and preventing the concentration of blocks in any one area.

IC Compiler includes complete hierarchical design planning for both channeled and abutted layout styles. For more information, see ["Supported Types of Floorplans" in Chapter 2](#).

Using a Hierarchical Methodology

Use a hierarchical design methodology to enable a “divide and conquer” approach for large designs. By dividing the design into multiple blocks, sometimes referred to as subblocks, modules, or lower-level blocks, you can work on the blocks separately and in parallel from RTL through physical implementation. Working with smaller blocks can reduce overall runtime.

Consider using a hierarchical methodology in the following scenarios:

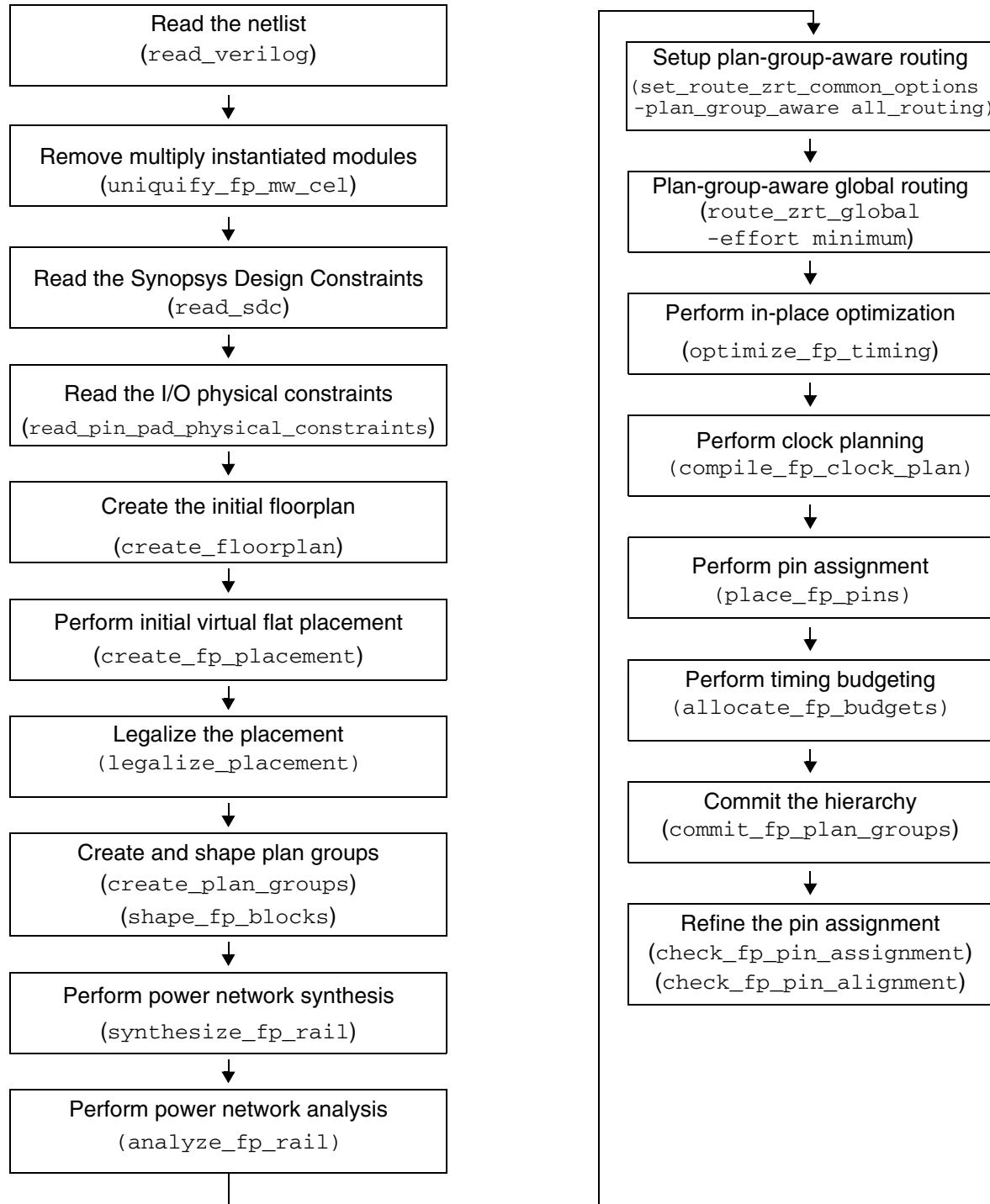
- The design is large, complex, and requires excessive computing resources to process the design.
- You anticipate that a number of blocks might have problems, which might cause the schedule to slip. Using a hierarchical methodology allows late design changes to be made to individual blocks without disturbing the rest of the design.
- The design contains hard intellectual property (IP) macros such as RAMs, or the design was previously implemented and can be converted and reused.

Hierarchical Methodology for Design Planning

After the initial design netlist is generated in Design Compiler topographical mode, you can use the hierarchical methodology for design planning in IC Compiler. Design planning is performed during the first stage of the hierarchical flow to partition the design into blocks, generate hierarchical physical design constraints, and allocate top-level timing budgets to lower-level physical blocks.

Figure 1-1 shows the hierarchical flow steps for design planning.

Figure 1-1 Hierarchical Design Planning Flow



This section includes a more detailed discussion of the following uses of design planning.

- [Design Partitioning](#)
- [Hierarchical Floorplanning](#)
- [Hierarchical Timing Closure](#)

Design Partitioning

The first step in a hierarchical design project is to determine the physical partitioning. When deciding on the physical partition of a large design, you should consider the following factors:

- Size

Ensure the size of the blocks is appropriate for the virtual flat IC Compiler implementation flow. It is easier to floorplan with blocks of similar size, so small blocks should be grouped and large blocks divided when appropriate.

- Datapaths

Partition your design using its functional units for verification and simulation purposes. Consider top-level connectivity and minimal block pin counts to avoid congestion and timing issues.

- Floorplan style

Different floorplan styles require different physical hierarchies to support them. An abutted floorplan style has no top-level logic and a channeled floorplan has either a small or large amount of top-level logic.

- Connection with the hierarchical Design Compiler topographical mode

To exchange SCANDEF information at the block level and the top level, the physical hierarchy used in Design Compiler topographical mode must also be used in IC Compiler.

During the design planning stage, a physical partition is represented by a plan group. A plan group is a physical constraint for placing the cells of the same group together physically. Each plan group represents a module in the logic hierarchy that you want to implement as a physical block and contains only cells that belong to that module. IC Compiler supports both rectangular and rectilinear plan groups. For more information, see [“Creating Plan Groups” in Chapter 7](#).

Deciding on the Physical Partitions

IC Compiler provides the following features to help you decide on the physical partitions:

- Using the Hierarchy Browser

You can use the hierarchy browser to navigate through the design hierarchy and to examine the logic design hierarchy and display information about the hierarchical cells and logic blocks in your design. You can select the hierarchical cells, leaf cells, or other objects you want to examine in layout or schematic views.

To open the hierarchy browser, select one of the following menu items:

- 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.

When you create the physical hierarchy, you should use the existing modules in the original logical hierarchy for the physical partitions. If you cannot use the original logical hierarchy, you can use the `merge_fp_hierarchy` and `flatten_fp_hierarchy` commands to modify the logical hierarchy.

For more information, see “[Analyzing and Manipulating the Hierarchy](#)” in Chapter 7.

- Performing Virtual Flat Placement

You can run virtual flat placement by using the `create_fp_placement` command to identify the logical hierarchy modules that can be used as physical partitions. If you do not know how the blocks are to be partitioned, you can run virtual flat placement without any partition constraints. The resulting log file contains information about the hierarchy nodes that are automatically selected for physical partitioning. The suggested physical partitioning is based on size so that the physical hierarchy is well balanced.

The hierarchical gravity feature keeps cells of a hierarchical design logic block physically together in the chip layout. This type of grouping usually results in better placement because designs tend to partition well along hierarchical boundaries.

For more information, see “[Virtual Flat Placement Overview](#)” in Chapter 6.

- Placing and Shaping Plan Groups

Based on the distribution of cells resulting from the initial virtual flat placement, you can use the `shape_fp_blocks` command to place, shape, and size plan groups automatically inside the core area. After placement and shaping, you can adjust the boundaries of the plan groups to finalize the floorplan.

For more information, see “[Automatically Placing and Shaping Objects in a Design Core](#)” in Chapter 7.

Hierarchical Floorplanning

Hierarchical floorplanning includes the following steps:

- [Padding the Plan Groups](#)
- [Shielding Plan Groups or Soft Macros](#)
- [Assigning Pins](#)
- [Creating Soft Macros From the Plan Groups](#)
- [Pushing Down Physical Objects](#)

Padding the Plan Groups

In the hierarchical design phase, you should add padding around plan group boundaries by using the `create_fp_plan_group_padding` command. Plan group padding sets placement blockages on the internal and external edges of the plan group boundary to prevent cells from being placed in the space around the plan group boundaries. If you place cells too close to the plan group boundaries, congestion and implementation issues might occur.

The plan group padding is visible in the layout window. The plan group padding is dynamic, which means that it follows the changes of the plan group boundaries.

For more information, see “[Adding Padding to Plan Groups](#)” in Chapter 7.

Shielding Plan Groups or Soft Macros

To avoid congestion and crosstalk issues between a block and the top level, you can add modular block shielding to plan groups and soft macros by using the `create_fp_block_shielding` command. The shielding creates rectangular metal layers around the outside of the soft macro boundary or plan group in the top level of the design or around the inside boundary of the soft macro or plan group, or both.

Usually, you can selectively create metal layers to block the router from routing long nets along the block boundaries. Note, however, that even if you block layers in the preferred direction, nets can still sometimes be routed along the block boundaries. In that case, you can consider blocking all layers. The block shielding is created along the soft macro boundaries, but it leaves narrow openings to access the pins.

For more information, see “[Adding Block Shielding to Plan Groups or Soft Macros](#)” in Chapter 7.

Assigning Pins

You can assign pins for soft macros in your design during floorplanning. Pins can be constrained to fixed locations, or adjusted to align with the pin placement of pins in other macros. You can create pin guides to restrict the possible placement locations of soft macro

pins. Feedthrough nets can be used to route signals through soft macros and reduce the total path length. You can also block feedthrough creation to prevent a net from entering a soft macro.

For more information, see “[Setting Pin Assignment Constraints](#)” in Chapter 11.

Creating Soft Macros From the Plan Groups

You can convert selected plan groups or all plan groups with placement constraints into soft macro CEL views by using the `commit_fp_plan_groups` command. Each soft macro is a CEL view of the block in the same Milkyway design library. All related design information, including the netlist, floorplan, pin assignment, and power plan is saved in the CEL view. You can use the CEL view directly for block implementation in IC Compiler.

For more information, see “[Converting Plan Groups to Soft Macros](#)” in Chapter 13.

Pushing Down Physical Objects

In the hierarchical design planning phase, most of the changes that are made to the floorplan and the netlist are done at the full-chip level and inherited by the soft macros after committing the plan groups. In some cases, however, changes to your design must be made at the top level. You can do this by using the `push_down_fp_objects` command to push down the physical objects such as routing, route guides, blockages, cells, and cell rows, from the top level to the soft macro level. You can also use the `push_up_fp_objects` command to push objects up from a soft macro back up to the top level.

For more information, see “[Pushing Physical Objects Down to the Soft Macro Level](#)” in Chapter 13 and “[Pushing Physical Objects Up to the Top Level](#)” in Chapter 13.

Hierarchical Timing Closure

Timing closure in a hierarchical design is more challenging than that in a flat design. In a hierarchical design, timing optimization might not be able to analyze or change all the logic of the timing paths that are causing violations. Therefore, a hierarchical design needs a more sophisticated flow to achieve timing closure.

The timing closure tasks in a hierarchical design include the following features:

- [Early Chip-Level Timing Feasibility Check](#)
- [Timing and Signal Integrity Budgeting](#)
- [Clock Planning](#)

Early Chip-Level Timing Feasibility Check

You can check the timing of the entire design at various stages of the design planning flow by using the `check_timing` and `report_timing` commands. Depending on the options you set, you can get reports on valid paths for the entire design or for specific paths, unconstrained paths, and timing slack. The timing report helps evaluate why some parts of a design might not be optimized. You can also use the `check_fp_timing_environment` command to check zero wire delay timing and timing bottlenecks.

If the early timing check shows serious timing violations, you should run in-place optimization by using the `optimize_fp_timing` command to improve the timing of your design. When performing timing optimization, the command honors the plan groups defined in the hierarchical design as well as the scan chain connections from the SCANDEF. The result predicts potential critical paths in the design. Negative slack on interblock paths after in-place optimization should not exceed 20 percent of the clock cycle time.

For more information, see “[Running In-Place Timing Optimization](#)” in Chapter 10.

You should analyze the results, identify the root causes of the timing issues, and resolve them before proceeding to the next steps.

Timing and Signal Integrity Budgeting

After the design passes the early timing feasibility checks, you can perform timing budgeting on plan groups by using the `allocate_fp_budgets` command. The timing budgeting determines the corresponding timing boundary constraints for each top-level soft macro or plan group (block) in the design. If your design meets the timing boundary constraints for each block when the block is implemented, the top-level timing constraints are satisfied.

Timing budgeting generates the SDC constraints and attributes for all the individual plan groups. These constraints and attributes are then transferred to the individual soft macro blocks when the hierarchy is committed by using the `commit_fp_plan_groups` command. Timing budgeting can also consider crosstalk effects across soft macro boundaries when generating SDC constraints.

Timing budgeting on plan groups runs with full-chip timing. It honors pin locations and feedthrough nets assigned to the plan groups. Reasonable timing budgets result from good chip-level timing.

For more information, see “[Performing Timing Budgeting](#)” in Chapter 12.

Clock Planning

Clock planning is an optional step in the design planning flow. You can use clock planning to estimate the clock tree insertion delay and skew in a hierarchical design. It also helps you determine the optimal clock pin locations on blocks.

If you use clock planning, run it before pin assignment. Clock planning inserts anchor cells to isolate the clock trees inside the plan group from the top-level clock tree. It then runs fast clock tree synthesis for each plan group to estimate the clock skew and insertion delay. The tool annotates clock tree synthesis results on floating pins that are defined on the anchor cells. Clock planning synthesizes the top-level clock tree to the anchor cell floating pins.

For hierarchical timing closure, clock planning is also used to generate realistic clock latency through budgeting to fix timing violations. Top-level timing violations result not only from delays on combinational logic between registers but also from clock skew between launching and capturing registers. In a hierarchical design, clock tree synthesis inside each soft macro cannot consider clock skews with the top level and other soft macros. In the top-level integration phase, clock skews among different soft macros can cause setup and hold violations on interblock paths. It is very hard to fix these violations without changing the soft macros. To estimate the chip-level clock skew issues for block-level implementation, consider using clock planning.

For more information, see [“Performing Clock Planning” in Chapter 9](#).

Using the GUI to Manage Tasks in Parallel

You can use the IC Compiler graphical user interface (GUI) to help manage the execution of a job script on a number of jobs that can be done in parallel by using the Load Sharing Facility (LSF) queuing system. Parallel job execution is typically used on a group of subblocks in a hierarchical design generated after committing plan groups to soft macros.

Using parallel processing can improve performance by allowing you to run tasks on a group of subblocks in parallel much more quickly than if they were run serially on a single machine. By breaking up tasks into stages and running them in parallel, you can find and correct problems earlier in the design flow.

To use the GUI to manage tasks that can run in parallel,

1. Choose File > Run Parallel Jobs

The Run Parallel Jobs dialog box appears.

Alternatively, you can use the `run_parallel_jobs` command.

2. Set the options, depending on your requirements.

Job title – Type the name of the job set that is used for the title bar when you invoke the GUI.

Job names – Type a list of names for the jobs. This labels the information and keeps it separate from other jobs that might be run in the same parent session. These names have meaning only when taken together with the batch script and executable that the tool launches in parallel. In many cases they are the names of subblocks in a hierarchical design but they can also refer to any task that can benefit from parallel execution.

Stage names – Type a list of stage names. The tool displays the stages in a GUI panel (monitor). The GUI visually records the progress of the child jobs through the stages of status information communicated back through the child modules. If no stages are defined, a single stage called “Job Status” is defined for each running job and is used to collect updated status information.

Batch script – Type the path to a batch script, which is the script that is provided to the child jobs. It contains commands and is passed to the executable on each parallel job.

Working directory – Type the name of the temporary working directory for job-related files. The default directory is ./sub_blocks.

Environment variables – The environment variables and their settings communicate information from the parent to the child processes. The values in the list become UNIX environment variable data that can customize the behavior of the batch script. The command passes the environment variables and settings through the queuing system to the child jobs. You can add new environment variables by selecting the “New” button and typing the name and value in the New Environment dialog box. You can also edit and remove the environment variables.

Restrict the number of active jobs – Select this option to restrict the number of active jobs that are submitted in parallel. If this option is not selected, all jobs are launched in parallel by default.

Show monitor – Select this option to open a GUI table view that shows the progress of the jobs that are currently running in parallel and enables you to interactively control the jobs.

Verbose – Select this option if you want to view more detailed status information. Use this option when you are debugging the logistics of setting up a new flow and you need detailed information about communication with the parent.

Advanced options – If you select the “Advanced options” button, a second dialog box appears which handles the more advanced options of the `run_parallel_jobs` command.

- Executable – This option allows you to specify the path to the executable that is run in parallel. The default is “IC Compiler.” If you select the “Specified” option, type the path to the executable in the corresponding text box.

The variables that are available to the executable are passed through the UNIX environment defined in the “Environment variables” list. The batch script is also passed to the executable on the command line by using the path to the script.

- Job submission script – This option allows you to specify the path to the job permission script. You can select the default script or you can customize your own job submission script.

The default job submission script interfaces with the Load Sharing Facility and is located at `$root/auxx/syn/chipopt/job_run.pl`.

Select the “Custom” option if you want to construct your own job submission script. It can be in any language and it can interface to any job-control system, proprietary or commercial.

Note:

Because it can be difficult to correctly create such a script, you should use the default script.

- Job queue – Type the name of the queue from which the jobs are dispatched. The queue name has meaning only to a given installation using a specified job submission script. In the default job script, which interfaces to the Load Sharing Facility (LSF), the default LSF queue is used.
- Runtime timeout – Select this option to specify the maximum amount of elapsed time that a job can run. You can use this option to prevent stuck or runaway jobs that can prevent a flow from running to completion.

Maximum runtime – Enter a maximum runtime value in the format: [[HH:] MM:] SS [.SS]. If the elapsed time exceeds the amount you enter, the job is terminated. This is the default.

For example, a runtime time-out value of 30 represents 30 seconds and a value of 2:30 represents 2 minutes and 30 seconds. If you specify hours, the maximum number of minutes is 59; otherwise, you can specify any number of minutes.

Timeout factor – Enter a factor used to determine the maximum elapsed time that a job can run. The value must be greater than 1. The default is 4. The factor relates to the average elapsed time of completed jobs. When one or more jobs have finished successfully, an average elapsed time of completion is computed. When a job’s elapsed time rises above this average times the factor, the command terminates job.

In most cases, parallel jobs should be arranged so that they run for approximately the same time. If you enter a value ranging from 5 through 10, you can effectively terminate runaway jobs without fear of terminating long-running jobs that will ultimately succeed.

- Prepare setup scripts only – Select this option if you want to set up all the batch scripts for the parallel run in the temporary working directory, but not actually run the scripts. This is useful if you want to view the scripts to make sure that they are correct.

3. Click OK or Apply.

You can use the `run_parallel_jobs` command to run parallel jobs controlled by the IC Compiler parent and to communicate and return status information of the child processes to the parent. The command syntax can be used to submit jobs and to check the status of the jobs.

The syntax to submit jobs is

```
run_parallel_jobs
-batch_script script_path
-exec exec_path
[-host_options host_options_name]
[-job_script script_path]
[-job_title job_title]
[-monitor]
[-run_timeout timeout | -run_timeout_factor factor]
[-setup_for_run_only]
[-stage_names stage_name_list]
[-temp_dir directory]
[-var_list name_value_pairs]
[-verbose]
job_name_list
```

You can use the `send_flow_status` command to send status information from a child job started through the `run_parallel_jobs` command back to the parent job.

The `send_flow_status` command uses the following syntax:

```
send_flow_status
[-job_name job_name]
[-host host_name]
[-port port_number]
-stage_name stage_name
-status current_status
[-eof]
[-verbose]
```

2

Creating a Floorplan

This chapter describes how to create and refine a floorplan from an existing netlist or floorplan description. The floorplan describes the size of the core; the shape and placement of standard cell rows and routing channels; standard cell placement constraints; and the placement of peripheral I/O, power, ground, corner, and filler pad cells.

This chapter includes the following sections:

- [Supported Types of Floorplans](#)
- [Preparing the Design](#)
- [Initializing the Floorplan](#)
- [Refining the Floorplan](#)
- [Adjusting the I/O Placement](#)
- [Saving the Floorplan Information](#)
- [Reading In an Existing Floorplan](#)
- [Copying a Floorplan From Another Design](#)

Supported Types of Floorplans

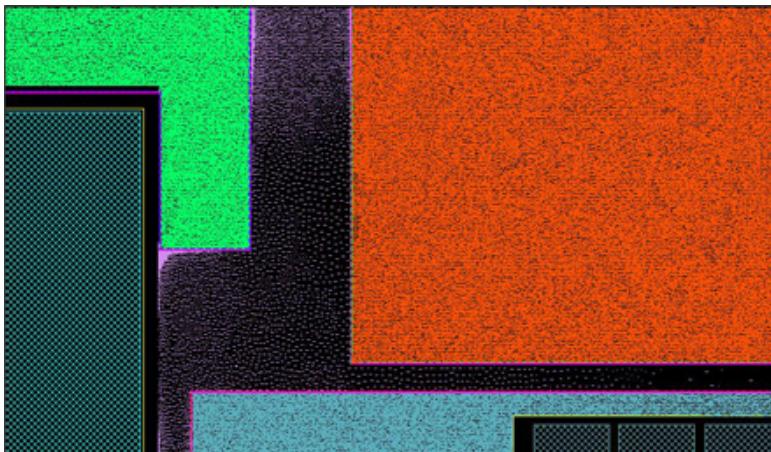
IC Compiler supports different floorplan methodologies to meet the requirements of your design. The channeled, abutted, and narrow-channel floorplan methodologies are described in the following sections.

- [Channeled Floorplans](#)
- [Abutted Floorplans](#)
- [Narrow-Channel Floorplans](#)

Channeled Floorplans

Channeled floorplans contain spacing between blocks for the placement of top-level macro cells, as shown in the floorplan example in [Figure 2-1](#).

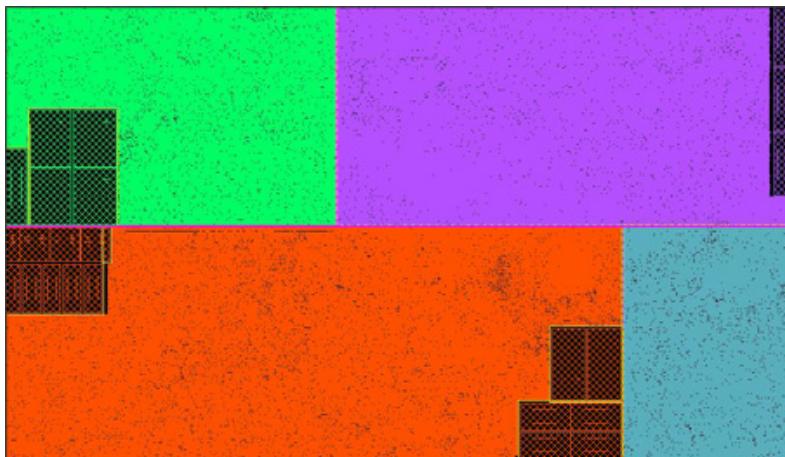
Figure 2-1 Channeled Floorplan



Abutted Floorplans

In the abutted floorplan methodology, blocks are touching and the tool does not allocate space for macro cell placement between blocks. Designs with abutted floorplans do not require top-level optimization, as all logic is pushed down into the blocks. However, abutted floorplans might require over-the-block routing to meet design requirements. This floorplan style also requires more attention to feedthrough management. Routing congestion might also become an issue for abutted floorplan designs. An example of an abutted floorplan is shown in [Figure 2-2 on page 2-3](#).

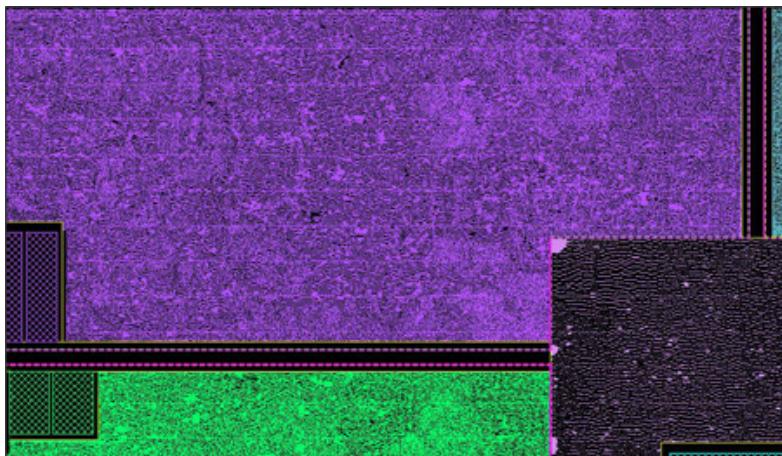
Figure 2-2 Abutted Floorplan



Narrow-Channel Floorplans

The narrow-channel floorplan methodology provides a balance between the channelled floorplan and abutted floorplan methodologies. You can abut certain blocks in the design where top-level cells are not needed. In other areas, you can reserve a channel between certain blocks for top-level clock routing or other special purpose cells. An example of a narrow-channel floorplan is shown in [Figure 2-3](#).

Figure 2-3 Narrow-Channel Floorplan



Preparing the Design

Before initializing the floorplan, you must read the design netlist and create unique instances of any multiply instantiated modules. For more information about this operation, see the *IC Compiler Implementation User Guide*. After reading the design, you must prepare the power and ground ports and set the constraints for the design.

- [Connecting Power and Ground Ports](#)
 - [Adding Power, Ground, and Corner Cells](#)
 - [Setting the I/O Pad Constraints](#)
 - [Setting the Pin Constraints](#)
 - [Saving the Pin and Pad Constraints](#)
 - [Reading an Existing Pad and Pin Constraints File](#)
 - [Reporting the Pad and Pin Constraints](#)
 - [Removing the Pad and Pin Constraints](#)
-

Connecting Power and Ground Ports

Use the `derive_pg_connection` command to create logical connections between power and ground pins on standard cells and macros and the power and ground nets in the design. For more information about the `derive_pg_connection` command, see “Creating Logical Power and Ground Connections” in the *IC Compiler Implementation User Guide*.

Adding Power, Ground, and Corner Cells

Physical-only cells for power, ground, and corner placement might not be part of the synthesized netlist and must be added to design. Use the `create_cell` command to add a leaf or hierarchical cell to the current design.

```
create_cell
  cell_list
  [reference_name]
  [-view view_name]
  [-freeze_silicon]
  [-hierarchical]
```

Setting the I/O Pad Constraints

Before initializing the floorplan, you can create placement and spacing settings for I/O pads by using the `set_pad_physical_constraints` command.

```
set_pad_physical_constraints
  objects | -pad_name pad_name
  [-side side_number]
  [-order order_number]
  [-offset offset_distance]
  [-orientation reflect | optimizeReflect]
  [-min_left_iospace min_space_left]
  [-min_right_iospace min_space_right]
  [-chip_level_distance {keyword_value_pairs}]
  [-lib_cell]
  [-lib_cell_orientation {l_orient t_orient r_orient b_orient}]
```

This command specifies the pad cell ordering, orientation, placement side, offset from die edge, and pad-to-pad spacing for each I/O pad. After setting the constraints with the `set_pad_physical_constraints` command, the `create_floorplan` command places the I/O pad cells accordingly. The constraints are stored in the Milkyway database when you save the design.

The `create_floorplan` command places constrained pads first. Any unconstrained pads are placed next, using any available pad location. The tool does not place unconstrained pads between consecutively ordered constrained pads.

The `initialize_rectilinear_block` command places pads based on constraints set by using the `set_pad_physical_constraints` command, as well as pins based on constraints set by using the `set_pin_physical_constraints` command. IC Compiler first checks that the constraints set by both commands are correct. If no constraint errors exist, the tool places the pins and then places the pads.

Note:

Pins and pads cannot be placed on the same side of a block created by using the `initialize_rectilinear_block` command.

The following example script portion describes two corner pad locations and several I/O pad locations for the floorplan.

Example 2-1 Pad Constraints Script Example

```
create_cell {cornerll cornerlr cornerul cornerur} cornercell
create_cell {vss1left vss1right vss1top vss1bottom} vsscell
# additional create_cell commands not shown

set_pad_physical_constraints -pad_name "cornerul" -side 1
set_pad_physical_constraints -pad_name "cornerur" -side 2
# additional corner pad constraints not shown

set_pad_physical_constraints -pad_name "pad_iopad_0" -side 1 -order 1
set_pad_physical_constraints -pad_name "pad_iopad_1" -side 1 -order 2
# additional left side pad constraints not shown

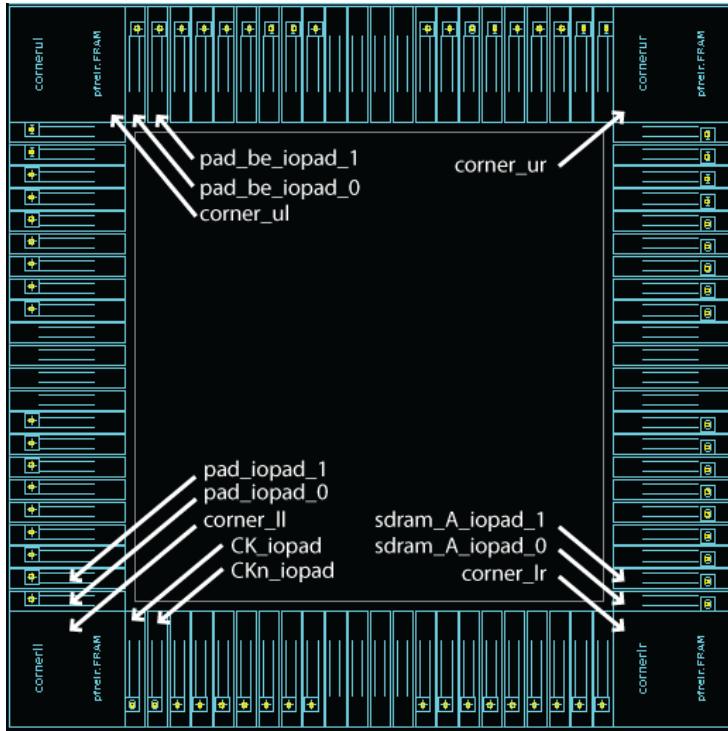
set_pad_physical_constraints -pad_name "pc_be_iopad_0" -side 2 -order 1
set_pad_physical_constraints -pad_name "pc_be_iopad_1" -side 2 -order 2
# additional top side pad constraints not shown

set_pad_physical_constraints -pad_name "sdram_A_iopad_0" -side 3 -order 1
set_pad_physical_constraints -pad_name "sdram_A_iopad_1" -side 3 -order 2
# additional right side pad constraints not shown

set_pad_physical_constraints -pad_name "CK_iopad" -side 4 -order 1
set_pad_physical_constraints -pad_name "CKn_iopad" -side 4 -order 2
# additional bottom side pad constraints not shown
```

[Figure 2-4](#) shows the floorplan created by these constraints.

Figure 2-4 Floorplan With Pad Placement



Setting the Pin Constraints

You can use the `set_pin_physical_constraints` command to set constraints on individual pins or nets. Use the `set_fp_pin_constraints` command to set global constraints for a block. If a conflict arises between the individual pin constraints and the global pin constraints, the individual pin constraints have higher priority. The constraints are stored in the Milkyway database. For more information about setting pin constraints with the `set_pin_physical_constraints` command, see “[Specifying Pin Physical Design Constraints](#)” on page 11-4.

Saving the Pin and Pad Constraints

You can save the current pin and pad constraints for your design with the `write_pin_pad_physical_constraints` command, or by choosing *Floorplan > Write Pin/Pad Physical Constraints* in the GUI.

```
write_pin_pad_physical_constraints
[-library lib_name]
[-cell cell_name]
[-constraint_type side_only | side_order | side_location]
[-pin_only | -pad_only]
[-objects object_list]
file_name
```

This command creates a constraints file that contains the `set_pin_physical_constraints` and `set_pad_physical_constraints` commands that you can use to reapply pin and pad constraints. The positional information for the pins and pads is based on their current location in the design.

Reading an Existing Pad and Pin Constraints File

Use the `read_pin_pad_physical_constraints` command (or choose Floorplan > Read Pin/Pad Physical Constraints in the GUI) to read a file that contains the `set_pin_physical_constraints` and `set_pad_physical_constraints` commands. The `read_pin_pad_physical_constraints` command applies the constraints to the current design or to another design that you specify.

```
read_pin_pad_physical_constraints
[-append]
[-cell name]
constraints_file
```

The constraints defined in the file can either remove the current constraints or append them, based on the behavior you choose. For example, if physical constraints for pin A are specified and you do not define any constraints for pin A in the constraints file, the `read_pin_pad_physical_constraints` command removes the existing physical constraints on pin A by default. To maintain the existing constraints and append new constraints contained in the constraints file, specify the `-append` option (or click the “Append” check box in the GUI).

Reporting the Pad and Pin Constraints

Use the `report_pin_pad_physical_constraints` command to display a list of `set_pin_physical_constraints` and `set_pad_physical_constraints` commands that define the pin and pad constraints for the current design.

```
report_pin_pad_physical_constraints
[-cell name]
[-pin_only | -pad_only | -chiplevel_pad_only]
[objects]
```

You can report only pin constraints, only pad constraints, only chip-level pad constraints, or all constraints depending on the command options you specify.

Removing the Pad and Pin Constraints

Use the `remove_pin_pad_physical_constraints` command to remove all constraints previously set by using the `set_pin_physical_constraints` and `set_pad_physical_constraints` commands.

```
remove_pin_pad_physical_constraints
[-cell name]
[-pin_only | -pad_only | -chiplevel_pad_only]
[objects]
```

You can remove only pin constraints, only pad constraints, only chip-level pad constraints, or all constraints based on the command options you specify. You can also remove constraints from a design other than the current design.

Initializing the Floorplan

After setting constraints, you can create the initial floorplan. The floorplan can be a rectangular or rectilinear shape.

- [Creating a Rectangular Floorplan](#)
- [Creating a Simple Rectilinear Floorplan](#)
- [Creating a Complex Rectilinear Floorplan](#)

Creating a Rectangular Floorplan

Use the `create_floorplan` command (or choose Floorplan > Create Floorplan in the GUI) to define a rectangular chip boundary and periphery based on aspect ratio, width and height, or number of cell rows. The command also places I/O pad and corner cells based on the constraints you defined by using `set_pin_physical_constraints` commands.

```
create_floorplan
[-bottom_io2core distance]
[-control_type aspect_ratio | boundary]
[-core_aspect_ratio ratio]
[-core_utilization ratio]
[-flip_first_row]
[-keep_io_place]
[-keep_macro_place]
[-keep_std_cell_place]
[-left_io2core distance]
[-min_pad_height]
[-no_double_back]
[-pad_limit]
[-right_io2core distance]
[-start_first_row]
[-top_io2core distance]
[-use_vertical_row]
```

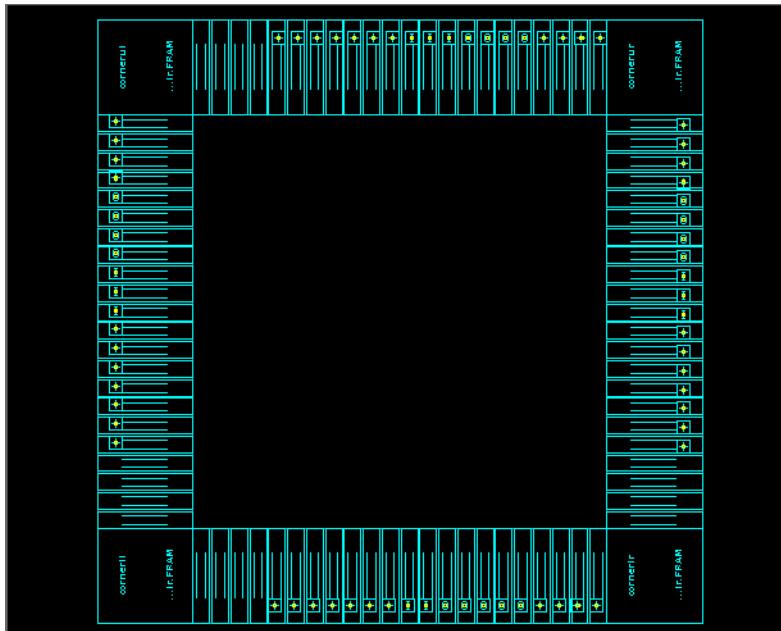
Note:

If your floorplanning flow uses the legacy `initialize_floorplan` command, you can continue to use the `initialize_floorplan` command for the F-2011.09 and F-2011.09-SP releases. The `create_floorplan` command contains the same functionality as the `initialize_floorplan` command.

The following examples are created by using different options with the `create_floorplan` command. [Figure 2-5 on page 2-11](#) shows a floorplan created by using the `create_floorplan` command with no options.

```
icc_shell> create_floorplan
```

Figure 2-5 Floorplan With No Options Specified



[Figure 2-6 on page 2-12](#) shows a floorplan created by using the `-control_type aspect_ratio` option. The following example shows the complete `create_floorplan` command. These two options set the aspect ratio for the design. In this example, the aspect ratio is 2.0, which creates a floorplan with a height equal to twice the width.

```
icc_shell> create_floorplan -control_type aspect_ratio \  
-core_aspect_ratio 2.0
```

Figure 2-6 Floorplan With Aspect Ratio of 2.0

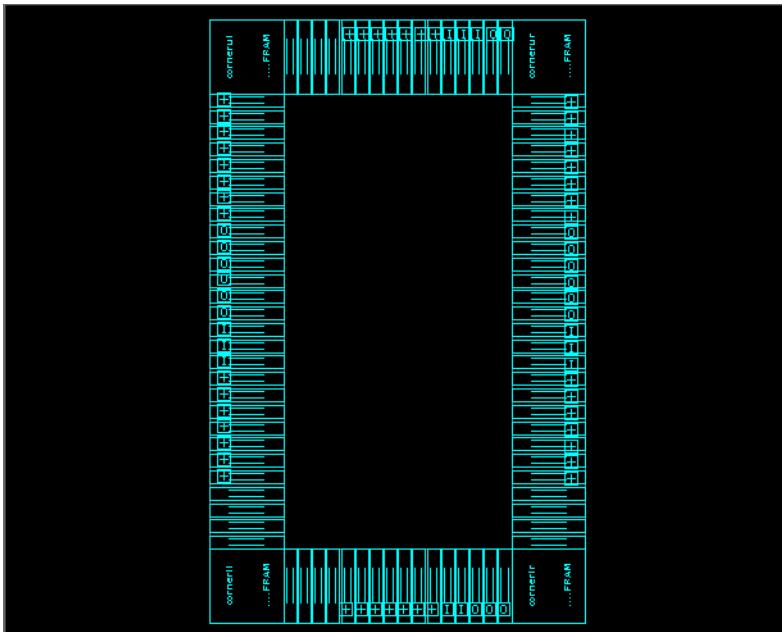


Figure 2-7 shows the first few rows of a floorplan created by using the `create_floorplan -start_first_row` command with the `-start_first_row` option. The `-start_first_row` option begins row pairing at the bottom of the core area.

```
icc_shell> create_floorplan -start_first_row
```

Figure 2-7 Floorplan With Row Pairing at the Core Bottom



[Figure 2-8](#) shows the first few rows of a floorplan created by using the `create_floorplan` command with the `-flip_first_row` option. The `-flip_first_row` option flips the first row at the bottom of the floorplan.

```
icc_shell> create_floorplan -flip_first_row
```

Figure 2-8 Floorplan With First Row Flipped



[Figure 2-9](#) shows the first few rows of a floorplan created by using the `create_floorplan` command with the `-no_double_back` option. The `-no_double_back` option prevents flipping or pairing of standard cell rows.

```
icc_shell> create_floorplan -no_double_back
```

Figure 2-9 Floorplan With No Row Flipping

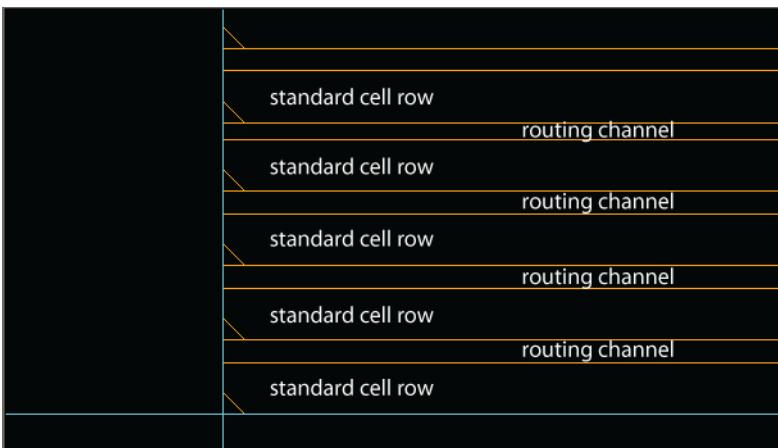


Figure 2-10 shows the first few rows of a floorplan created by using the `create_floorplan` command with the `-use_vertical_row` option. The `-use_vertical_row` option creates standard cell rows in a vertical orientation.

```
icc_shell> create_floorplan -use_vertical_row
```

Figure 2-10 Floorplan With Vertical Rows

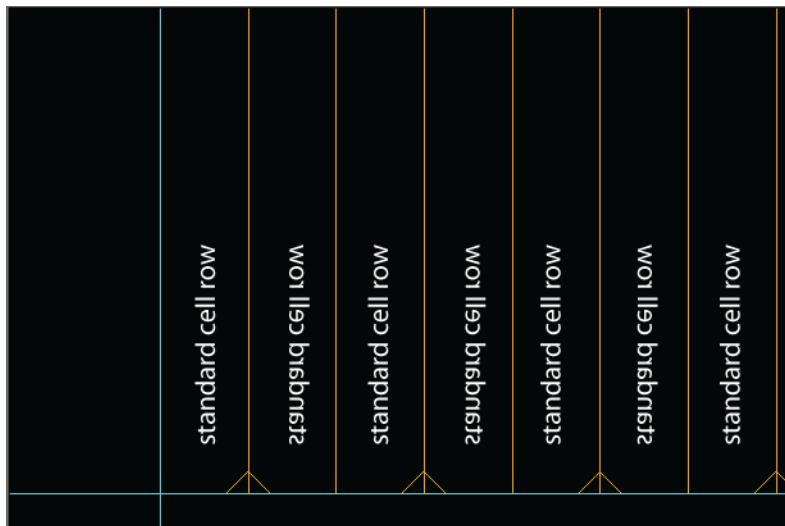
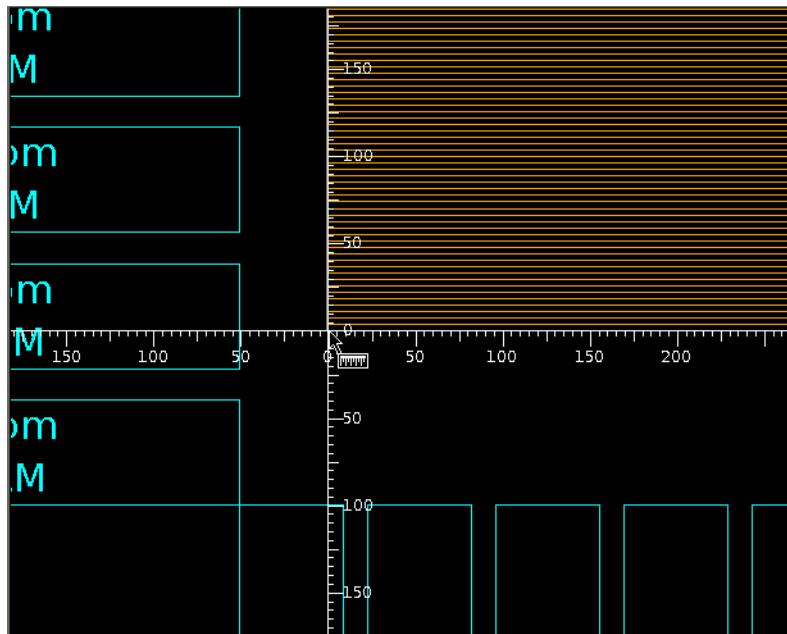


Figure 2-11 on page 2-15 shows a zoomed-in view of a floorplan created by using the `create_floorplan` command with the `-bottom_io2core` and `-left_io2core` options. These options set the distance between the core and the pad cells on the bottom and left sides of the die.

```
icc_shell> create_floorplan -core_utilization 0.8 \
-left_io2core 50 -bottom_io2core 100
```

Figure 2-11 Floorplan With I/O Pad to Core Spacing



Creating a Simple Rectilinear Floorplan

Use the `initialize_rectilinear_block` command (or choose Floorplan > Initialize Rectilinear Block in the GUI) to create L-shaped, T-shaped, U-shaped, or cross-shaped rectilinear blocks.

```
initialize_rectilinear_block
  [-bottom_io2core distance]
  [-control_type ratio | length]
  [-core_side_dim { side_a side_b side_c
                    side_d [side_e side_f] }]
  [-core_utilization ratio_val]
  [-flip_first_row]
  [-keep_io_place]
  [-keep_macro_place]
  [-keep_std_cell_place]
  [-left_io2core distance]
  [-no_double_back]
  [-orientation N | W | S | E ]
  [-right_io2core distance]
  [-row_core_ratio ratio_val]
  [-shape L | T | U | X]
  [-start_first_row]
  [-top_io2core distance]
  [-use_current_boundary]
  [-use_vertical_row]
```

You can vary the edge dimensions and rotation by specifying command options. The `initialize_rectilinear_block` command shares several options with the `create_floorplan` command; for more information, see “[Creating a Rectangular Floorplan](#)” on page 2-9. For complex rectilinear floorplans, use the `create_boundary` command with the `initialize_rectilinear_block` command, as described in “[Creating a Complex Rectilinear Floorplan](#)” on page 2-20.

The `initialize_rectilinear_block` command honors constraints set by using the `set_pad_physical_constraints` and `set_pin_physical_constraints` commands. Use these two commands to restrict the location of pins and pads on the periphery of the floorplan. Note that pins and pads cannot be placed on the same side of a block created by using the `initialize_rectilinear_block` command. For more information about the `set_pad_physical_constraints` command, see “[Setting the I/O Pad Constraints](#)” on page 2-5. For more information about the `set_pin_physical_constraints` command, see “[Setting the Pin Constraints](#)” on page 2-7.

Figure 2-12 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape T` option, which creates a T-shaped rectilinear block.

```
icc_shell> initialize_rectilinear_block -shape T
```

Figure 2-12 T-Shaped Rectilinear Block

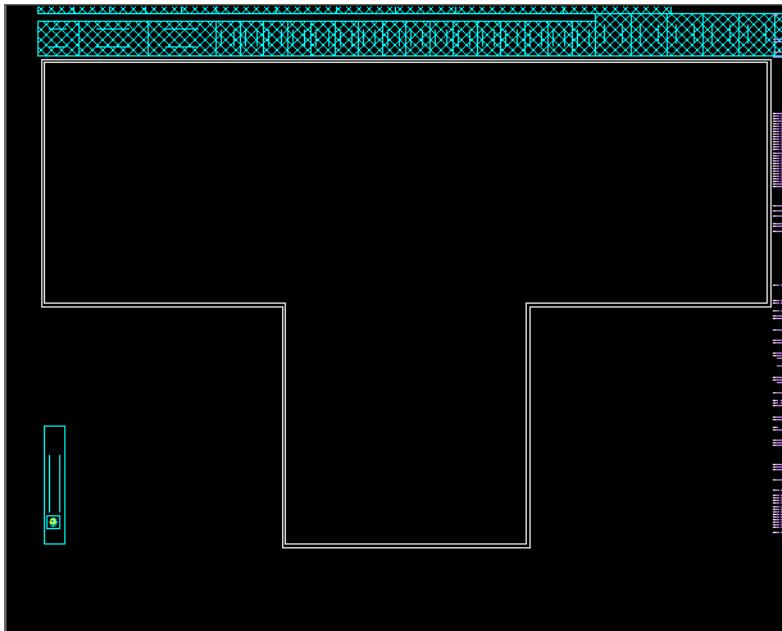


Figure 2-13 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape L` option, which creates an L-shaped rectilinear block.

```
icc_shell> initialize_rectilinear_block -shape L
```

Figure 2-13 L-Shaped Rectilinear Block

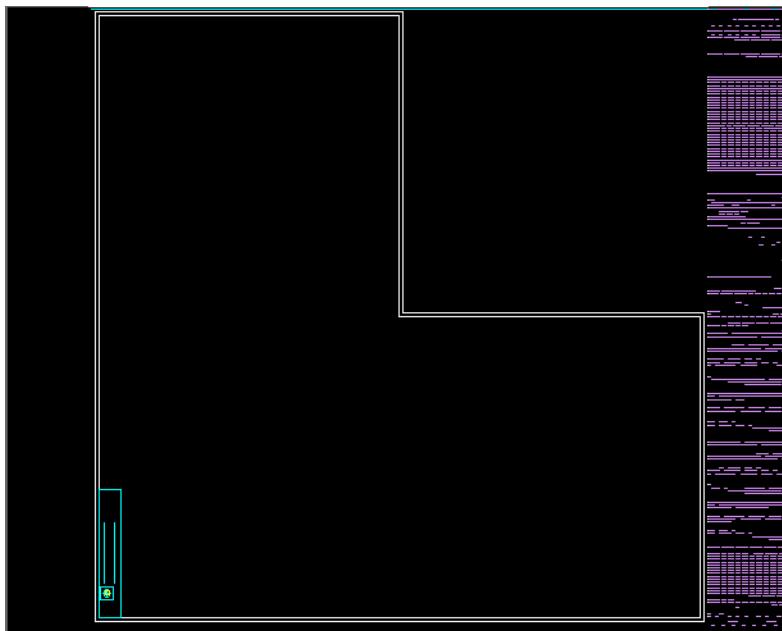


Figure 2-14 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape X` option, which creates a cross-shaped rectilinear block.

```
icc_shell> initialize_rectilinear_block -shape X
```

Figure 2-14 Cross-Shaped Rectilinear Block

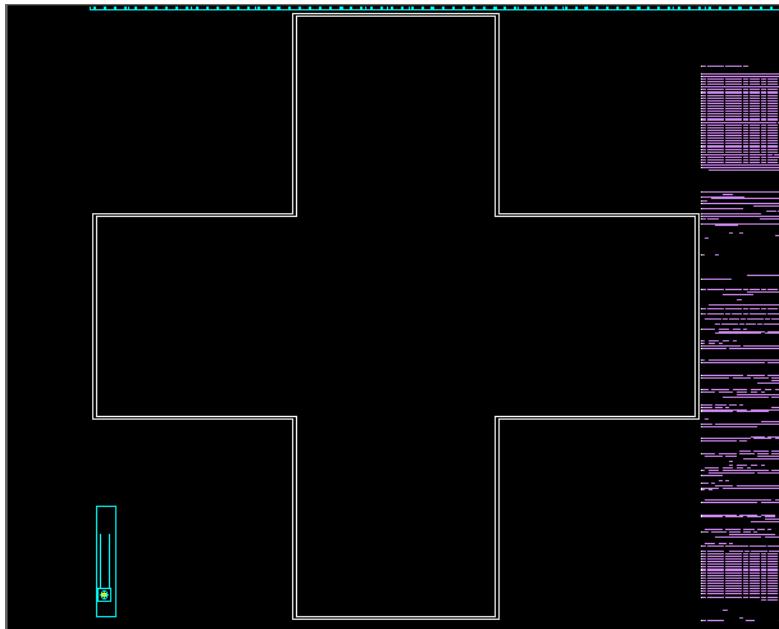


Figure 2-15 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape T` and `-orientation S` options, which creates a T-shaped rectilinear block that is rotated 180 degrees.

```
icc_shell> initialize_rectilinear_block -shape T -orientation S
```

Figure 2-15 T-Shaped Rectilinear Block Rotated 180 Degrees

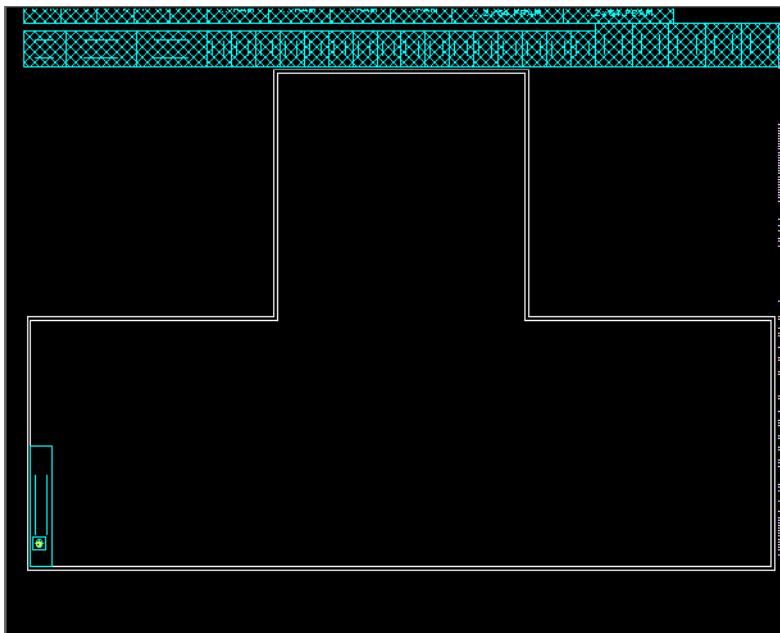
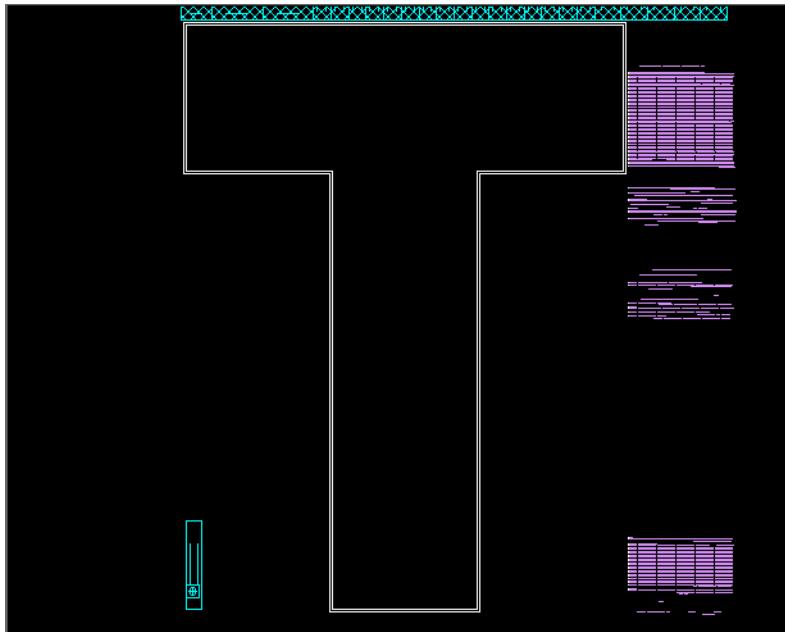


Figure 2-16 on page 2-20 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape`, `-control_type ratio`, and `-core_side_dim` options. The following command creates a T-shaped floorplan with a base section three times longer than the top section. For more information about which dimension argument controls which edge, see the man page or the graphic in the dialog box in the GUI.

```
icc_shell> initialize_rectilinear_block -shape T -control_type ratio \
           -core_side_dim { 1 1 3 1 3 1 }
```

Figure 2-16 T-Shaped Rectilinear Built Using User-Specified Dimensions



Creating a Complex Rectilinear Floorplan

You can create complex rectilinear floorplans by using the `create_boundary` command.

```
create_boundary
  [-coordinate rectangle |
   -poly {point point ...} | -by_terminal]
  [-core]
  [-left_offset l_offset]
  [-right_offset r_offset]
  [-top_offset t_offset]
  [-bottom_offset b_offset]
  [design | -lib_cell_type type]
```

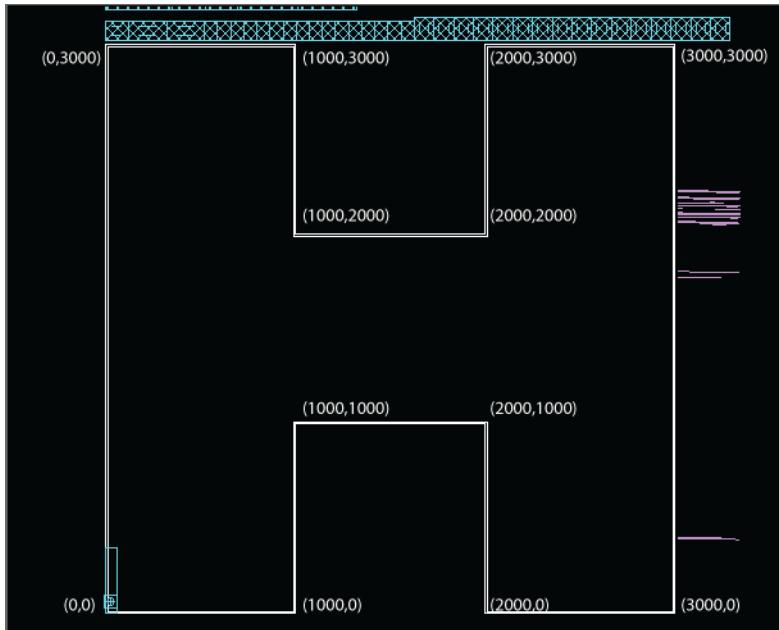
Use the `create_boundary` command together with the `initialize_rectilinear_block` command. The `create_boundary` command defines a floorplan based on a simple rectangle or a more complex shape based on the points of a polygon.

To create a complex rectilinear floorplan, define the perimeter by using the `create_boundary` command. Next, use the `initialize_rectilinear_block` `-use_current_boundary` command to initialize the floorplan using your boundary specification.

[Figure 2-17](#) shows a floorplan created by using the `create_boundary` command and the `initialize_rectilinear_block -use_current_boundary` command.

```
icc_shell> create_boundary -poly {{0 0} {0 3000} {1000 3000} \
{1000 2000} {2000 2000} {2000 3000} {3000 3000} {3000 0} \
{2000 0} {2000 1000} {1000 1000} {1000 0}}
icc_shell> initialize_rectilinear_block -use_current_boundary
```

Figure 2-17 H-Shaped Floorplan



Refining the Floorplan

After creating a floorplan, you can refine the floorplan with the `adjust_fp_floorplan` command or by making manual adjustments by using the tools in the GUI.

- [Adjusting the Floorplan](#)
- [Manually Modifying the Floorplan](#)

Adjusting the Floorplan

After you have created the floorplan by using the `create_floorplan` command or the `initialize_rectilinear_block` command, you can make adjustments by using the `adjust_fp_floorplan` command. The `adjust_fp_floorplan` command supports many of the same arguments as the `create_floorplan` command.

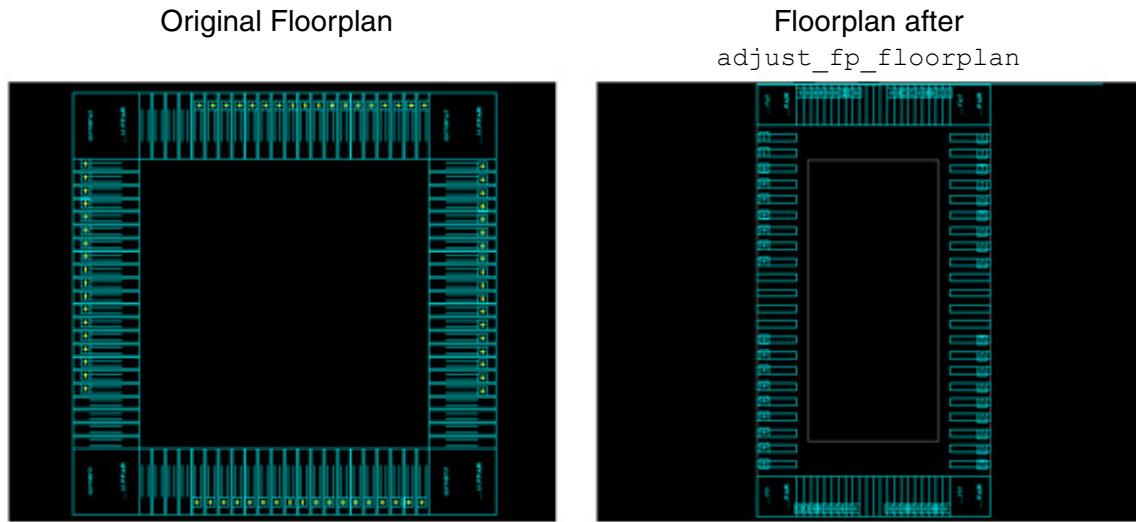
```
adjust_fp_floorplan
[-bottom_io2core distance]
[-core_aspect_ratio ratio]
[-core_height height]
[-core_utilization utilization]
[-core_width width]
[-die_height height]
[-die_origin {x y}]
[-die_width width]
[-fc_in_core number | -fc_periphery number]
[-flip_first_row true | false]
[-left_io2core distance]
[-maintain_placement]
[-min_pad_height true | false]
[-no_double_back true | false]
[-number_rows rows]
[-remove_filler_io]
[-right_io2core distance]
[-row_core_ratio ratio]
[-sm_utilization utilization]
[-start_first_row true | false]
[-top_io2core distance]
[-use_vertical_row true | false]
```

[Figure 2-18 on page 2-23](#) shows a floorplan that was initialized by using the `create_floorplan` command, and modified by using the `adjust_fp_floorplan` command. The `adjust_fp_floorplan` command makes the following changes:

- Changes the row orientation to vertical
- Expands the spacing between the core and I/O pads from 30 microns to 100 microns on the left and right
- Expands the spacing between the core and I/O pads from 30 microns to 300 microns on the top and bottom
- Changes the core aspect ratio to 3.0

```
icc_shell> create_floorplan -core_utilization 0.8 -left_io2core 30 \
    -bottom_io2core 30 -right_io2core 30 -top_io2core 30
icc_shell> adjust_fp_floorplan -use_vertical_row true -left_io2core 100 \
    -bottom_io2core 300 -right_io2core 100 -top_io2core 300 \
    -core_aspect_ratio 3.0
```

Figure 2-18 Adjusted Floorplan



Manually Modifying the Floorplan

You can perform detailed corrections to the floorplan, such as changing the die area, modifying standard cell rows, and changing wire tracks.

- [Defining the Die Area](#)
- [Removing Cell Rows](#)
- [Adding Cell Rows](#)
- [Defining the Wire Tracks](#)

Defining the Die Area

You can redefine the die area by using the `set_die_area` command. This command accepts two coordinates that define the lower-left corner and upper-right corner of the die specified by using the `-coordinate {x1 y2 x2 y2}` option. The coordinate specifies the lower-left and upper-right corner of the die. For more information about this option, see the man page.

For example, the following command defines a new die area rectangle with a lower-left corner at `{0 0}` and an upper-right corner at `{3000 3000}`. The `get_attribute` command confirms the die area set by the `set_die_area` command.

```
icc_shell> set_die_area -coordinate {0 0 3000 3000}
1
icc_shell> get_attribute [get_die_area] bbox
{0.000 0.000} {3000.000 3000.000}
```

Removing Cell Rows

You can remove cell rows from the floorplan by using the `cut_row` command (or by choosing Floorplan > Cut Site Row in the GUI).

```
cut_row
[-all | -within {coordinates}]
```

This command removes all cell rows in the design or only the rows in the specified area. For example, the following command removes the cell rows in the die area bounded by the coordinates `{0 0}` and `{1000 1000}`.

```
icc_shell> cut_row -within {{0 0} {1000 1000}}
1
```

Adding Cell Rows

You can add cell rows to the floorplan by using the `add_row` command (or by choosing Floorplan > Add Site Row in the GUI).

```
add_row
-within { {llx lly} {urx ury} } |
    { {x1 y1} {x2 y1} {x2 y2} ... {x1 y1} }
[-allow_overlap]
[-bottom_offset bottom_offset]
[-direction horizontal | vertical]
[-flip_first_row]
[-left_offset left_offset]
[-minimal_channel_height channel_height]
[-no_double_back]
[-no_start_from_first_row]
[-right_offset right_offset]
[-snap_to_orthogonal_row_direction {existing_row | wire_track | none}]
[-snap_to_row_direction {wire_track | tile_width | none}]
[-tile_name tile_name]
[-top_offset top_offset]
```

The command inserts cell rows based on the configuration options you provide and implements the rows based on the current floorplan. If you specify a design area that already contains rows, the `add_row` command does not create an overlap condition with the existing rows.

To create cell rows in a rectangular region, use the `-within` option and specify the lower-left and upper-right points of the rectangle. To create cell rows in a rectilinear region, use the `-within` option and specify the points that define the boundary of the region. To align the cell rows, use the `-snap_to_row_direction` and `-snap_to_orthogonal_row_direction` options.

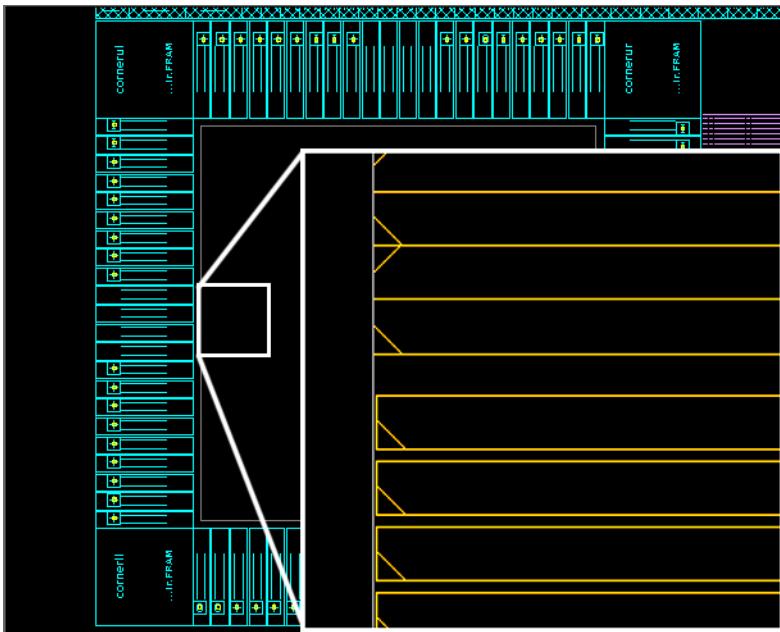
By default, new rows snap to the wire track in the direction of the row, and snap to existing rows in the orthogonal direction. If there are no existing rows with the same unit tile, the new rows snap to the wire track in the orthogonal direction instead. If there are rows that exist in the area specified by the `-within` option, IC Compiler issues an error and exits the command. You can create overlapping rows by specifying the `-allow_overlap` option to the `add_row` command. For more information, see the man page.

For example, the following commands remove the cell rows from the die area bounded by the coordinates `{375.770 375.770}` and `{1778.790 1000.200}`, and then add new rows with the `-no_double_back` option.

```
icc_shell> cut_row -within {{375.770 375.770} {1778.790 1000.200}}
1
icc_shell> add_row -within {{375.770 375.770} {1778.790 1000.200}} \
    -no_double_back
Info: 138 rows are added.
1
```

[Figure 2-19 on page 2-26](#) shows the section of floorplan modified by using the `add_row` command. Note that the rows in the bottom half of the cutout are inserted by using the `add_row -no_double_back` command, while the rows in the top half of the cutout were originally inserted without using the `-no_double_back` option.

Figure 2-19 Floorplan After Reinserting Rows



Defining the Wire Tracks

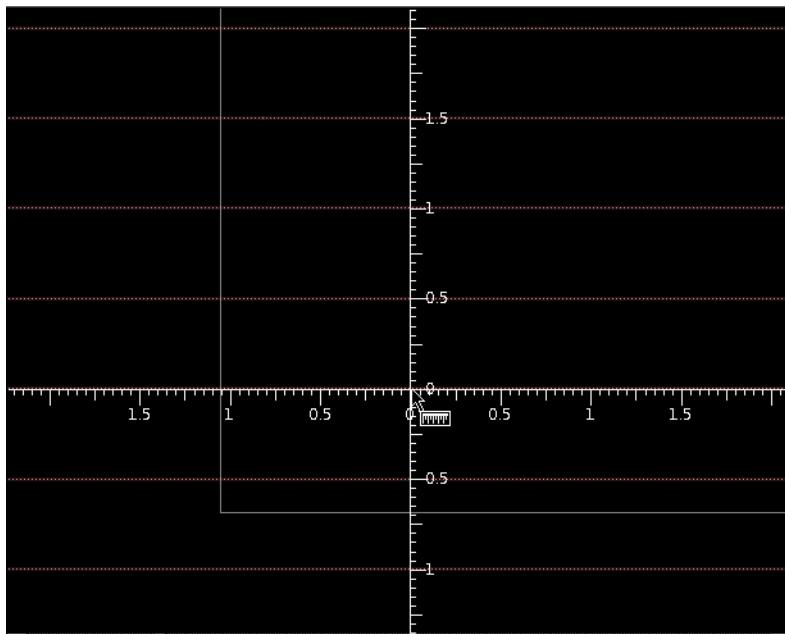
You can create routing tracks in the floorplan by using the `create_track` command (or by choosing Floorplan > Create Track in the GUI).

```
create_track
  -layer layer
  [-space track_pitch]
  [-count number_of_tracks]
  [-coord start_x_or_y]
  [-dir X | Y]
  [-bounding_box track_boundary_box]
```

[Figure 2-20 on page 2-27](#) shows a floorplan created with the `create_track` command. In this example, the `remove_track` command removes all existing routing tracks, and then the `create_track` command creates routing tracks on METAL3 in the preferred direction with a pitch of 0.5. The `report_track` command reports the routing track direction, startpoint, number of tracks, and pitch for each set of routing tracks in the design.

```
icc_shell> remove_track -all
1
icc_shell> create_track -layer METAL3 -space 0.5
1
icc_shell> report_track
*****
Report track
Design : TEST
Version: F-2011.09
Date  : Mon Aug 1 14:50:01 2011
*****
Layer      Direction   Start      Tracks      Pitch      Attr
-----
Attributes :
    usr : User defined
    def : DEF defined
METAL3      Y          0.250     4307      0.500      usr
1
```

Figure 2-20 Floorplan After Inserting Wire Tracks



Adjusting the I/O Placement

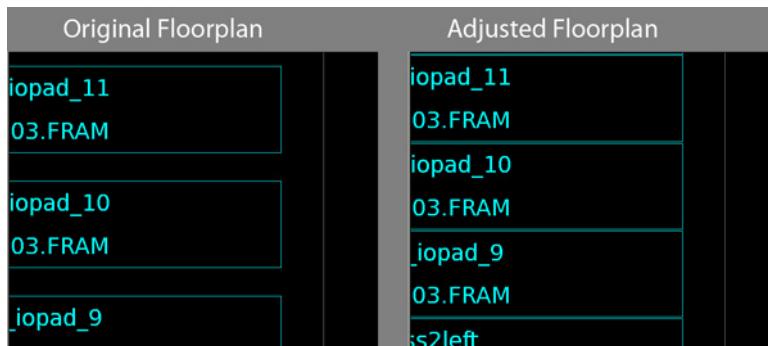
You can refine the spacing and relative locations of the I/O pads in your design by using the `adjust_fp_io_placement` command (or by choosing Floorplan > Adjust I/O Placement in the GUI).

```
adjust_fp_io_placement
[-side l | r | t | b]
[-spacing float]
[-pitch float]
[-offset float]
[-undo]
[list of IO cells]
```

The `adjust_fp_io_placement` command operates on one side of the chip at a time and supports an `undo` function that reverts the design to its previous state. [Figure 2-21](#) shows a floorplan adjusted by using the `adjust_fp_io_placement` command. In this example, the original pad spacing is 30 microns as specified by using the `set_pad_physical_constraints -min_left_iospace` command. The `adjust_fp_io_placement -side l -spacing 1` command changes the spacing between the I/O pads to 1 micron for pads on the left side of the chip.

```
icc_shell> adjust_fp_io_placement -side l -spacing 1
adjust_fp_io_placement -side l -spacing 1
Information: Adjusting placement of IO cells on specified sides ( left )
Information: Using spacing constraint, spacing = 1.000000 microns.
Information: packing IOs to the center.
1
```

Figure 2-21 Floorplan With Adjusted I/O Pads



Saving the Floorplan Information

After you create the floorplan, you can save a Tcl script that describes the floorplan by using the `write_floorplan` command or save a DEF file by using the `write_def` command.

- [Writing the Floorplan File](#)
- [Writing the Floorplan to a DEF File](#)
- [Floorplan Physical Constraints in Design Compiler](#)

Writing the Floorplan File

You can write the current floorplan to a Tcl file by using the `write_floorplan` command (or by choosing Floorplan > Write Floorplan in the GUI).

```
write_floorplan
  [-all]
  [-cell design_name]
  [-create_bound]
  [-no_bound]
  [-no_create_boundary]
  [-no_placement_blockage]
  [-no_plan_group]
  [-no_route_guide]
  [-no_voltage_area]
  [-objects object_collection]
  [-pin_guide]
  [-placement {placement_info_types} [-create_terminal]]
  [-preroute]
  [-user_shape]
  [-net_shape]
  [-row]
  [-sm_all]
  [-sm_bound]
  [-sm_cell_row]
  [-sm_placement {placement_info_types}]
  [-sm_placement_blockage]
  [-sm_plan_group]
  [-sm_preroute]
  [-sm_route_guide]
  [-sm_track]
  [-sm_voltage_area]
  [-track]
  file_name
```

The `write_floorplan` command creates a Tcl file that contains objects, placement, and attribute information for I/Os, terminals, standard cells, hard macros, and routing tracks in the floorplan. The Tcl file is used to rebuild the floorplan by using the `read_floorplan` command. You can control the information written to the Tcl file by using command options. The command does not write relative placement group and relative placement keepout information.

The `write_floorplan` command does not write standard cell placement information in the floorplan exploration flow if the database is created by the Design Compiler tool in topographical mode. The `write_floorplan` command issues the following message to alert you that standard cell placement information is not written to the floorplan Tcl script:

Warning: Standard cell placement output was suppressed in
`write_floorplan` because this is a DCG database (APL083)

The `write_floorplan` command does not write standard cell placement information in the floorplan exploration flow if the database is created by the Design Compiler tool in topographical mode. The `write_floorplan` command issues the following message to alert you that standard cell placement information is not written to the floorplan Tcl script:

Warning: Standard cell placement output was suppressed in `write_floorplan` because this is a DCG database (APL-083)

The following example uses the `write_floorplan` command with no options to create die area information.

```
icc_shell> write_floorplan floorplan_1.tcl
1
```

Writing the Floorplan to a DEF File

You can save the current floorplan to a DEF file by using the `write_def` command (or by choosing File > Export > Write DEF in the GUI).

```
write_def
  -output output_file_name
  [-version def_version]
  [-unit conversion_factor]
  [-compressed]
  [-rows_tracks_gcells]
  [-vias]
  [-all_vias]
  [-nondefault_rule]
  [-lef lef_file_name]
  [-regions_groups]
  [-components]
  [-macro]
  [-fixed]
  [-placed]
  [-pins]
  [-blockages]
  [-specialnets]
  [-notch_gap]
  [-pg_metal_fill]
  [-nets]
  [-routed_nets]
  [-diode_pins]
  [-floating_metal_fill]
  [-scanchain]
  [-no_legalize]
  [-verbose]
```

The `write_def` command writes the floorplan information, including the design netlist, layout, and constraints, to a DEF file. The following example uses the `write_def` command to write the floorplan to a DEF file.

```
icc_shell> write_def -version 5.6 -vias -all_vias -pins -nets \
           -diode_pins -specialnets -notch_gap -pg_metal_fill -scanchain \
           -no_legalize -output "chip.def"
```

Floorplan Physical Constraints in Design Compiler

To improve timing, area, and power correlation between Design Compiler and IC Compiler, you can read your mapped Design Compiler netlist into IC Compiler, create a basic floorplan in IC Compiler, export this floorplan from IC Compiler, and read the floorplan back into Design Compiler. The `write_def` and `write_floorplan` commands write floorplan information that can be used to create a floorplan that supports this flow. The following sections describe the commands and options necessary to write the floorplan file by using the `write_def` and `write_floorplan` commands.

Extracting Physical Constraints

To export floorplan information from IC Compiler for use in Design Compiler topographical mode, you can use the `write_def` command in IC Compiler. The `write_def` command exports physical design data and writes this data to a Design Exchange Format (DEF) file, which you can read into Design Compiler in topographical mode. The following example uses the `write_def` command to write physical design data to the `floorplan_for_DC.def` file.

```
icc_shell> write_def -version 5.7 -rows_tracks_gcells -macro -pins \
           -blockages -specialnets -vias -regions_groups -verbose \
           -output floorplan_for_DC.def
```

Exporting Physical Constraints

To export floorplan information from IC Compiler for use in Design Compiler topographical mode, you can use the `write_floorplan` command in IC Compiler. The `write_floorplan` command writes a Tcl script that you read into Design Compiler in topographical mode to re-create elements of the floorplan of the specified design. The following example uses the `write_floorplan` command to write out all placed standard cells to a file called `floorplan_for_DC.tcl`.

```
icc_shell> write_floorplan -placement {io hard_macro soft_macro} \
           -create_terminal -row -create_bound -preroute floorplan_for_DC.tcl
```

Reading In an Existing Floorplan

You can read existing floorplan data saved in a DEF file or a Synopsys Tcl file.

- [Reading DEF Files](#)
 - [Reading Floorplan Files](#)
-

Reading DEF Files

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

```
read_def
[-check_only]
[-enforce_scaling]
[-no_incremental]
[-verbose]
[-turn_via_to_inst]
[-inexactly_matched_via_to_inst]
[-lef lef_file_names]
[-snet_no_shape_as_user_enter]
[-snet_no_shape_as_detail_route]
[-preserve_wire_ends]
def_file_names
```

The following command reads the `chip.def` DEF file.

```
icc_shell> read_def chip.def
```

By default, the `read_def` command is additive; it adds the physical data in the DEF file to the existing physical data in the design. To replace rather than add to existing data, use the `-no_incremental` option on the command line (or deselect Incremental in the dialog box in the GUI).

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 about 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
```

Reading Floorplan Files

You can use the `read_floorplan` command to import a previously saved floorplan file. A floorplan file is Tcl script file created by using the `write_floorplan` command (or by choosing Floorplan > Write Floorplan in the GUI). For more information about creating a floorplan file, see “[Writing the Floorplan File](#)” on page 2-29. To display each command as it is executed, specify the `-echo` option. For more information about this command, see the man page.

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 more information about the `derive_pg_connection` command, see “[Creating Logical Power and Ground Connections](#)” in the *IC Compiler Implementation User Guide*.

The following example reads a floorplan from the `chip.tcl` file. The file was previously saved by using the `write_floorplan` command.

```
icc_shell> read_floorplan chip.tcl
```

Copying a Floorplan From Another Design

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).

```
copy_floorplan  
  [-library design_library_name]  
  -from design_name  
  [-macro]  
  [-filler]  
  [-pad]  
  [-power_plan]  
  [-verbose]
```

Specify the design from which to copy the physical data by using the `-from` option (or by specifying the design name in the “From cell” box in the GUI). The following example copies the floorplan from the design named `design1`.

```
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 shielding.

- Power net connections

- Plan groups

- Macro cell placement (optional)

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

- I/O pad placement (optional)

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

- Filler cell placement (optional)

To copy the filler cell placement information, use the `-filler` option (or choose “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 choose “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 choose “Incremental (non-overwriting) in the GUI).

Note:

You must ensure netlist consistency between the designs. The command does not perform netlist consistency checks during the copy process.

The `copy_floorplan -incremental` command copies the rows from the source cell to the destination cell, even if there are existing rows in the destination cell.

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

3

Automating Die Size Exploration

This chapter describes how to use MinChip technology in IC Compiler. MinChip technology automates die size exploration and determines the smallest routable die size for your design while maintaining the relative placement of hard macros and I/O cells. You can enable power network synthesis to ensure that the floorplan meets voltage drop requirements. The technology is integrated into the Design Planning tool through the `estimate_fp_area` command. The input is a flat Milkyway CEL view.

MinChip technology uses the `estimate_fp_area` command to iterate through different design size estimates to identify the smallest die or block size that meets the routability target. If the input CEL view (floorplan) is easy to route, the `estimate_fp_area` command finds the smallest routable die size. If the initial floorplan cannot be routed, the `estimate_fp_area` command produces a routable floorplan with a larger core area.

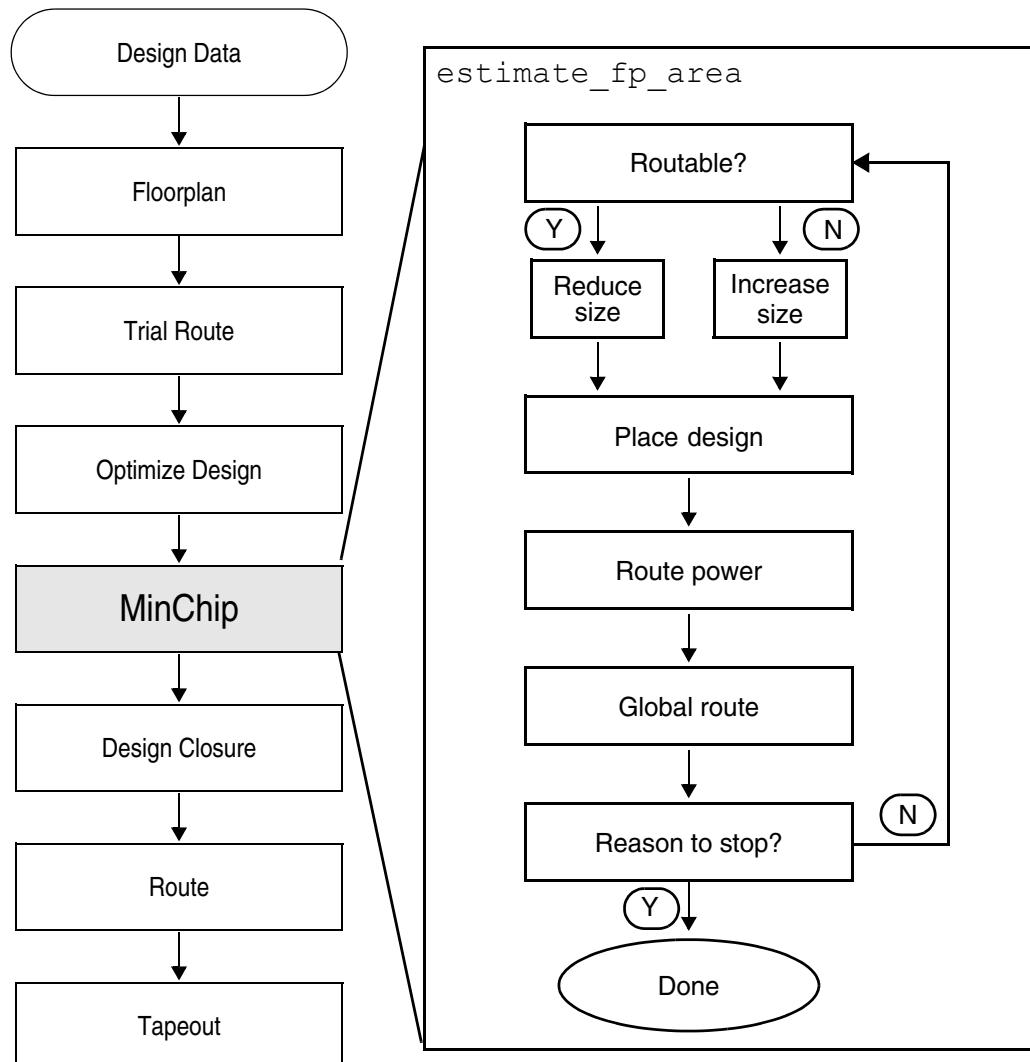
This chapter includes the following sections:

- [MinChip Design Flow](#)
- [Preparing the Design for MinChip Technology in IC Compiler](#)
- [Using the Estimate Area GUI](#)
- [Using the `estimate_fp_area` Command and Options](#)
- [Using MinChip Technology With Multivoltage Designs](#)

MinChip Design Flow

Figure 3-1 shows design flow using MinChip technology, and the steps that occur inside the MinChip technology flow.

Figure 3-1 MinChip Technology Flow and Steps



The `estimate_fp_area` command performs the following steps in the MinChip flow:

1. Estimates the core size and shape for a given core and cell utilization.

During the search for a minimal die size, filler pads and filler cells are automatically deleted to minimize the pad-limited characteristics of the design. When the size of the die changes, any previous placement blockages, preroutes, and routing guides are also automatically deleted.

To ensure that the results are consistent from each die size change, the following are maintained as constants:

- The original side and placement order of the I/O pads on the input design cell
- The relative placement of the hard macros
- The shape (aspect ratio) of the core area

2. Places all the design cells.

For a given core size, it runs virtual flat placement in incremental mode to place all the design cells.

3. Routes the design.

After cell placement, it automatically routes the design as follows:

- Performs a fast, virtual route. It estimates the wire length and calculates the routing resources to identify designs that are easy to route.
- Performs prototype global routing until the specified global route cell overflow percentage is reached. The command quits routing when the design is “unroutable.”
- Performs prototype global routing to report that the design is “unroutable” when local hot spots cause a very long runtime.

4. Analyzes the routing.

After the design is routed, the tool performs routing analysis. The analysis reports the number and percentage of nets routed, and the overall global route cell overflow percentage when a design is “unroutable.”

5. The `estimate_fp_area` command stops when any of the following occur:

- The routing target is met.
- The utilization bounds are exceeded.
- Overlaps of hard macros would be formed.
- The IR drop is exceeded.
- The design is pad limited.

Preparing the Design for MinChip Technology in IC Compiler

The MinChip flow operates based on the command options you specify. To enable the best possible die size optimization in the shortest amount of time, it is important for you to understand the nuances of your design. The quality of the results depends on the input CEL view and the `estimate_fp_area` command options you select.

For the best chip size reduction, make sure that your design is routable, can achieve timing closure, and the netlist is stable with no major logic changes expected in the gate-level netlist.

Note:

You should apply MinChip technology only to netlists and floorplans that have completed detailed implementation in IC Compiler. If the final netlist is larger than the netlist input to MinChip technology, it might not fit the optimized floorplan.

This section describes how to prepare your design to achieve the best QoR from MinChip technology.

- Floorplan
 - Use a design that has gone through initial detailed implementation and optimization in IC Compiler. This technology supports full-chip, rectangular block, and rectilinear block floorplans.
 - Correct all design rule violations.
 - Verify the cause of any routing congestion. Routing congestion or violations might exist because of floorplan issues, such as macros overlapping other macros, or existing power routing that makes pins inaccessible. MinChip technology reduces the size of routable designs and increases the size of unroutable designs.
- Power network synthesis scripts

Power network synthesis is used within MinChip technology primarily to consume routing resources used by power routing. This ensures an accurate view of the available routing resources.

To create an accurate power mesh and rings, use the `-run_pns_script` option.

- To estimate the power network, use the power network synthesis script file that you developed during the detailed implementation of the floorplan.
- If a custom power structure is required, create a power network synthesis script by using the `synthesize_fp_rail` command to emulate the power network. The script models the custom power structure by specifying the width, pitch, and offset of the core rings, block rings, and power straps. The objective is to accurately forecast the routing resources that are required by the power network.

Run power network synthesis on the existing floorplan before running the `estimate_fp_area` command to ensure that all the power network synthesis options are set properly. To run power network synthesis using the default configuration, use the `-power_net_names` and `-ir_drop_value` options.

- Preroute standard cell scripts

Use a preroute standard cells script to preroute to the supply pins of standard cells. Do this after you run power network synthesis by using the `synthesize_fp_rail` command. This is important if you use power network synthesis to form a mesh on the uppermost layers. Stacked vias used to connect the metal1 preroutes to the upper layer mesh are formed during the preroute process.

To run the preroute script within the `estimate_fp_area` command, specify the script name by using the `-run_preroute_script` option. Running the `estimate_fp_area` command with the `-run_preroute_script` option accounts for metal1 routing resources and stacked vias during die size reduction.

- Relative hard macro placement

The `estimate_fp_area` command maintains the relative placement of all fixed macros in terms of alignment and location. When the command reduces the chip size, the locations of its macro cells are automatically adjusted to the relative locations. The relative horizontal and vertical ordering of the macros is maintained. You can also reduce the die or core size until the fixed macros abut.

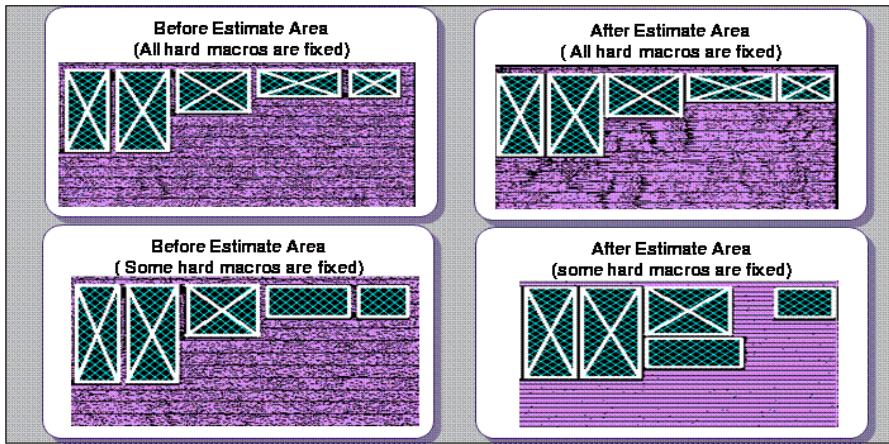
The `estimate_fp_area` command attempts to maintain the relative placement of unfixed macros, but that relative placement cannot be guaranteed. Macros that are not fixed can move to nearby locations.

Fix all macros whose placement is critical, especially any analog blocks or other macros partially outside the core area.

If there are fixed placed standard cells in your floorplan, the `estimate_fp_area` command might put hard macros on top of them. Unfix all standard cells before running the command.

[Figure 3-2 on page 3-6](#) shows before and after examples of maintaining relative hard macro placement.

Figure 3-2 Maintaining Relative Hard Macro Placement



- Hard macro padding

The `estimate_fp_area` command maintains the hard macro padding set by using the `set_keepout_margin` command.

- By default, IC Compiler calculates the padding for hard macros. You can set specific values for padding by instance or reference name.
- You can specify different padding values for different edges of the hard macro.
- If the padding of adjacent macros abuts or overlaps before running the `estimate_fp_area` command, the macros can be packed closer together.
- If the padding of adjacent macros does not overlap before you run the `estimate_fp_area` command, there is no overlap after you run the `estimate_fp_area` command.
- If the macro padding overlaps with other padding before you run the `estimate_fp_area` command, the macros will still overlap other padding after you run the command, but it will not overlap another macro.

- Sliver size

The `-sliver_size` option of the `set_fp_placement_strategy` command defines a spacing dimension for the `estimate_fp_area` command. If the distance between two objects (core edges, blockages, or macros) is less than or equal to the specified sliver size before running the `estimate_fp_area` command, this distance is not reduced by the `estimate_fp_area` command.

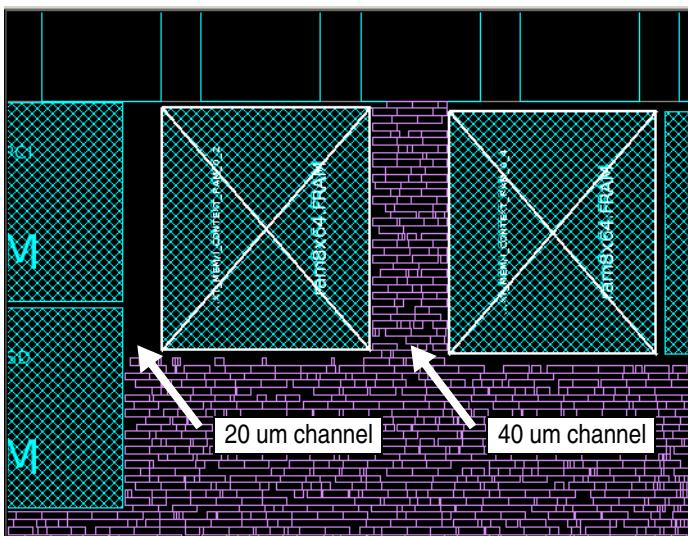
The default for the `-sliver_size` option is 0.

Analyze your floorplanned design to determine an appropriate value for the sliver size and set the strategy by using the `set_fp_placement_strategy -sliver_size` command. The following example sets a sliver size of 30 microns for the current design.

```
icc_shell> set_fp_placement_strategy -sliver_size 30
```

[Figure 3-3](#) shows an example placement result for a design that contains channels after setting a sliver size of 30 microns. The image shows two vertical channels between the macros. The channel on the left is 20 microns and the channel on the right is 40 microns. After setting the channel size with the `set_fp_placement_strategy -sliver_size 30` command, IC Compiler only places standard cells in channels when the distance between the blockage or macros that form the channel is 30 microns or greater.

Figure 3-3 Sliver Size Packing Limit



- Placement blockages

The `estimate_fp_area` command honors both soft and hard blockages. Blockages can be assigned to an edge or macro based on the relationship between the blockage and the edge or macro. You should delete all nonessential blockages. If the intent of a blockage is to reserve a space around a macro, delete the blockage because it is nonessential (macro padding performs this function) and define the appropriate macro padding or sliver size.

- Edge blockages

A blockage that covers any side of a core area is called an “edge blockage.” Edge blockages must extend over the complete edge dimension. In some cases, edge blockages are used to account for implementation requirements. For example, consider a design that requires a core ring on M1 and M2 as part of the power mesh. The `estimate_fp_area` command forms the power structure after placement. If the core

rings are likely to cover the placement area, a problem exists. Standard cells might have been placed in these areas. Edge blockages can reserve the placement area for the rings. During the `estimate_fp_area` iterations, the relative placement of edge blockages is maintained in relation to the core area.

- **Blockages over fixed hard macros**

Macro blockages are placement blockages assigned to macros. If at least 70 percent of the blockage area covers a macro, the `estimate_fp_area` command assigns the blockage to the macro. During die size reduction, macro blockages move together with assigned macros. If a blockage covers two or more macros, it is randomly assigned to one macro.

If macro padding cannot be used, you should create a hard macro blockage over each macro rather than one blockage over a set of macros.

- **Filler cells**

The `estimate_fp_area` command deletes I/O cells marked as type `filler`. Some I/O cell libraries might contain pads that are marked this way. To ensure that the I/O pads are not deleted, change these cell types before you run the `estimate_fp_area` command.

Note:

If the design has standard cells marked as type `filler`, you must explicitly delete these before running the `estimate_fp_area` command. The `estimate_fp_area` command does not automatically delete standard cell fillers in the core.

Using the Estimate Area GUI

You can use the GUI to automate the placement and routing iterations used to find the smallest routable die size. The tool automatically maintains relative placement, optionally uses power network synthesis for power routing, and uses routing congestion estimation technology to assess routing feasibility.

To explore valid die areas to determine the smallest routable die size,

1. Choose Floorplan > Minchip in the GUI.

The Estimate Area dialog box appears.

Alternatively, you can use the `estimate_fp_area` command.

The Estimate Area dialog box options are grouped into three categories:

- Floorplan Control
- Search Control
- Power Network Control

2. Set the floorplan control options, depending on your requirements.

Note:

If you run Estimate Area with the default values, die size reduction is observed in the width and height of the die until one of the search control criteria is achieved. Blockages are also removed during the default run.

- Select a sizing type to define constraints on the search space of aspect ratios that are used to reduce the die.

Variable aspect ratio – Varies the aspect ratio. This is the default behavior.

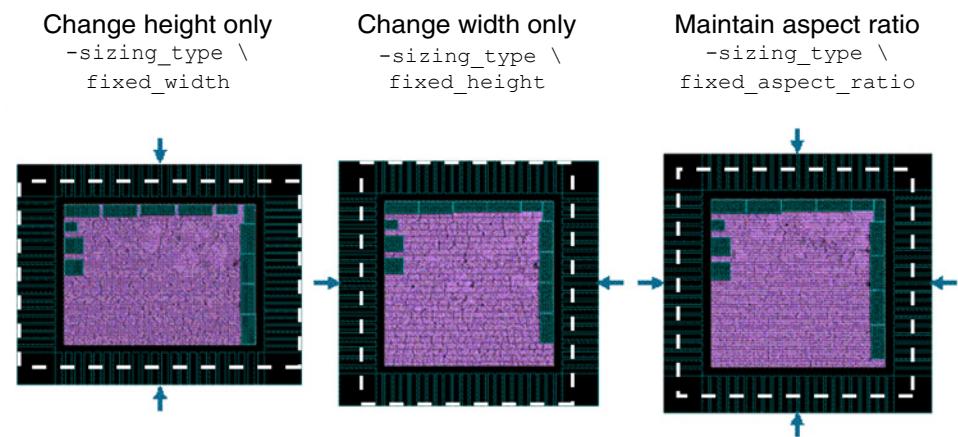
Fixed aspect ratio – Honors the aspect ratio of the original design size and stops shrinking the width or height of the die when the width or height becomes pad or macro limited.

Fixed width – Preserves the original design width and performs and reduces the chip size in the y-direction only. While reducing the chip size in the y-direction, the command might abut the macros if any of the stop criteria have not been met.

Fixed height – Preserves the original design height and reduces the chip size in the x-direction only. While reducing the chip size in the x-direction, the relative macro locations and orientations are maintained. The chip size reduction continues until the command meets any one of the stop criteria you have specified.

[Figure 3-4](#) shows an example of controlling the chip sizing.

Figure 3-4 Controlling the Chip Size



- Specify the core area boundaries.

Minimum width – Specifies the lower bound of the core width in microns to stop the die search process. By default, the command does not consider die sizes that result in utilizations greater than 0.95, or 95 percent.

Minimum height – Specifies the lower bound of the core height in microns to stop the die search process. By default, the command does not consider die sizes that result in utilizations greater than 0.95, or 95 percent.

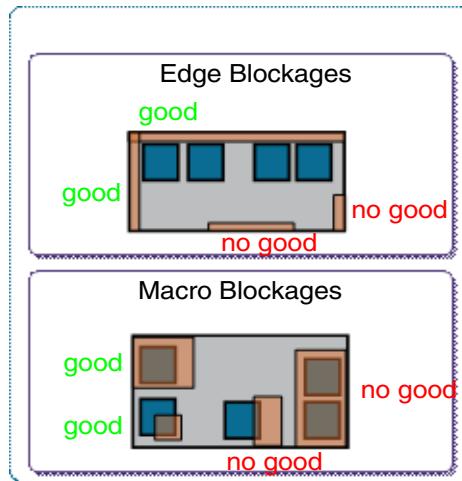
Maximum width – Specifies the upper bound of the core width in microns to stop the die search process. By default, the command does not consider die sizes that result in utilizations less than 0.30, or 30 percent.

Maximum height – Specifies the upper bound of the core height in microns to stop the die search process. By default, the command does not consider die sizes that result in utilizations less than 0.30, or 30 percent.

- Keep blockages – Keeps the placement blockages in the input floorplan. By default, the `estimate_fp_area` command deletes all placement blockages before it starts iterating through design sizes.

Some blockages are recognized as macro blockages or edge blockages, as shown in [Figure 3-5](#).

Figure 3-5 Blockage Types



A blockage that covers any side of a core area is called an “edge blockage.” Edge blockages keep cells and macros from being placed right at the edge of the core area.

Macro blockages are placement blockages assigned to macros. If a placement blockage overlaps a macro such that it covers 70 percent or more of the macro area, the placement blockage is assumed to be associated with the macro. The purpose of a macro blockage is to keep standard cells from being placed too close to the macro.

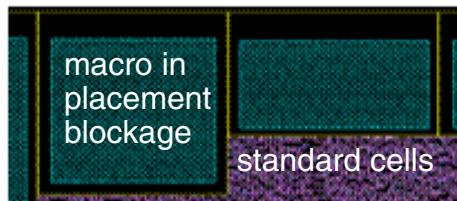
If a blockage covers two or more macros, it is randomly assigned to one macro.

You should create a separate placement blockage for each hard macro rather than one placement blockage over a set of macros.

During die size reduction, macro blockages move along with the macros, which might result in blockage overlaps, as shown in [Figure 3-6](#). Blockage overlaps are allowed. Macro overlaps, however, are not allowed.

Figure 3-6 Blockage Overlaps Allowed; Macro Overlaps Not Allowed

Blockages and macros do not overlap



Blockages overlap; macros do not



Note:

For most designs, the minimum allowable macro spacing can be achieved by using the `set_keepout_margin` command and the `sliver_size` parameter value. Creating and maintaining placement blockages to successfully complete estimate area iterations is not necessary. However, the `estimate_fp_area` command honors both soft and hard placement blockages.

If you specify a `sliver_size` parameter value, and the distance between two objects, such as macros, core edges, or blockages, is less than or equal to the `sliver_size` value before running the `estimate_fp_area` command, this distance is maintained by the `estimate_fp_area` command.

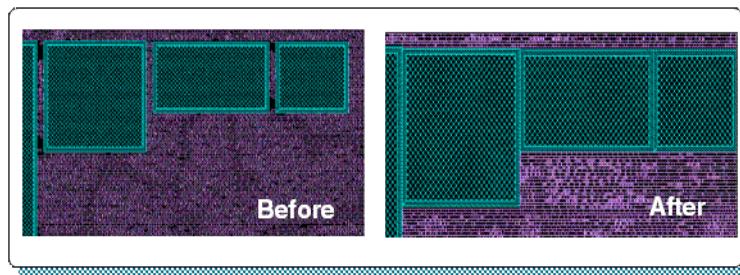
If there are no overlaps between the macro padding, and the distance between the macros is less than or equal to the `sliver_size` value, the `estimate_fp_area` command does not reduce the space between the hard macros.

If there are overlaps between the macro padding, and the distance between the macros is less than or equal to the `sliver_size` value, the `estimate_fp_area` command does not reduce the space between the hard macros, and the macro padding remains overlapped with the other padding.

If there is no `sliver_size` parameter value or `set_keepout_margin` specified for a fixed macro and one placement blockage is created over a set of multiple hard macros, the command abuts the macros. When the `estimate_fp_area` command is run, it does not reduce the placement blockage. Instead, it blocks some of the core area, preventing the placement of the standard cells.

If there is a hard macro keepout, macros can be packed until the keepout region abuts, as shown in [Figure 3-7](#).

Figure 3-7 Set Keepout Region Packing Limit



- Maintain I/O pad alignment – Maintains I/O pad alignment of designs with complex structures, such as rectilinear I/Os, staggered I/Os, multiring I/Os, and macros, encroaching into the I/O area, and I/O cells encroaching into the core area.

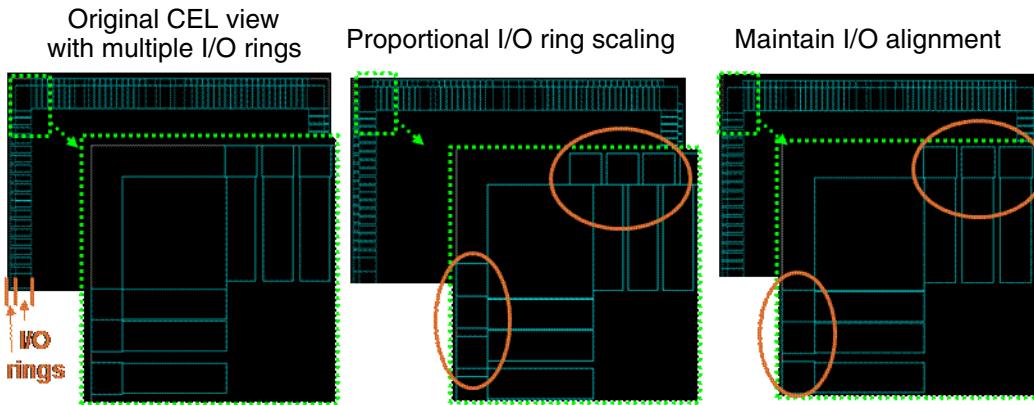
For multiring I/O placement, the `estimate_fp_area` command scales only the spaces that are shared by all I/O rings on the same side of the chip. This helps maintain the desired center or edge alignments between I/O pads across different I/O rings. By default, each I/O ring is scaled separately and proportionally.

[Figure 3-8 on page 3-13](#) shows an example of maintaining I/O pad alignment.

The image on the left is the original CEL view with multiple I/O rings. The pads between the I/O rings are aligned.

When you run the `estimate_fp_area` command with the default options, the command scales the I/O rings proportionally. As a result, the alignment between the I/O pads and the I/O rings is lost, as shown in the center image.

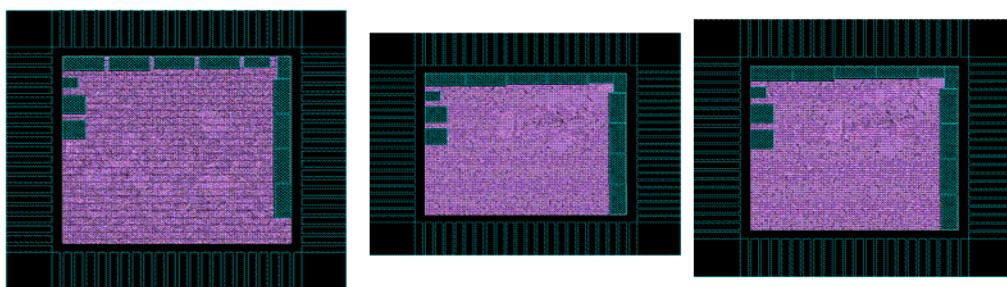
When you run the `estimate_fp_area` command and select the “Maintain I/O pad alignment” option, the shrinkage occurs by eliminating the unnecessary space in the I/O ring and by maintaining the I/O alignment between the I/O rings, as shown in the image on the right.

Figure 3-8 Maintaining I/O Alignment

- Replace I/O – Replaces all the I/O pads and pins based on physical constraints. If the physical constraints file does not exist, the command creates the file with side and order constraints. By default, the relative I/O placement of the input floorplan is maintained.
- Increase area – Reserves some extra space in the core area for cells to be added later in the flow. This increased space can help accommodate any clock tree synthesis or optimization buffers that might be inserted later during detailed implementation. Specify the area in square microns. The default is 0. [Figure 3-9](#) shows an example of reserving some space in the core area for future use.

Figure 3-9 Increasing Space in the Core Area

Original size = 1746 x 1741 Default sizing 1567 x 1490 Extra area 1617 x 1540



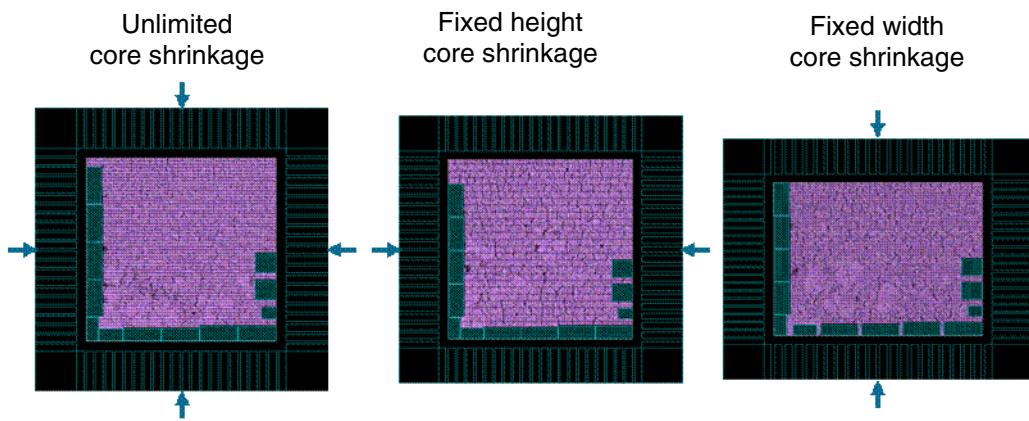
- Core sizing only – Core sizing applies only to full-chip designs. In designs that have complex, custom I/O structures, the tool does not modify the I/O structure. You can shrink the size of the core area only and ignore the I/O pad ring in the input floorplan. Note that if you increase the core size with this option, the core might overlap the I/O structure.

This option is provided to support a what-if type of analysis.

By default, the block or chip shrinks with the core and the I/O pad or pin placement is scaled.

[Figure 3-10](#) shows an example of core sizing within the I/O ring.

Figure 3-10 Core Sizing Only; Core Sizing With Fixed Height and Fixed Width



3. Set the search control options, depending on your requirements.

- Acceptable overflow – This is the main stop criteria. Enter a global route cell (GRC) overflow percentage in the range of 0.0 to 0.25 to determine if the design is routable. When a tested floorplan has the largest achievable global route cell (GRC) acceptable overflow percentage that does not exceed the value you specify, it is saved as the smallest routable floorplan.

Auto Routability – This is the default search control option for estimating routability. No user input is needed, and it runs fast.

Auto routability helps you to better predict a routable floorplan by:

- Predicting the completion rate of detail routing based on global routing results.
- Reducing the pessimism caused by global route cell based local congestion hotspots.
- Predicting the routability more accurately than the global route cell based on analyzing the neighborhoods of individual global route cells.

- Max allowable IR drop value – Allows you to set the acceptable IR drop criteria used by power network synthesis when it creates a power plan during `estimate_fp_area`. This option is not valid unless the “Power and ground net names” text box is also used. To use “Power and ground net names” with the `estimate_fp_area` command, you should first run a trial power network synthesis on the existing floorplan to ensure that all the power network synthesis options are set properly before starting `estimate_fp_area`.

Enter a value in mv units. By default, 10 percent of the power supply voltage is used as the target IR drop value.

4. Set the power network control options, depending on your requirements.

- Automatically create script to mimic power network – Creates a power network synthesis script file that is automatically sourced within MinChip technology to replicate the power structures in the input floorplan for a new die size. The input floorplan should contain a regular power structure with straps or rings. The power structures in the input floorplan are analyzed and the power network synthesis constraints are extracted. Irregular power structures and multivoltage design power structures are not extracted.

MinChip technology creates the `EA_pns_script.tcl` script file in the current directory.

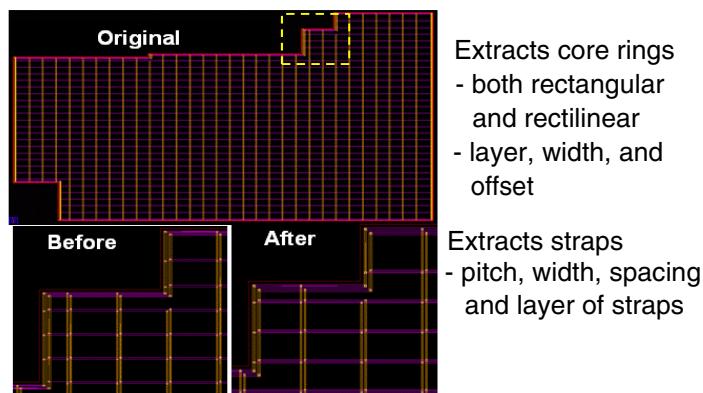
When you select this option, you must also include the power and ground net names.

The extracted power network synthesis script creates core rings with fixed layers and widths, straps with fixed pitch, widths and layers, block rings with fixed layers and widths, and standard cell preroutes.

The die that results should have a power plan that is very similar to that of the input floorplan but scaled to the new die size.

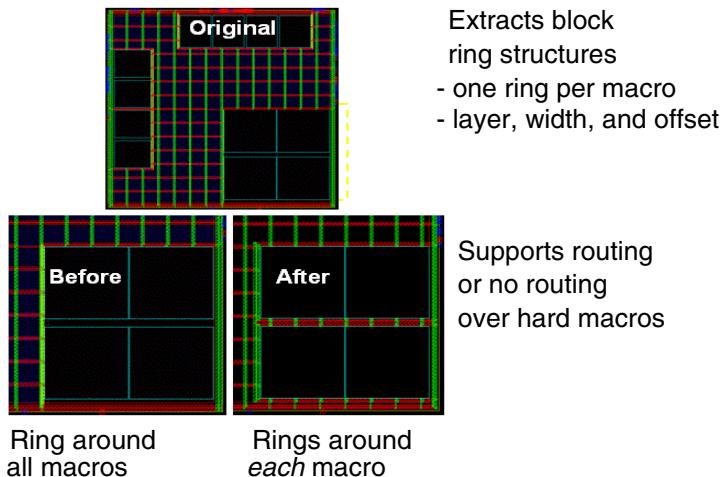
[Figure 3-11](#) shows an example of extracting rectangular and rectilinear core rings, and straps.

Figure 3-11 Extracting the Power Structure: Core Rings and Straps



[Figure 3-12](#) shows an example of extracting block ring structures. Routing or no routing over hard macros is also supported.

Figure 3-12 Extracting the Power Structure: Block Rings and Routing Over Macros



MinChip technology honors the virtual pad locations, which are taken from the power network synthesis constraints. The virtual pad file is read and the locations of the virtual pads are automatically scaled to the new die size.

- Power and ground net pairs as {pwr gnd} – Performs power network synthesis by using the power network synthesis default settings during each new die size exploration. The default power network synthesis settings form a two-layer mesh on the uppermost layers in the technology that meets the specified voltage drop (IR drop) target. If you do not specify a voltage drop target, the default assumes the target is 10 percent of the supply voltages. If this type of power structure does not accurately represent the power structure used for your design, you should use the “Custom PNS script” option.

The power plan routes provide a more realistic routing overflow estimate because the routing resources used by the power preroutes are taken into account during the global route cell overflow calculations.

You must specify a power and ground net name pair. A comma must be used to separate the names of the power net and ground net. For example,

VDD, VSS

By default, no power net synthesis is run.

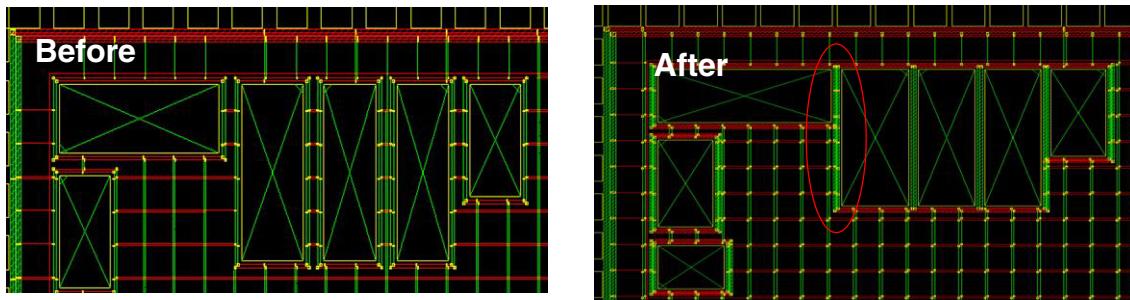
- Custom PNS script – Runs a custom power network synthesis script instead of the default settings used to run power network synthesis.

By default, no power network synthesis is run.

It is important that the script does not contain any absolute coordinates. They do not apply to every design iteration that the `estimate_fp_area` command creates.

If you provide a power network synthesis script file with hard macro block rings, but with no `sliver_size` or macro padding, as shown in [Figure 3-13](#), the `estimate_fp_area` command can reduce the die size. The die size is reduced by eliminating the space between the hard macros until the macro block power network synthesis rings are merged with one set instead of two sets.

Figure 3-13 Estimate Area Run With Power Network Synthesis and Fixed Hard Macros



- Preroute script – Runs a user-defined power routing script used to form preroutes to connect the power and ground pins of standard cells (preroute standard cells). During this process, stacked vias are formed to connect the preroutes to the upper layer power routes, if applicable.

It is important that the script does not contain any absolute coordinates. They do not apply to every design iteration that the `estimate_fp_area` command creates.

By default, no preroutes are created.

5. Save the result of the die size exploration.

- Save as – Explicitly name the saved CEL view. By default, the tool saves the last routable result as a CEL view named *design_name* EstimateArea.

6. Select OK or Apply.

The `estimate_fp_area` command performs the following tasks:

- Saves the starting CEL view with the original cell name and starts working on this CEL view.
- Deletes both signal and power routing.
- Maintains the orientation of the fixed macros; the command does not rotate the macros.

- Tests the routability of the input CEL view by using the current utilization of the CEL core to set the die size. The `estimate_fp_area` command then attempts to make the die size as small as possible.
- Stops if the design is pad limited. If the design becomes pad limited during estimate area iterations, the command stops and informs you that sizing stopped because the design had become pad limited.

If the original design has pad minimum spacing constraints, the command cannot create a pad-limited floorplan; the command cannot abut the pads due to the constraints.

Using the `estimate_fp_area` Command and Options

This section describes how to use the `estimate_fp_area` command and its options. The command options are grouped into three categories:

- Search Control
- Floorplan Control
- Power Network Control

The `estimate_fp_area` command automates the placement and routing iterations targeted at finding the smallest, routable die size. It automatically maintains relative placement of fixed hard macros, uses routing congestion estimation technology to assess routing feasibility, and optionally uses power network synthesis for power routing.

The `estimate_fp_area` command uses the following syntax:

```
estimate_fp_area
[-acceptable_overflow_ratio]
[-core_sizing_only]
[-estimate_optimization]
[-fixed_edges list_of_edges]
[-increase_area area]
[-ir_drop_value target_value]
[-keep_blockages]
[-maintain_iopad_alignment]
[-maintain_power_structure]
[-max_height max_height]
[-max_width max_width]
[-min_height min_height]
[-min_width min_width]
[-power_net_names list_of_names]
[-replace_io]
[-run_pns_script pns_script_name]
[-run_preroute_script preroute_script_name]
[-save_as name]
[-shrink_only]
[-sizing_type fixed_width | fixed_height | fixed_aspect_ratio]
```

Table 3-1 describes the MinChip technology search control options.

Table 3-1 estimate_fp_area Command Options

Command option	Description
<code>-min_height min_height</code>	Specifies the lower bound of the core height, in microns, required to stop the die search process. The command stops reducing the core size if the total utilization for the core is less than the target value you specify.
	By default, no die sizes that have utilizations greater than 95 percent are considered.
<code>-max_height max_height</code>	Specifies the upper bound of the core height, in microns, required to stop the die search process. The command stops increasing the core size if the total utilization for the core is greater than the target value you specify.
	By default, no die sizes that have utilizations greater than 30 percent are considered.
<code>-min_width min_width</code>	Specifies the lower bound of the core width, in microns, required to stop the die search process. The command stops reducing the core size if the total utilization for the core is less than the target value you specify.
	By default, no die sizes that have utilizations greater than 95 percent are considered.
<code>-max_width max_width</code>	Specifies the upper bound of the core width, in microns, required to stop the die search process. The command stops increasing the core size if the total utilization for the core is greater than the target value you specify.
	By default, no die sizes that have utilizations greater than 30 percent are considered.

Table 3-1 estimate_fp_area Command Options (Continued)

Command option	Description
-acceptable_overflow_percentage	<p>This is the main stop criterion for the estimate_fp_area command.</p> <p>Auto Routability: This is the default search control option for estimating routability. No user input is needed and it runs quickly. Auto routability helps you to better predict a routable floorplan by:</p>
	<p>Predicting the completion rate of detail routing based on global routing results</p>
	<p>Reducing the pessimism caused by global route-based local congestion</p>
	<p>Predicting the routability more accurately than the global route cell based on neighborhood global route cell analysis</p>
	<p>Acceptable Overflow: If you specify a value with the -acceptable_overflow option, the estimate_fp_area command specifies a global route cell overflow percentage to determine if the design is routable. This is the main stop criteria for the estimate_fp_area command. The Auto Routability feature is disabled.</p>
	<p>When a tested floorplan has the largest achievable global route cell acceptable overflow percentage without exceeding the value you specify, it is saved as the smallest routable floorplan.</p>
	<p>Enter the value as a floating point number. The default is 0.018 (1.8%).</p>
-ir_drop_value target_value	<p>Specifies the acceptable IR drop criteria used by power network synthesis when it creates a power plan during estimate_fp_area. This option is not valid unless the -power option is used. To use the -power option with estimate_fp_area, you should first run a trial power network synthesis on the existing floorplan to ensure that all the power network synthesis options are set properly before starting estimate_fp_area.</p>
	<p>The value is in mv units. If you do not specify a value, 10 percent of the power supply voltage is used as the target IR drop value.</p>

Table 3-2 describes the MinChip Technology floorplan control options.

Table 3-2 estimate_fp_area Command Options for Floorplan Control

Command option	Description
-sizing_type fixed_width fixed_height fixed_aspect_ratio	Defines constraints on the search space of aspect ratios. Select fixed_width if you want to preserve the original design width and perform sizing in the y-direction only. Select fixed_height if you want to preserve the original design height and perform sizing in the x-direction only. Select fixed_aspect_ratio if you want the estimate_fp_area command to honor the aspect ratio of the original design size and to stop shrinking the width or height of the die when the width or height becomes pad or macro limited.
-increase_area area	By default, the estimate_fp_area command ignores the aspect ratio and continues shrinking the width or height of the die until the width or height becomes pad or macro limited.
	By default, the estimate_fp_area command finds the smallest, routable die size. Specify this option to reserve some extra space for cells to be added later in the flow.
	This increased space can help accommodate any clock tree synthesis or optimization buffers that might be inserted during the detailed implementation.

Table 3-2 estimate_fp_area Command Options for Floorplan Control (Continued)

Command option	Description
-maintain_iopad_alignment	<p>Aligns the I/O pads in different rings when multiring I/O structures are present. For multiring I/O placement, the <code>estimate_fp_area</code> command scales only the spaces that are shared by all I/O rings on the same side of the chip. This helps maintain the desired center or edge alignments between I/O pads across different I/O rings.</p>
	<p>The <code>estimate_fp_area</code> command scales only the spaces that are shared by all I/O rings on the same side of the chip. This helps maintain the desired center or edge alignments between I/O pads across different I/O rings.</p>
	<p>It also handles designs with complex I/O structures, such as, rectilinear I/Os, staggered I/Os, multiring I/Os, and macros, encroaching into the I/O area, and I/O cells encroaching into the core area.</p>
	<p>Shrinkage occurs by eliminating the unnecessary space in the I/O ring and by maintaining the I/O alignment between the I/O rings.</p>
	<p>By default, each I/O ring is scaled separately and proportionally.</p>
-keep_blockages	<p>Keeps the placement blockages in the input floorplan. The blockages are not deleted.</p>
	<p>By default, the <code>estimate_fp_area</code> command deletes all placement blockages in the input floorplan before it starts iterating through design sizes.</p>
-replace_io	<p>Replaces all I/O pads and pins based on physical constraints. If no physical constraints file exists, one is created with side and order constraints.</p>
	<p>By default, the <code>estimate_fp_area</code> command maintains the relative I/O placement of the input floorplan.</p>

Table 3-2 estimate_fp_area Command Options for Floorplan Control (Continued)

Command option	Description
<code>-core_sizing_only</code>	Applies only to full-chip designs. It modifies the size of the core area and ignores the I/O pad ring in the input floorplan. In designs with complex custom I/O structures, the <code>estimate_fp_area</code> command might not modify the I/O structure appropriately. This option can be used to ignore the I/O structure and modify the size of the core only. Note that if you increase the core size, the core might overlap the I/O structure.
<code>-estimate_optimization</code>	By default, the block or chip shrinks with the core and the I/O pad or pin placement is scaled.
<code>-shrink_only</code>	Estimates the number of buffers added during optimization and reserves the area needed for buffer insertion. Therefore, the final die size after running the <code>estimate_fp_area</code> command includes the area needed for the current netlist plus some additional area reserved for optimization. MinChip technology outputs the amount of area reserved for buffer insertion.

[Table 3-3](#) describes the MinChip Technology power network control options.

Table 3-3 estimate_fp_area Command Options for Power Network Control

Command option	Description
<code>-power_net_names {power_net ground_net}</code>	Performs power network synthesis during each new die size exploration in <code>estimate_fp_area</code> . The power plan routes provide a more realistic routing overflow estimate because the routing resources used by the power preroutes are taken into account during the global route cell overflow calculations. You must specify a power and ground net name pair. For example,

```
-power_net_names {VDD VSS}
```

Table 3-3 estimate_fp_area Command Options for Power Network Control (Continued)

Command option	Description
<code>-run_pns_script <i>script_name</i></code>	<p>Performs power estimation with MinChip technology by using a user-defined power network synthesis script instead of the default settings to run power network synthesis.</p> <p>By default, power network synthesis is not run unless you use either the <code>-run_pns_script</code> or the <code>-power_nets</code> option.</p> <p>The script should not contain any absolute coordinates as these absolute coordinates do not apply to the various design sizes that the <code>estimate_fp_area</code> command iterates through.</p>
<code>-run_preroute_script <i>preroute_script_name</i></code>	<p>Performs MinChip estimation by using a user-defined power routing script to form preroutes to connect the power and ground pins of standard cells. During this process, stacked vias are formed to connect the preroutes to the upper layer power routes, if applicable.</p> <p>The script should not contain any absolute coordinates as these absolute coordinates do not apply to the various design sizes that the <code>estimate_fp_area</code> command iterates through.</p>
<code>-maintain_power_structure</code>	<p>Creates a power network synthesis script file that is automatically sourced within MinChip technology to replicate the power structures in the input floorplan for a new die size.</p> <p>The resultant die should have a power plan that is very similar to that of the input floorplan but scaled to the new die size.</p>
<code>-save_as <i>cell_name</i></code>	<p>Use this option to name the saved CEL explicitly.</p> <p>By default, the <code>estimate_fp_area</code> command saves the last routable result as a CEL named <i>design_name</i> EstimateArea.</p>

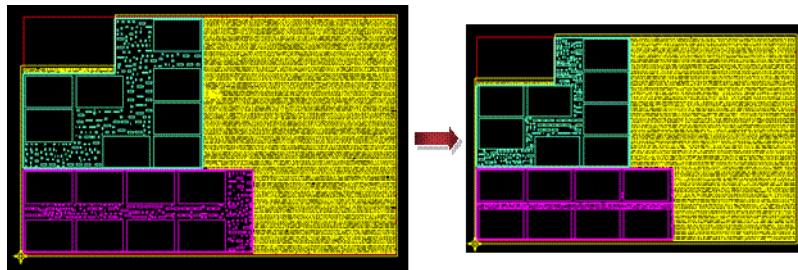
Using MinChip Technology With Multivoltage Designs

MinChip technology supports designs with voltage areas. When the `estimate_fp_area` command iterates through different design size estimates to find the smallest die or block size that meets the routability target, it automatically shrinks rectangular and rectilinear voltage area boundaries by a similar percentage, as shown in [Figure 3-14](#).

The `estimate_fp_area` command shrinks the voltage area boundary by the same amount as the top-level block or cell. The relative macro placement of the top-level hard macros as well as the macros inside the voltage areas is maintained.

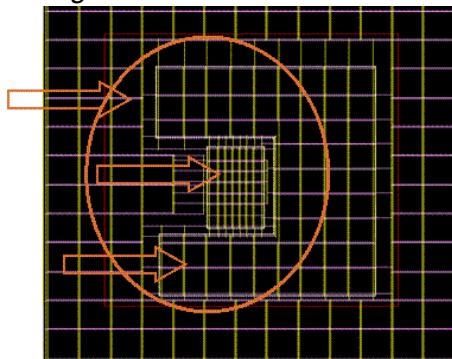
If a voltage area cannot be placed correctly because the utilization bounds are too high, the `estimate_fp_area` command stops shrinking the design. It also ensures that the aspect ratio of the voltage area changes in accordance with any changes to the aspect ratio of the top-level block or cell.

Figure 3-14 Voltage Area Sizing



Based on the power budget requirements, each voltage area in the core area can have different power structures, as shown in [Figure 3-15](#). Therefore, you should use a power network synthesis script because the default run might not generate different power structures for different voltage areas.

Figure 3-15 Multivoltage Design Power Structures



4

Handling Black Boxes

This chapter describes the various operations you can perform with black box modules and cells in IC Compiler, using a virtual flat approach to creating the floorplan.

Milkyway defines a black box as any instance in the netlist that has neither a corresponding leaf cell in a reference library nor a netlist module defined in terms of leaf cells. A black box can have an empty module represented in the netlist where its ports and their directions are defined. If the black box instance in the netlist does not have a corresponding module, the direction of the ports in the black box are defined in the Milkyway database as bidirectional.

A black box can be represented in the physical design as either a soft macro or hard macro. A black box macro has a fixed height and width. A black box soft macro sized by area and utilization can be shaped to best fit the floorplan.

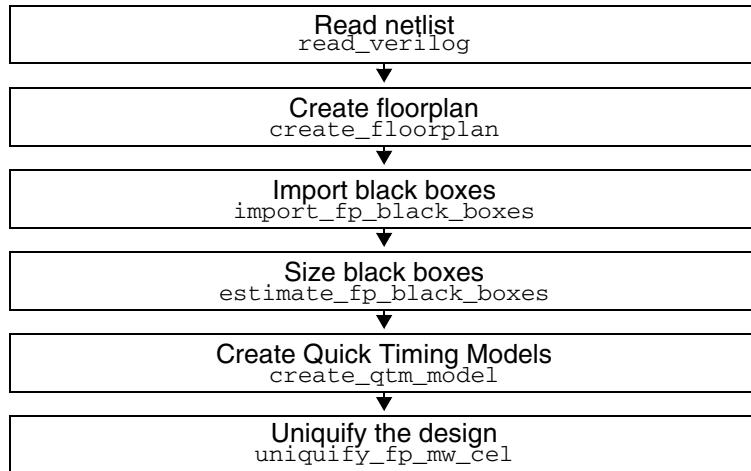
This chapter includes the following sections:

- [Black Box Flow](#)
- [Reading Netlists Containing Black Boxes](#)
- [Importing Black Boxes](#)
- [Sizing Black Boxes](#)
- [Configuring Black Boxes](#)
- [Creating Quick Timing Models](#)
- [Creating Unique Cell Instances](#)

Black Box Flow

The black box flow includes the following steps:

Figure 4-1 Black Box Flow



For information about creating the floorplan, see “[Initializing the Floorplan](#)” on page 2-9. The remaining steps are described in the following sections.

Reading Netlists Containing Black Boxes

To read a Verilog netlist, use the `read_verilog` command.

```
icc_shell> read_verilog verilog_file
```

IC Compiler reads the Verilog netlist and assigns a black box type to each black box module. [Table 4-1](#) describes the different types of black box modules read by the `read_verilog` command.

Table 4-1 Black Box Types

Black box type	black_box_type attribute	Description	Example Verilog module
Data flow	DF	A data flow module contains assign statements for some module ports; other ports are not connected.	module BB_DATAFLOW (IN, OUT); input [7:0] IN; output [7:0] OUT; assign OUT=1'b0; endmodule
Empty	Empty	An empty module contains only port statements; there are no assign or tie-off statements.	module BB_EMPTY (IN, OUT); input [7:0] IN; output [7:0] OUT; endmodule
Feedthrough	Feedthru	A feedthrough module contains assign statements; each input and output is connected.	module BB_FEEDTHRU (IN, OUT); input [7:0] IN; output [7:0] OUT; assign OUT = IN; endmodule
Missing	Missing	A missing module is not defined in the Verilog netlist; you must use the <code>-dirty_netlist</code> option with the <code>read_verilog</code> command to import this type of module.	Module is not in the Verilog netlist
Tie-off	Tie-Off	A tie-off module contains only outputs with each output assigned to either 1'b0 or 1'b1.	module BB_TIEOFF(OUT) ; output [7:0] OUT; assign OUT=1'b0; endmodule

When the `read_verilog` command detects a logical black box module, it sets the `is_logical_black_box` attribute to `true` and the `black_box_type` attribute to DF, Empty, Feedthru, Missing, or Tie-Off. You can query these attributes with the `get_cells` command to identify the logical black boxes in your design. For example, to retrieve all black box modules of type Missing, use the following command:

```
icc_shell> get_cells -hierarchical \
    -filter "is_logical_black_box==true && black_box_type==Missing"
```

Importing Black Boxes

To use black boxes in the design planning flow, you must import them by converting them from logical black boxes to physical black boxes. After conversion to a physical black box, the black box module is treated as a soft macro.

Note:

After you convert the logical black boxes to physical black boxes, you must treat the design as a hierarchical design. This means that you must use the `save_mw_cel -hierarchy` command when you save the design to ensure that both the top-level design and the black box cells are saved.

To convert the black boxes, use the `import_fp_black_boxes` command (or choose Floorplan > Blackboxes > Import Blackboxes in the GUI). You must specify the list of black boxes to convert. On the command line, you can specify the black boxes by using the `get_cells` command with the `is_logical_black_box` and `black_box_type` attributes to identify the black boxes. In the GUI, select the black box instances from the “Importable black boxes” list in the dialog box. To create a CEL view for every black box listed in the “Importable black boxes” list, click Select All.

For example, to create a CEL view for all empty modules, enter the following command:

```
icc_shell> import_fp_black_boxes [get_cells -hier \
    -filter "is_logical_black_box==true && black_box_type==Empty"]
```

The `import_fp_black_boxes` command performs the following attribute changes to the converted black box cells:

- Changes the `is_logical_black_box` attribute to `false`.
- Changes the `is_physical_black_box`, `is_black_box`, and `is_soft_macro` attributes to `true`.

By default, a maximum of 100 black boxes can be imported at one time. If you try to import more than the maximum number of black boxes, the import is canceled. The limit of 100 black boxes prevents you from accidentally selecting all black boxes in a script when you might have forgotten to specify one or more reference libraries. To import more than 100 black boxes, increase the number of importable black boxes by specifying the `-max_count` option or by entering the value in the “Maximum number of black boxes to import” text box in the GUI.

The `import_fp_black_boxes` command sets the size for each black box. By default, IC Compiler creates a bounding box of 300 microns on each side for the black box. You can change the default black box size by using one of the following methods:

- Specify the side length

To create a square boundary with a side length other than 300 microns, use the `-side_length` option.

- Specify the equivalent gate count and utilization

To create a boundary based on the number of equivalent gates in the black box and a utilization value, use the `-gate_count` and `-utilization` options. If you specify only the `-gate_count` option, IC Compiler uses the default utilization value of 0.7 to determine the size of the black box.

Note:

By default, IC Compiler uses the standard gate area of a 2-input NAND gate as the gate-equivalent size. You can change the gate-equivalent size by using the `set_fp_base_gate` command. For information about the `set_fp_base_gate` command, see [“Defining the Size by Gate Equivalents” on page 4-7](#).

Sizing Black Boxes

When you convert a logical black box to a physical black box, IC Compiler assigns a size to the black box. You can change the size of the black box cells to provide a more accurate boundary by using the `estimate_fp_black_boxes` command (or by choosing Floorplan > Blackboxes > Estimate Blackboxes in the GUI). You can specify a boundary for the black box, the number of gate equivalents, or the hard macro contents for the black box. You can also specify that the shape is fixed and have IC Compiler treat the black box as a hard macro rather than a soft macro.

A black box that is converted to a soft macro has a CEL view and is displayed in the layout window with a thick boundary outline. A black box that is converted to a hard macro has a FRAM view and is displayed in the layout window with a thin boundary outline.

Specifying the Black Box Boundary

You can specify either a rectangular black box boundary or rectilinear black box boundary.

Specifying a Rectangular Black Box Boundary

To specify the black box boundary by width and height, specify the width and height of the rectangle in microns by using the `-sm_size {width height}` option with the `estimate_fp_black_boxes` command. In the GUI, select “Soft macro size” and enter the values in the Width and Height text boxes.

By default, IC Compiler uses a utilization value of 1.0 when you specify the `-sm_size` option. You can change the utilization by using the `-sm_util` option with the `estimate_fp_black_boxes` command.

Specifying a Rectilinear Black Box Boundary

To specify a rectilinear black box boundary, specify the coordinates of the polygon by using the `-polygon` option with the `estimate_fp_black_boxes` command. In the GUI, select “Soft macro polygon” and enter the coordinates in the associated text box. Specify the polygon in the following format for both the command line and the GUI:

```
 {{x1 y1} {x2 y2} ...{xn yn} {x1 y1}}
```

The coordinates are in microns.

Sizing Black Boxes by Content

To estimate the size of a black box by its content, IC Compiler uses the area required for any hard macros contained in the black box cell plus the gate-equivalent area.

Defining the Size by Hard Macro Content

To specify the hard macros contained in the black box cell, use the `-hard_macros` option with the `estimate_fp_black_boxes` command or choose “Estimate by enclosed hard macros” in the GUI. IC Compiler uses the area of the specified hard macros, but not the shape, when estimating the black box size. Specify the hard macros in a comma-separated list. You can specify multiple instances of the same macro by appending the macro count in parentheses after the macro reference name. For example, to specify that the alu_bb black box contains two instances of the ram16x128 hard macro and one instance of the ram32x64 hard macro, enter the following command:

```
icc_shell> estimate_fp_black_boxes {alu_bb} \
-hard_macros "ram16x128(2), ram32x64"
```

Defining the Size by Gate Equivalents

IC Compiler computes the gate-equivalent area by multiplying the size of the gate-equivalent cell by the number of cells in the black box and dividing by the utilization. You specify these components in the following manner:

- Gate-equivalent size

The default gate-equivalent size is the standard gate area of a 2-input NAND gate. You can change the gate-equivalent size by using the `set_fp_base_gate` command to specify either a gate from the library or an area.

- To specify a gate from the library, use the `-cell` option. For example, to set the gate-equivalent size to the area of the `nd02` library cell, enter

```
icc_shell> set_fp_base_gate -cell nd02
```

- To specify an area, use the `-area` option. For example, to set the gate-equivalent size to 24 um^2 , enter

```
icc_shell> set_fp_base_gate -area 24
```

- Gate-equivalent count

The gate-equivalent count is the number of gate equivalents that is needed to occupy the same area as the actual cells in the black box. You must specify this value by using the `-sm_gate_equiv` option with the `estimate_fp_black_boxes` command.

- Utilization

The default utilization is 0.7. To change the utilization, use the `-sm_util` option with the `estimate_fp_black_boxes` command.

Fixing the Black Box Shape

By default, you can change the shape of a black box. To prevent changing of the black box shape after size estimation, use the `-fixed_shape` option with the `estimate_fp_black_boxes` command, or choose “Mark as fixed shape after estimation” in the GUI.

Configuring Black Boxes

You can perform additional configuration on the black boxes in your design.

Creating a FRAM View

If you are using a timing-driven flow, you must create a FRAM view for the black box after performing black box pin assignment using the non-timing-driven flow. After you generate the FRAM view, set it as a reference library and continue the flow from timing-driven placement. For information about creating a FRAM view, see the *Library Data Preparation for IC Compiler User Guide*.

Constraining Routing Over Black Boxes

By default, routing is not allowed over black boxes; routing resources are reserved for the contents of the black box. You can make routing resources available over black boxes by setting the `min_layer` and `max_layer` attributes on the black box cell.

For example, to allow routing over a black box on all layers above M6, select the black box in the GUI and enter the following commands:

```
icc_shell> save_mw_cel -hierarchy
icc_shell> set bb_refname [get_attribute \
    [get_selection] ref_name]
icc_shell> open_mw_cel ${bb_refname}
icc_shell> set_attribute [current_mw_cel] min_layer M1
icc_shell> set_attribute [current_mw_cel] max_layer M6
icc_shell> save_mw_cel
icc_shell> close_mw_cel ${bb_refname}
```

Note:

You must first save the top-level design, including the black boxes, before you can set these attributes on black boxes.

Setting Black Box Status

You can set the status of black boxes in your design by using the `set_fp_black_boxes_estimated` and `set_fp_black_boxes_unestimated` commands. These commands set the `sm_estimation_mode` attribute on the specified black boxes. At a later time, you can query the status of the black boxes by using the `get_attribute` command.

The following sequence of commands shows how the `import_fp_black_boxes`, `estimate_fp_black_boxes`, `set_fp_black_boxes_estimated`, and `set_fp_black_boxes_unestimated` commands affect the `sm_estimation_mode` attribute setting.

```
icc_shell> import_fp_black_boxes {alu_blackbox}
icc_shell> echo "sm_estimation_mode = [get_attr {alu_blackbox} \
    sm_estimation_mode]"
sm_estimation_mode = unestimated

icc_shell> estimate_fp_black_boxes {alu_blackbox} -sm_size {500 500}
icc_shell> echo "sm_estimation_mode = [get_attr {alu_blackbox} \
    sm_estimation_mode]"
sm_estimation_mode = area

icc_shell> estimate_fp_black_boxes {alu_blackbox} \
    -hard_macros "ram16x128(1), ram32x64"
icc_shell> echo "sm_estimation_mode = [get_attr {alu_blackbox} \
    sm_estimation_mode]"
sm_estimation_mode = gate_equivalent

icc_shell> set_fp_black_boxes_unestimated {alu_blackbox}
icc_shell> echo "sm_estimation_mode = [get_attr {alu_blackbox} \
    sm_estimation_mode]"
sm_estimation_mode = unestimated

icc_shell> set_fp_black_boxes_estimated {alu_blackbox}
icc_shell> echo "sm_estimation_mode = [get_attr {alu_blackbox} \
    sm_estimation_mode]"
sm_estimation_mode = area
```

Converting Physical Black Boxes to Logical Black Boxes

When you import a black box module, the layout window contains an instance of the physical black box. You can use the `flatten_fp_black_boxes` command to remove the physical black box instance from the top-level design in the layout window and restore it to a logical black box. The CEL view for the black box is retained in the library, but it does not have any effect on the top-level design where it was flattened. After you run the `flatten_fp_black_boxes` command, the logical black box continues to exist as a module in the Verilog netlist output.

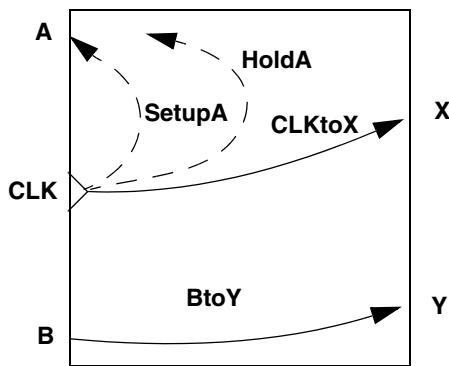
Creating Quick Timing Models

By default, black boxes cannot be used for timing analysis. However, you can enable timing analysis in designs that contain black box cells by setting the `fp_bb_flow` variable to `true` and creating quick timing models for the black boxes.

A quick timing model is an approximate timing model that is useful early in the design cycle to describe the approximate timing of a black box. You create a quick timing model for a black box by specifying the model ports, the setup and hold constraints on the inputs, the clock-to-output path delays, and the input-to-output path delays. You can also specify the loads on input ports and the drive strength of output ports.

[Figure 4-2](#) shows a quick timing model representation of a black box. The constraint arcs appear as dashed lines and the delay arcs appear as solid lines. Port CLK is a clock port, ports A and B are input ports, and ports X and Y are output ports.

Figure 4-2 Quick Timing Model Representation of a Black Box



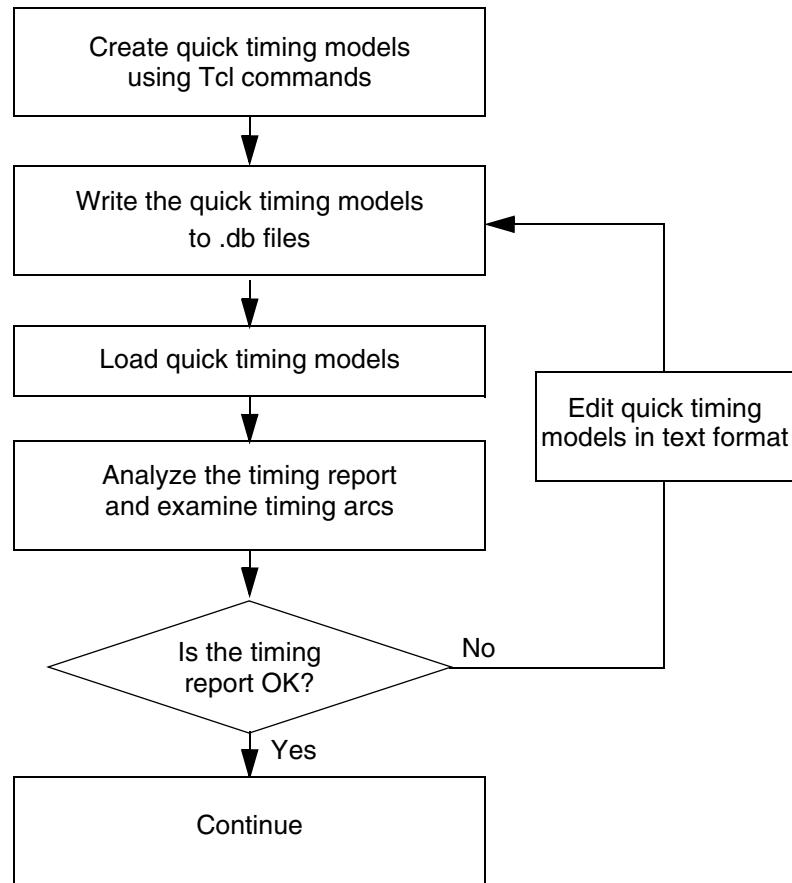
The arcs in this model are

- SetupA, which constrains port A. The constraining port is clock CLK.
- HoldA, which constrains port A. The constraining port is clock CLK.
- CLKtoX, which is a sequential delay arc from CLK to X.
- BtoY, which is a combinational delay arc from B to Y.

Note:

After all of the black boxes in the design are replaced with content, you should reset the `f_p_bb_flow` variable to `false`.

[Figure 4-3 on page 4-11](#) shows the flow used to create a quick timing model for a black box. You create the quick timing model by specifying the model ports, the setup and hold constraints on the inputs, the clock-to-output path delays, and the input-to-output path delays. You can also specify the loads on input ports and the drive strength of output ports. After creating the quick timing model, test it and refine it until the timing report meets your requirements. You can continue to update the black box timing arcs by performing timing budgeting iteratively until the full top-level timing analysis results are clean.

Figure 4-3 Quick Timing Model Flow

To create a quick timing model for a black box,

1. Specify that a new model is being created.

```
icc_shell> create_qtm_model gtm_bb
```

2. Specify the technology information, such as the name of the technology library, the maximum transition time, the maximum capacitance, and the wire load information.

```
icc_shell> set_qtm_technology -library library_name
icc_shell> set_qtm_technology -max_transition trans_value
icc_shell> set_qtm_technology -max_capacitance cap_value
icc_shell> set_qtm_technology -wire_load_model wlm_name
```

3. Specify global parameters, such as setup and hold characteristics.

```
icc_shell> set_qtm_global_parameter -param setup -value setup_value
icc_shell> set_qtm_global_parameter -param hold -value hold_value
icc_shell> set_qtm_global_parameter -param clk_to_output \
-value cto_value
```

4. Define the ports.

```
icc_shell> create_qtm_port -type input in_port
icc_shell> create_qtm_port -type output out_port
icc_shell> create_qtm_port -type inout inout_port
icc_shell> create_qtm_port -type clock clk_port
```

5. Define the timing information for each port.

```
icc_shell> create_qtm_constraint_arc -setup -edge rise \
    -from clk_port -to in_port -value arc_value
icc_shell> create_qtm_constraint_arc -hold -edge rise \
    -from clk_port -to in_port -value arc_value
icc_shell> create_qtm_delay_arc -from_edge rise \
    -from clk_port -to out_port -value arc_value
icc_shell> set_qtm_port_load -value drive_value in_port
icc_shell> set_qtm_port_drive -value drive_value out_port
```

6. (Optional) Generate a report that shows the defined parameters, the ports, and the timing arcs in the quick timing model.

```
icc_shell> report_qtm_model
```

7. Save the quick timing model.

```
icc_shell> save_qtm_model
```

Writing a Quick Timing Model to a .db File

To write out the .db file for the quick timing model, use the `write_qtm_model` command.

```
icc_shell> write_qtm_model -out_dir dp_for_qtm -text
```

Loading a Quick Timing Model

To load the quick timing model, you need to add the .db file to the `link_library` variable, as shown in the following example:

```
icc_shell> lappend link_library "dp_for_qtm/qtm_bb.db"
```

Propagating Timing Information to Soft Macros

In the typical hierarchical design flow, the `commit_fp_plan_groups` command converts plan groups to soft macros and propagates TLUPlus model information to the soft macros. As a result, soft macros in the typical hierarchical design flow contain TLUPlus model information.

In the black box flow, TLUPlus model information is not propagated to the individual black boxes. To use TLUPlus model information, you must manually reload the TLUPlus files by using the `set_tlu_plus_files` command before working on a soft macro created from a black box.

Creating Unique Cell Instances

After importing and sizing the black box cells, use the `uniquify_fp_mw_cel` command to create unique instances of multiply instantiated cells in the design as shown in the following example.

```
icc_shell> uniquify_fp_mw_cel
```


5

Using the On-Demand-Loading Flow for Large Designs

The on-demand-loading flow is a virtual flat design planning flow developed to handle large designs. The basic concept of this flow is to abstract the netlists of the plan groups with smaller netlists that contain the plan group interface logic. These abstractions are also known as reduced plan groups. The on-demand-loading flow enhances capacity and reduces runtime during certain design planning steps in IC Compiler without any degradation in the quality of results.

The concepts and tasks related to using the on-demand-loading flow in IC Compiler are presented in the following sections:

- [Overview of On-Demand-Loading Design Planning Flow](#)
- [Creating On-Demand Netlists](#)
- [Removing On-Demand Netlists](#)
- [Reporting and Querying Information About On-Demand Netlists](#)
- [Cell Placement in the On-Demand-Loading Flow](#)
- [Timing Budgeting Using On-Demand Netlists](#)
- [Committing Plan Groups Within the On-Demand-Loading Flow](#)

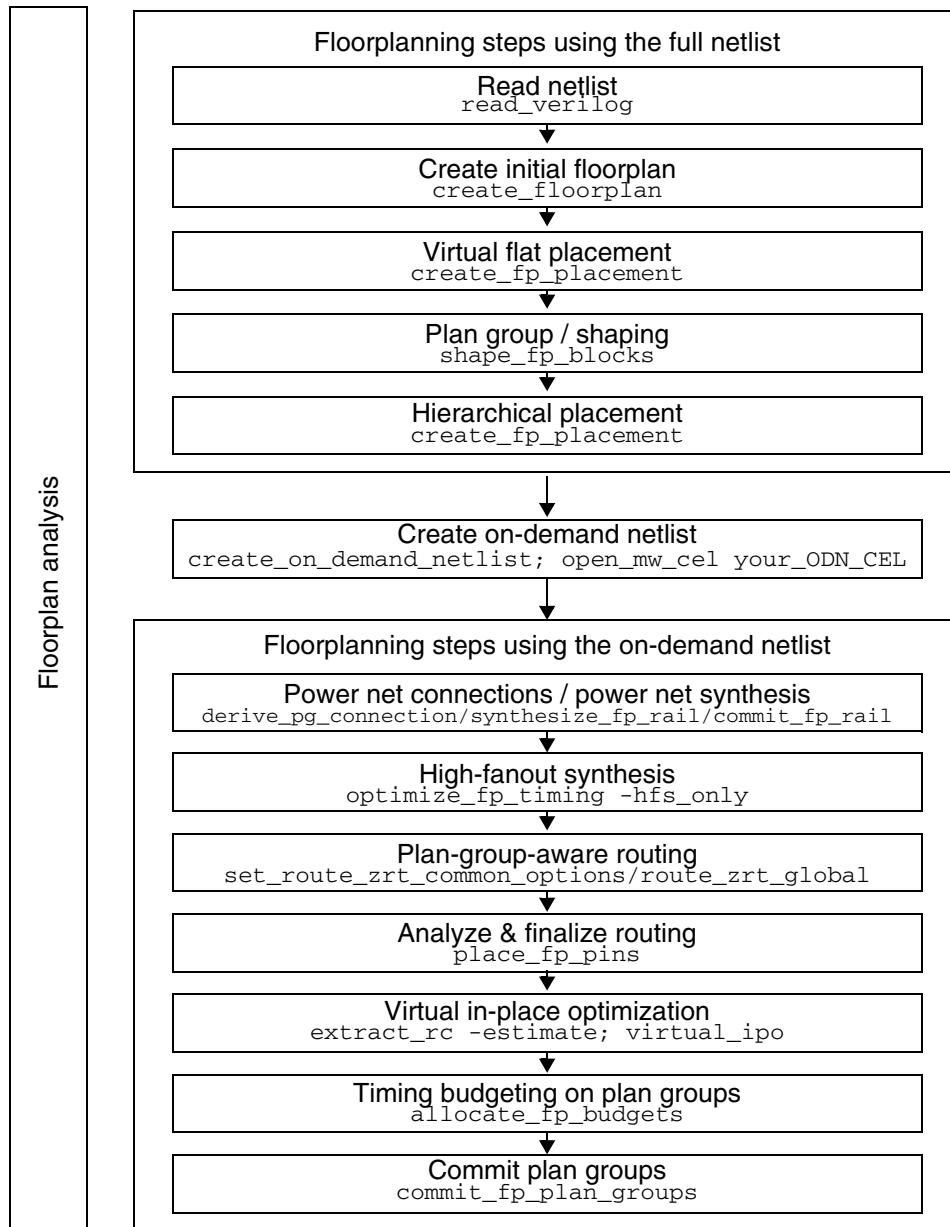
Overview of On-Demand-Loading Design Planning Flow

The on-demand-loading flow is designed to handle design planning for very large designs. This flow involves the following five steps:

1. Create the floorplan using the full netlist. Use the design planning virtual flat commands to create and shape the plan groups, and perform macro placement.
2. Create the on-demand netlist. This is a reduced netlist that contains the complete top-level netlist, the interface netlists of the plan groups, and all the macros in the design.
3. Continue with the rest of the floorplanning steps, such as high-fanout synthesis, routing, in-place optimization, budgeting, using the on-demand netlist.
4. Commit plan groups. In this step, the reduced plan groups are converted back to their corresponding full netlists.
5. Perform block implementation and top-level timing closure tasks as normal.

[Figure 5-1 on page 5-3](#) shows a typical on-demand-loading flow, using the design planning virtual flat commands.

Figure 5-1 A Typical On-Demand-Loading Flow Using the Design Planning Commands



Creating On-Demand Netlists

The on-demand netlist is a reduced netlist that contains the complete top-level netlist, the reduced plan groups, and all the macros in the design. The netlist of the reduced plan groups contain the following components:

- Cells, pins, and nets on the combinational input-to-output paths from the input ports to the output ports
- Cells, pins, and nets on the combinational paths from the input ports to the edge-triggered registers within the plan group
- Cells, pins, and nets on the combinational paths from edge-triggered registers within the plan group to the output ports
- All macro cells residing in the plan groups

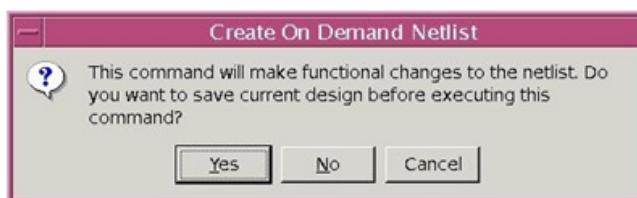
To create the on-demand netlist, use the `create_on_demand_netlist` command as follows or choose Partition > Create On Demand Netlist in the GUI.

```
create_on_demand_netlist
  -plan_groups plan_groups
  [-on_demand_cell cell_name]
  [-full_sdc_file sdc_file_name]
  [-mcmm_map_file mcmm_map_file_name]
  [-mcmm_setup_file mcmm_setup_file_name]
  [-save_current_design]
  [-work_dir dir_name]
  [-host_options host_options_name]
  [-test_host_options]
  [-netlist_reduction_only | -save_block_netlist_only |
   -save_internal_constraints_only]
```

When creating an on-demand netlist, IC Compiler

- Warns you that the tool is about to make a functional change to the netlist by displaying the message box shown in [Figure 5-2](#)

Figure 5-2 Create On Demand Netlist Warning Message Box



- Converts the current Milkyway design to a new Milkyway design that contains a reduced netlist

The tool replaces the full netlist of each plan group with the corresponding reduced plan group netlist. The new Milkyway design contains the complete top level, reduced plan groups netlists, and all the macros.

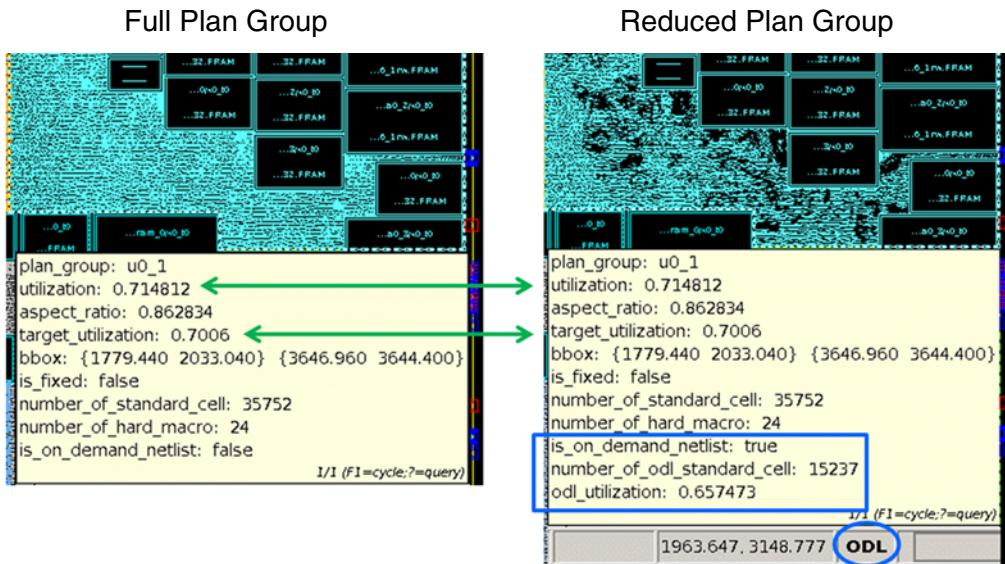
- Saves all the floorplan information, such as the plan group shapes, locations, and placement information, in the internal database
- Saves all the constraints residing in the core of the plan groups in the internal database
- Sets the `is_on_demand_netlist` attribute to `true` for the reduced plan groups
- Closes the design after creating the on-demand netlist

Before you can use this on-demand netlist for the rest of the floorplanning flow, you need to open the CEL view by using the `open_mw_cel your_ODN_CEL` command.

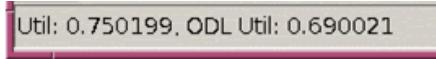
Displaying Plan Group Information in the GUI

The IC Compiler GUI displays utilization information based on the original, nonreduced set of standard cells and macros for designs that contain an on-demand netlist. In [Figure 5-3 on page 5-6](#), the “utilization: 0.714812” entry in the InfoTip in both plan group images refers to the utilization of the plan group before netlist reduction. This number is computed based on the original, nonreduced set of standard cells and macros. The “number_of_odl_standard_cell: 15237” and “odl_utilization: 0.657473” entries in the InfoTip for the reduced plan group are computed based on the standard cell count in the reduced plan group. These two entries are displayed only for designs that contain an on-demand netlist. The GUI displays the “ODL” text on the status bar to indicate that the current design contains reduced plan groups.

The InfoTip also displays the status of the `is_on_demand_netlist` attribute for the plan group under the mouse. The attribute is set to `false` for designs that contain the full netlist, and `true` for designs that contain reduced plan groups.

Figure 5-3 GUI Display for On-Demand-Loading Flow

The lower-left corner of the layout window displays the utilization of both the original and reduced plan groups. By using this feature, you can dynamically resize a reduced plan group and view both utilization numbers at the same time. [Figure 5-4](#) shows the lower-left corner of the layout window that is displayed when you resize a reduced plan group.

Figure 5-4 Display for Plan Group Resize in On-Demand-Loading Flow

Using Distributed Processing in the On-Demand-Loading Flow

The on-demand-loading flow reduces the runtime of the design planning flow. You can further reduce the runtime of this flow by using distributed processing. The `create_on_demand_netlist -host_options option_name` command performs the on-demand-loading flow by using distributed processing. The profile name `-host_options option_name` option selects the distributed processing profile that is used by the command. You must define the distributed processing profile by using the `set_host_options` command. Distributed processing speeds up the creation of the reduced plan groups by creating them in parallel. By default, distributed processing is not used and the reduced plan groups are created sequentially.

The `create_on_demand_netlist` command supports the following mechanisms for distributed processing:

- Load Sharing Facility (LSF)
- Oracle Grid Engine
- rsh

To use distributed processing, you must have at least one of these distributed processing systems configured in your compute environment. The following example uses the `set_host_options` command to enable four distributed processes using Oracle Grid Engine and uses the `create_on_demand_netlist` command to create the on-demand netlist.

```
icc_shell> set_host_options -name my_grd_host_options \
    -pool grd -submit_options {-P bnormal -l arch=glinux, \
    cputype=amd64,mem_free=14G,mem_avail=14G -l \
    qsc=f -cwd}
icc_shell> create_on_demand_netlist -on_demand_cell ORCA_ODL \
    -plan_groups [get_plan_groups *] \
    -work_dir ODL_GRD -host_options my_grd_host_options
```

Note:

Distributed processing requires one IC Compiler license for every four parallel tasks. For example, to run eight parallel tasks, you must have two IC Compiler licenses. In the previous GRD example, if only one IC Compiler license is available and there are more than four plan groups, the tool begins processing four plan groups by using parallel processing. When processing completes for one plan group, the tool submits the fifth plan group to the GRD queue. The tool continues until all plan groups are processed. You can also limit the number of submitted jobs by using the `-num_processes` option to the `set_host_options` command; see the man page for more information about this option.

Another technique to speed up the creation of the reduced plan groups is to use the `-netlist_reduction_only` option. Using this option in conjunction with the distributed processing capability enabled by the `-host_options` option creates the on-demand netlist in the fastest possible manner.

Note:

You must create the full block-level plan group netlists by using the `-save_block_netlist_only` option to commit the design. You must also annotate the full block-level plan group netlists with internal constraints by using the `-save_internal_constraints_only` option before budgeting the design.

The following example illustrates the use of the `create_on_demand_netlist -netlist_reduction_only` command.

```
icc_shell> set_host_options -name my_LSF_host_options -pool lsf \
    -submit_options {-q bnormal -R "rusage[mem=12000] \
    cputype==emt64 cpuspeed==EMT3000 qsc==f"}
icc_shell> create_on_demand_netlist -plan_groups [get_plan_groups] \
    -on_demand_cell Design_ODL \
    -netlist_reduction_only -host_options my_LSF_host_options
```

Note that you can create the block netlist with internal constraints only after performing netlist reduction by using the `create_on_demand_netlist -netlist_reduction_only` command as shown in the previous example. Creating the block netlist allows you to commit the design and proceed with block implementation. To create the block netlist and constraints, use the following `create_on_demand_netlist` commands.

```
icc_shell> create_on_demand_netlist -plan_groups [get_plan_groups] \
    -save_block_netlist_only
icc_shell> create_on_demand_netlist -plan_groups [get_plan_groups] \
    -save_internal_constraints_only -full_sdc_file sdc_file.sdc
```

You can test your distributed processing profile before you create the on-demand netlist by using the `create_on_demand_netlist -test_host_options` command. The `create_on_demand_netlist -test_host_options` command attempts to submit the commands to the distributed processing network by using the distributed processing profile specified by the `-host_options` option.

The following example uses the `create_on_demand_netlist -test_host_options` command to test the `my_host_options` distributed processing profile. The host options test successful message indicates that the test completed successfully.

```
icc_shell> create_on_demand_netlist -plan_groups [get_plan_groups] \
    -on_demand_cell ODN -work_dir ODL_work -netlist_reduction_only \
    -test_host_options -host_options my_host_options
ODL: Work directory is /u/user/ODL_work
Information: Master creating task.
Information: Launching parallel job.
Block: N/A Stage:Launch_Parallel_Job Status: Succeeded : Fri Oct  8 14:28:39 2010
Information: host options test successful.
```

Using Multicorner-Multimode in the On-Demand-Loading Flow

The on-demand-loading flow supports multicorner-multimode (MCMM) designs. If you are performing design planning for a multicorner-multimode design, you must provide an MCMM setup file and an MCMM map file by using the `-mcmm_setup_file` and `-mcmm_map_file` options of the `create_on_demand_netlist` command. Alternately, you can enter the setup file name in the “MCMM setup file name” box in the GUI and enter the map file name in the “MCMM map file name” box in the GUI. The command processes the design by using the setup file and map file names that you specify.

The following example shows the contents of an MCMM setup file, mcmmsetup.tcl.

```
set scenario1 ../inputs/mode_norm.sdc
set scenario2 ../inputs/mode_slow.sdc
create_scenario mode_norm
set_operating_conditions -analysis_type on_chip_variation \
    -max_library lib1 -min_library lib1 -max WCCOM -min WCCOM
set_tlu_plus_files -max_emulation_tluplus ../libs/f_MF.tluplus \
    -max_tluplus ../libs/f.tluplus -tech2itf_map ../libs/star.map_8M
read_sdc $scenario1
create_scenario mode_slow
set_operating_conditions -analysis_type on_chip_variation \
    -max_library lib1 -min_library lib1 -max WCCOM -min WCCOM
set_tlu_plus_files -max_emulation_tluplus ../libs/f_MF.tluplus \
    -max_tluplus ../libs/f.tluplus -tech2itf_map ../libs/star.map_8M
read_sdc $scenario2
```

The MCMM map file contains a listing of scenario names and corresponding variable names that represent the SDC file for the scenario. Note that both the MCMM setup file and the MCMM map file must specify all scenarios, and the variable names must match between the two files. The following example shows the corresponding MCMM map file, mcmm.map, for the MCMM setup file shown in the previous example.

```
mode_norm scenario1
mode_slow scenario2
```

Use the `-mcmm_setup_file` and `-mcmm_map_file` options as shown in the following example to create the on-demand netlist for a multicorner-multimode design. The `report_on_demand_netlist` command is used to check the amount of netlist compression gained by applying the `create_on_demand_netlist` command.

```
icc_shell> create_on_demand_netlist -plan_groups [get_plan_groups] \
    -mcmm_setup_file mcmmsetup.tcl -mcmm_map_file mcmm.map \
    -on_demand_cell ODN
icc_shell> open_mw_cel ODN
icc_shell> report_on_demand_netlist
icc_shell> source -echo -verbose mcmmsetup.tcl
```

Note:

Do not load the SDC constraints into the full design as this increases the memory required to process your design. Use the `-mcmm_setup_file` and `-mcmm_map_file` options of the `create_on_demand_netlist` command instead.

You might see warnings or errors when you source the MCMM setup file. This is due to the netlist objects that were removed by the `create_on_demand_netlist` command. Similar messages occur in the implementation flow when applying the full-chip SDC on a design that contains interface logic models (ILMs) or block abstraction models.

Features Not Supported by the `create_on_demand_netlist` Command

The following features are not supported by the on-demand-loading flow:

- IEEE 1801 Unified Power Format (UPF) objects
- Clock tree planning technology
- DFT operations
- Hierarchical manipulation of the on-demand netlist
- Commands that are not related to design planning in IC Compiler

Removing On-Demand Netlists

To remove the data related to the plan groups in an on-demand netlist, use the `remove_on_demand_netlist_data` command. During the creation of the on-demand netlist, the `create_on_demand_netlist` command saves data about the plan groups, such as the full netlists and the internal constraints. When you do not need the on-demand netlist, you can remove this data by using the `remove_on_demand_netlist_data` command. Note that you must open the Milkyway design library to use the `remove_on_demand_netlist_data` command.

You can also remove the on-demand-loading work directory that was created by the `create_on_demand_netlist -work_dir dir_name` command. The directory is a temporary working directory and is not needed after successful on-demand data creation. To remove the directory, use the `/bin/rm -R directory_name` command in a UNIX shell. By default, the `create_on_demand_netlist` command creates a directory named `ODL_work`.

Reporting and Querying Information About On-Demand Netlists

The `report_on_demand_netlist` command reports the following information about the plan groups in the on-demand netlist:

- List the plan groups in the current design and report which plan groups have the interface netlists created by the `create_on_demand_netlist` command
- Report which plan groups have the netlists saved during the on-demand netlist creation
- Report which plan groups have the internal SDC constraints saved during the on-demand netlist creation

- Report the standard cell and macro cell count for the original plan group
- Report the standard cell and macro cell count for the reduced, on-demand plan group
- Report the compression for the reduced plan group
- Report the area utilization for the original plan group
- Report other statistics such as top-level cell count, top-level plus on-demand plan group cell count, top-level plus original plan group cell count, and overall compression by using the on-demand netlist

Example 5-1 shows an example of the information reported by the `report_on_demand_netlist` command.

Example 5-1 Information Reported by report_on_demand_netlist

```
*****
Report      : report_on_demand_netlist
Design      : DesignA_ODL
Version     : F-2011.09
Date        : Tue Jul 26 17:08:14 2011
*****
Design DesignA_ODL is an On-Demand Netlist.

Plangroup  On Demand Block   Internal   On Demand Original   Compression  Original
          Netlist  Netlist  Constraints Cell Count Cell Count
-----
BlockA    true     true    true       78174    570344   0.86      59%
BlockB    true     true    true       78196    570344   0.86      59%
Audio     true     true    true      156296   1140688   0.86      59%
Video     true     true    true      312636   2281376   0.86      61%
BlockC    true     true    true      20388    1319456   0.98      62%
BlockD    true     true    true      20388    1319456   0.98      62%
BlockE    true     true    true      37697    3371188   0.99      60%
BlockF    true     true    true      37697    3371188   0.99      60%
-----
```

Top design leaf cell count statistics:

```
-----
PlanGroups (PGs)           :      8
Top Level Cell Count      : 1962607
Top + On Demand PG Cell Count : 2704079
Top + Original PG Cell Count : 15906647
Overall Compression        : 0.83
```

To check if the current design in memory is an on-demand netlist, use the `query_on_demand_netlist` command. This command returns a value 1 if the design is an on-demand netlist and returns a value of 0 otherwise.

Cell Placement in the On-Demand-Loading Flow

In the virtual flat placement flow, the `create_fp_placement` command spreads standard cells evenly throughout the design area. However, the on-demand-loading flow uses an on-demand netlist that does not require the even distribution of standard cells within the reduced plan groups. For these plan groups, IC Compiler gives higher priority to wire length

minimization and lower priority to the even distribution of standard cells. You can adjust the balance between wire length minimization and the even distribution of standard cells by setting the `placer_max_cell_density_threshold` variable.

The default `placer_max_cell_density_threshold` variable setting is 0.7 in the on-demand-loading flow. The following command lowers the priority of wire length minimization and increases the priority of the even distribution of standard cells:

```
icc_shell> set_app_var placer_max_cell_density_threshold 0.6  
0.6  
icc_shell> create_fp_placement
```

Note that the `placer_max_cell_density_threshold` variable does not affect the placement of the top-level cells in the on-demand-loading flow; the top-level cells are evenly distributed.

Timing Budgeting Using On-Demand Netlists

After you create the on-demand netlist, you can continue with the virtual flat design flow using the reduced netlist as shown in the flow diagram in [Figure 5-1 on page 5-3](#). During the budgeting step, the `allocate_fp_budgets` command checks the `is_on_demand_netlist` attribute on the plan groups in the current design. If a plan group is identified as a reduced netlist, the budgeter searches for its corresponding full netlist and internal constraints saved by using the `create_on_demand_netlist` command. The budgeter combines the constraints from the reduced netlist and the internal constraints from the full netlist to create the timing budget for the plan group. This complete set of budgeting constraints is passed to the next step for committing the plan groups.

You can perform timing budgeting on multiple scenarios in the on-demand-loading flow by specifying the `allocate_fp_budgets -add_scenarios` command. For more information, see [“Performing Budgeting on Multiple Scenarios” on page 12-25](#).

Committing Plan Groups Within the On-Demand-Loading Flow

During the plan group commit step within the on-demand-loading flow, the `commit_fp_plan_groups` command automatically checks if a plan group to be committed has the `is_on_demand_netlist` attribute. If a plan group is found to have a reduced netlist with only the interface logic, the plan group is committed into a soft macro with its corresponding full netlist. This full netlist is used to complete the commit plan group process.

Additional feedthroughs can be created in the plan groups during the on-demand-loading flow. These feedthroughs are not present in the original full netlist. In the commit plan group process, they are added to the final committed netlist of the plan group.

If you have run optimization in the on-demand-loading flow, only changes to buffering on the feedthrough nets are retained. All other changes are lost. Block implementation reoptimizes the netlist based on the created budgets. The budgets are valid because budgeting is done on the optimized netlist before the commit step.

Placement of the cells that were part of the on-demand netlist is maintained after the commit step. The placement of cells that were not in the on-demand netlist is not maintained. However, these cells do reside inside the plan group boundary; they are grouped together near the lower-left corner of the plan group.

6

Virtual Flat Placement

Virtual flat placement is the simultaneous placement of standard cells and macros for the whole chip. The initial virtual flat placement is very fast and is optimized for wire length, congestion, and timing. For designs with macros, plan groups, or voltage areas that have not already been placed, virtual flat placement can help you decide on the locations, sizes, and shapes of the top-level physical blocks.

This chapter includes the following sections:

- [Virtual Flat Placement Overview](#)
- [Virtual Flat Placement Command](#)
- [Virtual Flat Placement Strategy Options](#)
- [Hard Macro Constraints](#)
- [Blockages, Margins, and Shielding](#)
- [Relative Placement Groups](#)
- [Placement Evaluation](#)
- [Floorplan Editing](#)
- [Voltage Area Planning](#)
- [Placing Multiply Instantiated Modules](#)

Virtual Flat Placement Overview

Virtual flat placement is the simultaneous placement of standard cells and macros for the whole chip; it is a fast initial flat placement performed for design planning purposes. For designs with macros, plan groups, or voltage areas that have not already been placed, virtual flat placement can help you decide on the locations, sizes, and shapes of the top-level physical blocks. This placement is “virtual” because it temporarily considers the design to be entirely flat, without hierarchy. After you decide on the shapes and locations of the physical blocks, you restore the design hierarchy and proceed with the block-by-block physical design flow.

You can control the process of virtual flat placement to emphasize the most important qualities needed for your design, such as timing, congestion avoidance, hierarchical gravity, pin locations, scan chain connectivity, or fast turnaround. Because virtual flat placement is relatively fast, you can explore different tradeoffs between these qualities before you finalize the floorplan.

Virtual flat placement is typically done after the floorplan has been initialized with the `create_floorplan` command or by using Floorplan > Create Floorplan in the GUI, but before any macros or standard cells have been placed. The chip boundary should be defined, the I/O pad cells placed, and the site rows defined. The timing constraints should be set and meeting these constraints should be reasonably achievable. The power structure is typically not yet defined, except possibly for the I/O pads.

Using virtual flat placement typically consists of the following steps:

1. Set the virtual flat placement strategy options by using the `set_fp_placement_strategy` command.
2. Set the macro placement options by using the `set_fp_macro_options` command.
3. Define any necessary relative placement constraints.
4. Set the blockage, margin, and shielding options.
5. Perform virtual flat placement by using the `create_fp_placement` command.
6. Analyze the results for timing and congestion. Perform manual editing, if applicable.
7. If the results are not satisfactory, repeat steps 1 through 6.

Virtual Flat Placement Command

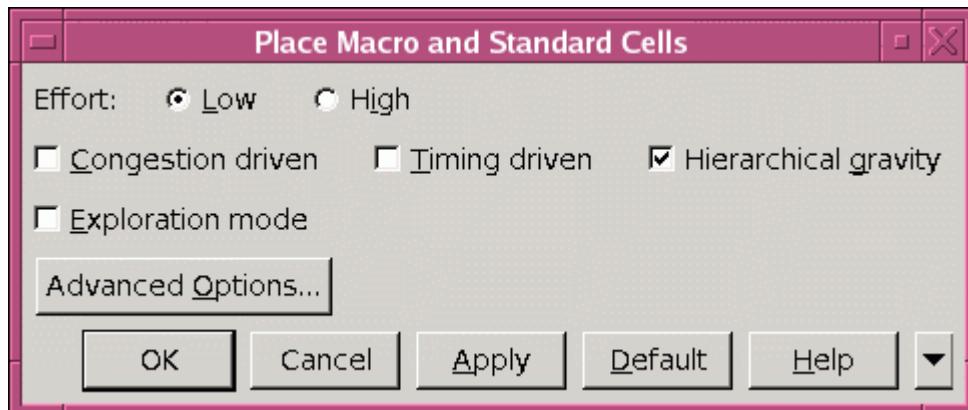
You can perform virtual flat placement by using the `create_fp_placement` command:

```
create_fp_placement
[-effort low | high]
[-max_fanout positive_integer]
[-no_hierarchy_gravity]
[-no_legalize]
[-incremental placement_string]
[-congestion_driven]
[-timing_driven]
[-num_cpus number_of_cpus]
[-plan_groups collection_of_plan_groups]
[-voltage_areas collection_of_voltage_areas]
[-optimize_pins]
[-write_placement_blockages]
[-exploration]
```

For information about the command options, see the `create_fp_placement` man page.

To perform virtual flat placement using the GUI layout window, first set the window task mode to Design Planning if it is not already in that mode (File > Task > Design Planning). Then choose Placement > Place Macro and Standard Cells. This opens the dialog box shown in [Figure 6-1](#).

Figure 6-1 Place Macro and Standard Cells Dialog Box



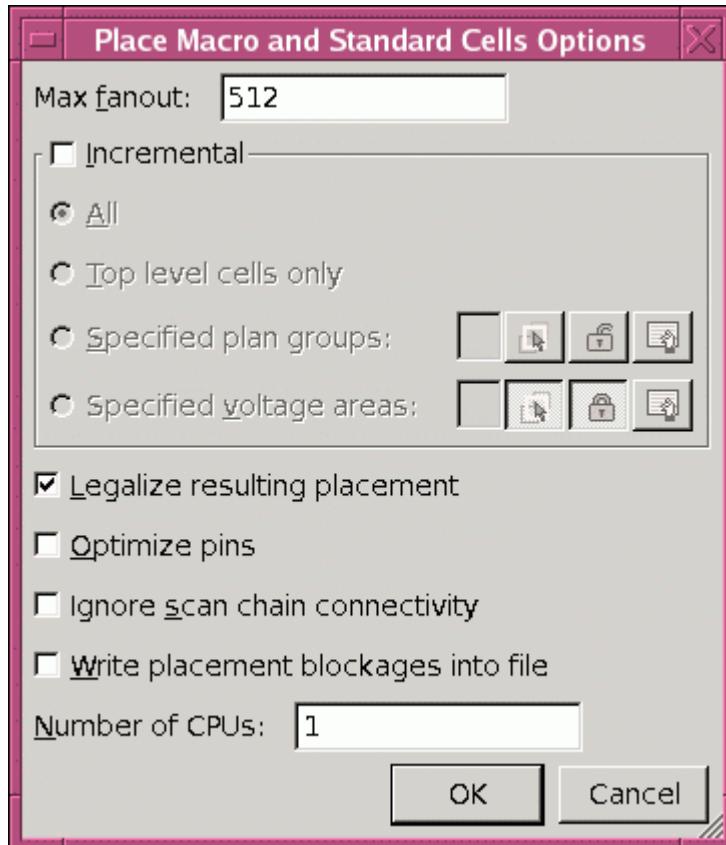
You can set the effort level to Low (the default) or High. The Low effort mode produces good placement results for hard macros and plan groups. Select the High effort mode for even better quality of results at the cost of more runtime.

Select one or more of the “Congestion driven,” “Timing driven,” and “Hierarchical gravity” options to specify the types of placement optimization to apply. Select “Exploration mode” to perform very fast placement, without legalization, so that you can quickly explore various placement alternatives. You can read more about these options in the following sections:

- [“Congestion-Driven Placement” on page 6-6](#)
- [“Timing-Driven Placement” on page 6-6](#)
- [“Hierarchical Gravity” on page 6-6](#)
- [“Exploration Mode” on page 6-7](#)

Click the Advanced Options button to open the dialog box shown in [Figure 6-2](#).

Figure 6-2 Virtual Flat Placement Advanced Options Dialog Box



In the “Max fanout” box, enter the maximum fanout that you want to be considered during placement optimization. The placer does not attempt to reduce the wire lengths of any net that has a fanout greater than this number.

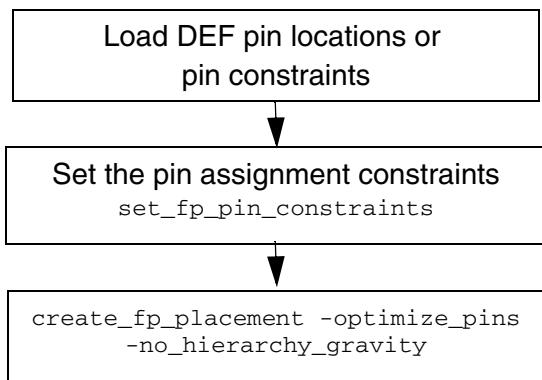
Select the “Incremental” option to perform incremental virtual flat placement on a design that has already had virtual flat placement performed, and you want the placer to keep the placement of existing cells where possible. You can choose the scope of the design on which to perform incremental placement:

- All – The whole design
- Top level cells only – Top-level cells that do not belong to any plan group or voltage area
- Specified plan groups – Cells belonging to specified plan groups
- Specified voltage areas – Cells belonging to specified voltage areas

By default, placed cells are legalized by resolving overlaps and moving standard cells to legal sites. To speed up the design planning flow, you can disable legalization of standard cell placement and use the unlegalized design during exploration mode plan-group-aware routing. To prevent legalizing, uncheck the “Legalize resulting placement” option. To disable legalization before high-fanout synthesis and in-place optimization, set the `fpopt_no_legalize` variable to `true`. After you get the desired floorplan, you can legalize the placement separately by using the `legalize_placement` command.

If you are performing virtual flat placement for a block and not for the whole chip, you can optimize the I/O pin locations by selecting the “Optimize pins” option. In that case, the placer locates the pins to minimize wire lengths, while observing any pin constraints that have been set for the block. Performing placement and pin assignment simultaneously results in smaller wire lengths for nets connected to the constrained pins. [Figure 6-3](#) shows the overall flow for using simultaneous placement and pin assignment at the block level.

Figure 6-3 Block-Level Simultaneous Placement and Pin Assignment Flow



Select the “Ignore scan chain connectivity” option to have the placer ignore scan chain connections during placement.

Select the “Write placement blockages into file” option to have the placer write all the internally computed placement blockages into a file called `design_planning_blockages.tcl`. These blockages include macro padding and channels narrower than the sliver size.

In the “Number of CPUs” box, enter the number of CPUs to be used in parallel to work on virtual flat placement.

Congestion-Driven Placement

The congestion-driven virtual flat placement flow is disabled by default. To enable it, use the `create_fp_placement -congestion_driven` command or select the “Congestion driven” option in the Place Macro and Standard Cells dialog box.

With congestion-driven placement, the placer makes a greater effort to reduce congestion at the cost of runtime. It places cells close together to minimize wire length, but in congested areas, it adds keepout padding around cells and moves the cells to make room for routing. This can affect the amount of area occupied by macro arrays.

The keepout areas generated during congestion-driven placement are automatically removed after placement is complete. If you plan to do incremental placement later, you can save the generated keepouts by using the `-write_placement_blockages` option of the `create_fp_placement` command or by selecting the “Write placement blockages into file” option of the advanced options dialog box. Later, when you want to perform incremental virtual flat placement, source the saved file, `design_planning_blockages.tcl`.

Timing-Driven Placement

The timing-driven virtual flat placement flow is disabled by default. To enable it, use the `create_fp_placement -timing_driven` command or select the “Timing driven” option in the Place Macro and Standard Cells dialog box.

With timing-driven placement, the placer makes a greater effort to improve timing by keeping cells closer together along critical paths, including both standard cells and macro cells. This takes more runtime because timing analysis must be performed along with placement.

Hierarchical Gravity

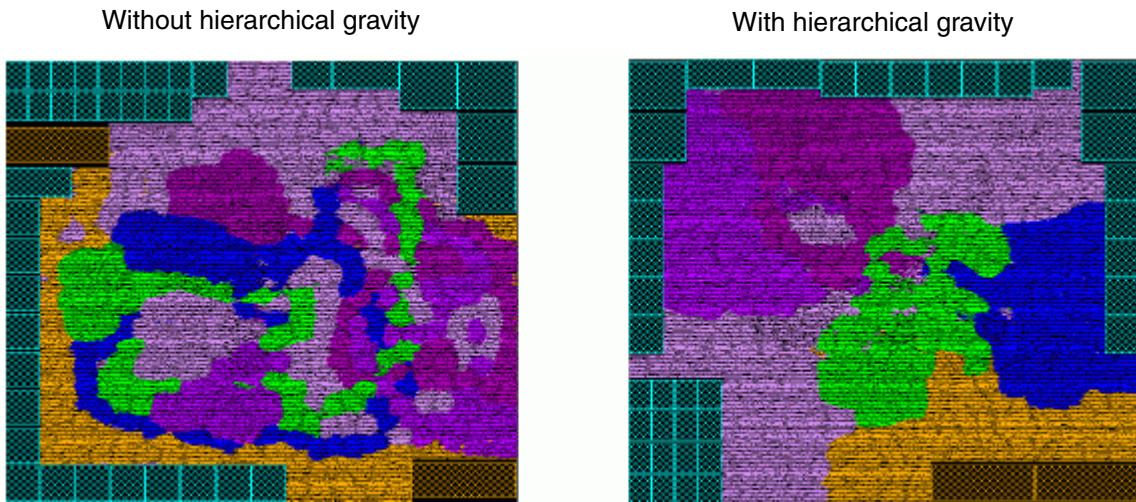
The hierarchical gravity option is enabled by default. To disable it, use the `create_fp_placement -no_hierarchy_gravity` command or uncheck the “Hierarchical gravity” option in the Place Macro and Standard Cells dialog box.

The hierarchical gravity option tends to keep the cells belonging to each hierarchical design logic block physically together. This type of grouping usually results in better placement because designs tend to partition well along hierarchical boundaries. If there are no user-created plan groups, the placement engine analyzes the logical hierarchy of the design and uses blocks with reasonable sizes as the default plan groups.

You should disable hierarchical gravity if the design does not have hierarchy or if you know that the existing logical hierarchy does not represent a good physical partitioning of the design. Disabling hierarchical gravity saves runtime and allows the placement engine more flexibility in placing the cells of a hierarchical block.

[Figure 6-4](#) shows the effects of hierarchical gravity on virtual flat placement. The colored regions represent the areas occupied by standard cells belonging to different hierarchical blocks.

Figure 6-4 Placement With and Without Hierarchical Gravity



Exploration Mode

The exploration mode is disabled by default. To enable it, use the `create_fp_placement -exploration` command or select the “Exploration mode” option in the Place Macro and Standard Cells dialog box.

In the exploration mode, the placer performs virtual flat placement very fast so that you can quickly explore different placement strategies. It uses the low-effort mode for speed and places most cells coarsely, without considering timing or congestion, and does not legalize the placement.

In a typical placement exploration flow, you perform an initial virtual flat placement in exploration mode to determine the best plan group sizes, shapes, and locations. Using that information, you create and shape the plan groups by using the `create_plan_groups` and `shape_fp_blocks` commands. After that, you run virtual flat placement in exploration mode again to get good-quality placement of hard macros, top-level cells, and interface cells.

During the initial flat placement stage before plan group creation, the exploration mode groups together logic from the same hierarchy as in the standard design planning flow, although overlaps can occur because the placement is not legalized. This placement provides sufficient information to determine plan group sizes, shapes, and locations.

At the hierarchical placement stage, when plan groups are in the core, the exploration mode performs good-quality placement of top-level interface cells and macro cells. It places hard macros with no overlap, but standard cells might overlap. It applies more effort to the placement of top-level cells, interface cells, and any hard macros inside the plan groups. At the end of hierarchical placement in exploration mode, the placement of hard macros, top-level cells, and interface cells has the same quality as regular hierarchical placement. The placement results are sufficient for you to perform top-level routability analysis.

Note:

The fast exploration flow does not support multiply instantiated modules, black boxes, or multicorner-multimode operation, and it cannot be used with the `-effort high`, `-timing_driven`, `-congestion_driven`, `-incremental`, or `-optimize_pins` options.

To get the best results in the least amount of time, observe the following guidelines in the exploration flow:

- High-fanout synthesis

To perform high-fanout synthesis after virtual flat placement, use the `optimize_fp_timing -hfs_only` command. When you use this command in the exploration flow (after `create_fp_placement -exploration`), the tool performs high-fanout synthesis in incremental mode, which is more efficient for analyzing the clumped placement of existing buffers, and does not legalize the newly added buffers.

- Routing

The `route_zrt_global -exploration true` command performs fast, low-effort routing for design exploration.

- Optimization

The `optimize_fp_timing -feedthrough_buffering_only` command, followed by virtual in-place optimization, performs optimization without running the complete in-place optimization flow. The `virtual_ipo` and `virtual_ipo -end` commands perform in-place virtual timing optimization for fast optimization during design exploration.

- **Timing budgeting**

The `allocate_fp_budgets -exploration` command performs fast timing budgeting during design exploration.

The following script shows a typical command sequence in a timing-driven exploration flow.

```
read_verilog ...
derive_pg_connection ...
create_floorplan

# Perform clumped, nonlegalized placement, hard macros overlapped
create_fp_placement -exploration
...
shape_fp_blocks
...
# Perform clumped, nonlegalized placement, hard macros NOT overlapped
create_fp_placement -exploration

# Power network synthesis
derive_pg_connection
synthesize_fp_rail ...
commit_fp_rail

# High-fanout synthesis, incremental, no legalization
optimize_fp_timing -hfs_only

# Plan-group-aware routing
mark_clock_tree
set_route_zrt_common_options -plan_group_aware ...

# Routing in exploration mode, routing done on nonlegalized cells
route_zrt_global -exploration true
place_fp_pins [get_plan_groups *]

# Virtual in-place optimization
optimize_fp_timing -feedthrough_buffering_only
virtual_ipo
...
virtual_ipo -end

# Timing budgeting
allocate_fp_budgets -exploration

# Commit plan groups
command_fp_plan_groups
```

Simultaneous Placement and Pin Assignment

You can simplify your design flow by performing simultaneous placement and pin assignment for block-level designs. To do this, choose Placement > Place Macro and Standard Cells, click Advanced Options, and then select “Optimize pins” in the dialog box.

Alternatively, you can use the `create_fp_placement -optimize_pins` command.

The placement engine places the cells near the port locations according to the pin constraints (side, side and order, and side and location) that you have set. The result is a virtual flat placement and pin assignment for block-level designs. Therefore, the overall quality of simultaneous placement and pin assignment is superior to running placement and pin assignment separately. The quality is measured by a smaller wire length of nets connected to the constrained pins.

[Figure 6-3 on page 6-5](#) shows the design flow for the simultaneous placement and pin assignment for block-level designs.

Virtual Flat Placement Guidelines

For better results, observe the following guidelines for virtual flat placement:

- Place and fix any macros that must be located in specific positions before invoking virtual flat placement.
- Disable the hierarchical gravity feature by using `create_fp_placement -no_hierarchy_gravity` if the design does not have hierarchy.
- Use the default dialog box settings and commands settings if you are not sure about what settings to use. Use lower-effort settings initially and higher-effort settings for detailed floorplanning.
- Consider using high-effort settings for small or medium-sized designs, even at early stages of physical prototyping, for higher quality of results at the cost of slightly longer runtimes.
- Increase the pin-count-based macro padding to at least 1.0 track per pin (`set_keepout_margin -tracks_per_macro_pin 1.0`) to prevent congestion when using the macros on edge feature (`set_fp_placement_strategy -macros_on_edge`).

Virtual Flat Placement Strategy Options

You perform virtual flat placement by using the `create_fp_placement` command or by choosing Placement > Place Macro and Standard Cells in the GUI. The command options let you select the effort level, the types of optimization performed, and a few other options.

Several additional virtual flat placement strategy options are available by using the `set_fp_placement_strategy` command. These settings are not saved in the Milkyway database; you must set placement strategy options during each session. This is the command syntax:

```
set_fp_placement_strategy
  [-adjust_shapes on | off]
  [-auto_grouping none | user_only | low | medium | high]
  [-block_constraint_file file_name]
  [-congestion_effort low | medium | high]
  [-default]
  [-fix_macros none | soft_macros_only | all]
  [-flip_chip off | on | honor_reserved]
  [-force_auto_detect_hierarchy on | off]
  [-hierarchy_gravity_blocks list_of_names]
  [-hierarchy_gravity_multi_level on | off]
  [-honor_mv_cells on | off]
  [-IO_net_weight weight]
  [-macro_orientation automatic | all | orientation]
  [-macro_setup_only on | off]
  [-macros_on_edge on | off | auto]
  [-min_distance_between_macros distance]
  [-macros_min_distance_applies_to collection]
  [-pin_routing_aware on | off]
  [-plan_group_interface_net_weight weight]
  [-sliver_size distance]
  [-snap_macros_to_user_grid on | off]
  [-spread_spare_cells on | off]
  [-virtual_IPO on | off]
```

After you set these options, the settings apply to future virtual flat placement performed by using the `create_fp_placement` command or by choosing Placement > Place Macro and Standard Cells in the GUI. To view the current settings for these options, use the `report_fp_placement_strategy` command. For example,

```

icc_shell> report_fp_placement_strategy
...
*** Macro related parameters ***
set_fp_placement_strategy -macro_orientation automatic | all | N
    current setting: automatic
    default setting: automatic
...

*** Congestion driven placement related parameters ***
set_fp_placement_strategy -congestion_effort low | medium | high
    current setting: low
    default setting: low
...

*** Net weighting related parameters ***
set_fp_placement_strategy -IO_net_weight <float>
    current setting: 1.00
    default setting: 1.00
...
...

```

The virtual flat placement strategy options are divided into the following categories:

- [Macro-Related Options](#)
- [Congestion-Related Options](#)
- [Net Weighting Options](#)
- [Miscellaneous Options](#)

Macro-Related Options

These are the strategy options that control hard macro placement:

```

set_fp_placement_strategy
[-auto_grouping none | user_only | low | medium | high]
[-fix_macros none | soft_macros_only | all]
[-macro_orientation automatic | all | orientation]
[-macro_setup_only on | off]
[-macros_min_distance_applies_to collection]
[-min_distance_between_macros distance]
[-macros_on_edge on | off | auto]
[-pin_routing_aware on | off]
[-sliver_size distance]
[-snap_macros_to_user_grid on | off]

```

Additional macro placement constraints can be set with the `set_fp_macro_options` command. For details, see “[Hard Macro Constraints](#)” on page 6-26.

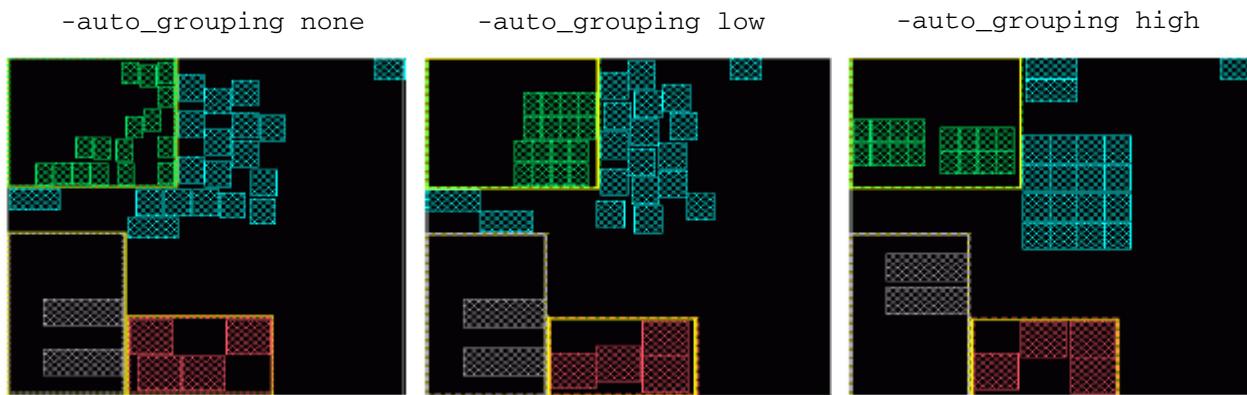
Automatic Grouping of Macros Into Arrays

The `-auto_grouping` option controls the packing of macros into arrays. It has five possible settings:

- `none` – No macros are placed into arrays. User-defined arrays are not honored.
- `user_only` – Only user-defined arrays are honored.
- `low` (the default behavior) – Small macros of the same reference cell are arrayed, but larger macros are not arrayed. User-defined arrays are honored.
- `medium` – Small and medium-sized macros of the same reference cell are arrayed, but larger macros are not arrayed. User-defined arrays are honored.
- `high` – Small macros and some larger macros are arrayed. User-defined arrays are honored.

[Figure 6-5](#) shows examples of macro array grouping results for the `none`, `low` (default), and `high` settings for the `-auto_grouping` option.

Figure 6-5 Macro Array Grouping Options



Placing macros into arrays reduces the scattering of macros, leaving larger continuous areas for other placement. It can reduce wire lengths when a single net drives the same pin of multiple macros in the array. Automatic grouping of macros is hierarchy-aware; macros in the same physical hierarchy are more likely to be arrayed.

Fixing Macros in Place

The `-fix_macros` option specifies which types of macros, if any, should be fixed in position (not moved) during virtual flat placement. It has three possible settings:

- `none` (the default behavior) – None of the macros are fixed in position. All macros can move during virtual flat placement.

- `soft_macros_only` – Soft macros are fixed in position. Hard macros can move during virtual flat placement.
- `all` – All macros are fixed in position and cannot move during virtual flat placement.

This option setting affects macros during virtual flat placement only. The `is_fixed` attribute status of each macro is not affected. Optimization commands used later in the flow might still move macros that are fixed during virtual flat placement.

Macro Orientation

The `-macro_orientation` option can be used to restrict the allowed orientation of macros during virtual flat placement. It can be set as follows:

- `automatic` (the default behavior) – The tool determines a good orientation to use for each reference library cell based on the pin locations and uses that same orientation for all instances of that macro.
- `all` – The tool can use any orientation for each macro instance.
- `N` – All macros are placed in the north orientation.

This option setting affects macros during virtual flat placement only.

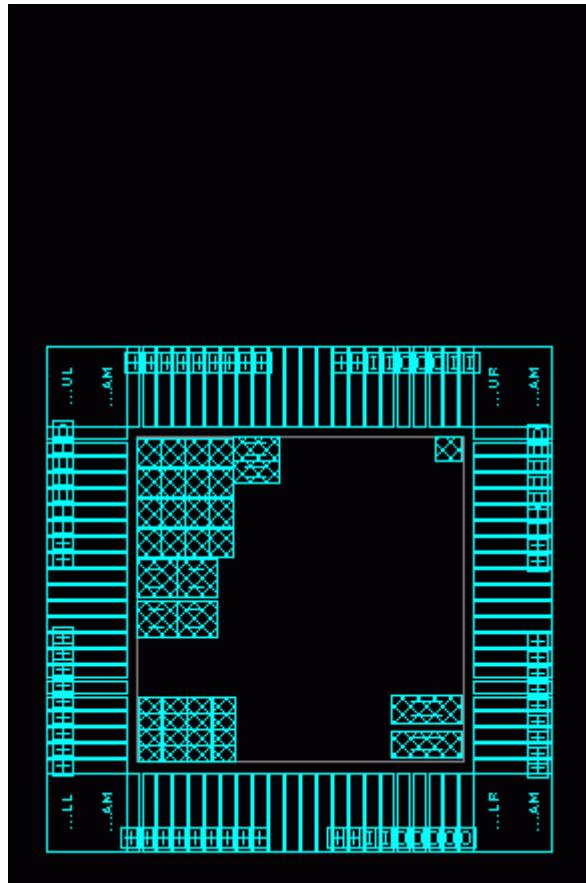
Macro Setup Only

Use the `-macro_setup_only on` option to have the placer place the macros only (not standard cells) and to quickly place them outside the core area, except for macros with relative location constraints, which are still placed inside the core area. After you view the results, you can use the Edit Toolbar to manually move the macros into the core area as desired. You can lock the location of macros that you want to stay in place by setting the `is_fixed` attribute to `true` or by using the GUI, and allow the placer to finish any remaining placement.

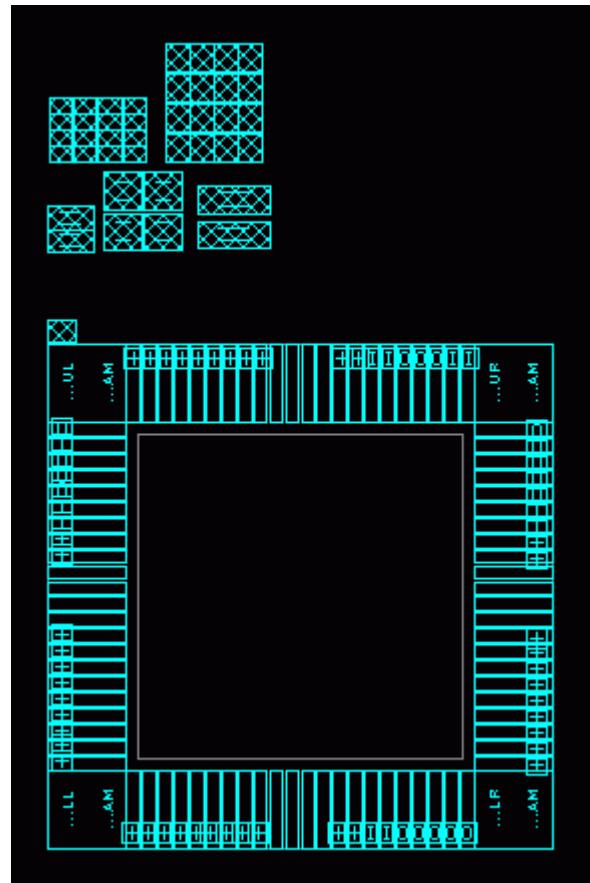
[Figure 6-6 on page 6-15](#) shows the resulting macro placements with the macro setup-only mode set to `off` and set to `on`.

Figure 6-6 Macro Setup-Only Option

`-macro_setup_only off` (default behavior)



`-macro_setup_only on`



Macro Minimum Distance

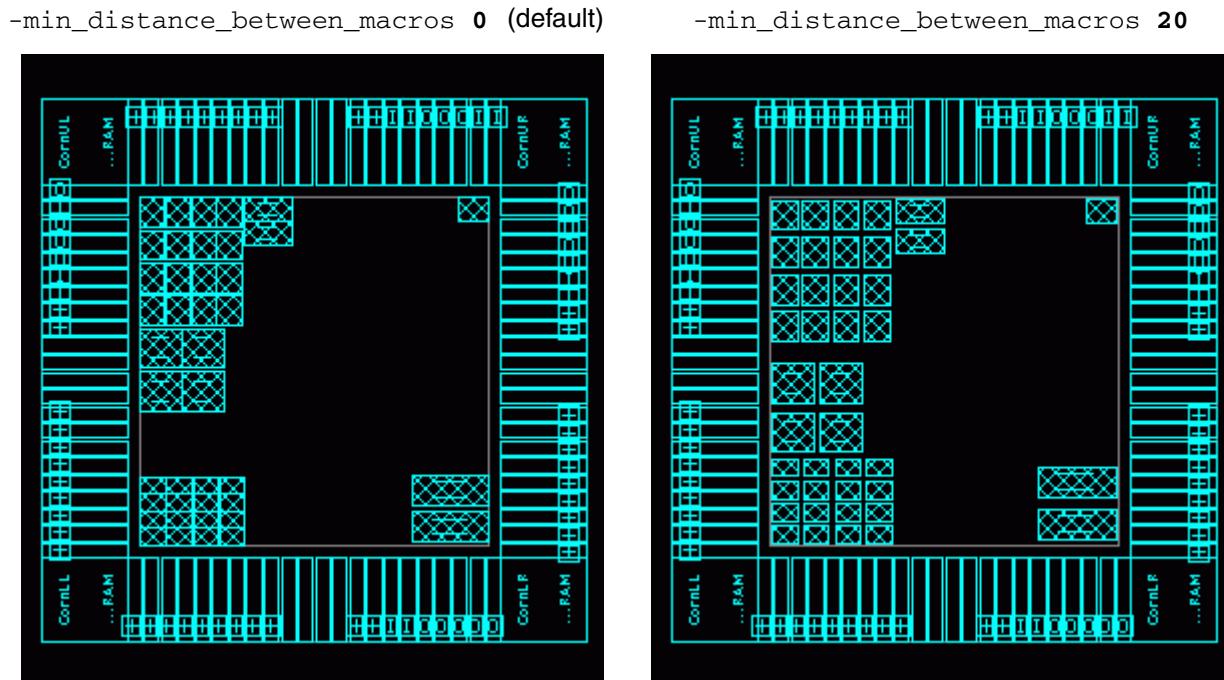
Macros are often placed as close together as possible. To specify a minimum distance between macros, use the `-min_distance_between_macros` option and specify the minimum distance in microns, as in the following example:

```
icc_shell> set_fp_placement_strategy -min_distance_between_macros 8.0
```

Macros are then placed with a minimum of 8.0 microns of spacing between them, as measured from the macro padding, if any, to ensure that at least 8.0 microns of space is available for standard cell placement.

[Figure 6-7 on page 6-16](#) shows the resulting macro placements with the distance between macros set to 0 microns and 20 microns.

Figure 6-7 Minimum Distance Between Macros Option



By default, the minimum spacing applies to all macros. To apply a minimum spacing only to certain macros, use the `-macros_min_distance_applies_to` option and supply a collection of macros, as in the following example:

```
icc_shell> set lev3RAM [get_cells {*//*/*RAM*}]
{I_ORCA_TOP/I_CONT_MEM/I_CONT_RAM_2_3 I_ORCA_TOP/I_CONT_MEM/...
 ...
icc_shell> set_fp_placement_strategy -min_distance_between_macros 8.0 \
-macros_min_distance_applies_to $lev3RAM
```

Then all of the macros belonging to the collection are placed with a minimum spacing of at least 8.0 microns between them. The minimum spacing between one of these macros and a macro not belonging to the collection is one-half of the specified value, or 4.0 microns in this example.

If the collection is empty or if the `-macros_min_distance_applies_to` option is not used, the minimum spacing value applies to all macros.

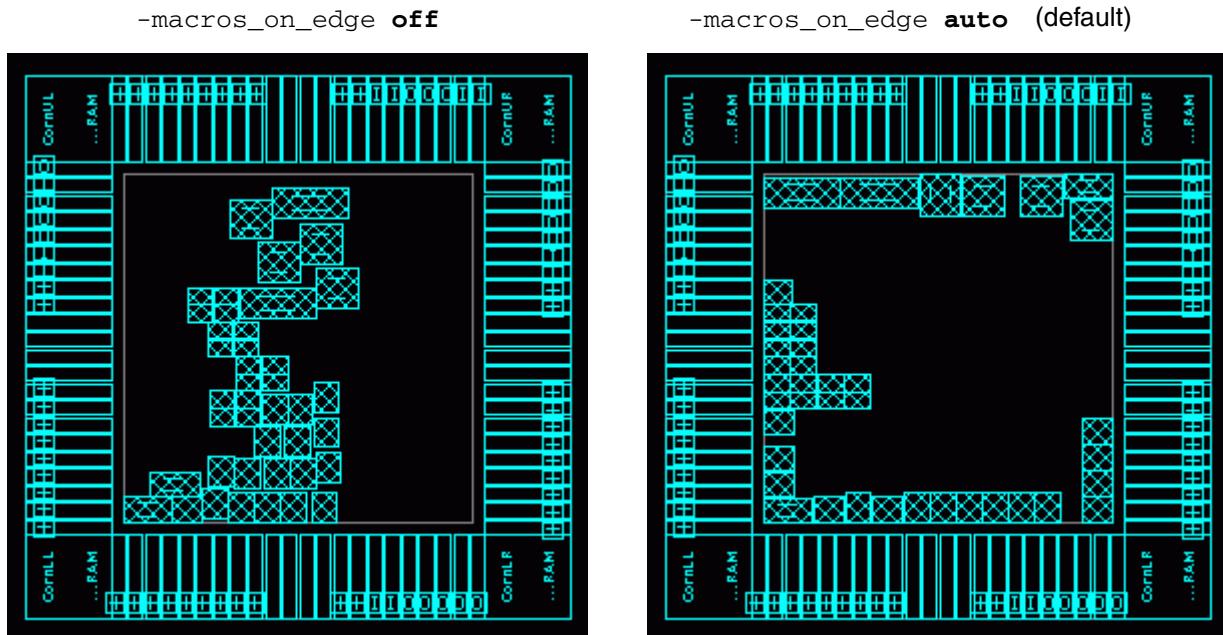
Macros on Edge

Use the `-macros_on_edge` option to specify whether to place macros along boundary edges or to allow them to be placed anywhere. This option can be set to `off`, `on`, or `auto`. The `off` setting allows macros to be placed anywhere. The `on` setting forces placement of macros

along the boundary edges of the chip or plan group whenever possible. The `auto` setting (which is the default behavior) tends to perform placement along the chip or plan group boundary edges, but allows some macros to be placed away from edges.

[Figure 6-8](#) shows examples of placement using `-macros_on_edge off` and `-macros_on_edge auto`.

Figure 6-8 Macros on Edge Option



Placing macros along boundary edges reduces the occurrence of fragment spaces and leaves a larger open space in the middle for placement and routing.

If hard macros occupy a large percentage of the chip area, consider using the `on` setting. Placing hard macros along the edges generally reduces congestion and improves routability.

For a hierarchical design, set this option to `off` until you have shaped and placed the plan groups inside the core area of the chip.

Macros-on-edge placement behavior depends on the hierarchical gravity placement feature. When hierarchical gravity placement is enabled, the macros are placed on the edges of the hierarchical boundaries of their respective plan groups. Otherwise, they are placed along the edges of the chip. Hierarchical gravity is enabled by default, but it can be turned off by using the `-no_hierarchy_gravity` option of the `create_fp_placement` command or unchecking the “Hierarchical gravity” option in the Place Macro and Standard Cells dialog box.

Pin Routing Aware

The `-pin_routing_aware` option specifies whether macro placement considers reserving additional area for routing resources to pins of a block or plan group. By default, pin-routing-aware placement is disabled. To enable this feature, specify the `-pin_routing_aware on` option as in the following example:

```
icc_shell> set_fp_placement_strategy -pin_routing_aware on
```

With this option turned on, the placer considers pin information and creates macro-only blockages along the edges of each plan group and soft macro so that macro cells do not block the edges of the plan groups or soft macros. This reserves enough space along the edges of the plan group or soft macro for pin assignment to place pins and for the router to route nets to the pins.

Sliver Size

The `-sliver_size` option specifies the minimum channel size that is allowed to be populated by standard cells. A sliver is a channel that is considered too small to allow placement of any standard cells.

For example, suppose that you set the sliver size to 10 as follows:

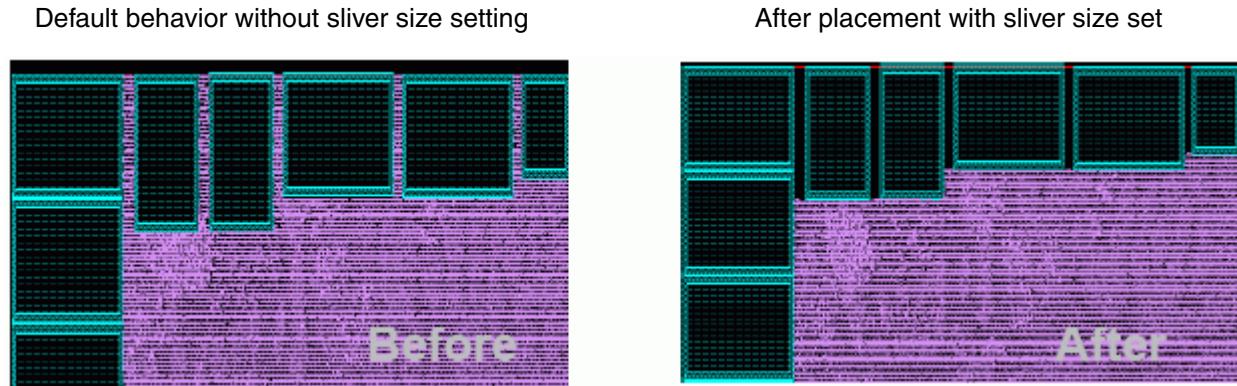
```
icc_shell> set_fp_placement_strategy -sliver_size 10.0
```

In that case, the placer does not place any standard cells in a channel narrower than 10.0 microns wide, even if there is room for them. The default is sliver size is 0.0, which means the placer can place standard cells in any channel that has room.

You can use the `-sliver_size` option to prevent congestion in narrow channels that currently contain standard cells. If there are several channels containing standard cells that have a lot of congestion, set the sliver size to the largest width of those channels. The next time you use the `create_fp_placement` command, the channels are cleared.

[Figure 6-9 on page 6-19](#) shows the placement of standard cells before and after setting the sliver size. After placement with the sliver size set, the channels narrower than the specified sliver size are left unfilled.

Figure 6-9 Channels Cleared by Setting the Sliver Size



A sliver is a channel between any two of the following types of objects:

- A fixed or movable hard macro occupying at least 1/100 of the chip area
- A placement blockage occupying at least 1/100 of the chip area
- An edge of a plan group occupying at least 1/100 of the chip area
- An edge of the core area

The keepout margin of a hard macro is not included in the sliver size. Hard macro slivers are measured from the edge of the keepout margin, not from the edge of the macro.

Snap Macros to User Grid

Use the `-snap_macros_to_user_grid` option to specify whether placed macros snap to the user grid. It can be set to `on` or `off`. The default setting is `off`.

If you turn on this feature, the placer places the lower-left corner of each hard macro's bounding box on one of the grid points defined by the `set_user_grid` command. This affects subsequent placement of macros but does not immediately affect existing placed macros. This feature can be used to favorably align hard macros to power and ground structures on upper metal layers.

Congestion-Related Options

These are the strategy options that control congestion-driven placement:

```
set_fp_placement_strategy
[-adjust_shapes on | off]
[-congestion_effort low | medium | high]
```

The `-adjust_shapes` option determines whether the placer adjusts plan group shapes to reduce congestion during congestion-driven placement. By default, this option is off and no plan group shaping is performed during congestion-driven placement.

The `-congestion_effort` option controls the amount of effort applied to preventing congestion during congestion-driven placement. It can be set to `low`, `medium`, or `high`. The default behavior is `low`, in which the placer analyzes congestion by using the placement congestion map. When this option is set to `medium`, the placer uses the routing congestion map generated by the Zroute global router working at the minimum effort level. The `high` setting is like the `medium` setting, except that the Zroute global router works at the default effort level (`medium`) instead of the minimum level.

Net Weighting Options

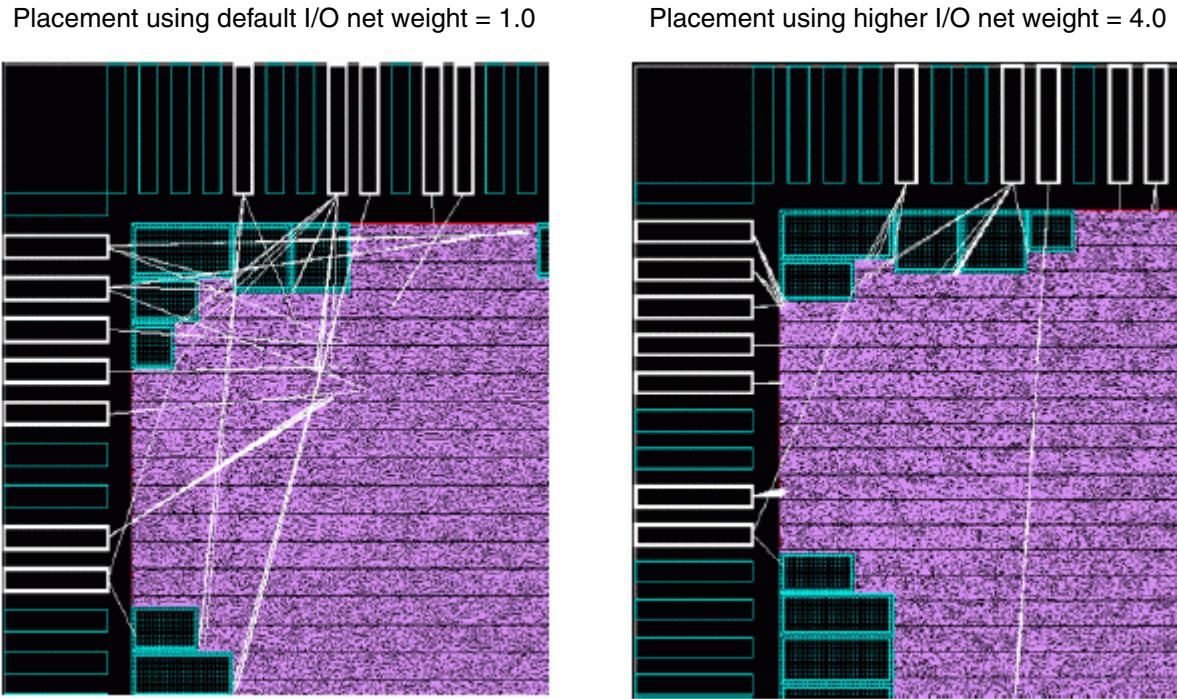
These are the strategy options that control the amount of effort applied to minimizing interface net lengths during virtual flat placement:

```
set_fp_placement_strategy
[-IO_net_weight weight]
[-plan_group_interface_net_weight weight]
```

The `-IO_net_weight` option specifies a weight applied to minimizing the lengths of I/O nets. The default weight is 1.0, which give I/O nets the same weight as all other nets. Specify a lower weight, between 0.0 and 1.0, to give less weight to I/O nets; or specify a higher weight, between 1.0 and 10.0, to give more weight to I/O nets. Set the weight higher to minimize I/O net lengths, possibly at the expense of having longer nets elsewhere. Set the weight to 0.0 if I/O block placement has not been finalized and you want the placer help determine the best I/O block locations.

[Figure 6-10 on page 6-21](#) shows flylines between I/O blocks and the rest of the chip after placement using an I/O net weight of 1.0 (default) and an I/O net weight of 4.0. The I/O net lengths are shorter when the higher weight is used.

Figure 6-10 I/O Net Weighting Results



The `-plan_group_interface_net_weight` option specifies a weight applied to minimizing the lengths of nets that cross the edges of exclusive plan groups. The default weight is 1.0. Specify a lower or higher weight, in the range from 0.0 and 10.0, to apply less or more effort in making these wires shorter. Specify a higher weight to pull cells closer to plan group boundaries where interface nets cross that boundary.

Miscellaneous Options

These are miscellaneous strategy options that control virtual flat placement:

```
set_fp_placement_strategy
  [-default]
  [-block_constraint_file file_name]
  [-flip_chip off | on | honor_reserved]
  [-force_auto_detect_hierarchy on | off]
  [-hierarchy_gravity_blocks list_of_names]
  [-hierarchy_gravity_multi_level on | off]
  [-honor_mv_cells on | off]
  [-spread_spare_cells on | off]
  [-virtual_IPO on | off]
```

Reset to Default Settings

The `set_fp_placement_strategy` command option settings are cumulative. You can use multiple commands to set multiple options. Each option remains in effect until you change it with another `set_fp_placement_strategy` command.

To return all the `set_fp_placement_strategy` options to their default settings, use the `-default` option:

```
icc_shell> set_fp_placement_strategy -default
```

To view the current settings, use the `report_fp_placement_strategy` command:

```
icc_shell> report_fp_placement_strategy
...
*** Macro related parameters ***
set_fp_placement_strategy -macro_orientation automatic | all | N
  current setting: automatic
  default setting: automatic
...
```

Block Constraint File

You can optionally set region and side-ordering constraints on specific blocks when macros-on-edge placement is enabled (see “[Macros on Edge](#)” on page 6-16). To set the constraints, create a block constraint file and use the `-block_constraint_file` option to specify the file name. For example,

```
icc_shell> set_fp_placement_strategy -block_constraint_file my_blk_con
```

The block constraint file is a text file that lists the blocks, their respective regions where they are to be placed, and an optional ordering number, in the following form:

```
fplModuleDestRegion block_name1 region_abbrev [order_number]
fplModuleDestRegion block_name2 region_abbrev [order_number]
fplModuleDestRegion block_name3 region_abbrev [order_number]
...

```

where `block_name` is the name of a plan group or second-level submodule, `region_abbrev` is one of the region strings `N`, `S`, `E`, `W`, `NE`, `NW`, `SE`, or `SW`. (representing north, south, east, and so on), and `order_number` is a relative ordering integer. For example,

```
fplModuleDestRegion PG_1 S 1
fplModuleDestRegion PG_2 S 2
fplModuleDestRegion PG_3 S 3
fplModuleDestRegion PG_4 S 4
```

In this example, the blocks `PG_1` through `PG_4` are placed along the south edge of the chip in numerical order.

Flip-Chip Driver Placement

The `-flip_chip` option specifies the placement method used for flip-chip drivers in the design. It can be set to `off`, `on`, or `honor_reserved`. By default, it is set to `off`, which disables flip-chip driver placement. Set it to `on` to enable flip-chip driver placement. The `honor_reserved` setting causes the placer to perform reserved-slot insertion and compression, if specified. Flip-chip driver placement occurs after placement of the signal drivers. You must use the `honor_reserved` setting if you want to reserve driver slots.

Automatic Hierarchy Detection

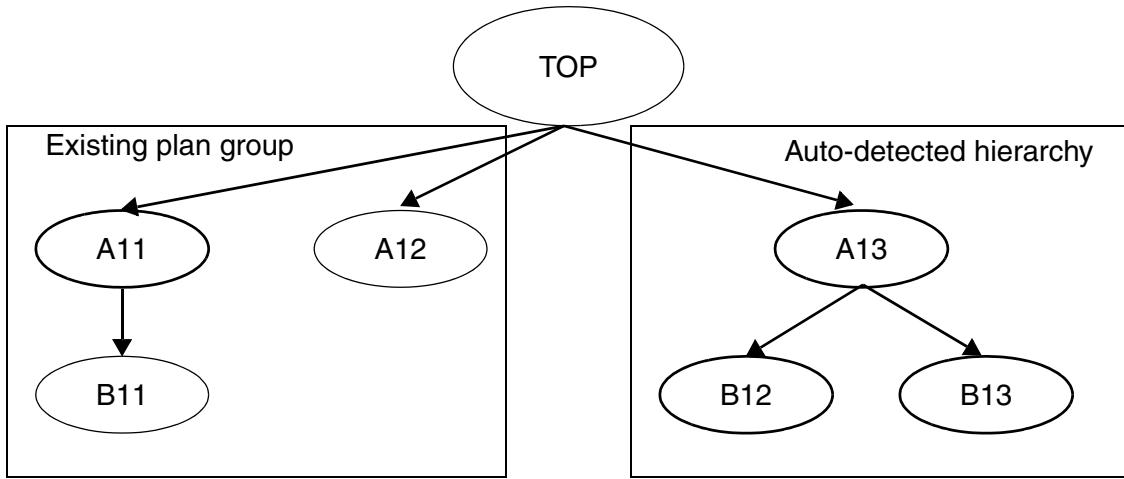
The `-force_auto_detect_hierarchy` option of the `set_fp_placement_strategy` command specifies whether the tool analyzes the logical hierarchy nodes that are left at the top level of the design to determine if they are candidates for hierarchical gravity. This option can be used on designs that contain existing plan groups either within the core area or outside the core area. If the design does not contain plan groups, this option has no effect. By default, this option is set to `off`.

If you set this option to `on`, the placement engine analyzes the logical hierarchy nodes that are left at the top level of the design and not already assigned to any plan group to determine if they are candidates for hierarchical gravity. If there are additional logical nodes outside of the existing plan groups that contain top-level modules for grouping, the `create_fp_placement` command places the cell instances inside these modules together.

This feature is based on the logical hierarchy only. Other placement constraints, such as power domains or relative placement groups, do not affect the results.

[Figure 6-11 on page 6-24](#) shows an example of the logical hierarchy of a design in which there is an existing plan group consisting of modules `top/A11` and `top/A12`. When the auto-detect hierarchy option is set to `on`, an additional logical hierarchy node outside of the existing plan group is extracted. This logical hierarchy node consists of modules `top/A13`, `top/A13/B12`, and `top/A13/B13`. The `create_fp_placement` command places the cell instances inside these modules together.

Figure 6-11 Example of Automatic Hierarchy Detection



Hierarchical Gravity

The `-hierarchy_gravity_blocks` option of the `set_fp_placement_strategy` command specifies a list of blocks for hierarchical gravity placement. If you specify this option, the `create_fp_placement` command does not perform automatic hierarchy detection. Instead, it applies hierarchical gravity only to the blocks specified by using this option and any plan groups already created in the design. This option can also be used on a netlist where the logical hierarchy is flattened.

The `-hierarchy_gravity_multi_level` option, when set to `on`, specifies that the `create_fp_placement` command consider multiple levels of logical hierarchy during macro placement. The tool determines which hierarchy levels to consider based on macro connectivity and places the macros from this hierarchy in close proximity. This option and the `-hierarchy_gravity_blocks` option are mutually exclusive. By default, this option is `off`.

Honor Multivoltage Cells

The `-honor_mv_cells` option of the `set_fp_placement_strategy` command specifies whether to bring level-shifter and isolation cells closer to their respective voltage areas. The default setting is `off`. Setting this option to `on` attaches a constraint to level-shifter and isolation cells that causes them to be placed closer to their voltage areas. After setting this option to `on`, you must use the `link -force` command to allow the placer to recognize the `dont_touch` attribute on these cells.

Spread Spare Cells

By default, the `create_fp_placement` command spreads spare cells evenly across the available space on the chip. Spare cells are cells that have no net connectivity, including filler cells, and cells that connect only to high-fanout nets. To prevent even spreading of spare cells, set the `-spread_spare_cells` option to `off` in the `set_fp_placement_strategy` command. The default setting is `on`.

Virtual In-Place Optimization

The `-virtual_ipo` option of the `set_fp_placement_strategy` command specifies whether virtual in-place optimization is performed during timing-driven virtual flat placement. The default setting is `off`. Set this option to `on` to obtain better final timing results at the cost of more runtime during timing-driven virtual flat placement.

Timing-driven placement is performed when you use the `-timing_driven` option with the `create_fp_placement` command or select the “Timing driven” option in the Place Macro and Standard Cells dialog box. With timing-driven placement, the placer makes a greater effort to improve timing by keeping cells closer together along critical paths.

Virtual in-place optimization is the process of estimating the timing effects of optimization performed by driver resizing and buffer insertion. This optimization is “virtual” because the cells are not actually resized and buffers are not actually inserted. Instead, the tool estimates the timing effects of optimization and annotates the design with the new timing information. Estimating the timing effects of optimization is much faster than actually performing the optimization.

The purpose of virtual in-place optimization is to allow the timing-driven placer to concentrate its efforts on true critical paths and ignore timing paths that can be fixed later by driver resizing or buffer insertion.

For example, a high-fanout net has a long delay that is easily fixed by buffer insertion. Without virtual in-place optimization, the placer might see the long delay as a critical path. As a result, it would try to place all the driven cells closer to the driver, an unnecessary action that could hurt the timing of other paths by increasing the lengths of other wires connected to the driven cells. However, with in-place optimization, the high-fanout net is annotated with its optimized delay, so the placer ignores the timing of that net, and instead focuses its effort on nets belonging to true critical paths.

Virtual in-place optimization can also be invoked as a separate process at any time in the design planning flow by using the `virtual_ipo` command.

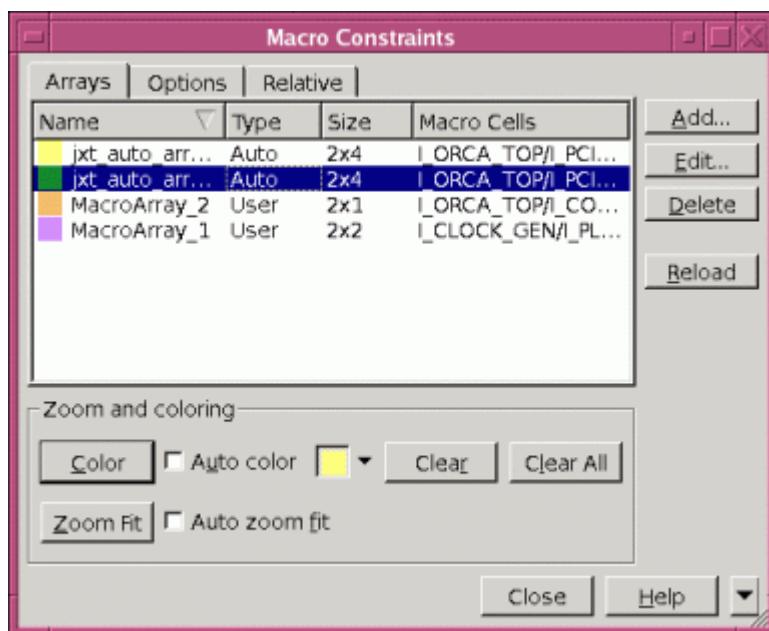
Hard Macro Constraints

You can control several aspects of hard macro placement during virtual flat placement by using the `set_fp_placement_strategy` command. In addition, you can control other aspects of hard macro placement by using the following commands:

- `set_fp_macro_array` – Sets macro array constraints
- `set_fp_macro_options` – Sets macro placement options
- `set_fp_relative_location` – Sets relative location constraints

The IC Compiler GUI makes it easy to set these constraints. To use the GUI, first make sure that the command menus are in design planning mode (File > Task > Design Planning). Then choose Placement > Macro Constraints, which opens the Macro Constraints dialog box, shown in [Figure 6-12](#).

Figure 6-12 Macro Constraints Dialog Box



The dialog box has three tabs, labeled Arrays, Options, and Relative. Click a tab to view a list of macro constraints of that type that have been defined: array constraints, macro placement options, or relative location constraints, respectively.

To create a new constraint, click the Add button. This opens another dialog box containing the options for setting the constraints of the current tab setting.

To edit or delete an existing constraint, select that constraint in the Macro Constraints list and then click the Edit or Delete button.

The Color button applies a specified color to the display of the macros belonging to the currently selected constraint. This feature helps you identify the location of the macros in the layout view. The Zoom Fit button zooms the layout view to show just those macros.

The Reload button creates user-defined array constraints from automatically created arrays. You can then edit these constraints to modify the automatically placed arrays.

For online help on using the dialog box, click its Help button.

The following subsections describe how to set the hard macro constraints:

- [“Macro Array Constraints” on page 6-27](#)
- [“Macro Placement Options” on page 6-31](#)
- [“Relative Location Constraints” on page 6-35](#)

After you set the hard macro constraints, close the Macro Constraints dialog box. When you perform virtual flat placement by using the `create_fp_placement` command or by choosing Placement > Place Macro and Standard Cells, the placer places the macros according to the constraints you have set.

Macro Array Constraints

You can create user-defined arrays of hard macro cells. The cells are placed in the array according to the constraints you specify.

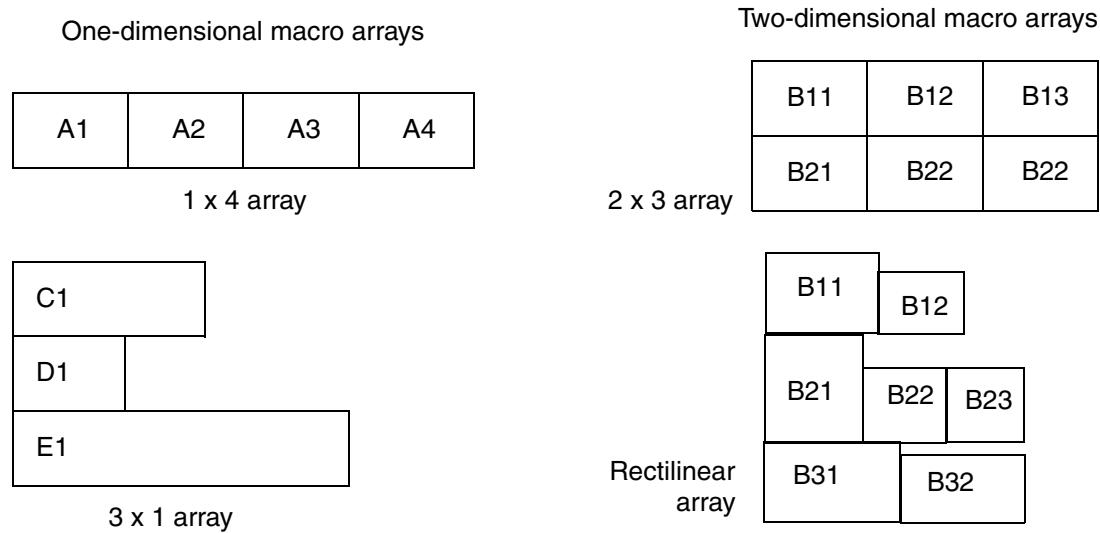
When you create a user-defined array of hard macros, keep the following points in mind:

- The core area must be large enough to accommodate large macro arrays.
- You must specify the order of the macros for the array.
- You can group macros to form a one-dimensional or a two-dimensional array.
- Align one-dimensional and two-dimensional heterogeneous macro arrays by using the “Align edges” option.
- Hard keepout areas are automatically created between the elements in the array.

If there are conflicting constraints on the macros, error messages are printed during placement. However, the error messages do not print details about the conflicts.

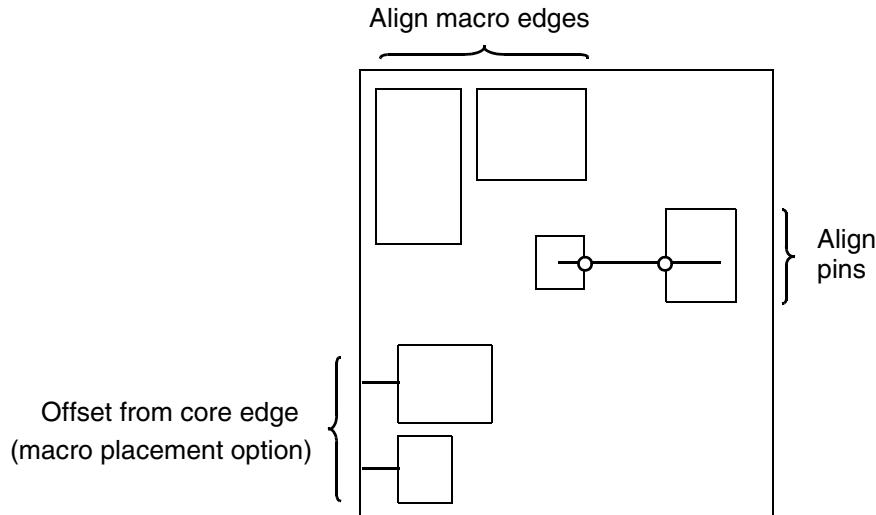
[Figure 6-13](#) shows some examples of one-dimensional and two-dimensional macro arrays.

Figure 6-13 One-Dimensional and Two-Dimensional Macro Arrays



[Figure 6-14](#) shows different methods for aligning macros in an array: by macro edges, pin, or offsets from core edges.

Figure 6-14 Alignment to Other Macros, Pins, and Edges



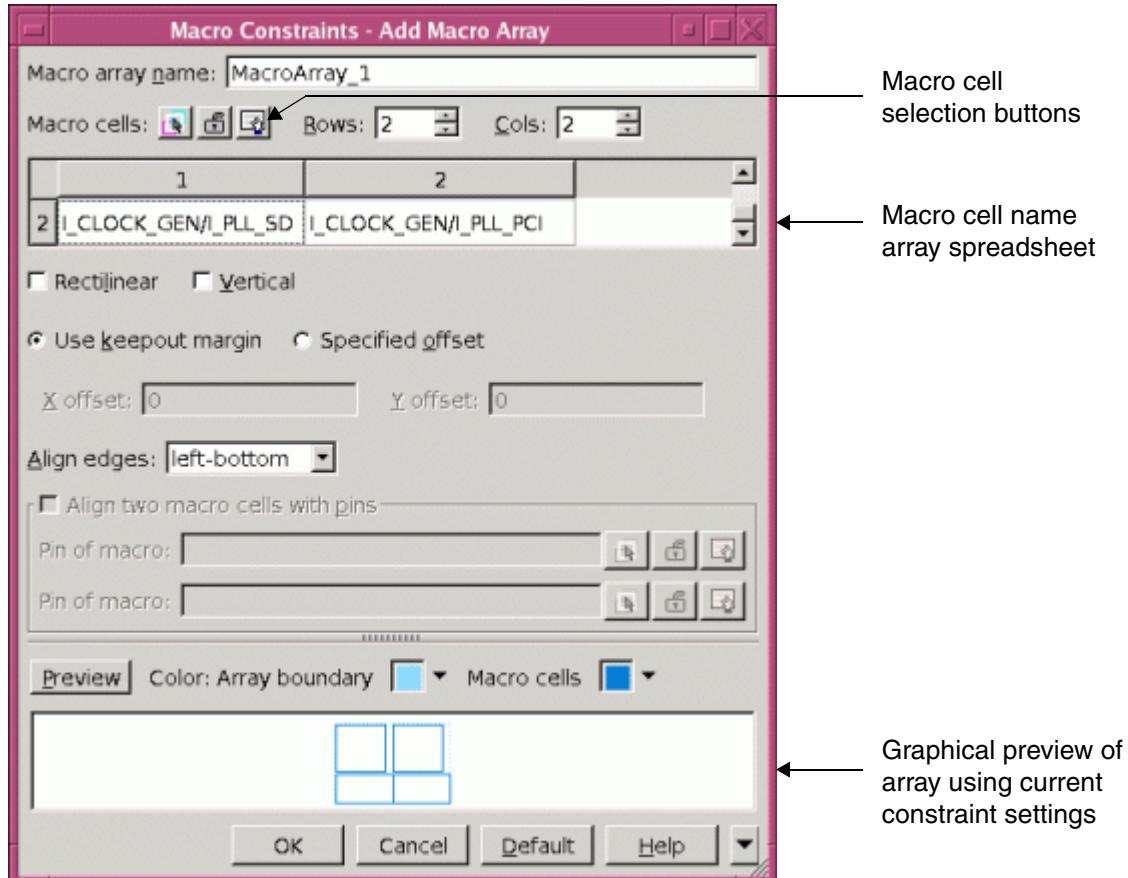
You can specify macro array constraints with the `set_fp_macro_array` command:

```
set_fp_macro_array
  -name string
  [-elements collection_of_macro_cells]
  [-align_edge t | b | l | r | c | top | bottom | left | right | center]
  [-align_pins {list_of_two_pin_objects}]
  [-x_offset float] [-y_offset float]
  [-use_keepout_margin]
  [-vertical] [-rectilinear]
  [-align_2d lb | lc | lt | rb | rc | rt | cb | cc | cr |
    left-bottom | left-center | left-top |
    right-bottom | right-center | right-top |
    center-bottom | center-center | center-top]
  [-reset]
```

To report macro array constraints that have been set, use the `report_fp_macro_array` command. For details, see the man pages for the `set_fp_macro_array` and `report_fp_macro_array` commands.

To specify macro array constraints by using the GUI, first make sure that the command menus are in design planning mode (File > Task > Design Planning). Then choose Placement > Macro Constraints. In the Macro Constraints dialog box, click the Array tab if it is not already selected. Then click the Add button. This opens the Macro Constraints - Add Macro Array dialog box. See [Figure 6-15 on page 6-30](#).

Figure 6-15 Macro Constraints - Add Macro Array Dialog Box



The “Macro array name” box specifies a name for the macro array constraint. You can use the default name or enter any name to identify this constraint.

Enter the dimensions of the array in the Rows and Cols boxes. A table displays an array of cells having the specified dimensions. Enter the names of the macro cell instances into the cells of the array. You can use the “Macro cells” selection buttons to select the macros graphically from the layout view or select from a list of macro names.

Select the Rectilinear option if you want to allow rows or columns to have different numbers of cells. In that case, you can leave some cells in the spreadsheet empty.

Select the Vertical option if you want a one-dimensional array to be placed as a vertical column of cells rather than a row of cells.

The “Use keepout margin” option causes the macros to be placed as close together as possible while observing the macro keepout margins. Alternatively, use the “Specified offset” option to explicitly specify the minimum allowed spacing, in microns, between macro edges in the x- and y-directions.

The “Align edges” option setting determines how to align the edges of macro cells of different sizes in each row (or in each column if the Vertical option is selected). For a one-dimensional array, the options are left, right, bottom, top, and center. For a two-dimensional array, the options are left-bottom, left-top, left-center, and so on, so you can specify the alignment in the x- and y-directions.

For an array containing exactly two macros, you can use the “Align two macro cells with pins” option to align a specified pin of one macro to a specified pin of the other macro.

Click the Preview button to get a graphical preview of the array. The preview shows the relative sizes, shapes, and alignments of macros in the array.

If you are satisfied with the preview, click the OK button to add the array constraint to the list shown in the Macro Constraints dialog box. To edit this constraint, select it in the list in the Macro Constraints dialog box and then click the Edit button.

Macro Placement Options

You can define placement constraints that restrict specified macro cells and macro arrays to particular regions, orientations, and alignments. During virtual flat placement, macro cells are flipped or rotated according to legal orientation constraints from the library or by constraints you set.

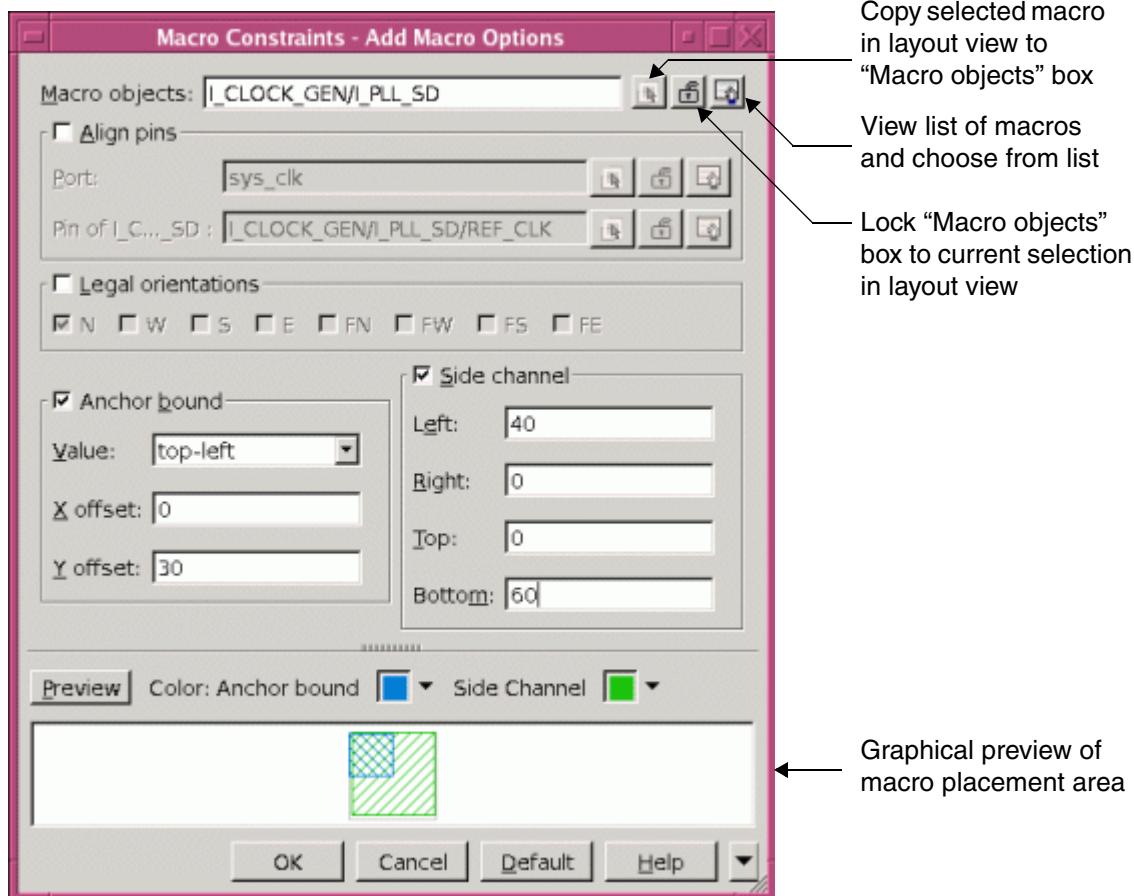
You can specify the macro placement options with the `set_fp_macro_options` command:

```
set_fp_macro_options
  collection_of_macro_objects
  [-legal_orientations list]
  [-anchor_bound t1 | t | tr | r | br | b | bl |
   l | tm | bm | lm | rm | c ]
  [-x_offset float]
  [-y_offset float]
  [-align_pins list]
  [-side_channel {left right top bottom}]
  [-reset]
```

To report macro placement option settings, use the `report_fp_macro_options` command. For details, see the man pages for the `set_fp_macro_options` and `report_fp_macro_options` commands.

To specify macro placement options by using the GUI, first make sure that the command menus are in design planning mode (File > Task > Design Planning). Then choose Placement > Macro Constraints. In the Macro Constraints dialog box, click the Options tab if it is not already selected. Then click the Add button. This opens the Macro Constraints - Add Macro Options dialog box. See [Figure 6-16](#).

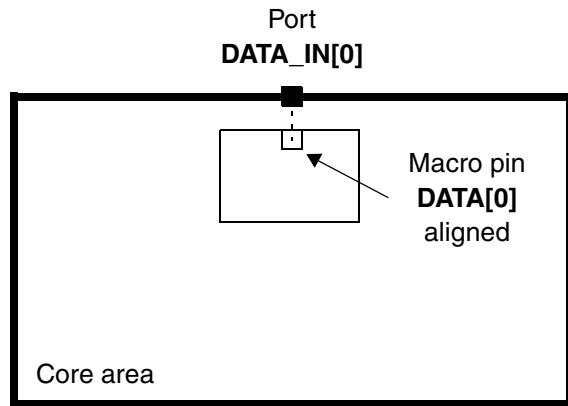
Figure 6-16 Macro Constraints - Add Macro Options Dialog Box



In the “Macro objects” box, enter the names of one or more macro cell instances or macro arrays to which the constraints apply. You can use the buttons to select the macros graphically from the layout view or select from a list of macro instances and macro array names.

You can use the “Align pins” option to place a single macro cell so that a specified pin of the macro is aligned to a specified port, as shown in [Figure 6-17 on page 6-33](#). To do so, enter a single macro name in the “Macro objects” box, select the “Align pins” option, and enter the port name and the macro pin name. You can use the buttons to select the port and pin graphically or select from a list of port names and macro pin names.

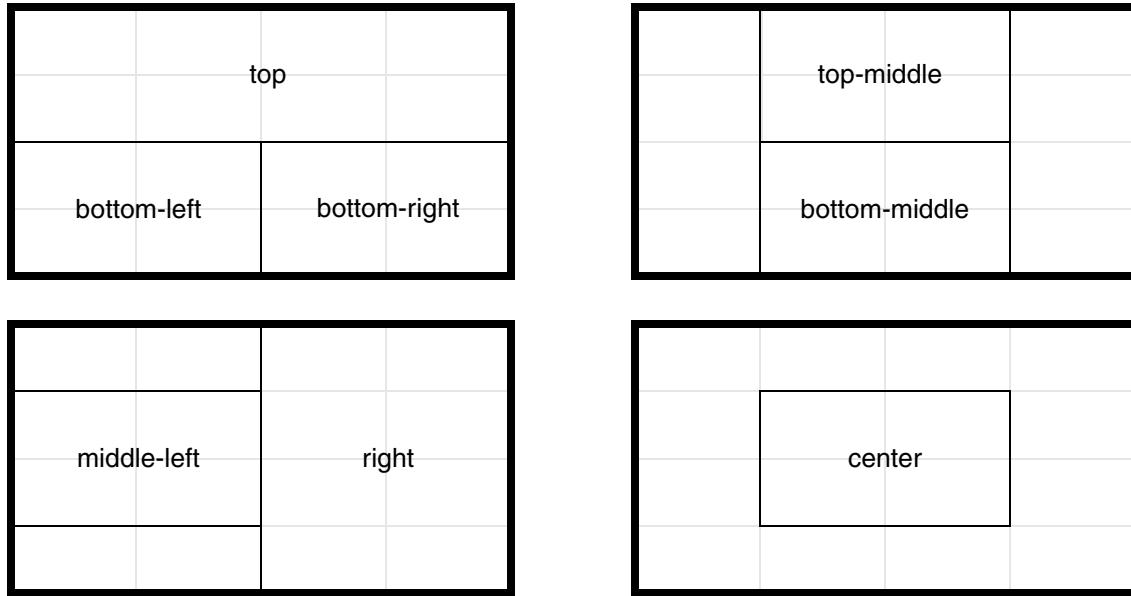
Figure 6-17 Aligning a Macro Pin to a Port



Use the “Legal orientations” options to restrict the allowed orientations of macro cells. If you do not use this option, all orientations allowed in the physical library apply to the macro. This option can be used for macro cells only, not macro arrays.

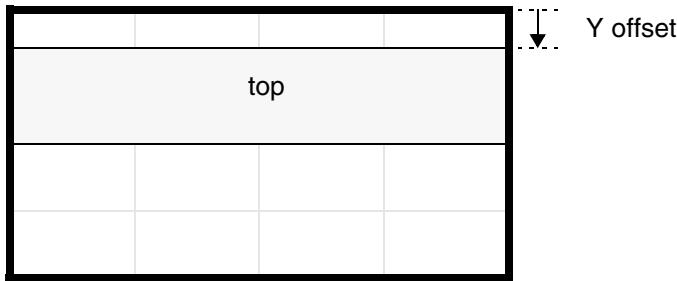
You can restrict placement of the macros or macro arrays to a specified region of the core area such as the top half, the left half, or the bottom-right quadrant. To make such a constraint, select the “Anchor bound” option and select the desired region by using the “Value” list. Some of the possible choices are shown in [Figure 6-18](#).

Figure 6-18 Anchor Bound Areas



For each anchor bound region setting, you can optionally specify an offset from the outside edge or edges, further restricting placement of the macros or macro array. For example, if you select “top” as the anchor bound, restricting placement to the top half of the core area, you can also specify a “Y offset” or minimum distance away from the top core edge, as shown in [Figure 6-19](#).

Figure 6-19 Anchor Bound Offset From Outside Edge

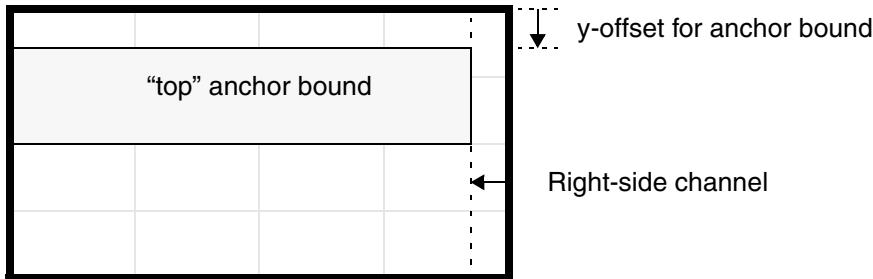


The “Anchor bound” is a soft bound, which means that the placer can move the macro cell outside of the boundary area if there is not enough space available or if the cost is too high.

Whether or not you use the “Anchor bound” option, you can define side channels, which are regions along the core edges where placement of macros is not allowed. To define side channels, select the “Side channel” option and specify the desired channel sizes for the Left, Right, Top, and Bottom edges of the core area, in microns. The macros are then placed at least the specified distances away from the core edges. If a channel is not required on a side, leave the channel at its default setting of zero. For example, if you set the Right channel to a nonzero value, macros cannot be placed along the right edge within the specified distance.

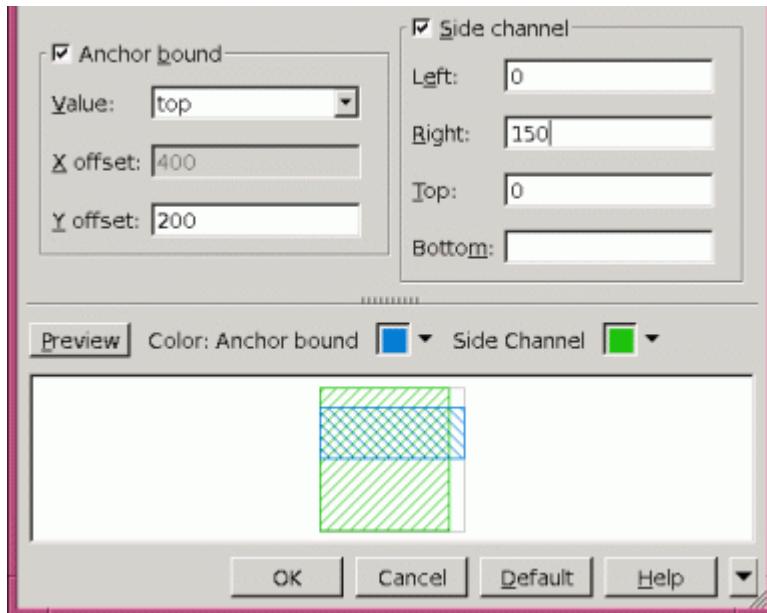
If both anchor bound and side channel constraints are specified, both types of constraints apply at the same time, as shown in the example in [Figure 6-20](#).

Figure 6-20 Anchor Bound and Side Channel Constraints



Click the Preview button to get a graphical preview of the anchor bound and side channel constraints. In the example shown in [Figure 6-21](#), the anchor bound constrains placement to the blue area and the side channel constrains placement to the green area.

Figure 6-21 Anchor Bound and Side Channel Constraint Preview



If you are satisfied with the preview, click the OK button to add the placement constraint to the list shown in the Macro Constraints dialog box. To edit this constraint, select it in the list in the Macro Constraints dialog box and then click the Edit button.

Relative Location Constraints

A relative location constraint specifies the exact location of a macro cell with respect to an object that has a fixed location, expressed as an x- and y-offset from the fixed object. The macro to which the constraint applies is called the target macro. The fixed-location object is called the anchor object. The anchor object can be the core area, a plan group, a fixed macro cell, or a macro cell that is effectively fixed in place by having its own relative location constraint.

You can specify relative location constraints with the `set_fp_relative_location` command:

```
set_fp_relative_location
  -name constraint_name
  -target_cell cell_name
  [-target_orientation N | S | E | W | FN | FS | FE | FW]
  [-target_corner bl | br | tl | tr]
  [-anchor_object object_name]
  [-anchor_corner bl | br | tl | tr]
  [-x_offset distance]
  [-y_offset distance]
```

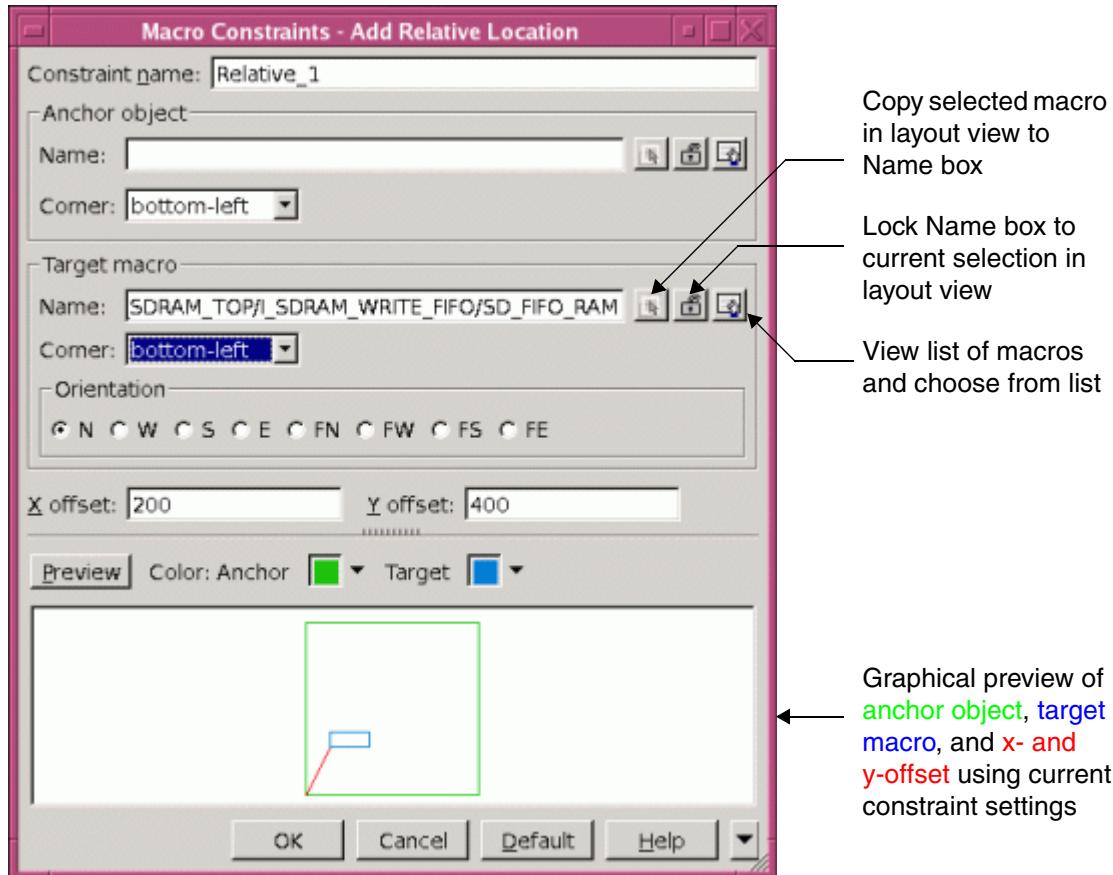
The following additional commands operate on relative location constraints:

- `report_fp_relative_location` – Reports the relative location constraints that have been set.
- `extract_fp_relative_location` – Extracts one or more relative location constraints from the current layout, producing a set of `set_fp_relative_location` commands that you can run later to restore the current placement.
- `remove_fp_relative_location` – Removes one or more relative location constraints that have been set.

For detailed information about using these commands, see their man pages.

To specify relative location constraints by using the GUI, first make sure that the command menus are in design planning mode (File > Task > Design Planning). Then choose Placement > Macro Constraints. In the Macro Constraints dialog box, click the Relative tab if it is not already selected. Then click the Add button. This opens the Macro Constraints – Add Relative Location dialog box. See [Figure 6-22 on page 6-37](#).

Figure 6-22 Macro Constraints - Add Relative Location Dialog Box



The “Constraint name” box specifies a name for the relative location constraint. You can use the default name or enter any name to identify this constraint.

Under “Anchor object,” in the Name box, enter the name of the anchor object, or leave this box blank if the anchor object is the core area. The anchor object can be the core area, a plan group, a fixed macro cell, or a macro cell that is effectively fixed in place by having its own relative location constraint. Also, using the Corner list, specify the corner of the anchor object from which to apply the x- and y-offsets.

Under “Target macro,” in the Name box, enter the name of the macro to which the constraint applies. Also, using the Corner list, specify the corner of the target macro to be placed at the x- and y-offsets from the anchor object. Select one Orientation option (N, W, S, E, FN, FW, FS, or FE) to specify the orientation of the target macro to be placed.

In the “X offset” and “Y offset” boxes, enter the amount of offset from the anchor object corner to the target macro corner, in microns. Each value can be positive, negative, or zero.

Click the Preview button to display a graphical preview of the anchor object, target object, and x- and y-offsets between the active corners of these objects. The preview shows the relative sizes, shapes, and alignments of the anchor and target objects.

If you are satisfied with the preview, click the OK button to add the relative location constraint to the list shown in the Macro Constraints dialog box. To edit this constraint, select it in the list in the Macro Constraints dialog box and then click the Edit button.

Blockages, Margins, and Shielding

A placement blockage is a region where hard macros cannot be placed. To create a placement blockage, use the `create_placement_blockage` command or choose Floorplan > Create Placement Blockage.

A keepout margin is a region along the edge of a macro where placement of other cells is not allowed. To define a keepout margin for one or more macros, use the `set_keepout_margin` command or choose Placement > Set Keepout Margin.

Block shielding is a strip of metal surrounding a macro that prevents crosstalk between wires inside and outside the macro. To add shielding around macros or plan groups, use the `create_fp_block_shielding` or choose Floorplan > Create Module Block Shielding.

Blockages and keepout margins are described in the following subsections. Shielding is described in “[Adding Block Shielding to Plan Groups or Soft Macros](#)” on page 7-6.

Placement Blockages

You might want to prevent the placement of hard macros in certain areas, for example, a channel between two plan groups or next to a power pad. You can specify placement blockages with the `create_placement_blockage` command:

```
create_placement_blockage
  -bbox {llx1 lly1 urx1 ury1}
  [-type {hard | soft | partial}]
  [-blocked_percentage percentage]
  [-no_register]
  [-no_pin]
  [-blocked_layers layers]
  [-no_hard_macro]
  [-name blockage_name]
```

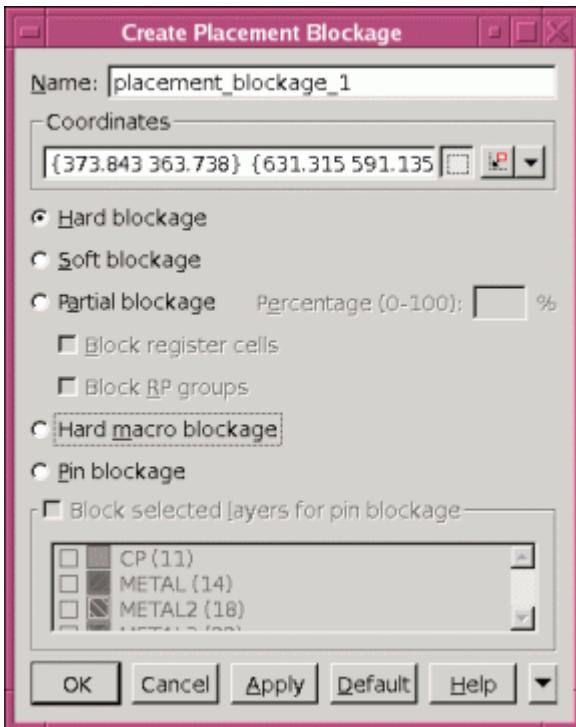
To report placement blockages that have been set, use the following script excerpt:

```
foreach_in_collection item [get_placement_blockages -quiet] {
    puts "Name: [get_attribute $item name]"
    puts " Type: [get_attribute $item type]"
    puts " bbox: [get_attribute $item bbox]\n"
}
```

To remove placement blockages, use the `remove_placement_blockage` command. For details on using these commands, see the man pages.

To specify placement blockages by using the GUI, choose Floorplan > Create Placement Blockage. This opens the Create Placement Blockage dialog box. See [Figure 6-23](#).

Figure 6-23 Create Placement Blockage Dialog Box



The Name box specifies a name for the blockage. You can use the default name or enter any name to identify the blockage.

In the Coordinates box, enter the coordinates of the lower-left corner and upper-right corner of the rectangular blockage area. To select the blockage area graphically in the layout view, use the rectangle button next to the box.

Select one of the following five blockage types:

- Hard blockage – No hard macros or standard cells are allowed in the blockage area.
- Soft blockage – No hard macros or standard cells are placed in the blockage area initially, but they can be moved into the area during optimization and legalization.
- Partial blockage – Hard macros or standard cells are allowed, but the placer prevents them from occupying more than the specified percentage of the blockage area.
- Hard macro blockage – No hard macros are placed in the blockage area, but standard cells and pins are allowed.
- Pin blockage – In the blockage area, the pin placer does not assign pins and the global router does not add routes using the specified layer or layers.

Click OK or Apply to create the blockage.

You can control the display of blockages in the layout window by using the View Settings panel (View > Toolbars > View Settings).

To remove blockages, use the `remove_placement_blockage` command.

Keepout Margins

You can specify keepout margins or padding around some or all macros. During virtual flat placement, all other cells, including standard cells and other macros, are placed outside the specified margin. This can help prevent congestion and DRC errors.

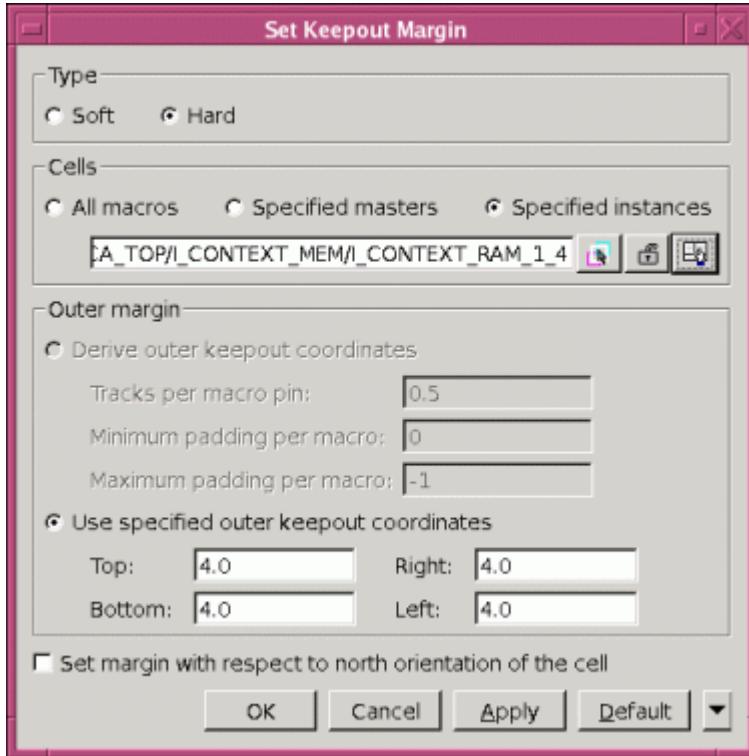
This is the syntax of the `set_keepout_margin` command:

```
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]
[-north]
[object_list]
```

To report keepout margins that have been set, use the `report_keepout_margin` command. To remove keepout margins, use the `remove_keepout_margin` command. For details on using these commands, see the man pages.

To specify keepout margins by using the GUI, choose Placement > Set Keepout Margin. This opens the Set Keepout Margin dialog box. See [Figure 6-24 on page 6-41](#).

Figure 6-24 Set Keepout Margin Dialog Box



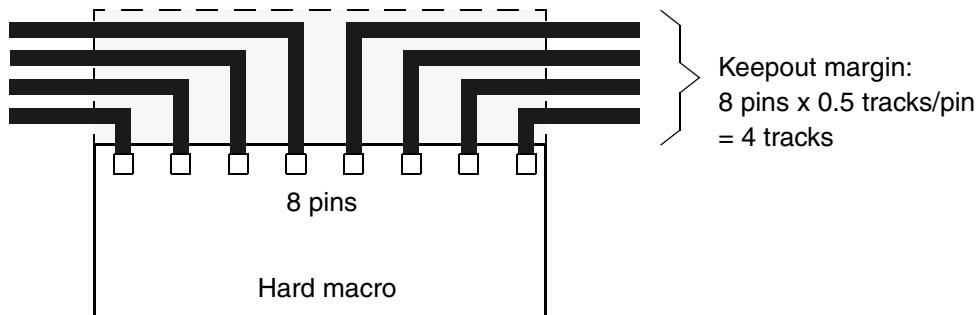
Under “Type,” select the keepout margin type, Soft or Hard. A soft keepout margin prevents other cells from being placed within the margin, but the cells can be moved in during optimization and legalization. A hard keepout margin never allows other cells within the margin.

Under “Cells,” specify the cells affected by the command. You can choose all macros, specified reference cells (“Specified masters” option), or specified cell instances. The buttons next to the text box let you choose the cells graphically in the layout view or choose from a list of cells.

Under “Outer margin,” select “Use specified outer keepout coordinates” to explicitly specify the margins on each side of the macro, in microns.

If you selected “All macros,” you can optionally select “Derive outer keepout coordinates” to automatically calculate the margins of all macro cells. In that case, the margin is set according to the number of pins on each side. An automatically calculated keepout is a hard keepout constraint. By default, the margin for each side is set to the number of macro pins multiplied by one-half the track width, which allows enough space for parallel connections to the pins from both sides, as shown by the example in [Figure 6-25 on page 6-42](#).

Figure 6-25 Automatically Derived Keepout Margin



You can specify the number of tracks per pin used in the calculation. For a macro with a large number of pins, use a value between 0.5 and 1.0 tracks per pin to reserve enough room for routing the connections. You can also specify a minimum margin and a maximum margin, in microns, to restrict the allowed range of calculated keepout margins. The default maximum margin setting, -1, causes the maximum limit to be 40 times the metal1 pitch.

Select the “Set margin with respect to north orientation of the cell” to keep the specified margin values with respect to the north orientation of the cell when the cell is rotated.

Click OK or Apply to create the keepout margins.

If you do not specify any keepouts, the placer automatically derives the keepouts using 0.5 tracks per pin by default.

You can control the display of keepout margins in the layout window by using the View Settings panel (View > Toolbars > View Settings). Only explicit (not automatically calculated) keepout margins can be displayed.

To remove explicit keepout margins, choose Placement > Report Keepout Margin.

Relative Placement Groups

The `create_rp_group` command creates relative placement groups. A relative placement group is an association of cell instances, other hierarchical relative placement groups, and placement keepouts. A group is defined by the number of rows and columns that it uses. Use the `add_to_rp_group` command to add leaf cells to a relative placement group. The relative placement information is automatically saved in the Milkyway design database when you save the design in Milkyway format using the `save_mw_cel` command. It contains the height and width of the top-level relative placement groups and the x- and y-offsets of the cell instances, placement keepouts, and hierarchical relative placement groups for each

top-level relative placement group. The x- and y-offsets are used to anchor the relative placement cell instance with respect to the lower-left corner (the relative placement's origin) of its corresponding relative placement group.

During the design planning flow, you can perform the following relative placement functions:

- Apply compression by using the `-compress` option.
- Specify the anchor location by using the `-x_offset` and `-y_offset` options.
- Specify the utilization percentage by using the `-utilization` option.
- Ignore the relative placement group by using the `-ignore` option.

However, you should not use the following relative placement functions during virtual flat placement:

- Specifying the group alignment pin by using the `-pin_align_name` option.
- Specifying the right alignment by using the `-alignment bottom_right` option.
- Allowing keepouts over tap cells by using the `-allow_keepout_over_tapcell` option.
- Legalizing individual objects in a relative placement group.

You can use the `create_fp_placement` command to place cell instances in each relative placement group during initial virtual flat placement. By using the `create_fp_placement` and `create_placement` commands to place the cells in each relative placement group in the same way, the `create_fp_placement` command can provide you with better wire length and congestion estimates for your floorplan.

To achieve a good correlation between the `create_fp_placement` and `create_placement` commands when placing the relative placement cells, the following conditions apply.

- If the cell instances in one relative placement group belong to different move bounds, the `create_fp_placement` command ignores this relative placement group. This also applies to plan groups automatically extracted by the `create_fp_placement` command based on the logical hierarchy.
- No cell instances are placed over relative placement keepouts.
- If any cell instance in one relative placement group is fixed, this relative placement grouping is ignored by the `create_fp_placement` command and the cells are treated as nonrelative placement cells.

The `create_fp_placement -no_legalize` command supports two types of relative placement groups, including the macro-only and standard-cell-only relative placement groups. The virtual flat placement stage allows the following relative placement objects to occupy multiple column and row positions: leaf cells, keepouts (hard, soft, and space), hierarchical groups, and relative placement cells.

Placing Relative Placement Groups

By default, relative placement is supported during initial virtual flat placement. The `create_fp_placement` command extracts information about the relative placement groups from the Milkyway database. A dummy cell, which contains both macros and standard cells in one relative placement group, is created for each top-level relative placement group. The Milkyway properties of the relative placement group are modified based on the new locations of the placed dummy cells. All relative placement cell instances are temporarily fixed. Any dummy cells related to the top-level relative placement groups are removed. This allows the `create_fp_placement` command to place other standard cell instances over the free spaces inside the relative placement bounding boxes.

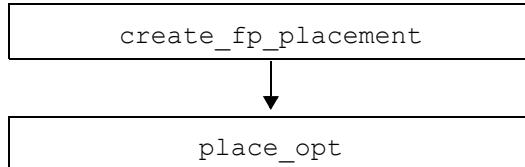
The `create_fp_placement` command honors the specified placement keepouts when it places the cell instances in the relative placement group, and no cell instances in the relative placement group are placed over the keepout areas.

Using the Default Relative Placement Flow

Use the default relative placement flow if you do not want to fix the relative placement groups after you run the `create_fp_placement` command.

[Figure 6-26](#) shows the default relative placement flow.

Figure 6-26 Default Relative Placement Flow



Use the `legalize_placement` command to automatically legalize the standard cells. The standard cells in relative placement groups might be moved outside of the corresponding relative placement bounding box during legalization.

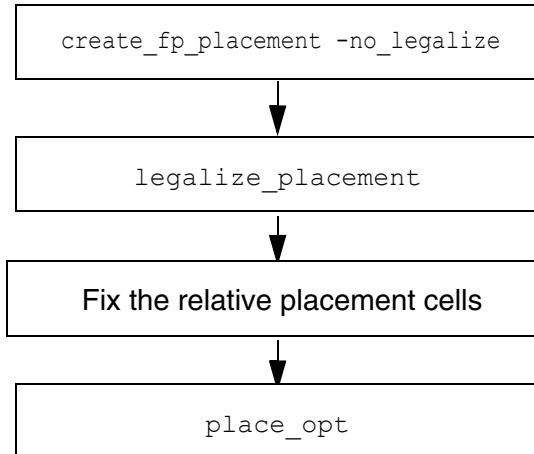
Using the Relative Placement Cells Flow

Use the relative placement cells flow if your design contains either macro-only or standard cell-only relative placement groups and you need to fix the relative placement groups after floorplanning to ensure the exact relative placement constraints are met.

1. Define the relative placement constraints.
 - Create the relative placement groups by using the `create_rp_group` command.
 - Add relative placement objects to the groups by using the `add_to_rp_group` command.
2. Run virtual flat placement by using the `create_fp_placement -no_legalize` command.
3. Set the `is_fixed` attribute to `true` for all macros.
4. Legalize your design by using the `legalize_placement` command.
5. Set the `is_fixed` attribute to `true` for all relative placement cells.
6. Perform physical placement by using the `place_opt` command.

[Figure 6-27](#) shows the flow for fixing the relative placement cells.

Figure 6-27 Fixing the Relative Placement Cells Flow



Use this flow to perform virtual flat placement if your design contains relative placement groups that have mixed macros and standard cells.

1. Run virtual flat placement by using the `create_fp_placement -no_legalize` command.
2. Set the `is_fixed` attribute to `true` for all macros.

3. Legalize your design by using the `legalize_placement` command.
4. Set the `is_fixed` attribute to `true` for all macros and standard cells in the relative placement groups.
5. Continue the design planning flow.

Propagating Relative Placement Groups by Commit and Uncommit

During the commit and uncommit hierarchy steps of the design planning flow, the `commit_fp_plan_groups` and `uncommit_fp_soft_macros` commands support relative placement groups, including hierarchical relative placement groups. The `commit_fp_plan_groups` command converts plan groups into new soft macros. If the plan group includes relative placement groups, they are automatically propagated from the top-level design to soft macros. The `uncommit_fp_soft_macros` command converts soft macros into top-level plan groups. If the soft macro includes relative placement groups, they are automatically propagated to the plan group that is at the top level.

Committing Relative Placement Groups

As described in the following steps, for each plan group that the `commit_fp_plan_groups` command processes, the relative placement groups are searched to determine if they belong to the plan group that is being committed.

1. All relative placement groups in the top-level design are examined and for each relative placement group, the cell instances of the hierarchy are examined to see if they belong to a descendant of a plan group's logical hierarchy.
2. If cell instances that belong to each relative placement group are determined to be logical descendants of the plan group's logical hierarchical cell instance, the relative placement group is processed by the `commit_fp_plan_groups` command, and an exact copy of it is created in the new soft macro, according to its placement within the plan group boundary.

Any keepout blockages associated with the relative placement group are pushed down, and the cell instances are assigned to the relative placement group. The keepout blockages are also checked to see if they are included within the plan group boundary. If they extend outside the plan group boundary, a warning message is issued, but the commit process continues.

3. After all relative placement groups that belong to the plan group are pushed down, a link step occurs that populates hierarchical attachment lists in some of the relative placement groups that have hierarchical trees. Relative placement group instances are also instantiated.

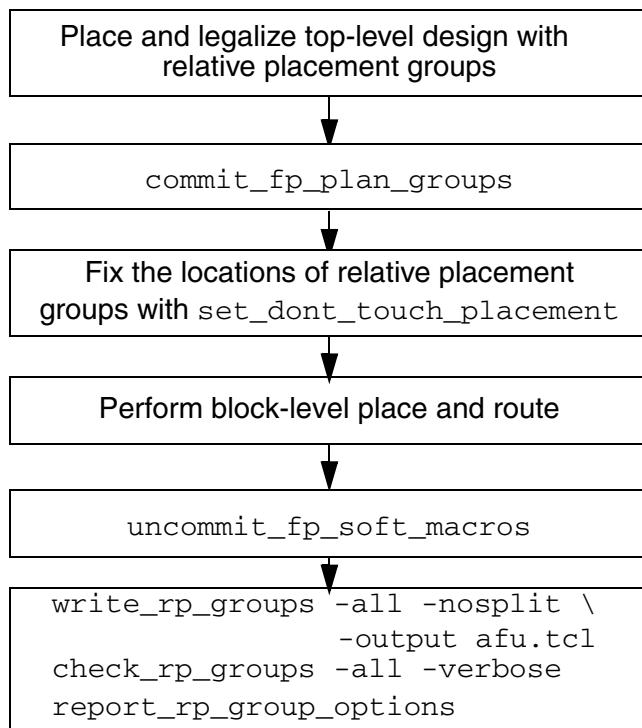
Uncommitting Relative Placement Groups

For each soft macro that the `uncommit_fp_soft_macros` command processes, all relative placement cells in the relative placement groups are replicated in the top-level design according to the hierarchical cells and the hierarchical cell instance references within the soft macro.

Design Flow for Hierarchical Relative Placement Groups

[Figure 6-28](#) shows the flow for designs that contain hierarchical relative placement groups.

Figure 6-28 Hierarchical Relative Placement Group Design Flow



Use the following flow for designs that contain hierarchical relative placement groups.

1. Place and legalize the design with hierarchical relative placement groups by using the `create_fp_placement -no_legalize` and `legalize_placement` commands.
2. Convert plan groups into soft macros, also called child cells or physical blocks, by using the `commit_fp_plan_groups` command.

The command extracts relative placement groups that belong to the plan group from the top CEL view and pushes them down to soft macros. Each soft macro is a CEL view of the block in the same Milkyway design library.

3. Fix the locations of the relative placement groups, which include macros, by using the `set_dont_touch_placement` command.

4. Perform place and route on each soft macro independently.

5. Convert all soft macros into plan groups by using the `uncommit_fp_soft_macros` command.

During uncommit hierarchy, the soft macro child cell instances are pushed back up to the top CEL view.

6. After the uncommit process, you can check the validity of the relative placement groups by using the following commands:

```
icc_shell> write_rp_groups -all -nosplit -output afu.tcl  
icc_shell> check_rp_groups -all -verbose  
icc_shell> report_rp_group_options
```

Placement Evaluation

No straightforward criteria exist for evaluating the initial placement of hard macros.

Measuring the quality of results (QoR) is subjective and often depends on practical design experience. You should observe some basic rules when measuring QoR, such as pushing hard macros to the core boundary and aligning similar hard macros. However, the most important issues are timing and routability.

QoR can be measured by the following criteria:

- Routability

You can measure the routability of your design by analyzing the routing congestion produced by the global router. The data used to analyze congestion consists of a textual report and visual heat maps that show where the routing congestion hotspots exist in the design. Highly congested designs might not be routable. Hard macros tend to create congestion around their edges and corners. You should analyze hard macro placements for routability early in the design cycle.

- Timing

Timing calculations can use the placement information to better estimate interconnect loading. The interconnect timing calculations should take into account detours in routes caused by the presence of hard macros. Good placement minimizes timing violations.

- Wire length

Smaller values of total wire length are a good indicator of placement quality. The total wire length is reported in the placement log file. You should monitor this number when you assess multiple placement solutions.

Another aspect of wire length is localized to hard macros. By looking at the signal (flyline) connections of a hard macro, you can quickly determine whether the hard macro is placed in an optimal location. For example, a hard macro placed in the lower-left corner of the core area that is extensively connected to logic in the upper-right corner of the die indicates a poor location for the hard macro.

- Data flow

If you know the logic and intended operation of the design, you can assess the data flow by using the hierarchical browser at any stage in the design flow to observe where logical modules and hard macros are physically placed. Using this technique can help you make sensible placement decisions.

- Standard cell placement areas

You can visually assess the placement of macros and standard cells. Small areas surrounded by hard macros usually cause congestion hot spots. Unless the connections to and from standard cells in these areas are completely localized, it is difficult to complete the connections from within these areas to the objects outside these areas. Generally, a contiguous standard cell placement area without bottlenecks is desirable.

After passing these QoR measurements, the placement result qualifies as a good starting point for further manual tuning.

Placement Quality of Results Report

You can generate a quality of results (QoR) report for virtual flat placement by using the `report_fp_placement` command. The command reports the following information:

- Total wire length
- Number of regions with high density
- Number of regions with low density
- Number of overlaps between hard macros
- Number of overlaps between hard macros and standard cells
- Number of overlaps between plan groups
- Number of cells that are not inside their respective plan groups (or core area)
- Number of cells violating the core area

If your design does not have channels, you can use the `-check_abutment` option to check for the abutment of each soft and hard macro with the surrounding macro cell instances. The abutment check reports gaps detected between the macro cell instances.

Evaluating Macro Placement

Poor macro placement might cause high congestion, timing problems, and other design issues. You can quickly evaluate the quality of the macro placement in your design and report a placement score by using the `evaluate_macro_placement` command.

```
evaluate_macro_placement
[-ignore_std_cell_placement]
[-misaligned_channel_relative_weight weight]
[-net_detour_relative_weight weight]
[-swimming_pool_relative_weight weight]
[-thin_channel_relative_weight weight]
[-wirelength_relative_weight weight]
```

You can use the `evaluate_macro_placement` command to measure macro placement quality and compare the results among several different design iterations. The command produces a score between 0 and 100 that represents the relative level of placement issues in your design. A lower score represents a relatively better placement.

You can fine-tune the weighted macro placement score by specifying the following options to the `evaluate_macro_placement` command. [Table 6-1](#) describes the command options that control the weight settings.

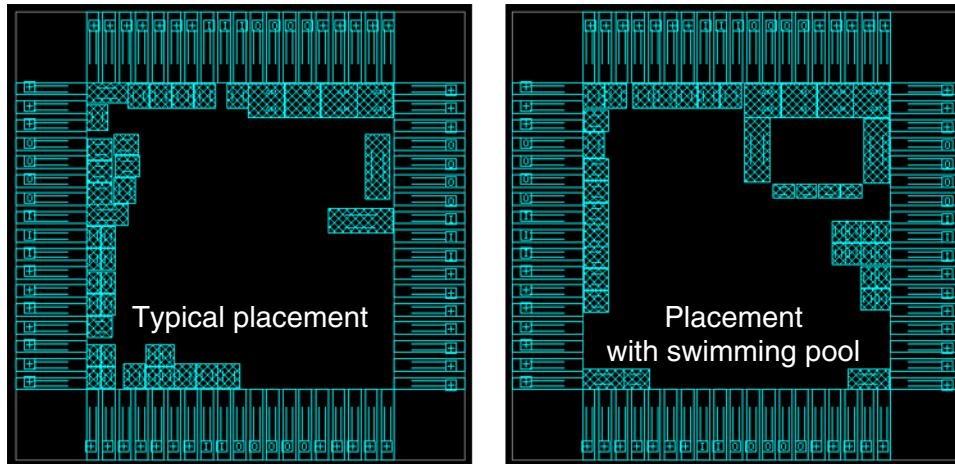
Table 6-1 evaluate_macro_placement Weight Options

Option	Description
<code>-misaligned_channel_relative_weight</code>	Weight for consecutive channels in the same direction that are not aligned.
<code>-net_detour_relative_weight</code>	Weight for expected net detours due to macro placement.
<code>-swimming_pool_relative_weight</code>	Weight for regions surrounded by macros.
<code>-thin_channel_relative_weight</code>	Weight for narrow channels between macros that are wide enough for standard cell placement.
<code>-wirelength_relative_weight</code>	Weight for the wirelength of nets connected to macros.

By default, the weight is 0.2 for each of the previous options. If you specify a cumulative weight that is not equal to 1.0, the tool adjusts each weight proportionally such that the sum of all weights is 1.0.

The following examples illustrate the results of the `evaluate_macro_placement` command on two different placements. In [Figure 6-29](#), the design on the left contains a typical placement. The design on the right of the figure contains a swimming pool in upper right of the layout. This swimming pool region is a standard cell placement area that is completely surrounded by hard macros.

Figure 6-29 Example Macro Placements



The output of the `evaluate_macro_placement` command for the typical layout shows a weighted cost of 6. The command output is as follows:

```
icc_shell> evaluate_macro_placement
Reading database ...
Checking for macro overlaps ... none
Swimming pools ... area: 1506
Net detours ... length: 158215
Wirelength (nets connected to macros) ... 362009
Thin channels ... 4
Misaligned channels ... 1 pair
```

metric	cost	relative weight	actual weight	weighted cost
swimming pools	3	1.000	0.200	0.600
net detours	1	1.000	0.200	0.200
wirelength	23	1.000	0.200	4.600
thin channels	2	1.000	0.200	0.400
misaligned channels	3	1.000	0.200	0.600
overall cost				6

1

The output of the `evaluate_macro_placement` command for the layout that contains the swimming pool shows a weighted cost of 11. The command output is as follows:

```
icc_shell> evaluate_macro_placement
Reading database ...
Checking for macro overlaps ... none
Swimming pools ... area: 130578
Net detours ... length: 265286
Wirelength (nets connected to macros) ... 325924
Thin channels ... 2
Misaligned channels ... 1 pair

          relative    actual    weighted
metric        cost      weight      weight      cost
-----
swimming pools     28      1.000      0.200      5.600
net detours       2       1.000      0.200      0.400
wirelength        20      1.000      0.200      4.000
thin channels      2       1.000      0.200      0.400
misaligned channels 4       1.000      0.200      0.800
-----
overall cost           11
```

1

Total Wire Length Estimate

You can use the `get_fp_wirelength` command to evaluate the overall placement quality. The command generates vertical, horizontal, and total wire length statistics for the top-level signal nets in your design. It also calculates the virtual route wire length as a minimum length needed to connect all the pins.

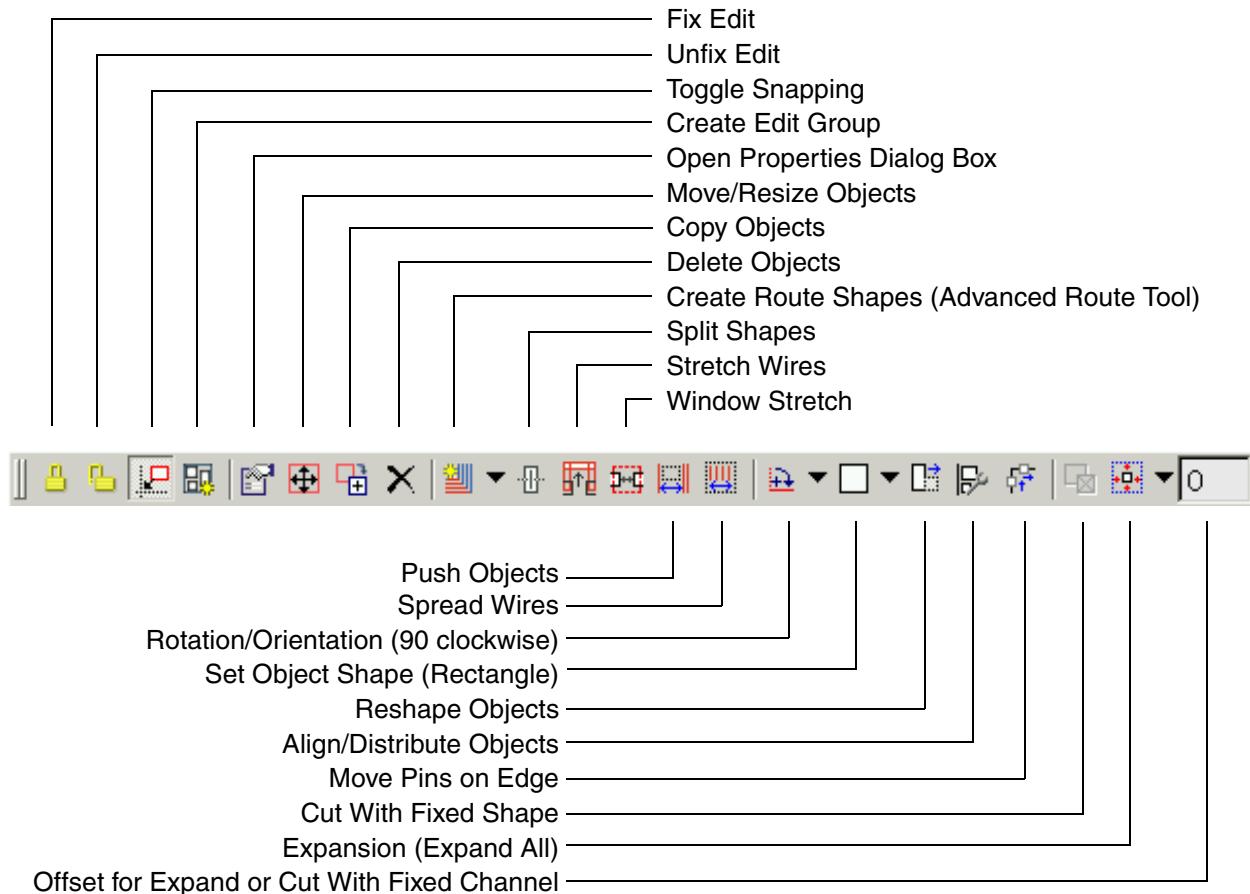
Floorplan Editing

You can use IC Compiler to perform the following floorplan editing operations:

- Create and delete objects
- Move, rotate, align, distribute, and spread objects
- Fix and unfix objects for editing
- Undo and redo edit changes
- Resize, expand, reshape, and split objects

You can perform many types of editing in the GUI layout view. To display or remove the toolbars containing the editing command icons, use View > Toolbars > Edit and View > Toolbars > Advanced Edit. The editing icons and their functions are shown in [Figure 6-30](#).

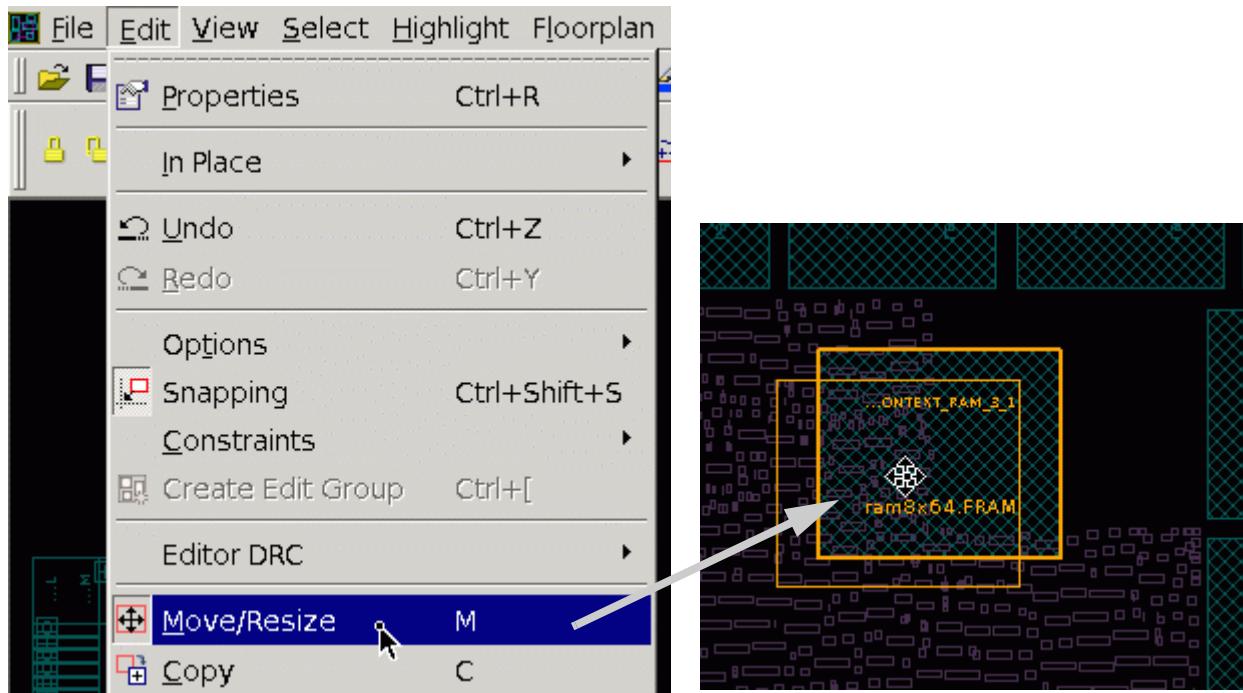
Figure 6-30 GUI Editing Tools



For example, to move a hard macro, use the selection tool (arrow icon) to select the macro. Click the move and resize tool icon. The cursor changes to a four-way arrow shape. Place the cursor on the macro, and drag it to the desired location.

Instead of clicking the move/resize icon, you can go to the command menu and choose Edit > Move/Resize, or you can type the M key on the keyboard. The command menu shows the icon and keyboard shortcut associated with the command, as shown in [Figure 6-31 on page 6-54](#).

Figure 6-31 Moving an Object With the Move/Resize Tool



For detailed information about using the GUI to perform editing tasks, see the “Editing a Design” topic in IC Compiler online Help and the Using GUI Tools appendix in the *IC Compiler Implementation User Guide*.

Removing Cell Placement

You can move (unplace) all movable macro cells or standard cells and place them in a new location by using the `remove_placement` command. The command does not affect cells with `is_fixed true` or `dont_touch` attributes assigned to them. This is the command syntax:

```
remove_placement
[-object_type standard_cell | macro_cell | all]
[-new_location top_right | origin | center]
```

By default, the `remove_placement` command by itself moves all movable standard cells and macro cells to outside of the chip and spreads them out, effectively taking you back to the starting point of the placement task.

To move only the standard cells or only the macro cells, use the `-object_type` option. To move the cells to the origin, chip center, or top and right sides of the chip, use the `-new_location` option.

To delete other floorplan objects at the design planning stage such as placement blockages, bounds, plan groups, voltage areas, plan group padding, and block shielding, choose Floorplan > Delete Floorplan Objects in the GUI and the select the type of objects you want to delete. You can also use one or more of the following commands to remove floorplan objects: `remove_placement_blockage`, `remove_bounds`, `remove_plan_groups`, `remove_route_guide`, `remove_voltage_area`, `remove_fp_plan_group_padding`, `remove_fp_block_shielding`, or `remove_track`.

Packing Hard Macros Into an Area

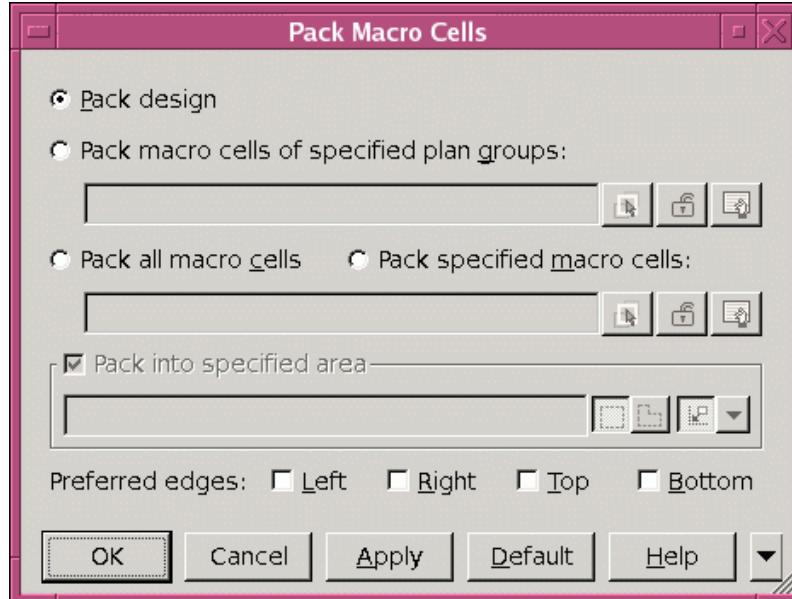
After performing virtual flat placement, you can specify an area (region) and automatically pack selected hard macros along the edges of that area.

To automatically pack hard macros into an area,

1. Choose Placement > Pack Macro Cells.

The Pack Macro Cells dialog box appears. See [Figure 6-32](#).

Figure 6-32 Pack Macro Cells Dialog Box



Alternatively, you can use the `pack_fp_macro_in_area` command.

2. Set the options, depending on your requirements.

- Pack design – Packs all objects in the core area.
- Pack macro cells of specified plan groups – All hard macros belonging to the selected plan groups are packed inside the plan group boundaries. If a plan group boundary is outside the core area, any macros belonging to that plan group are ignored. Top-level macros and macros belonging to other plan groups are also ignored.
- Pack all macro cells – Select this option to pack all macro cells inside the core area. Any top-level macros are also packed into the remaining area.
- Pack specified macro cells – Select this option to pack only the specified macro cells into the core area. If a polygon or rectangle is selected (it can be on any layer except a blockage layer) together with hard macros, the hard macros are packed into the selected polygon or rectangle.
- Preferred edges – Specify the edges along which to pack the macros.

You can specify one or more edges along which to pack the hard macros: Left, Right, Top, or Bottom. These are the preferred edges for placing the macro cells. For example, if you select Left, Top, and Bottom, the macro cells are placed along those edges first, arranging them in a “C” shape.

Selecting all four edges causes the macro cells to be placed along all four edges, arranging them in an “O” shape. This is different from specifying no edges, which causes placement to be based on other considerations, such as wire length.

3. Click OK or Apply.

Manually Adjusting the Hard Macros

To manually adjust the hard macros, you should observe the following principles, in addition to applying your own criteria:

- Follow the given relative locations from the hard macro placement command (`create_fp_placement`).
- Avoid isolated standard cell placement areas enclosed by a ring of hard macros.
- Avoid narrow standard cell placement areas in channels (slivers) between hard macros.
- Align similar hard macros.
- Push large hard macros to the core boundary.
- Create a rectangular placeable area for standard cell placement.

The “Align objects” and “Distribute objects” buttons on the Edit toolbar are particularly helpful for adjusting your hard macros.

Using Constraints With the Core and Die Editing Commands

You can use the `-keep_placement` and `-keep_pad_to_core_distance` options with the editing commands that are relative to the core and die. Using these options makes it easier and more convenient to refine and debug core and die commands.

These options are effective only when they are applied to the following core and die editing commands:

```
set_object_boundary  
resize_objects  
set_object_shape  
window_stretch  
move_objects  
cut_objects
```

When you move or change a core or die, you can use the `move_objects -keep_placement` command to ensure that standard cells and hard macros remain within the boundary of the parent object (for example, the core, the die, or a plan group).

You can use the `-keep_pad_to_core_distance` option to maintain the distance between the core and the I/O pad cells. This option also maintains the distance between the core and the die. This forces changes to the core and die to be made in tandem. If the I/O pad cells are absent, the core-to-die distance is still maintained.

Note:

After moving or resizing objects, the resulting placement is not legalized. You must explicitly legalize the placement by using the `legalize_placement` command.

Voltage Area Planning

You can automatically place and shape voltage areas, including voltage areas that are physically nested. You can also create voltage areas outside the chip area.

This section includes the following topics:

- [Supported Voltage Area Physical Boundary Scenarios](#)
- [Updating Voltage Areas in a Design](#)
- [Removing Voltage Areas](#)

In IC Compiler, you can define voltage areas at any level of a logic hierarchy. A voltage area can be a single rectangular or rectilinear shape or can consist of multiple disjoint rectilinear and rectangular shapes.

Power domains and voltage areas should maintain a one-to-one mapping relationship. A power domain and its matching voltage area should have the same name and same set of hierarchical cells. When a power domain is mapped with the voltage area, the associated logic information is automatically derived from the power domain.

Voltage areas can be explicitly associated with existing power domains, or they can be created without specifying any explicit association with power domains. They can also be created when no power domains have been defined for the design.

A given voltage area can contain one or more physically nested voltage areas. A voltage area must completely contain its nested voltage areas. These nested voltage areas correspond to logically nested power domains. Voltage areas that would overlap physically must be resolved into nonoverlapping, abutted rectangular or rectilinear subareas. Intersecting voltage areas are not supported.

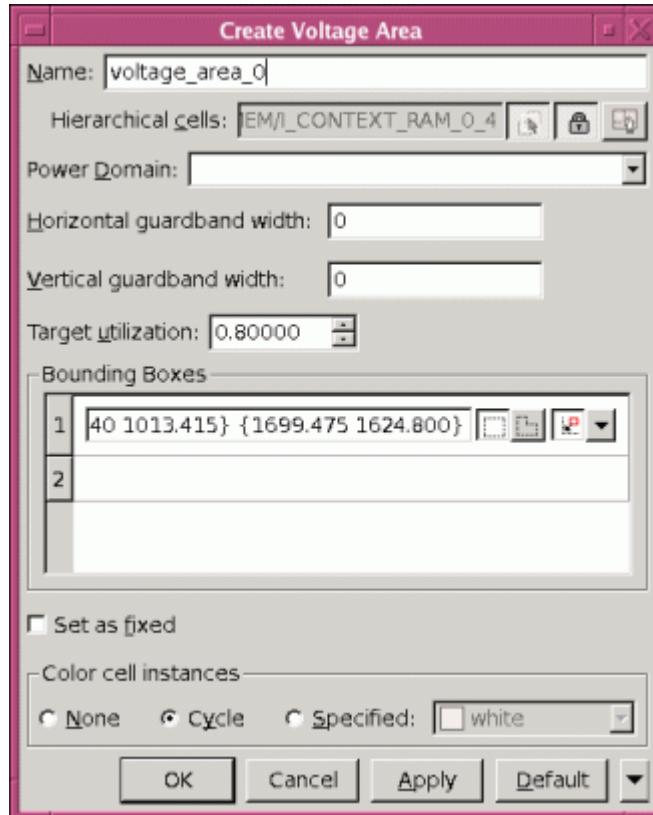
Planning the location and shape of voltage areas requires the following steps:

1. Create a voltage area of a specific geometry on the core area of the chip.

Choose Floorplan > Create Voltage Area.

The Create Voltage Area dialog box appears. See [Figure 6-33 on page 6-59](#).

Figure 6-33 Create Voltage Area Dialog Box



Alternatively, you can use the `create_voltage_area` command.

- Name – Specify the name of the voltage area you want to create.

If you specify this option, you must also provide a list of hierarchical cells that are contained within the voltage area. The logical hierarchies are assigned to specific voltage areas when you create or update these voltage areas.

- Power Domain – If your design contains power domains, you can use this option with the corresponding power domain name to create a voltage area. Then the power domain name is also the voltage area name. After the voltage area is created, the power domain contains the corresponding boundary information.

In this case, do not specify a list of hierarchical cells. The cell list is provided in the power domain definition. Power domains and voltage areas should maintain a one-to-one mapping relationship. A power domain and its matching voltage area should have the same name and same set of hierarchical cells. IC Compiler places these cells in the given voltage area. Using the power domain option ensures the correct alignment of the logical hierarchies with the voltage areas.

- Horizontal guard band width and Vertical guard band width – 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, use guard bands 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 specify a guard band width in the horizontal and vertical direction when creating or updating voltage areas.

- Target utilization – Specifies the target utilization for the voltage area during shaping.

You must specify a utilization value between 0.1 and 1.0. The default utilization is the same value used for the rest of the design. Larger utilization values result in smaller voltage areas, with less extra space allowed for routing.

To change the target utilization, you must use the `remove_voltage_area` command to remove all voltage areas or specified voltage areas from the design, and then re-create the voltage area and the new target utilization value.

- Bounding Boxes – If you know an ideal location for a voltage area, you can direct it by specifying explicit coordinates. The voltage area geometry can be a rectangle or a rectilinear polygon.

To define a rectangular voltage area, select the rectangular-shaped button, and type the x- and y-coordinates for the lower-left and upper-right corners of the rectangle in the Coordinates box.

To define a rectilinear voltage area, select the rectilinear-shaped button, and type the x- and y-coordinates for each corner of the polygon in the Coordinates box.

After the user-defined voltage areas are created, the tool automatically derives a default voltage area for placement of cells not specifically assigned to any voltage areas.

- Set as fixed – You can place newly created voltage areas inside the core in fixed locations. The voltage areas with the `is_fixed` attribute set to `true` are ignored by the `shape_fp_blocks` command and are not reshaped. By default, this option is off.

- Color cell instances – (Optional) Select a method for coloring cells in the voltage area.

To disable cell coloring, select None.

To cycle through the available colors, select Cycle. This is the default.

To apply a specific color, select Specified and select a color in the color list.

2. Click OK or Apply.

3. Use the `create_fp_placement` command to place the design, keeping voltage area cells together. You can also choose Placement > Place Macro and Standard Cells.
4. Use the `shape_fp_blocks` command and options to automatically shape and place the voltage area boundaries. You can also choose Placement > Place and Shape Plan Groups. For more information, see “[Automatically Placing and Shaping Objects in a Design Core](#)” on page 7-18.

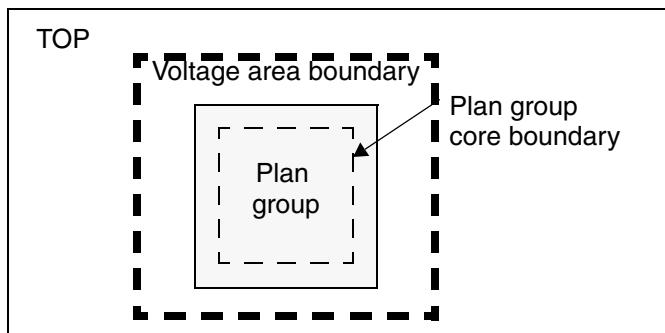
For more information about voltage areas in multivoltage designs, see Defining and Using Voltage Areas in the *IC Compiler Implementation User Guide*. For more information about nested voltage areas in multivoltage designs, see Handling Nested Voltage Areas in the *IC Compiler Implementation User Guide*.

Supported Voltage Area Physical Boundary Scenarios

The following voltage area physical boundary scenarios are supported.

- The voltage area boundary encloses the plan group core boundary, as shown in [Figure 6-34](#).

Figure 6-34 Voltage Area Boundary Encloses Plan Group Boundary

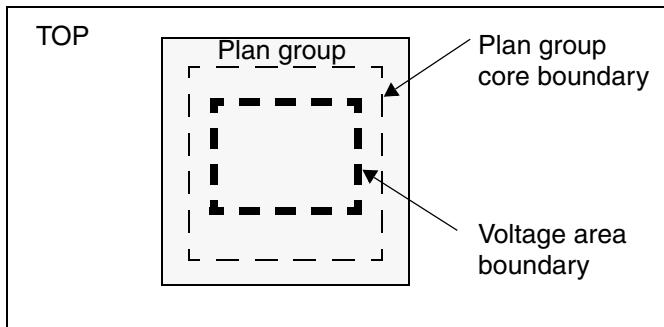


During commit hierarchy by using the `commit_fp_plan_groups` command, the physical voltage area boundary is not copied down into the soft macro, and child cell instances are not assigned logically to any nondefault voltage areas. The power domain is updated to swap the hierarchical cell with the new cell instance.

During uncommit hierarchy by using the `uncommit_fp_soft_macros` command, the soft macro child cell instances are pushed back up to the top CEL view, and are reassigned logically to the original top cell nondefault voltage area.

- The voltage area boundary is contained by the plan group core boundary, as shown in [Figure 6-35 on page 6-62](#).

Figure 6-35 Voltage Area Boundary Contained by Plan Group Boundary



During commit hierarchy, the physical voltage area boundary is copied down into the soft macro, and the child cell instances are assigned logically to their respective voltage areas. The effects on associated power domains are not considered.

During uncommit hierarchy, the soft macro child cells are assigned logically to the original top cell nondefault voltage area.

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_areas` 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.

Removing Voltage Areas

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 voltage area. If you remove a logically nested voltage area, the logical hierarchy inherits the voltage properties of the parent hierarchy.

Placing Multiply Instantiated Modules

You can perform virtual flat placement of multiply instantiated modules. Multiply instantiated modules are physically represented in the design as plan groups and have the same reference. A set of these instantiated modules is called a multiply instantiated module group. A design can have more than one multiply instantiated module group. If for example, modules A1 and A2 have reference A, and modules B1 and B2 have reference B, two

multiply instantiated module groups are formed. Many designs use multiple instances of the same logic block to create the required functionality. By using the same block for similar operations, the design time is reduced.

This section includes the following topics:

- [Identifying Multiply Instantiated Modules](#)
- [Identifying Multiply Instantiated Module Plan Groups](#)
- [Shaping Multiply Instantiated Module Plan Groups](#)
- [Placing and Analyzing Multiply Instantiated Module Plan Groups](#)
- [Flipping Multiply Instantiated Module Plan Groups](#)
- [QoR Analysis of Multiply Instantiated Module Plan Groups](#)

Identifying Multiply Instantiated Modules

Before you can place the multiply instantiated modules, you must first determine which plan groups and soft macros are to be considered as multiply instantiated modules. This can be done by using the `uniquify_fp_mw_cel -store_mim_property` command to store the multiply instantiated module information in the Milkyway database as a multiply instantiated module property. After unification, other IC Compiler design planning commands can access the multiply instantiated module information and treat the cell instances with these attributes as multiply instantiated modules in the virtual flat placement design planning flow.

The `uniquify_fp_mw_cel` command uses the following syntax:

```
uniquify_fp_mw_cel -store_mim_property cell_instances
```

For example:

```
icc_shell> uniquify_fp_mw_cel \
           -store_mim_property [get_cells {instA instB instC}]
```

If you specify the `-store_mim_property` option, the tool sets the `mim_master_name` attribute to the reference design name for the specified cell instances.

Removing the Multiply Instantiated Module Property

You can use the `remove_mim_property` command to remove the multiply instantiated module data for selected cell instances. You must specify a list of hierarchical cells for which the multiply instantiated module data is to be removed. If a cell is not found, a warning message is issued.

The cell instances are no longer treated as multiply instantiated modules; each cell instance is treated as a unique reference cell.

The `remove_mim_property` uses the following syntax:

```
remove_mim_property collection_of_cells
```

For example:

```
icc_shell> remove_mim_property [get_cells instA]
```

Identifying Multiply Instantiated Module Plan Groups

Use the `report_mim` command to print a list of all multiply instantiated module plan groups in the design. Multiply instantiated soft macro modules are also identified by this command, although for placement, they are treated as hard macros.

Here is an example of a report:

```
icc_shell> report_mim
*****
Report      :MIM
Design      :test.CEL;1
Version     :E-2010.12
Date        :Thu Oct 28 17:01:51 2010
*****
Master BLENDER is instantiated to:
  Soft Macro: 1_ORCA_TOP/1_BLENDER_4
  Soft Macro: 1_ORCA_TOP/1_BLENDER_3
  Soft Macro: 1_ORCA_TOP/1_BLENDER_2
  Soft Macro: 1_ORCA_TOP/1_BLENDER_1
  Soft Macro: 1_ORCA_TOP/1_BLENDER_5
  Reference Blender is instantiated 5 times.
  Total 1 MIM references and 5 MIM instances.
  MIM plan group flipping grid:x offset=425.5,x step=0.410
                                y offset=425.565,y step=7.380
  MIM Report End
```

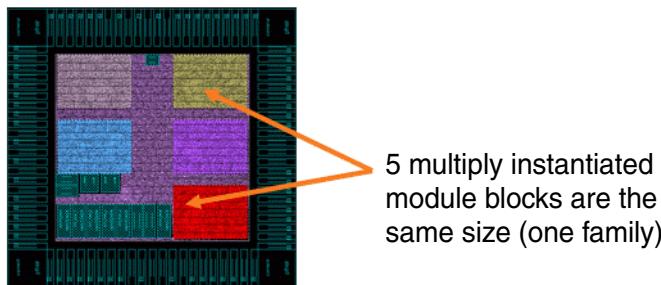
The report also provides information about the status of the copied placement data and suggests a multiply instantiated module plan group flipping grid on which to place a multiply instantiated module plan group. When the plan group is flipped, the standard cell placement remains legal.

Shaping Multiply Instantiated Module Plan Groups

After running virtual flat placement by using the `create_fp_placement` command, you can use the `shape_fp_blocks` command to shape the plan groups in each multiply instantiated module group automatically. For more information, see “[Automatically Placing and Shaping Objects in a Design Core](#)” on page 7-18.

The plan groups in each multiply instantiated module group are shaped and sized identically, as shown in [Figure 6-36](#).

Figure 6-36 Plan Groups of Multiply Instantiated Module Groups in the Same Size and Shape



All corners of the multiply instantiated module plan groups are aligned on their row and column placement boundaries and are placed in such a way that they can be legally flipped, provided the rows are uniform.

Note:

The default plan group locations might not be aligned.

Placing and Analyzing Multiply Instantiated Module Plan Groups

Run the `create_fp_placement` command to place the macros and standard cells into fixed placement plan group boundaries. However, the placement in the multiply instantiated module groups will not be identical.

You can use the `copy_mim` command and options to perform various manipulations during the virtual flat placement stage on the multiply instantiated modules to determine which of the plan groups in each multiply instantiated module group is the master instance of that multiply instantiated module group.

The placement of the master instance is copied to the other plan groups in the multiply instantiated module group. The copied information, which is saved to the Milkyway database, contains the original placement of all the cells in the copied plan multiply instantiated module plan groups relative to the lower-left corner of the plan group and the orientation of the plan group.

The `copy_mim` command uses the following syntax:

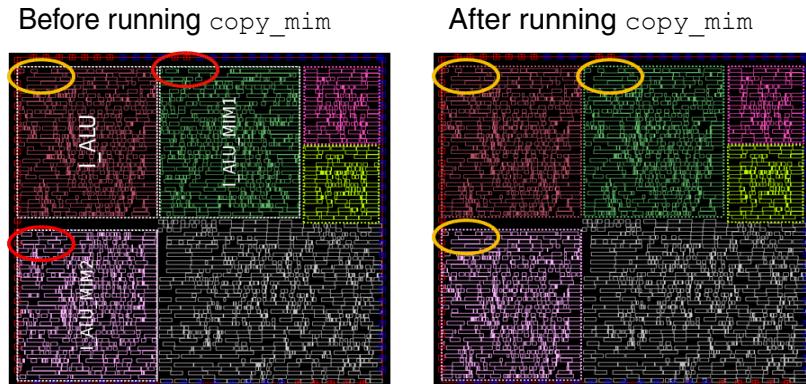
```
copy_mim
[-type placement | blockage | boundary]
[-restore_placement]
mim_plan_group
```

The `mim_plan_group` argument specifies a list containing one multiply instantiated module plan group.

In [Figure 6-37](#), the placement is different in every plan group before running the `copy_mim` command. After running the `copy_mim` command, the tool copies the cell placement of the `1_ALU` multiply instantiated module to the other multiply instantiated modules in the same group.

```
icc_shell> copy_mim -type placement 1_ALU
```

Figure 6-37 Before and After Running the copy_mim Command



As described in the following sections, you can create multiply instantiated modules with the same cell placement, blockages, and shapes.

Copying the Cell Placement of a Plan Group

Use the `copy_mim -type placement` command to copy the cell placement of the specified multiply instantiated module to all other multiply instantiated modules in the same group. The command replicates the original placement of all the cells in the specified multiply instantiated module relative to the lower-left corner of the plan group.

Note:

Placement blockages are not touched by this command. Errors are reported if the plan group is not part of a multiply instantiated module group or the plan groups in the plan group's multiply instantiated module group do not have the same size and shape.

You can make more than one copy of a multiply instantiated module group; however, previous versions are discarded because only one level of undo is supported.

You can use the `report_mim` command to list the copied multiply instantiated module groups and their corresponding reference plan group. If the shapes or sizes of the plan groups changed from the previous `copy_mim -type placement` command, this information is also reported.

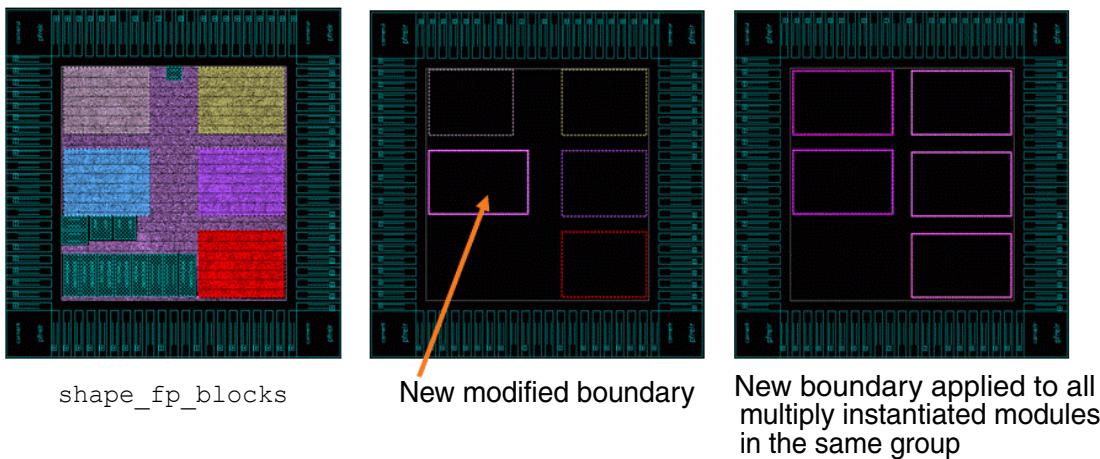
Copying Placement Blockages and Boundaries

Use the `copy_mim -type blockage` command to copy the placement blockages from the specified multiply instantiated module to all the other plan groups in the same multiply instantiated module group. This command uses the plan group orientation you set by using the `flip_mim` command. For more information, see “[Flipping Multiply Instantiated Module Plan Groups](#)” on page 6-68.

Use the `copy_mim -type placement` or `copy_mim -type blockage` command to reshape each plan group that belongs to a multiply instantiated module group so that their boundaries are identical to the specified plan group. The tool preserves the orientation of the individual plan groups.

[Figure 6-38](#) shows an example of a modified boundary applied to all the other multiply instantiated modules in the same group.

Figure 6-38 Modified Boundary Applied to Other Multiply Instantiated Modules in the Same Group



Restoring the Placement of Cells in Plan Groups

Use the `copy_mim -restore_placement` option to restore the placement of cells in the specified plan group and all other plan groups in the same multiply instantiated module group. The command also restores the orientation or flipping about the x- or y-axis.

Note:

Placement blockages are not touched by this command. Errors are reported if any size or shape does not correspond to the original size or shape in the specified multiply instantiated module group.

Flipping Multiply Instantiated Module Plan Groups

Use the `flip_mim` command to flip the placement of the cells in the multiply instantiated module plan group and set the orientation. The resulting pins on the multiply instantiated module are also flipped. After running the `flip_mim` command, the resulting placement is the same as if the plan group was converted to a soft macro, flipped, and then converted back to a plan group.

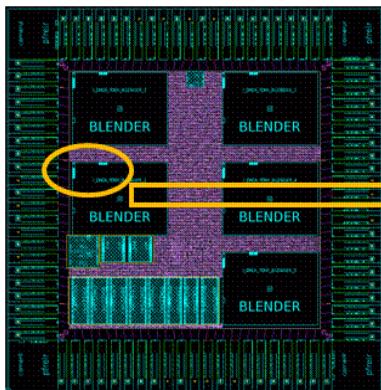
The `flip_mim` command uses the following syntax:

```
flip_mim
[-direction x | y]
plan_groups
```

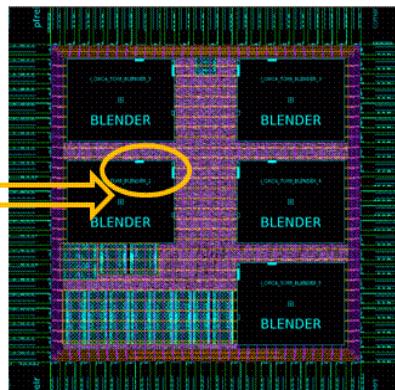
[Figure 6-39](#) shows a layout before and after running the `flip_mim` command.

Figure 6-39 Flipping Multiply Instantiated Modules

Before running `flip_mim`



After running `flip_mim`



Only mirroring and flipping about the x- or y-axis is allowed for multiply instantiated module plan groups. The command reports an error if the plan group is not rectangular.

During flipping, the bounding box of the plan group does not move.

Placement blockages are not flipped, however, the `copy_mim -type blockage` command recognizes the flipped orientation.

Horizontal Requirements for Plan Group Locations

Consider the following horizontal requirements for plan group locations, when running the `flip_mim` command. The requirements ensure that the cells in the plan group have the same relative distance to the left and right plan group boundaries after flipping the plan group horizontally.

- Snap the left and right boundaries of a plan group to the edge of a unit tile or to a location that is one-half the unit tile width.
- Ensure that the plan group width is a multiple of the unit tile width.

Vertical Requirements for Plan Group Locations

Consider the following vertical requirements for plan group locations when running the `flip_mim` command. The requirements ensure that the cells in a plan group have the same relative distance to the top and bottom plan group boundaries after flipping the plan group vertically. The cells are flipped and placed legally at the rows with the opposite orientation.

- Snap the top boundary of a plan group to the top-side of a row and snap the bottom boundary of a plan group to the bottom-side of a row. Alternatively, you can snap both the top and bottom boundaries of the plan group to a location that is one-half the row height.
- Ensure that plan group height covers an even number of rows. (It must not be an odd number of rows.)

QoR Analysis of Multiply Instantiated Module Plan Groups

Use the `get_fp_wirelength` command to run a quality of results (QoR) analysis of the different multiply instantiated module plan groups.

The following options determine which plan group out of a set of multiply instantiated module plan groups has the best placement with respect to timing.

- Plan Group Interface Nets – Reports the wire length of interface nets for all plan groups. Interface nets have at least one pin inside the plan group and one pin outside the plan group boundary.
- Plan Group Internal Nets – Reports the wire length of internal nets for all plan groups. Internal nets have all their pins inside the plan groups.

7

Plan Groups

You can create plan groups for logic modules that need to be physically implemented as a group. A plan group restricts the placement of cells to a specific region of the core area. You can automatically place and shape objects in a design core, add padding around plan group boundaries, and prevent crosstalk by adding block shielding to plan groups and soft macros.

This chapter includes the following sections:

- [Creating Plan Groups](#)
- [Adding Padding to Plan Groups](#)
- [Adding Block Shielding to Plan Groups or Soft Macros](#)
- [Analyzing and Manipulating the Hierarchy](#)
- [Automatically Placing and Shaping Objects in a Design Core](#)
- [Physical-Only Cells in Plan Groups](#)
- [DFT-Aware Design Planning Flow](#)

Creating Plan Groups

You create plan groups to manipulate the hierarchy. (See [“Analyzing and Manipulating the Hierarchy” on page 7-8](#).) A plan group is a partition with a physical boundary. Each plan group represents a module in the logic hierarchy that you want to implement as a physical block and contains cells that belong to that module. A plan group can have many logical modules. The physical block inherits the size and shape (rectangular or rectilinear) of the plan group. Timing budgeting, pin cutting, and commit hierarchy are applied on plan groups.

Plan group boundaries are created for the modules you select and the modules are placed outside the chip boundary. Each plan group boundary is displayed in a different color in the layout window and in the hierarchy tree to show the relationship between the logical hierarchy and the physical groups of standard cells.

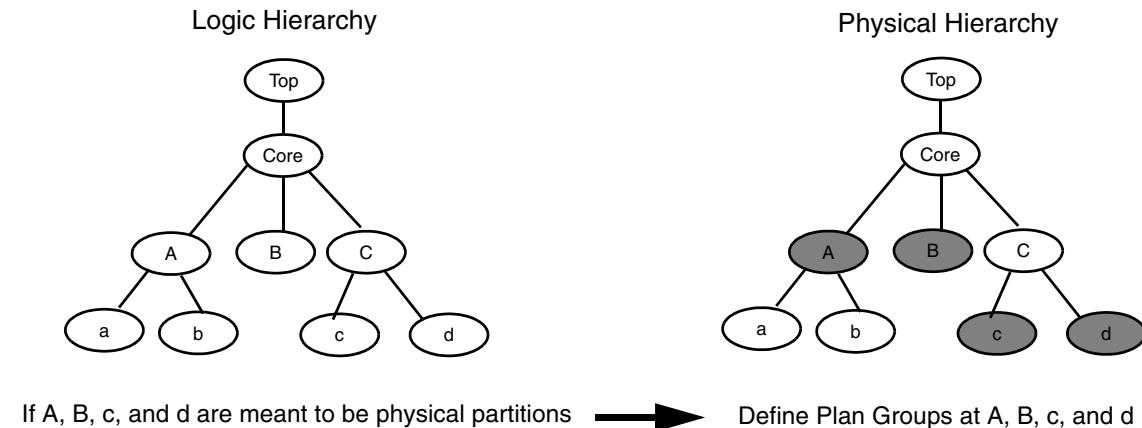
A plan group requires all of its cells to be placed inside of its boundary and prohibits the placement of other cells in the same area.

In the early stage of the design flow when your design contains the intended plan group partitions, you can analyze the pins and feedthrough pins to help identify issues before you fix the pin locations. Using this information, you can redefine route guides or constraints on the router to achieve better pin locations.

During commit hierarchy (`commit_fp_plan_groups` command), the plan groups are converted to soft macro cells or physical blocks. The command also creates pins on each soft macro. (See [“Converting Plan Groups to Soft Macros” in Chapter 13](#).) After the conversion is complete, the floorplan contains only the top-level standard cells and soft macro cells. Place and route can then work on each soft macro independently. These soft macros are then propagated to the top level in a FRAM view, and place and route is run on the whole chip design.

You can create plan groups for the logic modules that exist at any level in the logic hierarchy. [Figure 7-1 on page 7-3](#) shows an example of a relationship between a logic hierarchy and the corresponding physical hierarchy.

Figure 7-1 Relationship Between Logic Hierarchy and Physical Hierarchy



To create a plan group,

1. Choose Floorplan > Create Plan Group.

The Create Plan Group dialog box appears.

2. Click in the “Hierarchical cell” text box and enter the hierarchical cell names. A plan group is created for each valid hierarchical cell name in the list.

3. Define the plan group shape by doing one of the following:

- Select the Aspect option to define a plan group shape by its aspect ratio and utilization. This is the default.

Enter an aspect ratio (height divided by width) for the plan group area. If you specify a ratio of 1.00 (the default), the plan group is a square. If you specify a ratio of 3.00, for example, the height is three times the width.

Enter a utilization value. This is the area occupied by the cells of the plan group divided by the total plan group area.

- Select the Dimension option to define a plan group shape by its dimensions and enter the width and height in microns. The plan groups are placed outside the core area.
- Select the Region option to define a plan group by using coordinates.

To define a rectangular plan group shape, select the Rectangle button and draw the rectangle in the layout view, or enter the lower-left and upper-right coordinates in the Coordinates box.

To define a rectilinear plan group shape, select the Rectilinear button and draw the rectilinear shape in the layout view, or enter the lower-left and upper-right coordinates of the area rectangles in the Coordinates box.

4. (Optional) You can apply a specific color to a plan group. To do this, deselect the “Cycle” option to assign a new color to each new plan group automatically (the default). Select the “Specified” option to specify a particular color for the plan group color from the list. Select “None” for no color.
5. Click OK or Apply.

Alternatively, you can use the `create_plan_groups` command.

Removing the Plan Groups

To remove (delete) plan groups from the current design, choose Floorplan > Delete Floorplan Objects > Delete All Plan Groups. This command deletes all the plan groups.

Alternatively, you can use the `remove_plan_groups` command, which lets you selectively delete plan groups.

Note:

Deleting a plan group means canceling the grouping for future placement. The existing placed cells are not affected.

Adding Padding to Plan Groups

To prevent congestion and DRC violations, you can add padding around plan group boundaries. Plan group padding sets placement blockages on the internal and external edges of the plan group boundary to prevent cells from being placed there. Internal padding is equivalent to boundary spacing in the core area. External padding is equivalent to macro padding.

You can pad the plan groups to ensure that the placed standard cells do not extend over the plan group boundary and that cells belonging to the plan group remain inside the plan group. The cells that do not belong to the plan group remain outside the plan group when the physical hierarchy is committed. The boundary space is also needed for pins and the routing to those pins.

You should pad plan groups before refining the placement to ensure that the placement honors the plan group boundaries.

Padding the plan groups also reserves space for the pins on soft macros.

When the physical hierarchy is committed, the padding is transferred to a soft macro cell as core-boundary spacing and not as placement blockages in the child soft macro.

The plan group padding is visible in the layout window. The plan group padding is dynamic, which means that it follows the changes of the plan group boundaries.

To add padding to plan groups,

1. Choose Floorplan > Create Plan Group Padding.

The Create Plan Group Padding dialog box appears.

2. Specify the plan groups for which you want to add padding.

- All – Select this option to add padding for all plan groups. This is the default.
- Specified – Select this option to add padding for selected plan groups.

Enter the names of the plan groups.

3. Specify whether the padding is internal, external, or both.

- Internal – Select this option to add padding inside the plan group boundaries. Enter a width value. Internal padding should be at least 1 micron (the default) wide to allow space for pins and pin routing.
- External – Select this option to add padding outside the plan group boundaries. Enter a width value. The default is 0.

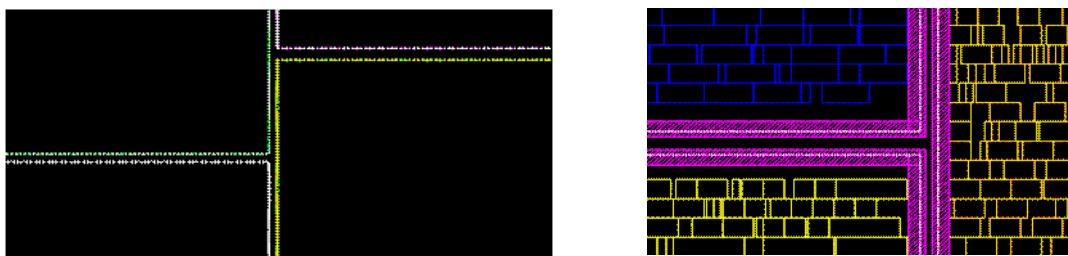
You can add both internal and external padding.

4. Click OK or Apply.

Alternatively, you can use the `create_fp_plan_group_padding` command.

[Figure 7-2](#) shows examples of plan groups with and without padding.

Figure 7-2 Example Floorplan With and Without Plan Group Padding



Removing the Plan Group Padding

To remove (delete) both external and internal padding for the plan groups, choose Floorplan > Delete Floorplan Objects > Delete Plan Group Padding. The Delete Plan Group Padding dialog box appears.

Alternatively, you can use the `remove_fp_plan_group_padding` command.

You can delete padding for selected plan groups or for all plan groups. The default is all plan groups.

Adding Block Shielding to Plan Groups or Soft Macros

When two signals are routed parallel to each other, crosstalk can occur between the signals, leading to an unreliable design. You can protect signal integrity by adding modular block shielding to plan groups and soft macros.

The shielding creates rectangular metal layers around the outside of the soft macro boundary or plan group in the top level of the design, and around the inside boundary of the soft macro or plan group, or both.

Usually you can selectively create metal layers to block the router from routing long nets along the block boundaries. However, note that even if you block layers in the preferred direction, nets can still be routed along the block boundaries. In that case, you can consider blocking all layers. The block shielding is created along the soft macro boundaries, but it leaves narrow openings to access the pins.

To handle signal integrity when creating hierarchical designs, the router must be prevented from laying down routes that run collinear to the soft macro or plan group boundary. To do this, rectangles that the router creates are placed along and inside the soft macro's or plan group's edges on metal layers whose preferred direction crosses the soft macro or plan group boundary. If the plan groups are committed, the inner-boundary shielding is then transferred to the soft macro.

The rectangles act as hierarchical signal shielding by allowing only routes that are perpendicular to the soft macro or plan group boundary to pass through the metal layers. By preventing collinear routing along the soft macro and plan group boundaries, signal crosstalk is minimized between routing in the soft macro and routing in the top-level channels.

To add block shielding for plan groups or soft macros,

1. Choose Floorplan > Create Module Block Shielding.

The Create Module Block Shielding dialog box appears.

2. Select the type of objects to which you want to add signal shielding.

You can add shielding for all plan groups and soft macros, or you can specify selected plan groups or soft macros. The default is "All".

3. Select a scope option:

- Inside – Select this option if you want to create metal route blockages (shielding) around the boundary inside the plan groups or soft macros.
- Outside – Select this option if you want to create metal route blockages (shielding) around the boundary outside the plan groups or soft macros.
- Both – Select this option if you want to create metal route blockages (shielding) around the boundary both inside and outside the plan groups or soft macros. This is the default.

4. Specify a shielding width.

Click in the Width box and enter a value. The shielding width is determined by multiplying the layer pitch with the value you specify. The default multiplier is 3.0. Alternatively, you can specify a value in microns.

5. Select the metal layers.

Specify the metal layer or layers on which to create the rectangles (shielding).

6. Specify the sides on which to create the shielding.

7. Use the “Tie to the net” option if you want to tie the shielding to a net such as a power or ground net.

8. Click OK or Apply.

Alternatively, you can use the `create_fp_block_shielding` command.

Removing Module Block Shielding

You can remove the signal shielding (route blockages) created by modular block shielding. Choose Floorplan > Delete Floorplan Objects > Delete Module Block Shielding.

Alternatively, you can use the `remove_fp_block_shielding` command.

You can remove shielding rectangles from all plan groups and soft macros or from specified plan groups or soft macros. You can also select the sides as well as the metal layers on which shielding should be removed.

Analyzing and Manipulating the Hierarchy

You can use the hierarchy browser to navigate through the design hierarchy and to examine the logic design hierarchy and display information about the hierarchical cells and logic blocks in your design. You can select the hierarchical cells, leaf cells, or other objects you want to examine in layout or schematic views.

If you are not familiar with a design, you can explore the logic hierarchy to understand its structure and gather information about the design objects. You can also use the hierarchical browser to

- Highlight cells or blocks (and their leaf cells) with different colors in both the hierarchy browser and the layout window
- Perform floorplan tasks such as creating plan groups and estimating black boxes
- Display design schematics of hierarchical cells
- View the hierarchical data interactively and switch between the layout view and the hierarchy browser view for selection and highlighting
- Perform probing between the logical hierarchy tree and the physical layout. This allows you to select cell instances, highlight and unhighlight cells, turn flylines on and off, and display the connectivity of selected modules and plan groups.

This section includes the following topics:

- [Opening the Hierarchy Browser](#)
 - [Exploring the Hierarchical Structure](#)
 - [Manipulating the Hierarchy](#)
-

Opening the Hierarchy Browser

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 hierarchy browser window shows 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.

[Figure 7-3 on page 7-9](#) shows an example of a hierarchy browser.

Figure 7-3 Hierarchy Browser

Name	Type	Ref Name	fp number of	fp nu
[T] clock_opt_route	Design	ORCA_TOP	12	1075
[H] I_PCI_TOP	Hierarchical	PCI_TOP	8	1376
[+][H] I_PCI_CORE	Hierarchical	PCI_CORE	0	834
[+][H] I_PCI_READ_...	Hierarchical	PCI_FIFO_0	4	269
[+][H] I_PCI_WRITE_...	Hierarchical	PCI_FIFO_1	4	272
[HM] PCI_FIFO_...	Hard macro	ram8x64.FRAM	0	0
[HM] PCI_FIFO_...	Hard macro	ram8x64.FRAM	0	0
[HM] PCI_FIFO_...	Hard macro	ram8x64.FRAM	0	0
[HM] PCI_FIFO_...	Hard macro	ram8x64.FRAM	0	0
[+][H] PCI_FIFO_...	Hierarchical	PCI_FIFO_1_DW_fifoc...	0	271
[H] I_PARSER	Hierarchical	PARSER	0	218

The hierarchy browser indicates object types by displaying icons next to the cell names as follows:

- Logical
 - T – Top-level design
 - BB – Black boxes
 - H – Hierarchical module
 - HM – Hard macro
 - IO – I/O cell
 - SC – Standard cell
 - ILM – Interface logic model or block abstraction model
- Physical
 - P – Plan group
 - SM – Soft macro

To color-code the icons, right-click the hierarchy browser view and choose Set Colors on Top Hierarchies.

Exploring the Hierarchical Structure

You can explore the complete hierarchical structure of your 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 of hierarchy. You can also right-click and select “Expand Tree Level” from the menu to display the Expand Tree dialog box.
- Select an instance to display leaf cell information in the object table. Shift-click or Control-click instance names to select combinations of instances. The information includes cell instance names, cell types, and cell reference names.
- Display port and pin names or net names by selecting a port, pin, or net (rather than a cell) in the text box above the object table list.

You can specify what types of information are displayed in the object table list. Right-click inside the table and then choose Columns > More. You can then select the details you want to display from the objects listed in the dialog box.

Note:

For more information about the hierarchy browser, see IC Compiler online Help.

Manipulating the Hierarchy

This section describes how to create a new level of hierarchy in the early design planning or prototyping phase and how to remove a level of hierarchy from the current design.

This section includes the following topics:

- [Merging the Hierarchy](#)
- [Flattening the Hierarchy](#)

Merging the Hierarchy

You can select any number of cells in the current design and merge (group) them together into a new cell, thereby creating a new level of logic hierarchy.

To create a new level of hierarchy,

1. Select Partition > Merge Hierarchy.

The Merge Hierarchy dialog box appears.

2. Set the options, depending on your requirements.

- New cell name – Enter the name of the new cell instance that replaces the merged cells in the design.

- New design name – Enter the reference name of the new cell design created by merging the existing cells.

The name you enter cannot already exist in the current design.

- Cells – Enter a list of cells (modules, plan groups, black boxes, hard macros, soft macros, or leaf cell instances) that you want to merge into a new level of logic hierarchy.

The cells you select must be from the same parent and reside at the same level of hierarchy.

- Update SDC – During the creation of the logical module, you might want to update your original Synopsys Design Constraints (SDC) file so that it matches the new logical hierarchy. Select this option to update the SDC file.

Click in the “Input SDC file” box and enter the name of the original SDC file to be updated to reflect the netlist changes. The file should contain valid SDC commands for the current design.

Click in the “Output SDC file” box and enter the name of the resulting SDC output file. This file contains the same information as the original SDC file but with updated top-level timing constraints and new hierarchical names for all the objects in the netlist.

- Load back updated SDC file – Select this option to reload the updated SDC file. The new SDC file is automatically reloaded into the CEL view.
- Create wrapper – Select this option to create a wrapper on any existing leaf cell or hierarchical logic modules. A wrapper is used to selectively expose internal connections and dangling pins onto a wrapper interface so that any future changes made to these internal objects do not impact the wrapper interface.

The new wrapper module retains the pin names from the logic modules. Note that if the new pin name on the wrapper is different from the one connected inside the original cell’s pins, a file describing the mapping of the pin names is automatically generated.

You can also use this option to create a wrapper module for a merged plan group before committing it to a soft macro.

Note:

Pin assignment and feedthrough generation occur only at the wrapper module. The original module under the wrapper is not affected by any feedthrough insertion changes.

If there are changes in the netlist of the original logic modules, but the module interface is still the same, you can swap the new netlist into the wrapper module, create a new block CEL view, and then link it to the top-level design. You do not have to run pin assignment or generate feedthroughs again for the new netlist.

- Port merging – Deselect this option if you do not want the ports connected by the same interface net to be merged under the wrapper.

3. Click OK or Apply.

The Milkyway hierarchical netlist is updated and a new hierarchical cell is created in the Hierarchical Preservation data.

This new logical module can be converted into plan groups, which you can then later commit (`commit_fp_plan_groups`) to soft macros (physical blocks).

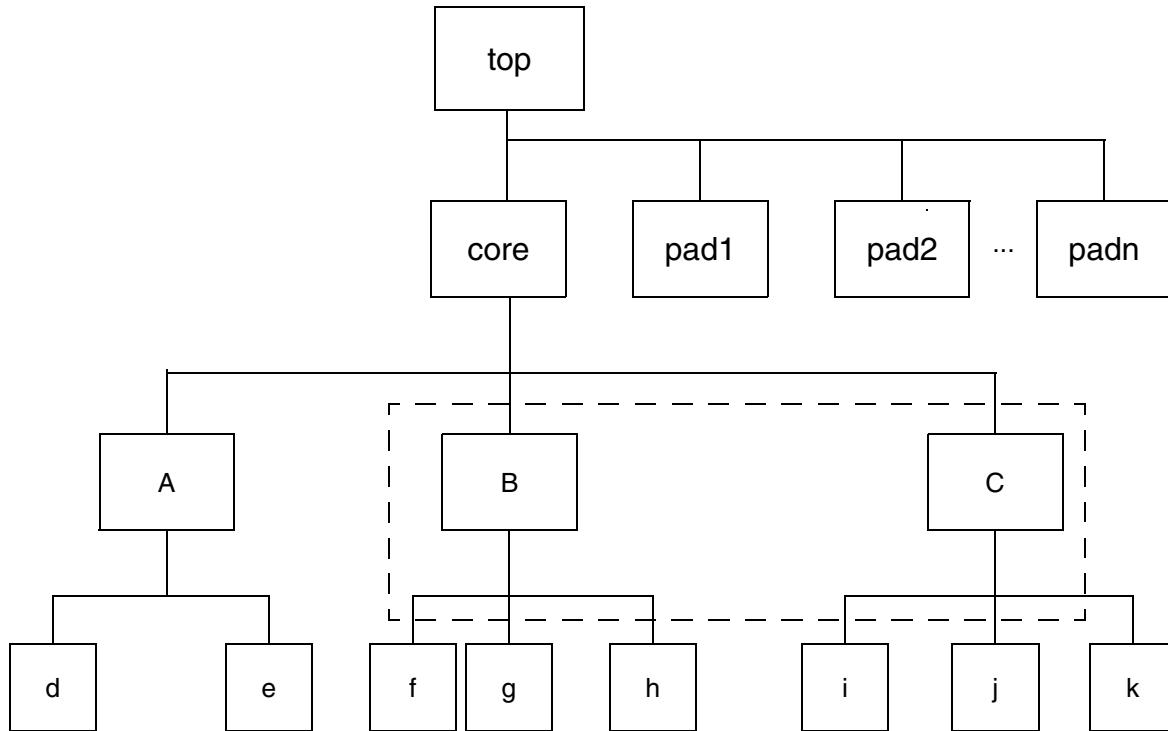
Alternatively, you can use the `merge_fp_hierarchy` command.

Logic Netlist Grouping (Merging) Examples

You can group (merge) modules at the same level of hierarchy and of the same parent in the logic hierarchy. This section has five examples showing how modules can and cannot be grouped.

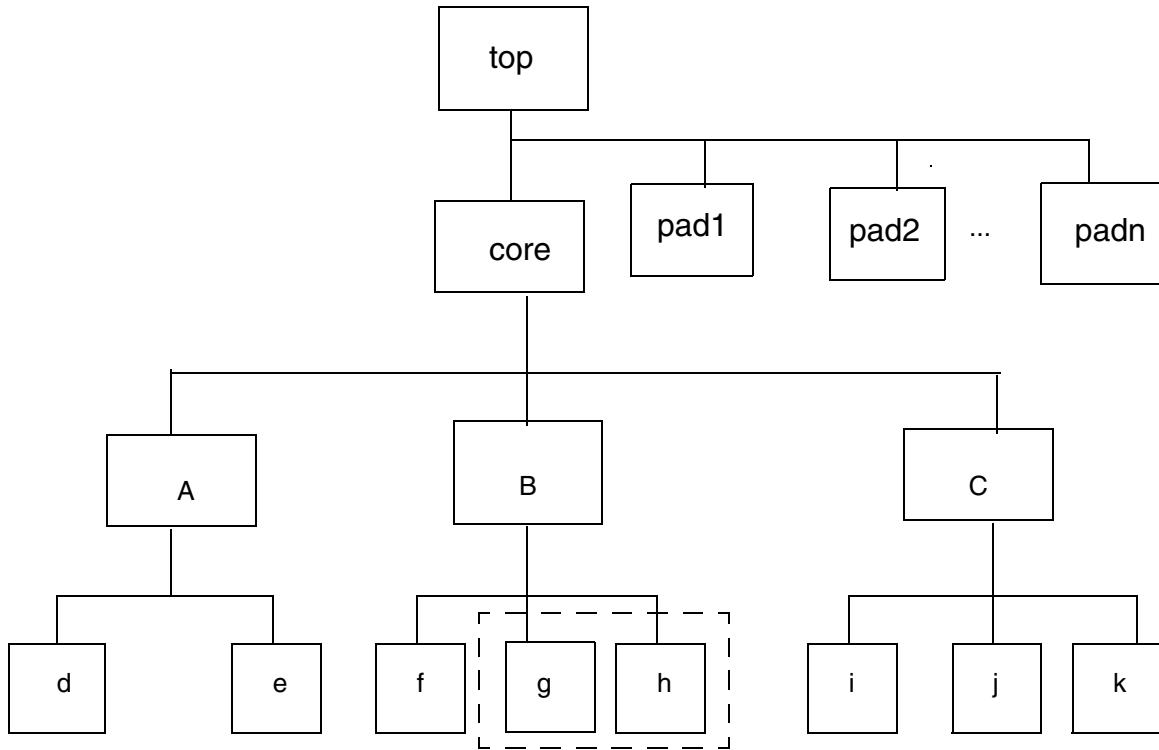
In example 1, shown in [Figure 7-4 on page 7-13](#), standard cells at the same level as A, B, and C are included in the merge.

Figure 7-4 Example 1: Logic Netlist Merging: Same Level of Hierarchy and Same Parent



In example 2, shown [Figure 7-5 on page 7-14](#), standard cells at the same level as f, g, and h and under B, are included in the merge. Standard cells and hard macros of B and f are placed at the top level.

Figure 7-5 Example 2: Logic Netlist Merging: Same Level of Hierarchy and Same Parent



The remaining three examples ([Figure 7-6 on page 7-15](#), [Figure 7-7 on page 7-16](#), and [Figure 7-8 on page 7-17](#)) show that you cannot merge modules that are in multiple levels of the hierarchy or that do not have the same parent.

Figure 7-6 Example 3: Multiple Levels of Hierarchy (Merging Not Possible)

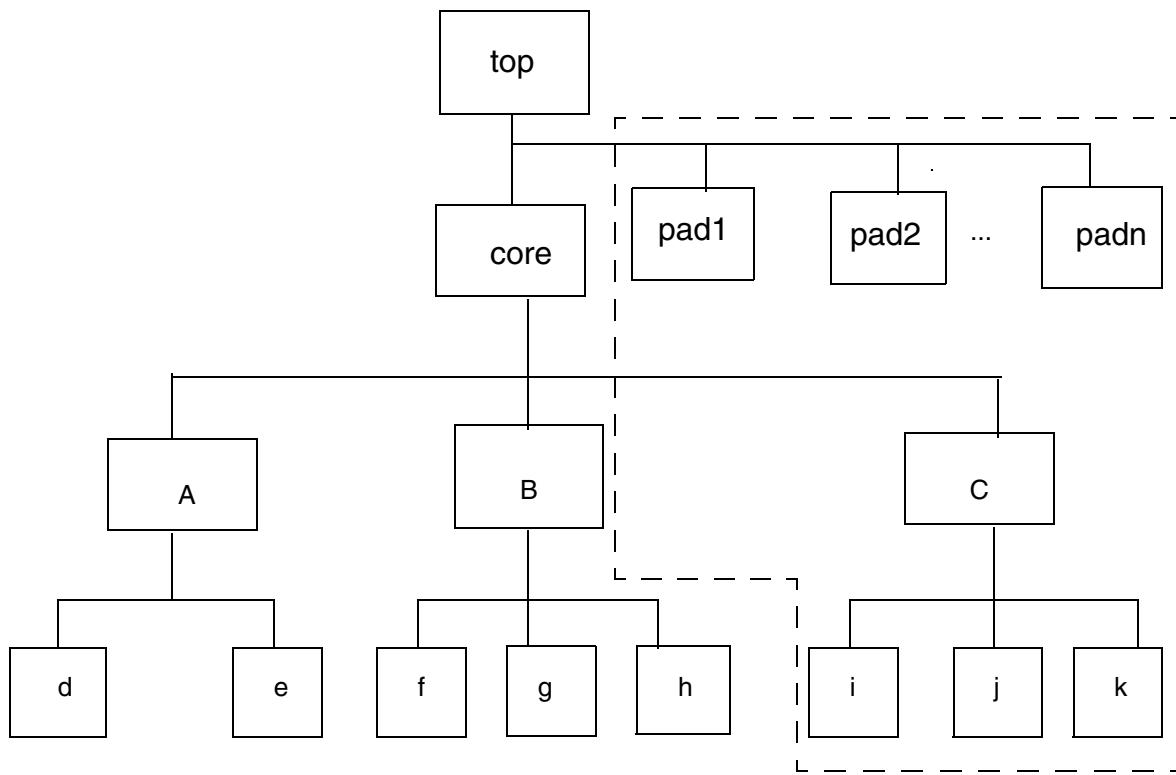


Figure 7-7 Example 4: Multiple Levels of Hierarchy, Not Same Parent (Merging Not Possible)

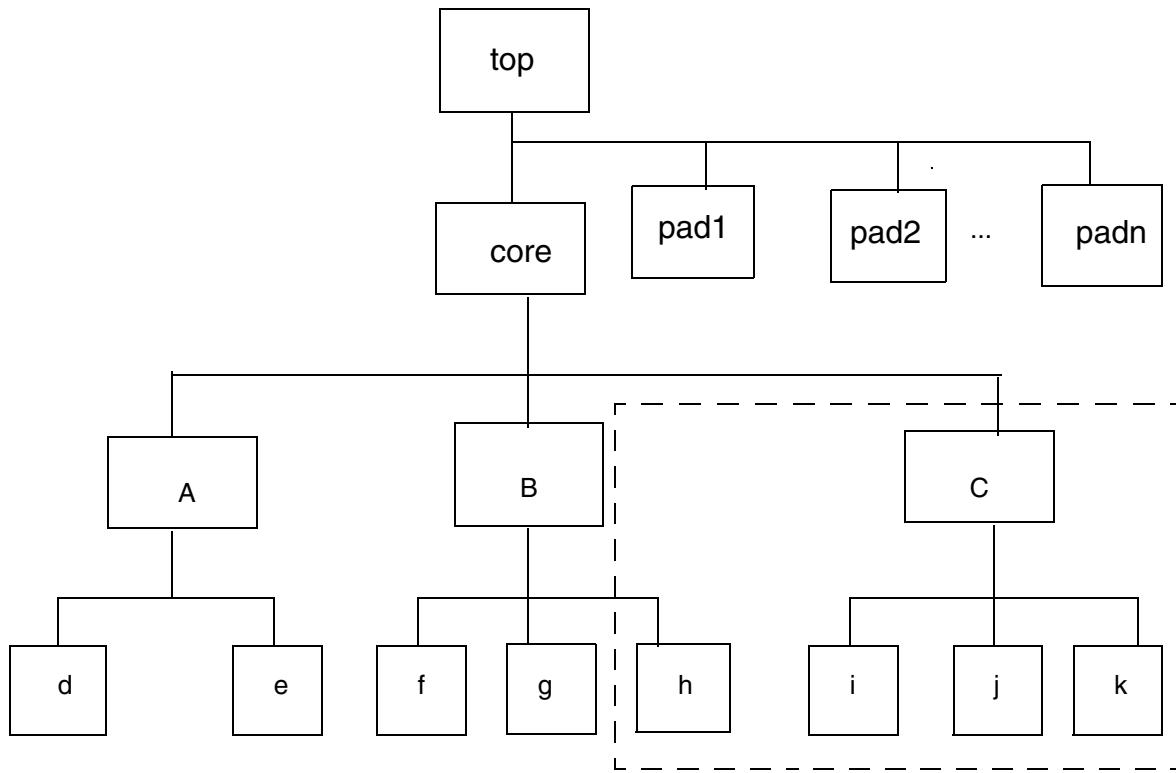
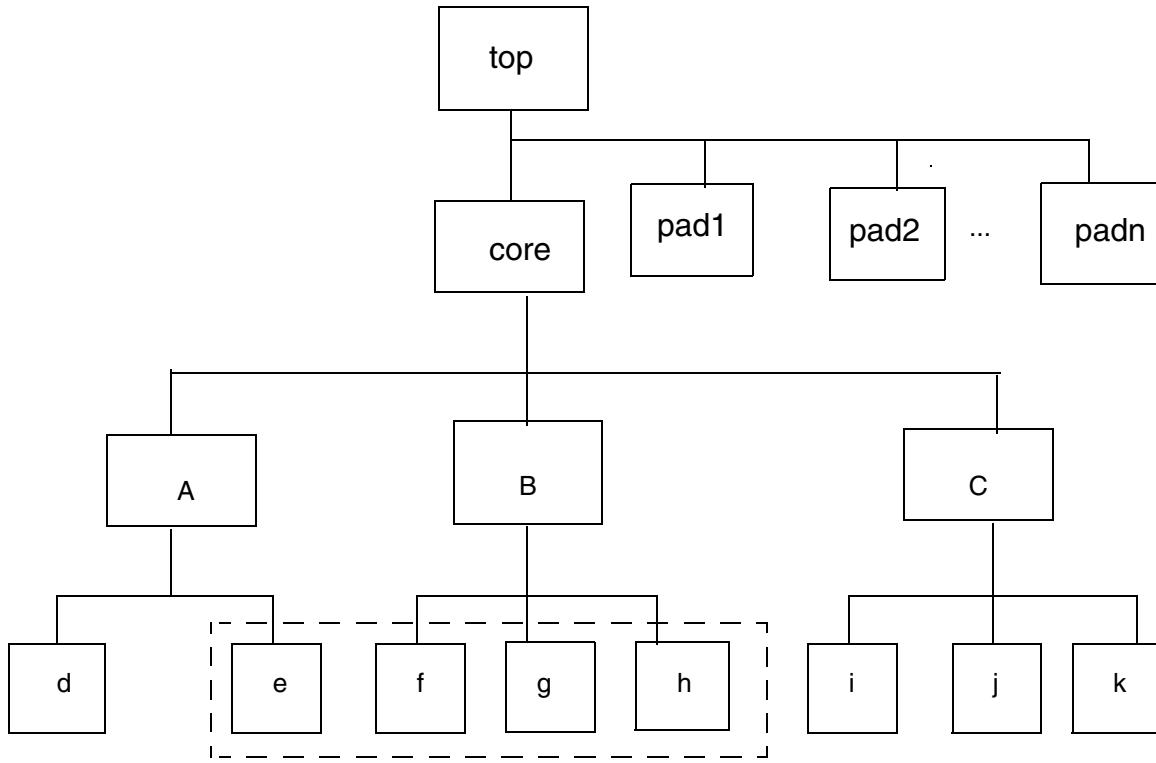


Figure 7-8 Example 5: Not the Same Parent (Merging Not Possible)



Flattening the Hierarchy

You can flatten (remove) a single level of hierarchy from your current design so that the logic modules at the next lower level of hierarchy are merged into the current level of hierarchy. This replaces a single hierarchical block with multiple blocks brought up from the lower level, which raises all of those blocks up by one level of hierarchy.

To flatten a level of hierarchy,

1. Select Partition > Flatten Hierarchy.

The Flatten Hierarchy dialog box appears.

2. Set the options, depending on your requirements.

- Cells – Enter a list of cells (modules, plan groups, black boxes, hard macros, soft macros, or leaf cell instances) in the current design that are to be flattened.
- Update SDC – Select this option to update the SDC file to reflect the changes to the netlist.

Click in the “Input SDC file” box and enter the name of the SDC file to be updated. The file should contain valid SDC commands for the current design.

Click in the “Output SDC file” box and enter the name of the resulting SDC output file into which the updated SDC timing constraints are written.

- Load back updated SDC file – Select this option to reload the updated SDC file. The updated SDC constraints are also automatically reloaded into the CEL view.
- Name Prefix – Select this option to specify the prefix to use when naming flattened cells.

Simple name – Select this option to indicate that simple, nonhierarchical names are to be used for cells that are flattened. The cells maintain their original names. This is the default.

No backslash – Select this option to indicate that a backslash should not be added before the hierarchy delimiter.

Specified prefix – Select this option to specify the prefix to be used when naming flattened cells. The naming style is:

<specified_prefix><original_cell_name>

3. Click OK or Apply.

Alternatively, you can use the `flatten_fp_hierarchy` command.

Automatically Placing and Shaping Objects in a Design Core

You can automatically place and shape plan group boundaries, black boxes, voltage areas, and other soft macros in a design core.

This section includes the following topics:

- [Controlling the Placement and Shaping of Objects](#)
- [Using Relative Placement Constraints](#)

Plan groups are automatically shaped, sized, and placed inside the core area based on the distribution of cells resulting from the initial virtual flat placement. Blocks (plan groups, voltage areas, and soft macros) marked as fixed (they have the `is_fixed` attribute set to `true`) remain fixed; the other blocks, whether or not they are inside the core, are subject to being moved or reshaped.

Note:

An initial virtual flat placement must have been previously run on the design.

To automatically place and shape objects in the design core,

1. Choose Placement > Place and Shape Plan Groups.

The Place and Shape Plan Groups dialog box appears.

2. Select the options as needed.

- Prefer rectilinear shapes – If you do not select this option, the `shape_fp_blocks` command creates only rectangular plan groups or soft macros boundaries unless they would overlap fixed objects, in which case the shape might need to be rectilinear if it needs to be cut out around fixed objects.

If you select this option, the virtual flat placer tries to put the plan group boundary around the set of preplaced cells associated with that plan group. However, in cases where you have one small block and one large block in your design, the small block is placed by one of the corners of the large block, creating a rectilinear L-shaped boundary. L-shaped plan groups can be created even if no fixed objects are encountered on the floorplan. The default is off.

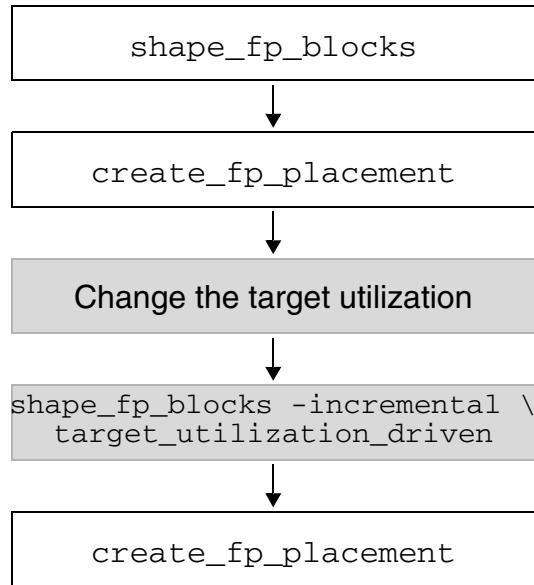
- Run incrementally – When you select this option, the floorplan is adjusted to meet the new plan group or voltage area target utilizations or to reduce the placement congestion in channels between blocks or inside the blocks themselves. The input is a legal, nonoverlapping floorplan and a congestion map if you select the congestion driven mode. The command incrementally resizes and reshapes the blocks (plan groups, voltage areas, and soft macros) in the design, changing the floorplan as little as possible, so that the estimates based on the congestion map are satisfied.

The run incrementally option runs in two modes: target utilization driven and congestion driven.

Target utilization driven – If you select this mode, the command adjusts the floorplan so that it meets the newly specified target plan group or voltage area utilizations. The command should achieve target utilizations within the `set_fp_shaping_strategy -utilization_slack` settings. The new floorplan should be similar to the original one. This mode is useful when you are satisfied with the overall locations of the plan groups, but realize that the sizes of the plan groups need to change.

[Figure 7-9 on page 7-20](#) shows the target utilization flow.

Figure 7-9 Target Utilization Shaping Flow

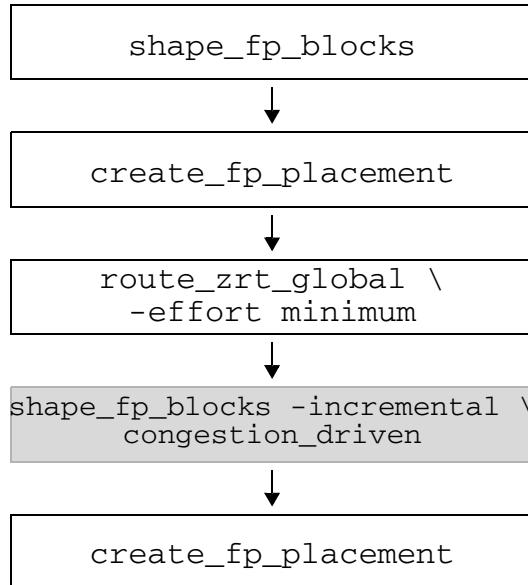


Congestion driven – If the channels between the blocks (plan groups, voltage areas, and soft macros) are too narrow or if the blocks themselves are too small, congestion in the design can often result. If you select this mode, the command first attempts to use the congestion map created by global routing. If a congestion map is not available, it tries to use a placement congestion map. If neither of the two congestion maps is available, an error message is generated. After a congestion map exists, you can use it to estimate the required sizes of the channels and blocks required to help reduce congestion.

The channels and blocks are resized and reshaped, changing the floorplan as little as possible, to reduce congestion in the channels between blocks and inside the blocks themselves so that the estimates based on the congestion map are satisfied. The objective is to reduce congestion by resizing channels as close to an optimal width as possible based on the existing congestion map.

[Figure 7-10](#) shows the congestion-driven shaping flow.

Figure 7-10 Congestion-Driven Shaping Flow



Occasionally, standard cells move into widened channels, which causes additional connections to pass through the channels, making the initial channel congestion estimates invalid. To reduce occurrences of this happening, you can use the `set_fp_shaping_strategy -add_channel_blockages` command to control the creation of blockages in the widened channels. You can use hard placement blockages or partial blockages with the congestion density reflecting the cell area that was initially inside the channel.

Note:

Because the placement legalizer does not obey partial blockages, using hard placement blockages in the channels is generally better, but only if there is enough space.

- Create routing channels – If you select this option, routing channels are created between plan groups, black boxes, and soft macros. The channel widths are based on pin counts with nonperpendicular connections to objects on the associated object edge. The default is off.

If you want channels between the top-level blocks in your design, but the design is not pad limited in size, set the initial core utilization slightly lower than the initial block utilizations. Begin by using a utilization of 0.65 for the core and a utilization of 0.7 for the blocks.

- Refine placement afterwards – If you select this option, the placement is refined after the placement and shaping of all unfixed plan groups, soft macros, and black boxes. (This is the equivalent of running the `create_fp_placement` command without any arguments.) The default is off.
- Top-down block placement mode – If you select this option, the command sets variables before it runs so that the placement and shaping of objects is constrained to top-down placement. Initial virtual flat placement is not assumed when you use this option. After the command runs, any variables that it set are restored to their previous settings. The default is off.
- Place sub macros – If you select this option, the command places hard macros in the top-level cell and in all unfixed soft macros. The default is off.
- Block small placement area – If you select this option, placement areas between blockages or fixed objects or both which are smaller than the threshold value are blocked out during shaping. This avoids skinny “fingers” that can occur when there are many blockages or when there are blockages with small channels between each other or the edge of the design core.

By default, the tool automatically calculates a reasonable value for the blockage threshold. To specify a threshold explicitly, select the “Narrower than (microns)” option and enter a number in microns.

- Use placement constraint file – If you select this option, you must enter the name of the file that contains the path and name of an ASCII file containing relative placement constraints to guide the placement and shaping of objects.

The format of the constraint file is one constraint per line. The first word in the line is always the name of the constraint. For example,

```
fplModuleDestRegion plan_group_name region
```

where *region* is one of N, S, E, W, NE, NW, SE, SW, or a coordinate specified as {*x*, *y*}

For a description of the relative placement constraints, see “[Using Relative Placement Constraints](#)” on page 7-25.

3. Click OK or Apply.

Alternatively, you can use the `shape_fp_blocks` command.

Controlling the Placement and Shaping of Objects

You can use the `set_fp_shaping_strategy` command to control how the `shape_fp_blocks` command and the Place and Shape Plan Groups dialog box place and shape objects in the design core. The command options are described in [Table 7-1](#). You can report the current settings with the `report_fp_shaping_strategy` command.

Note:

These settings are not saved in the Milkyway design library.

Table 7-1 set_fp_shaping_strategy Command Options

Command option	Usage
<code>-default</code>	Sets all the default parameter values.
<code>-avoid_power_grid true false</code>	Keeps the plan group edges away from the power grid during shaping when set to <code>true</code> . The default is <code>false</code> .
	This parameter applies to the <code>shape_fp_blocks</code> command without the <code>-incremental</code> option. During incremental shaping, it applies only to the edges that the <code>shape_fp_blocks</code> command decides to move. The command does not move an edge just to “legalize” its status with respect to the power grid, but it places all moved edges into legal locations.
<code>-distance_to_power_grid float</code>	Specifies the minimum required distance between the plan group boundary and the power grid. If the value is negative, IC Compiler uses one row height as the minimum distance. The default is <code>-1</code> .
<code>-keep_top_level_together true false</code>	Forces the <code>shape_fp_blocks</code> command to keep the top-level area of the design contiguous during nonincremental shaping when set to <code>true</code> . Keeping the top level contiguous enables the signals to reach all of the top-level area without going inside the plan groups, which is particularly useful if no feedthroughs are allowed. The default is <code>false</code> .

Table 7-1 set_fp_shaping_strategy Command Options (Continued)

Command option	Usage
<code>-min_channel_size float</code>	Specifies the minimum channel size between two plan groups if the <code>-channels</code> option is used with <code>shape_fp_blocks</code> command. The default is 0, which means plan groups can be abutted.
<code>-utilization_slack float</code>	Specifies how much a plan group is allowed to exceed the target utilization during incremental shaping. The default is 0.1, which means the target utilization can be exceeded by 10 percent.
<code>-add_channel_blockages none partial soft hard</code>	Controls the types of blockages that are added to the congested channels during incremental shaping. If adding hard or soft blockages causes overutilization, partial blockages are added instead. The default is soft.
<code>-adjust_macro_locations on off</code>	Specifies whether the <code>shape_fp_blocks -incremental congestion_driven</code> command moves the specified macros to ensure legal placement. The default is off.
<code>-preserve_abutment true false</code>	Specifies whether the <code>shape_fp_blocks -incremental target_utilization_driven</code> command preserves plan group abutment. The default is false.
<code>-max_shape_complexity integer</code>	Controls the maximum number of rectangles that can be used to make a plan group shape. For example, an L-shaped plan group consists of two rectangles and a U-shaped plan group consists of three rectangles. The default is 0, which means there is no limit.

Using Relative Placement Constraints

Relative placement constraints considered in a constraint file for guiding the placement and shaping of objects are described as follows:

- `fplNamedGroup groupName obj`

Note:

Groups must first be defined by using the `fplNamedGroup` constraint.

If `groupName` is not a name of an existing group, this constraint creates one and places `obj` in it. `Obj` can be a name of a block or another (already created) group. The block is a soft macro, black box, or plan group. Grouping, in the context of these relative constraints, can be viewed as a “pseudo hierarchy” because these are physical groups that do not have to be related to the logic hierarchy of the netlist.

- `fplRel relType obj1 obj2`

`relType` (a type of relation)

Supported types are

- `BT` (bottom top): This means `obj1` should be below `obj2`.
- `LR` (left right): This means `obj1` is to be placed to the left of `obj2`.

Care should be taken to ensure consistency of these relative constraints. There is no constraint “conflict resolution” up front when running the `shape_fp_blocks` command. For example, do not use both “`fplRel BT o1 o2`” and “`fplRel BT o2 o1`” or other, more complicated and conflicting constraint sets. Likewise, constraining some elements of group1 above elements of group2 while placing other elements of group1 below elements of group2 creates unpredictable results. Do not make relationships between an object and its group. Generally, relations should be on the same grouping level.

- `fplModuleDestRegion plan_group_name region`

where `region` is one of { N | S | E | W | NE | NW | SE | SW} or an x- and y-coordinate surrounded by braces, for example {10.700 1200.000}

The `fplModuleDestRegion` constraints can be applied to both top-level and lower-level modules.

Here is an example constraint file:

```
fplNamedGroup red plan_group1
fplNamedGroup red black_box_foo

fplNamedGroup blue plan_group2
fplNamedGroup blue plan_group3
fplNamedGroup green plan_group4
fplNamedGroup green black_box_bar
```

```
fplNamedGroup root red
fplNamedGroup root blue
fplNamedGroup root green

fplRel BT plan_group1 black_box_foo
fplRel LR plan_group3 plan_group2

fplRel BT green blue
fplRel LR red green
```

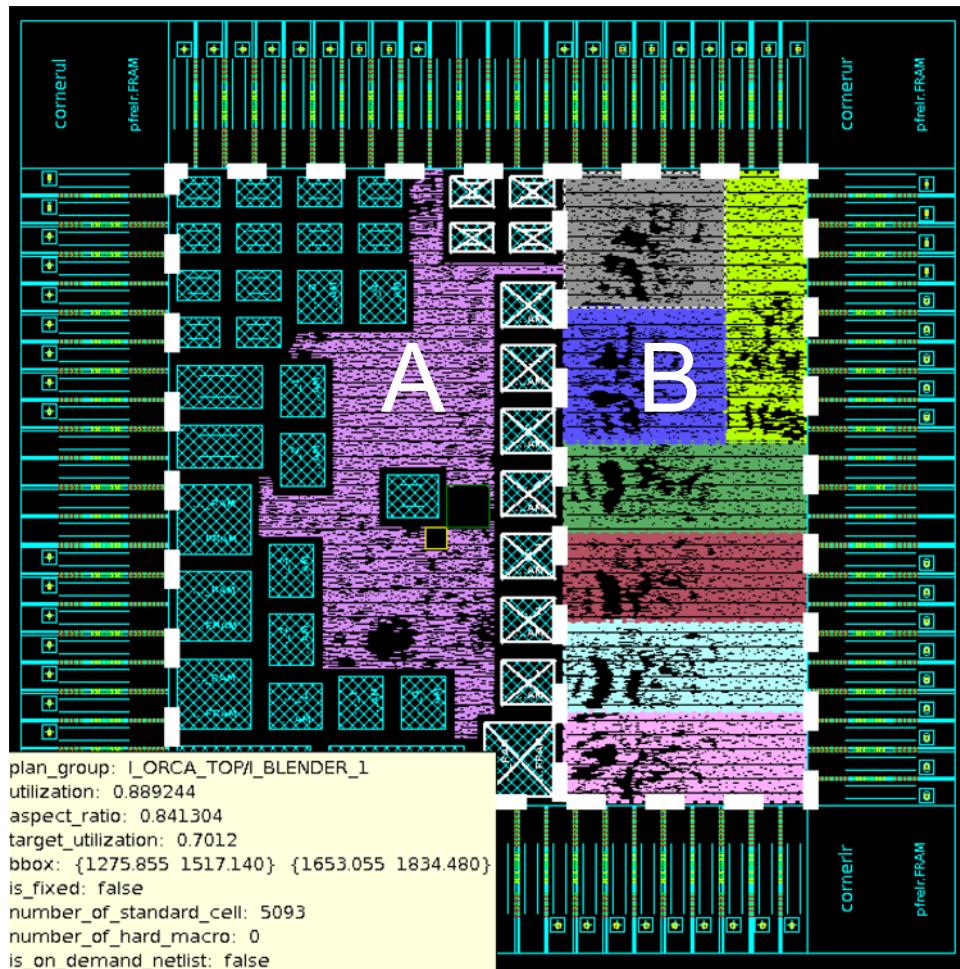
Plan Group Utilization

Utilization is the ratio of design area covered by cells to the available area for cell placement. You can use the `get_utilization` command to report a utilization value for the entire design or for a set of plan groups and soft macros. Command options specify the criteria used to calculate utilization; you can exclude or include blockages and keepouts, depending on your requirements.

```
get_utilization
[objects]
[-row_based]
[-consider_blockages none | hard | soft_and_hard]
[-consider_macro_keepouts]
[-treat_macros_like_blockages none | fixed | all]
[-flat]
```

The following examples use the design example in [Figure 7-11 on page 7-27](#). In the figure, region A is occupied by standard cells, macro cells, placement blockages, and keepout margins. Region B contains plan groups that contain only standard cells.

Figure 7-11 Utilization Calculation Design Example



To display the utilization value for individual plan groups and soft macros in the design, specify their names on the command line. The `get_utilization` command and the ToolTip window report the same utilization value for the plan group.

```
icc_shell> get_utilization [get_plan_groups *]
plan group <I_ORCA_TOP/I_BLENDER_1> utilization: 0.889244
plan group <I_ORCA_TOP/I_BLENDER_2> utilization: 0.887113
plan group <I_ORCA_TOP/I_BLENDER_3> utilization: 0.893797
plan group <I_ORCA_TOP/I_BLENDER_4> utilization: 0.893594
plan group <I_ORCA_TOP/I_BLENDER_5> utilization: 0.893594
plan group <I_ORCA_TOP/I_BLENDER_6> utilization: 0.893594
plan group <I_ORCA_TOP/I_BLENDER_7> utilization: 0.888865
average utilization of plan groups: 0.891400
overall utilization of plan groups: 0.891392
0.891392
```

To report the utilization for the top-level design, use the `get_utilization` command with no arguments. The `get_utilization` command with no options excludes plan groups from the utilization calculation; the command calculates the utilization for region A and excludes region B in [Figure 7-11 on page 7-27](#):

```
icc_shell> get_utilization
      Top level utilization: 0.588681
0.588681
```

Use the `-flat` option to ignore plan group boundaries and include the plan group area in the utilization calculation; the command includes both regions A and B in [Figure 7-11 on page 7-27](#) in the utilization calculation.

```
icc_shell> get_utilization -flat
      Top level utilization: 0.701213
0.701213
```

Use the `-row_based` option to exclude gaps between rows in the utilization calculation. If gaps exist between standard cell rows in your design, the `get_utilization -row_based` command reports a higher utilization than the `get_utilization` command.

The remaining options to the `get_utilization` command specify which objects are part of the occupied area and which are part of the available area. For example, the `-consider_blockages hard` option excludes hard blockages from being considered as part of the available area and causes a higher utilization value to be reported. You can also control the consideration of hard and soft blockages, macros, and macro blockages. For details, see the man page for the `get_utilization` command.

Physical-Only Cells in Plan Groups

You can place physical-only cells, such as tap or filler cells, inside specific plan groups during top-level virtual flat placement in design planning. After committing the plan groups, the physical-only cells remain with the plan groups, and are not moved to the top level of the design hierarchy. Note that physical-only cells that straddle the plan group boundary are inserted into the hierarchy of the plan group. [Table 7-2](#) summarizes the commands and options that perform plan-group-aware physical-only cell placement.

Table 7-2 Plan-Group-Aware Physical-Only Cell Placement Commands

Command and option	Description
<code>add_end_cap -at_plan_group_boundary</code>	Places end cap cells at both sides of the vertical boundary of the plan group. The tool inserts end cap cells within the plan group boundary at the hierarchy level of the plan group.
<code>add_tap_cell_array -plan_group {plan_groups}</code>	Inserts tap cells only within the specified plan groups. Tap cells that straddle the plan group boundary or are contained within the plan group are inserted into the hierarchy of the plan group.
<code>insert_stdcell_filler -plan_group {plan_groups}</code>	Inserts filler cells only within the specified plan groups. Filler cells that straddle the plan group boundary or are contained within the plan group are inserted into the hierarchy of the plan group.

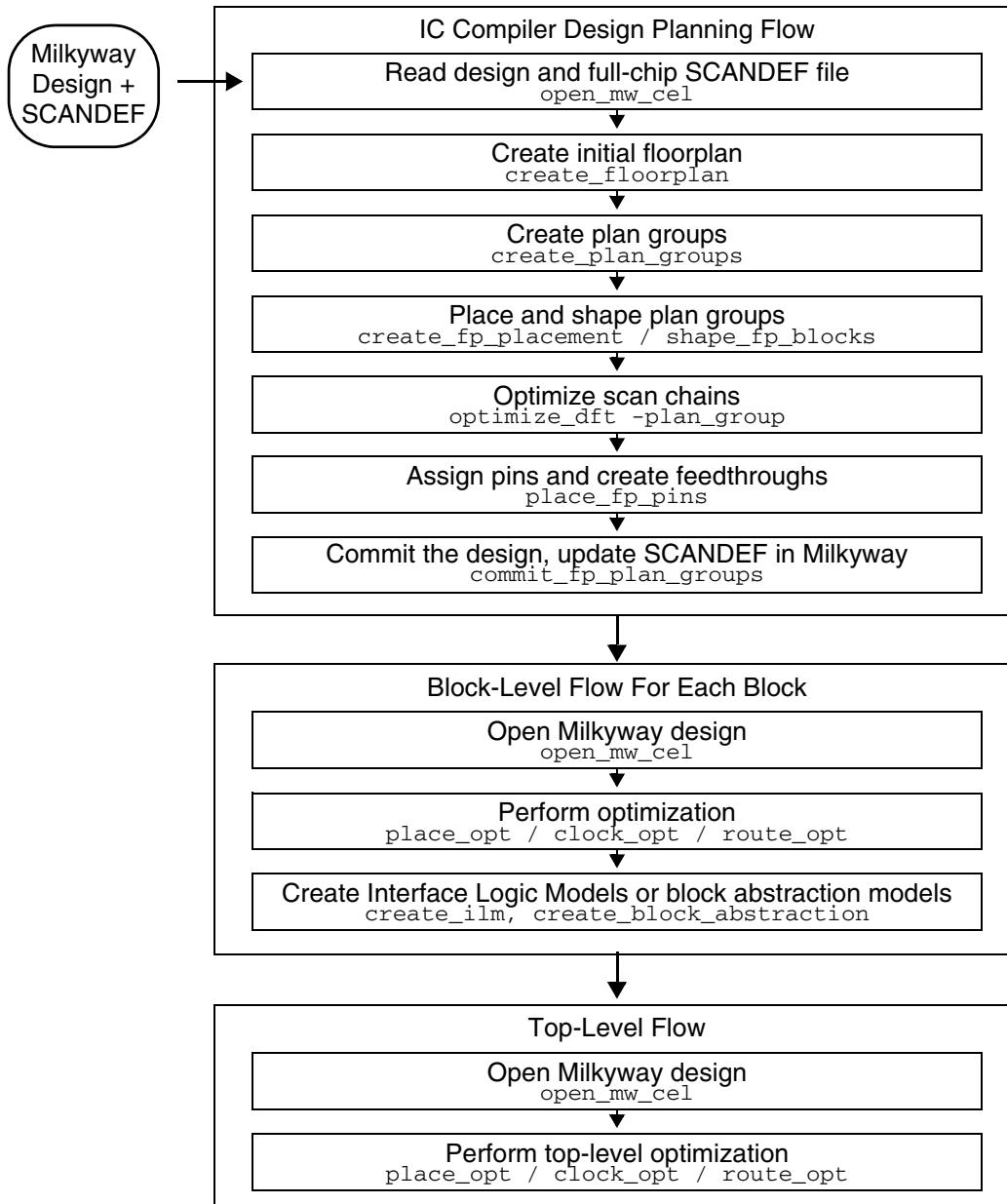
DFT-Aware Design Planning Flow

In the IC Compiler design planning flow, you can use the `optimize_dft -plan_group` command to optimize scan chains based on the physical hierarchy after you place and shape the plan groups but before you perform pin cutting. This is useful when logical hierarchy-based scan chains are suboptimal for the best physical design. The plan-group-aware scan chain takes advantage of the flexibility of scan chain connections to repartition and reorder the scan cells in a way that reduces the number of scan ports and feedthroughs.

Note:

Scan chains in designs that contain multiply instantiated modules cannot be optimized by using the `optimize_dft -plan_group` command.

[Figure 7-12 on page 7-30](#) shows where the `optimize_dft -plan_group` command is used in the DFT-aware design planning flow.

Figure 7-12 DFT-Aware Design Planning Flow

Generating and Using SCANDEF

DFT Compiler uses the `write_scan_def -expand list_of_instances` command to write the chip-level SCANDEF file with all the scan cells of the subblocks. You can load the full-chip SCANDEF at the beginning of the design planning process in IC Compiler as follows:

- If the design input is .ddc, read the .ddc file with the SCANDEF embedded in it.
- If the design input is Verilog, read the ASCII SCANDEF file, using the `read_def` command.

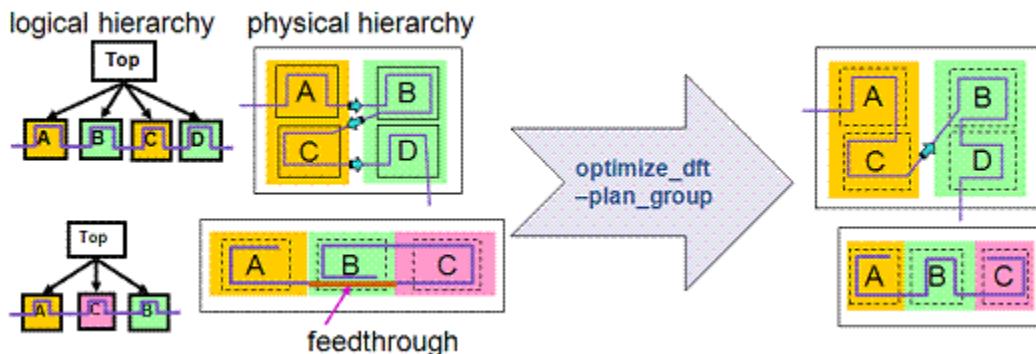
Reading the full-chip SCANDEF information enables scan chain elements to have `size_only` attributes applied. This prevents the scan chain elements from being reoptimized during in-place optimization. As a result, the consistency between the scan chain netlist and SCANDEF is maintained through the design planning stage.

Performing Plan Group-Aware Scan Chain Optimization

Using the `optimize_dft -plan_group` command performs scan chain repartitioning and reordering. Repartitioning reallocates the scan cells to chains so that scan cells in the same physical hierarchy that are close together form new scan chains. Reordering ensures that the scan chains in the same physical hierarchy are adjacent to each other to minimize net lengths that cross physical hierarchies. It provides the following benefits:

- The number of unnecessary top-level scan-chain wires crossing between the physical hierarchies is minimized during design planning.
- Congestion is minimized and routability is improved.
- After scan chain optimization, the scan cells of the same plan group are clustered together. The entry and exit points of the plan group and the top-level scan wires and ports are minimized, as shown in [Figure 7-13 on page 7-32](#).
- An optimal order of the physical hierarchies is determined for each top-level scan chain to minimize excessive feedthroughs created for scan wires, as shown in [Figure 7-13 on page 7-32](#).

Figure 7-13 Minimizing Top-Level Scan Wires and Feedthroughs



Using Plan Group-Aware Repartitioning

You can use the following command to control the plan group-aware repartitioning of the scan chains during design planning:

```
set_optimize_dft_options -repartitioning_method \
    none|single_directional|multi_directional|adaptive
```

For more information about this command and options, see the man page.

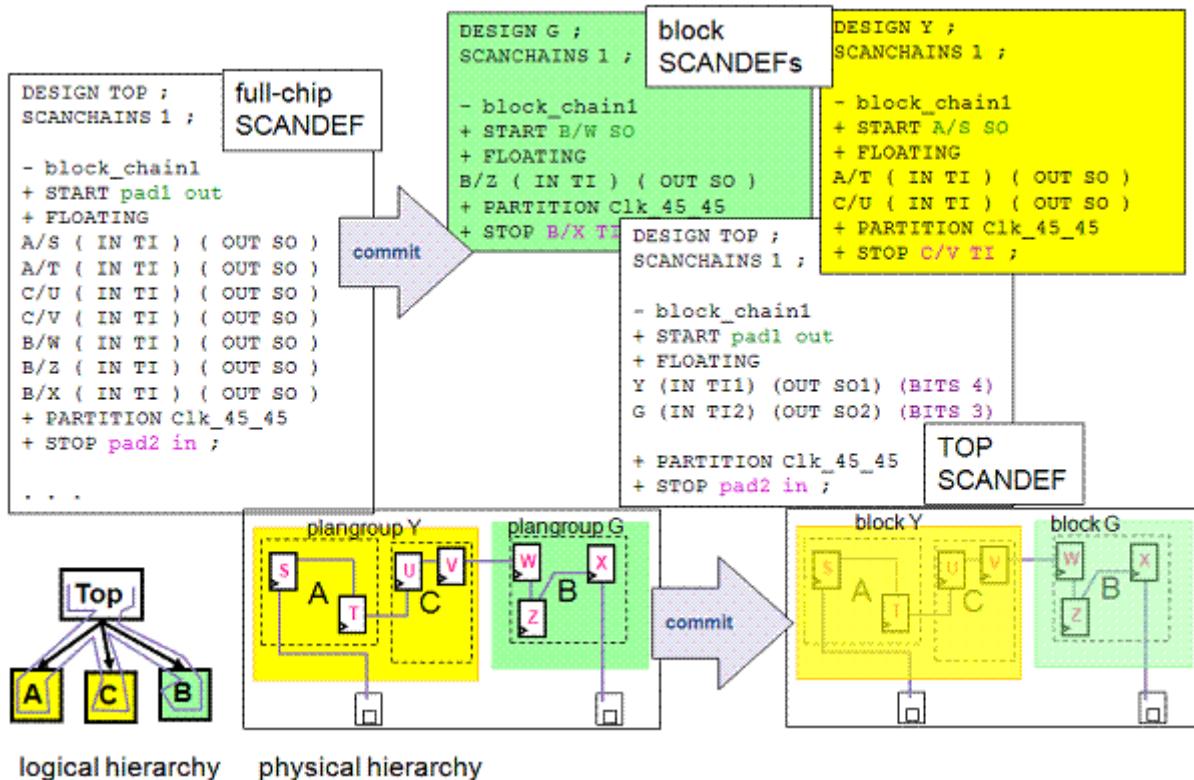
You can turn off repartitioning and perform only plan group-aware scan chain reordering by using the following setting before running the `optimize_dft -plan_group` command:

```
icc_shell> set_optimize_dft_options -repartitioning_method none
```

Committing Physical Hierarchy Changes After Scan Chain Optimization

When committing the physical hierarchy changes by using the `commit_fp_plan_groups` command, the SCANDEF information is updated based on the plan groups. The original logical hierarchy-based SCANDEF information is automatically updated to correspond to the physical hierarchy in the design planning phase, and block-level SCANDEF information is pushed down into the hierarchy, and the updated SCANDEF is stored in the Milkyway CEL views for the blocks and top level, based on the physical hierarchy, as shown in [Figure 7-14 on page 7-33](#).

Figure 7-14 Updating SCANDEF Data During Commit Hierarchy



When using the `uncommit_fp_soft_macros` command, note that the SCANDEF might be different from the original file. Nevertheless, this SCANDEF is functionally equivalent to the original.

Checking the SCANDEF Data Against the Netlist

IC Compiler maintains the SCANDEF data that describes the 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 in both the top level and the blocks. By default, a summary of the consistency of all the chains is displayed.

Consistency checking is performed 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.

8

Performing Power Planning

Power planning, which includes power network synthesis and power network analysis, is required to create a design with good power integrity. A design with a robust power and ground (PG) grid mitigates the impact of IR drop and electromigration by providing an adequate number of power and ground pads and rails. The power plan can be used to assess the routing resources consumed by the power nets and to determine the impact on routability due to the power plan. You can experiment with different power plans or fine-tune the existing power plan by modifying the replay script and regenerating the power plan.

Power network analysis extracts the power network and performs an IR drop analysis. You can use the analysis results to modify the power plan and reduce the IR drop to meet your specification.

This chapter includes the following sections:

- [Performing Power Network Synthesis on Single-Voltage Designs](#)
- [Performing Power Network Synthesis on Multivoltage Designs](#)
- [Performing Template-Based Power Planning](#)
- [Creating Power-Switch Arrays and Rings for Multithreshold-CMOS Designs](#)
- [Analyzing the Power Network](#)
- [Viewing the Analysis Results](#)

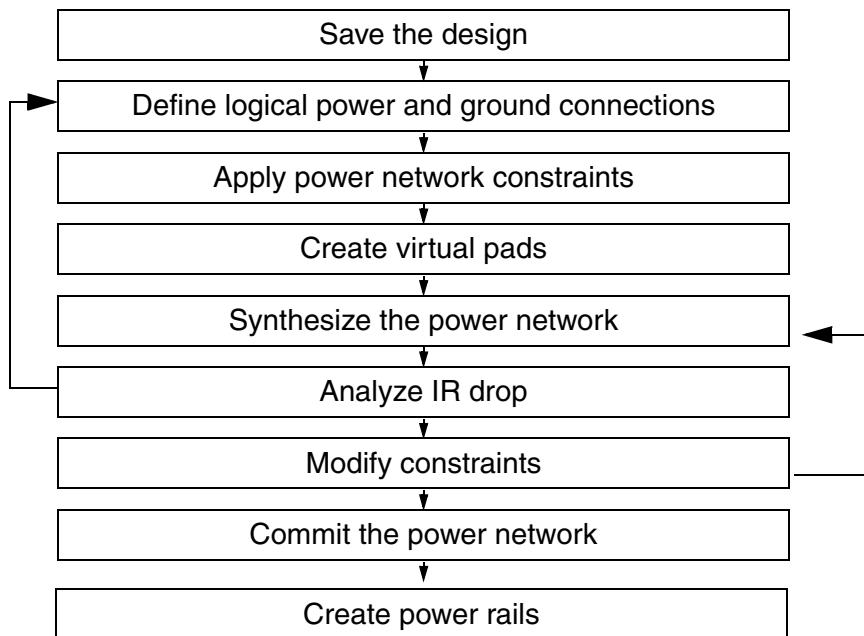
Performing Power Network Synthesis on Single-Voltage Designs

IC Compiler supports different flows for single-voltage designs and multivoltage designs. For information about performing power network synthesis on multivoltage designs, see “[Performing Power Network Synthesis on Multivoltage Designs](#)” on page 8-9. The following section outlines the steps to perform power network synthesis for a single-voltage design.

- [Saving the Design](#)
- [Defining Logical Power and Ground Connections](#)
- [Applying Power Rail Constraints](#)
- [Creating Virtual Pads](#)
- [Synthesizing the Power Network](#)
- [Committing the Power Plan](#)

The following illustration shows a typical power network synthesis flow for single-voltage designs.

Figure 8-1 Power Network Synthesis for Single-Voltage Designs



Saving the Design

Prior to running power network synthesis on your design, you should save the design by using the `save_mw_cel` command. If you are not satisfied with the results of power network synthesis, you can revert to the saved version and resynthesize using different constraints. The following command saves the design before power network synthesis.

```
icc_shell> save_mw_cel -as floorplan_prepns
```

Defining Logical Power and Ground Connections

Use the `derive_pg_connection` command to create logical connections between power and ground pins on standard cells and macros and the power and ground nets in the design. For more information about the `derive_pg_connection` command, see “Creating Logical Power and Ground Connections” in the *IC Compiler Implementation User Guide*.

Applying Power Rail Constraints

You must specify constraints before synthesizing the power network. The constraints specify the width, direction, spacing, offset, and other parameters IC Compiler uses when creating the rings and straps for the power network. Use the `set_fp_rail_constraints` command or choose one of the GUI forms by choosing Preroute > Power Network Constraints in the GUI and selecting one of the GUI forms to set the constraints. The `set_fp_rail_constraints` command syntax is as follows.

```

set_fp_rail_constraints
  [-add_layer | -remove_layer | -remove_all_layers |
   -set_ring | -skip_ring | -set_global]
  [-layer layer]
  [-direction vertical | horizontal]
  [-max_strap number]
  [-min_strap number]
  [-max_pitch distance]
  [-min_pitch distance]
  [-max_width distance]
  [-min_width distance]
  [-spacing distance | minimum | interleaving]
  [-offset distance]
  [-nets nets]
  [-horizontal_ring_layer layer]
  [-vertical_ring_layer layer]
  [-ring_width distance]
  [-ring_max_width distance]
  [-ring_min_width distance]
  [-ring_spacing distance]
  [-ring_offset distance]
  [-extend_strap core_ring | boundary | pad_ring]
  [-keep_floating_segments]
  [-no_stack_via]
  [-no_same_width_sizing]
  [-optimize_tracks]
  [-keep_ring_outside_core]
  [-no_routing_over_hard_macros]
  [-no_routing_over_plan_groups]
  [-no_routing_over_soft_macros]
  [-ignore_blockages]

```

The following example uses METAL2 as the vertical power rail and METAL3 as the horizontal power rail. The first command removes the constraints on all layers and then the next two commands constrain the power rails to limit the number of straps between 10 and 20 and sets a minimum width of 0.4 microns.

```

icc_shell> set_fp_rail_constraints -remove_all_layers
icc_shell> set_fp_rail_constraints -add_layer -layer METAL2 \
           -direction vertical -max_strap 20 -min_strap 10 \
           -min_width 0.4 -spacing minimum
icc_shell> set_fp_rail_constraints -add_layer -layer METAL3 \
           -direction horizontal -max_strap 20 -min_strap 10 \
           -min_width 0.4 -spacing minimum

```

Applying Power Ring Constraints

The `set_fp_block_ring_constraints` command defines the constraints for the power and ground rings that are created around plan groups and macros by using the `synthesize_fp_rail` command. You can also choose Preroute > Multi-Voltage Power Network Constraints > Block Rings Constraints in the GUI to set the constraints. The `set_fp_block_ring_constraints` command syntax is as follows.

```
set_fp_block_ring_constraints
  -add | -remove | -remove_all |
  -save_file file_name |
  -load_file file_name
  -block blocks
  -nets nets
  [-block_type master | instance | plan_group | voltage_area]
  [-all_blocks]
  [-horizontal_layer layer]
  [-vertical_layer layer]
  [-horizontal_width distance]
  [-vertical_width distance]
  [-horizontal_offset distance]
  [-vertical_offset distance]
  [-spacing distance]
```

The following example adds a ring constraint for the current design. The horizontal layer is constrained to the METAL5 layer with a width of 3 microns and an offset of 0.600 microns. The vertical layer is constrained to the METAL6 layer with a width of 3 microns and an offset of 0.600 microns. The ring constraint is applied around all instances with the cell names ALU, CPU, or RAM using the VDD and VSS nets.

```
icc_shell> set_fp_block_ring_constraints -add \
  -horizontal_layer METAL5 -vertical_layer METAL6 -horizontal_width 3 \
  -vertical_width 3 -horizontal_offset 0.600 -vertical_offset 0.600 \
  -block_type master -nets {VDD VSS} -block { ALU CPU RAM }
```

Creating Virtual Pads

You can create virtual pads to serve as temporary power sources for your design. Virtual pads provide additional sources of current for the power network and do not require modification of the floorplan. After experimenting with power network analysis using different placements and numbers of virtual pads, you can modify your floorplan and insert additional power pads based on your analysis.

To insert virtual pads into your design, use the `create_fp_virtual_pad` command or choose Preroute > Create Virtual Power Pad in the GUI. [Table 8-1](#) describes the `create_fp_virtual_pad` command options. For more information about these options, see the man page.

Table 8-1 create_fp_virtual_pad Command Options

Command option	Description
<code>-nets net_name</code> ("Power or Ground net" box in the GUI)	Specifies the net name of the net to attach the virtual pad.
<code>-layer layername</code> ("Specified" box in Layer area in the GUI)	Specifies the layer used to create the virtual pad.
<code>-point {x y}</code> ("Coordinates" text box in the GUI)	Specifies the x- and y-locations for the virtual pad.
<code>-load_file filename</code> ("Load" text box in the GUI)	Specifies the file name containing the list of virtual pads.
<code>-save_file filename</code> ("Save" text box in the GUI)	Specifies the file name used to write the location and net connections for the virtual pads.

The following example creates two virtual pads for the VDD net at coordinates 100,100 and 2000,2000.

```
icc_shell> create_fp_virtual_pad -nets VDD -point {100.000 100.000}
icc_shell> create_fp_virtual_pad -nets VDD -point {2000.000 2000.000}
```

You can save the current virtual pads to a file by using the `-save` option. The following example saves the current virtual pads to the *pads.rpt* file and displays the content of the file. To re-create the virtual pads, use the `create_fp_virtual_pad` command with the `-load_file` option.

```
icc_shell> create_fp_virtual_pad -save_file pads.rpt
Saving the virtual pad file pads.rpt successfully

icc_shell> sh cat pads.rpt
VDD
100.000 100.000 auto
2000.000 2000.000 auto
```

To remove the virtual pads created by using the `create_fp_virtual_pad` command, use the `remove_fp_virtual_pad` command. The `remove_fp_virtual_pad -all` command removes all virtual pads; you can also use the `remove_fp_virtual_pad` command with the `-net` and `-point` options to remove individual virtual pads. The following command removes all virtual pads from the design.

Synthesizing the Power Network

You can perform power network synthesis on your design by using the `synthesize_fp_rail` command. This command synthesizes the power network based on the constraints you define. To create the power network, use the `synthesize_fp_rail` command or choose Preroute > Synthesize Power Network in the GUI. The `synthesize_fp_rail` command syntax is as follows.

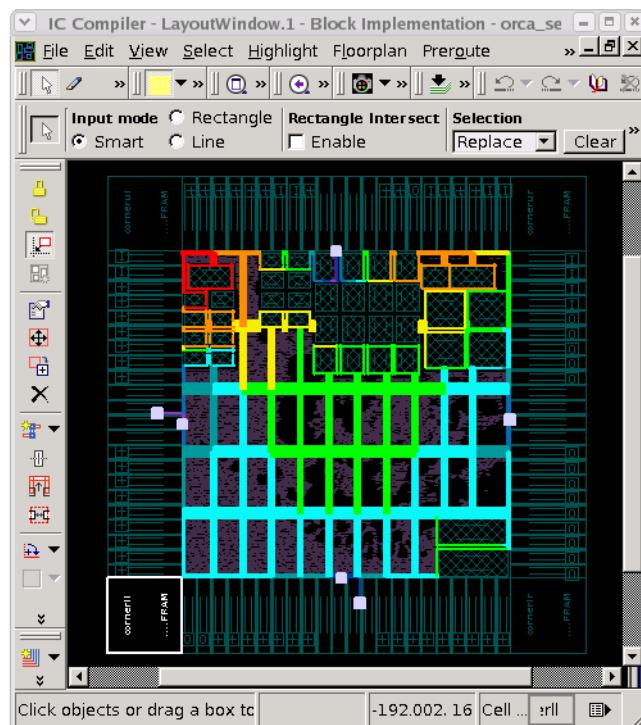
```
synthesize_fp_rail
  -nets nets
  -synthesize_voltage_areas | -synthesize_power_switch_array
  [-analyze_power]
  [-create_virtual_rails layer]
  [-honor_conn_view_layers conn_layer]
  [-ignore_blockages]
  [-lowest_voltage_drop]
  [-output_directory directory_name]
  [-pad_masters pad_reference_cells]
  [-power_budget power]
  [-read_default_power_file power_file_name]
  [-read_pad_instance_file pad_instance_file_name]
  [-read_pad_master_file pad_reference_cell_file_name]
  [-read_power_compiler_file power_compiler_report_file]
  [-read_prime_power_file prime_power_report_file]
  [-synthesize_power_pads]
  [-synthesize_power_plan]
  [-target_voltage_drop target_voltage]
  [-top_level_only]
  [-use_pins_as_pads]
  [-use_strap_ends_as_pads [-strap_ends_direction directions]]
  [-use_strap_ends_as_pads]
  [-voltage_areas voltage_areas]
  [-voltage_supply voltage]
```

The following example synthesizes the power plan based on the constraints set previously by using the `set_fp_rail_constraints` command. The command synthesizes a power plan using the VDD power net, the VSS ground net, and a supply voltage of 1.32V. The command writes the voltage drop and electromigration results to the `powerplan.dir` directory.

```
icc_shell> synthesize_fp_rail -power_budget 800 -voltage_supply 1.32 \
  -output_directory powerplan.dir -nets {VDD VSS} \
  -synthesize_power_plan
```

The `synthesize_fp_rail` command generates the power plan based on the constraints you provide. Rail analysis results for the design are saved to the `./pna_output` directory; you can use these results to examine the voltage drop, resistance, and electromigration and perform other analyses on the power network. See “[Analyzing the Power Network](#)” on [page 8-72](#) for more information. If the target IR drop is greater than your specification, change the power network constraints by using the `set_fp_rail_constraints` and `set_fp_block_ring_constraints` commands. [Figure 8-2](#) shows the power plan created by running the `synthesize_fp_rail` command.

Figure 8-2 Power Network Synthesis Results Display in IC Compiler



Committing the Power Plan

After creating a satisfactory power plan that meets your specifications for IR drop, you can commit the power plan to convert the virtual power straps and rings to actual power wires, ground wires, and vias. Use the `commit_fp_rail` command or click the Commit button on the Preroute > Synthesize Power Network form in the GUI.

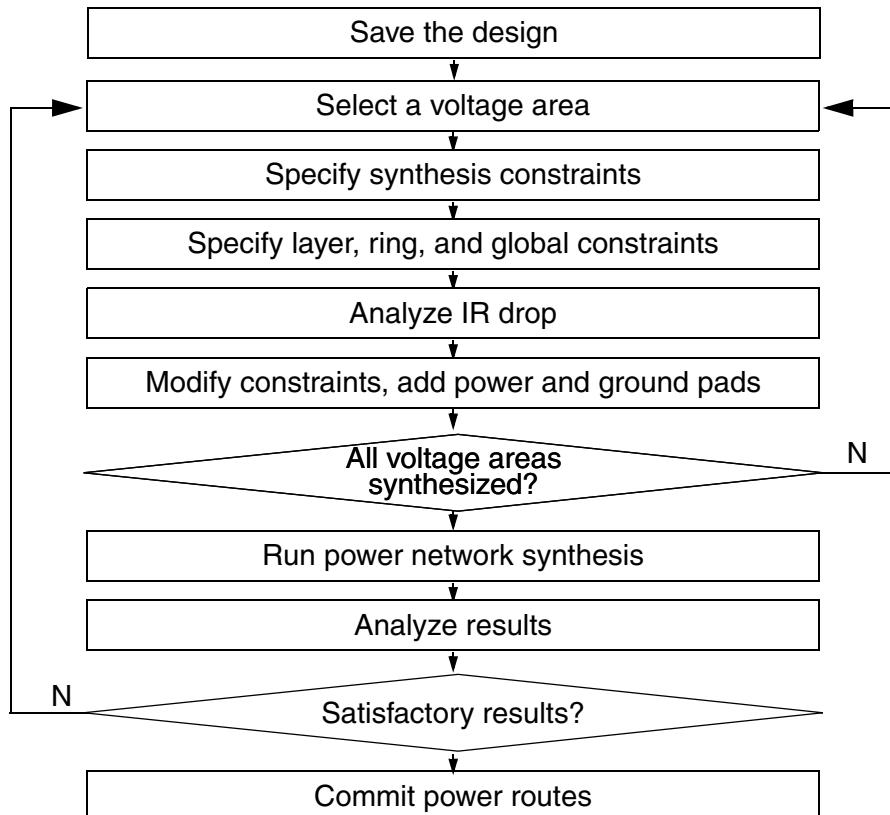
Performing Power Network Synthesis on Multivoltage Designs

The following section outlines the steps necessary to implement a power network in a multivoltage design.

- [Saving the Design](#)
- [Selecting a Voltage Area and Specifying Synthesis Constraints](#)

The following illustration shows a typical power network synthesis flow for a multivoltage design.

Figure 8-3 Multivoltage Power Network Synthesis Flow



Saving the Design

Prior to running power network synthesis on your design, you should save the design by using the `save_mw_cel` command. If you are not satisfied with the results of power network synthesis, you can revert to the saved version and resynthesize using different constraints. The following command saves the design before power network synthesis.

```
icc_shell> save_mw_cel -as floorplan_prepns
```

Selecting a Voltage Area and Specifying Synthesis Constraints

Begin power network synthesis by selecting a voltage area and setting synthesis constraints on that area. You must set synthesis constraints before setting the layer, ring, and global constraints on the voltage area. Synthesis constraints include settings for the power and ground net names, supply voltage, power budget, target IR drop, and power-switch name. Use the `set_fp_rail_voltage_area_constraints` command, or choose Preroute > Multi-Voltage Power Network Constraints > Voltage Area Constraints in the GUI to select the voltage area and set the constraints. [Table 8-2](#) describes the `set_fp_rail_voltage_area_constraints` command options used to set synthesis constraints for a voltage area.

Table 8-2 set_fp_rail_voltage_area_constraints Command Options for Power Network Synthesis

Command option	Description
<code>-voltage_area areaname</code> ("Voltage Area" selection box in the GUI)	Specifies the name of the voltage area.
<code>-nets net_names</code> ("Power Ground nets" text box in the GUI)	Specifies the power and ground net names.
<code>-voltage_supply voltage</code> ("Supply Voltage" text box in the GUI)	Specifies the supply voltage in volts for this voltage area.
<code>-power_budget power</code> ("Power Budget" text box in the GUI)	Specifies the power budget in mW for the voltage area.

*Table 8-2 set_fp_rail_voltage_area_constraints Command Options for Power Network Synthesis
(Continued)*

Command option	Description
<code>-target_voltage_drop target</code> ("Target IR drop" box in the GUI)	Specifies the target voltage drop in mV for the voltage area.
<code>-power_switch switchname</code> ("Power switch name" in the GUI)	Specifies the name of the power switch for this voltage area.

Note:

To set the constraint for a power-down voltage area, you must specify the name of the permanent power net followed by the name of the virtual power net. For example, use the `set_fp_rail_voltage_area_constraints -voltage_area VA1 -nets VDD+VDD_Virtual` command to assign a constraint on the VDD net for the power-down voltage area named VA1. IC Compiler synthesizes both nets connected by power-switch cells.

For a UPF design, IC Compiler inserts the power and ground net names automatically, although you must provide supply voltage information. If you do not set the power budget, power network synthesis assigns a power budget based on the ratio of the instance area for this voltage area divided by the area of the entire design.

The following example sets a power network synthesis constraint on the VA1 voltage area for the VDD and VSS power nets using a supply voltage of 1.5 V, a power budget of 200 mW, a target IR drop of 160 mV, and a power switch named SWITCH1.

```
icc_shell> set_fp_rail_voltage_area_constraints -voltage_area VA1 \
    -power_budget 200 -power_switch SWITCH1 -voltage_supply 1.5 \
    -target_voltage_drop 160 -nets {VDD VSS}
```

Setting Layer Constraints

To assign layer constraints for a voltage area, use the `set_fp_rail_voltage_area_constraints` command or choose Preroute > Multi-Voltage Power Network Constraints > Voltage Areas Constraints in the GUI and click the Layer constraints button. The layer constraints define the layer, direction, strap width, strap spacing or density, and net type for the voltage area. [Table 8-3](#) describes the options for the `set_fp_rail_voltage_area_constraints` command.

Table 8-3 set_fp_rail_voltage_area_constraints Options for Layer Constraints

Command option	Description
<code>-layer layer</code> ("Layer" text box in the GUI)	Specifies the layer to which to apply the constraint.
<code>-direction direction</code> ("Direction" options in the GUI)	Specifies the routing direction, horizontal or vertical, for the layer.
<code>-max_strap number</code> ("Density,By strap number,Max" text box in the GUI)	Specifies the maximum number of power and ground straps in the voltage area. Use the <code>-min_strap</code> option to specify the minimum number of straps.
<code>-max_pitch distance</code> ("Density,By pitch,Max" text box in the GUI)	Specifies the maximum pitch of the power and ground straps in the voltage area. Use the <code>-min_pitch</code> option to specify the minimum pitch.
<code>-max_width distance</code> ("Width,Max" text box in the GUI)	Specifies the maximum width of the power and ground straps in the voltage area. Use the <code>-min_width</code> option to specify the minimum strap width.
<code>-spacing minimum interleaving distance</code> ("PG spacing" options in the GUI)	Specifies the spacing between power and ground straps in the specified voltage area.
<code>-offset offset</code> ("Offset" text box in the GUI)	Specifies the distance from the voltage area boundary to the left or bottom power strap.
<code>-mcticmos_net_type permanent virtual</code> ("Net type" check box and options in the GUI)	Specifies the power-switch array synthesis net type, permanent or virtual, for the layer.

The following example specifies a power network layer constraint on the VA1 voltage area for layer METAL6. The layer direction is vertical, the maximum number of power straps is 128, the minimum number is 4, the minimum width is 0.1 microns, and the spacing is minimum.

```
icc_shell> set_fp_rail_voltage_area_constraints -voltage_area VA1 \
    -layer METAL6 -direction vertical -max_strap 128 -min_strap 4 \
    -min_width 0.1 -spacing minimum
```

Setting Ring Constraints

Ring constraints specify the layer, ring width, ring spacing, and strap extension parameters used when creating power and ground rings around a voltage area. To assign ring constraints for a voltage area, use the `set_fp_rail_voltage_area_constraints` command with the appropriate options or choose Preroute > Multi-Voltage Power Network Constraints > Voltage Areas Constraints in the GUI and click the Ring and Straps constraints button. [Table 8-4](#) describes the command options for the `set_fp_rail_voltage_area_constraints` command used to create power rings around voltage areas.

Table 8-4 set_fp_rail_voltage_area_constraints Options for Ring Constraints

Command option	Description
<code>-ring_nets net_names</code> ("Power Ground nets" in the GUI)	Specifies the power and ground net names for the power ring.
<code>-horizontal_ring_layer_layers</code> ("Horizontal layers" box in the GUI)	Specifies the layers to use for horizontal routes for power and ground nets. Use the <code>-vertical_ring_layer</code> option to specify the layers for vertical routes for power and ground nets.
<code>-ring_width width</code> ("Ring width, Fixed" text box in the GUI)	Specifies the power and ground ring width. Use the <code>-ring_max_width</code> and <code>-ring_min_width</code> options to specify a maximum and minimum width.
<code>-ring_spacing spacing</code> ("Ring spacing" text box in the GUI)	Specifies the distance between the power and ground rings in the voltage area.
<code>-ring_offset offset</code> ("Ring offset to IO or PNS region" text box in the GUI)	Specifies the distance from the voltage area boundary to the closest ring net.

Table 8-4 set_fp_rail_voltage_area_constraints Options for Ring Constraints (Continued)

Command option	Description
<pre>-extend_strap voltage_area_ring core_ring boundary ("Extend straps to" options in the GUI)</pre>	<p>Specifies how to extend power and ground nets in the voltage area. The <code>boundary</code> keyword is used primarily for block-level designs that do not contain I/O pad cells. Use the <code>core_ring</code> keyword at the chip-level for designs that contain I/O pad cells. Use the <code>voltage_area_ring</code> keyword to extend the straps to the ring inside the voltage area; this is the default behavior.</p>
<pre>-num_extend_straps number ("Number of straps ..." text box in the GUI)</pre>	<p>Specifies the number of power straps extended from each side of the voltage area to the core ring.</p>
<pre>-extend_strap_direction {left right top bottom all} ("Direction for extending straps" check boxes in the GUI)</pre>	<p>Specifies the directions in which to extend the straps from the voltage area to the core ring or boundary.</p>

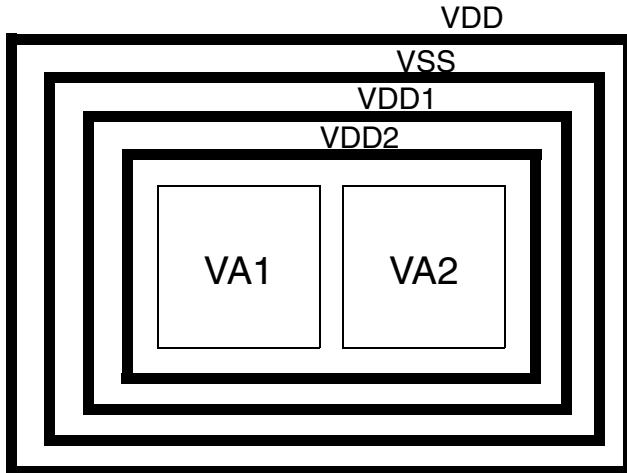
Note:

You can define constraints for the core ring only for the default voltage area. To generate the core ring, select the default voltage area in the Voltage Area selection box, choose the “Generate core rings” check box in the GUI, and specify the ring power and ground net names in the Power Ground nets text box. You can generate a permanent ring in the default voltage area.

[Figure 8-4 on page 8-15](#) shows a set voltage rings created by using the following `set_fp_rail_voltage_area_constraints` command:

```
icc_shell> set_fp_rail_voltage_area_constraints \
-voltage_area DEFAULT_VA -ring_nets {VDD VSS VDD1 VDD2} \
-horizontal_ring_layer { M9 } -vertical_ring_layer { M8 }
```

Figure 8-4 Core Rings for Default Voltage Area



IC Compiler supports different techniques for extending power straps from the voltage area. The `-extend_strap voltage_area_ring` option (“Voltage area ring” option in the GUI) extends the straps to the power and ground ring for the voltage area. The `-extend_strap core_ring` option (“Core ring” option in the GUI) extends the power and ground straps to the power ring for the default voltage area. The `-extend_strap boundary` option extends the straps to the top-level cell boundary. The `-extend_strap pad_ring` option extends the straps to the top-level pad ring, and the `-extend_strap voltage_area_boundary` option extends the straps to the boundary of the voltage area set by using the `create_voltage_area -coordinate` command.

You can also control the direction used for extending the straps. Set the direction for power rail extension by using the `-extend_strap_direction` option or by selecting the appropriate check boxes in the “Direction for extending straps” section of the GUI. By default, IC Compiler extends the power straps from all sides of the voltage area, unless they are blocked by a constraint, a hard macro, or another voltage area.

The ground net is not extended out from each voltage area to the core ring because the ground net is synthesized as a common net in the design. As a result, the ground net in each voltage area is extended to the nearest ground straps outside the voltage area boundary.

Setting Global Constraints

Define global constraints by using the `-global` option with the `set_fp_rail_voltage_area_constraints` command, or by clicking the Global constraints button in the GUI. The list of global constraint options is the same as the list for the `set_fp_rail_constraints` command.

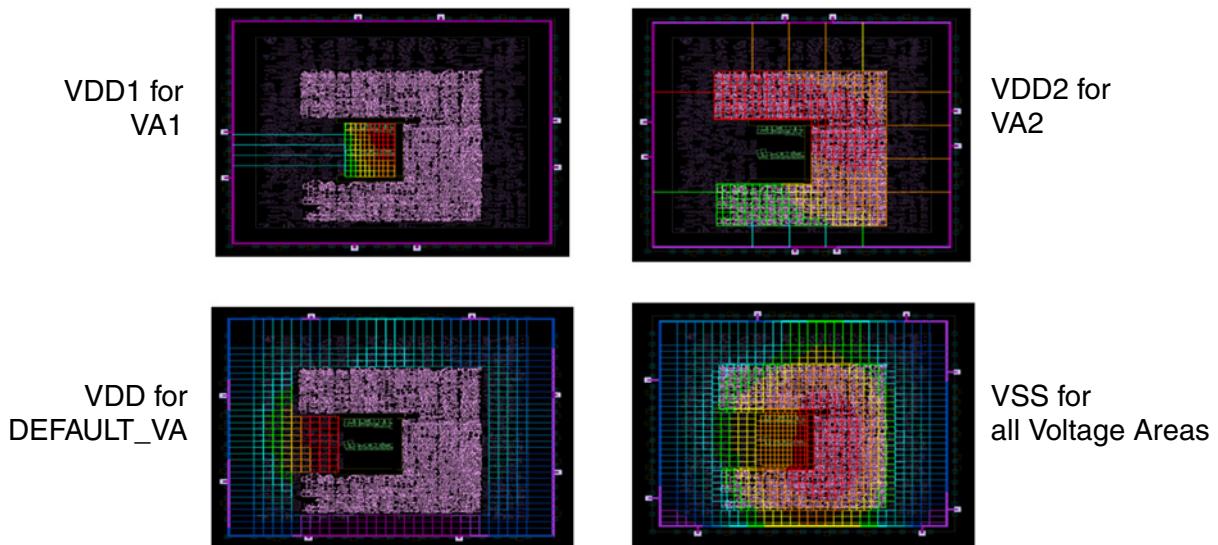
Synthesizing the Power Network

After you set the power network constraints by using the `set_fp_rail_voltage_area_constraints` command, you can perform power network synthesis on your multivoltage design. IC Compiler supports synthesis of all voltage areas at the same time, or synthesis of each voltage area individually. Use the `synthesize_fp_rail` command, or choose Preroute > Synthesize Power Network in the GUI.

To synthesize the power network for the voltage areas, choose the “Synthesize power network by voltage areas” option in the GUI. You can synthesize all voltage areas at the same time by selecting the “All voltage areas” option. To synthesize a specific set of voltage areas, select the “Specified voltage areas” option and enter the voltage area name in the text box. Use the `synthesize_fp_rail -synthesize_voltage_areas -voltage_areas { area_names }` command to synthesize the power network for the specified voltage areas from the command line, or do not use the `-voltage_areas` option to synthesize all voltage areas.

[Figure 8-5](#) shows the results of power network synthesis on multiple voltage areas. The figure contains an IR drop map for each power and ground net in the different voltage areas.

Figure 8-5 IR Drop Maps for Different Nets and Voltage Areas



Performing Template-Based Power Planning

The template-based power planning flow separates the description of the power and ground ring and mesh architecture from the implementation details. You define the implementation details in a template file to be used across multiple designs that share the same technology. A power ring or power mesh strategy associates the power nets in your design with the implementation details in the template file. Separating the ring generation and mesh generation steps simplifies the creation of the power network for your design.

The following sections describe the flow to create power rings and power meshes by using the template-based power network synthesis flow.

1. [Defining Power Plan Regions](#)
 2. [Creating the Power Ring Template File](#)
 3. [Creating the Power Ring Strategy](#)
 4. [Compiling the Power Rings](#)
 5. [Creating the Power Mesh Template File](#)
 6. [Creating the Power Mesh Strategy](#)
 7. [Compiling the Power Mesh](#)
-

Defining Power Plan Regions

In the template-based power network synthesis flow, power plan regions define the area used to create the power rings and power straps that form the power network. You can use a power plan region as a working area to create power plan strategies or blockage areas for other routing areas. Use the following commands to create and manipulate power plan regions:

- `create_power_plan_regions` - Create a power plan region
- `update_power_plan_region` - Reshape a power plan region
- `write_power_plan_regions` - Write out names and coordinates for all power plan regions
- `read_power_plan_regions` - Read in a power plan region names and coordinates file written by the `write_power_plan_regions` command
- `get_power_plan_regions` - Retrieve a collection of existing power plan regions

Creating a Power Plan Region

A power plan region can combine a group of macro cells or user-specified input polygons. You can also form a power plan region by upsizing or downsizing the boundary of the core area, a voltage area or a set of macro cells. The command syntax for the `create_power_plan_regions` command is as follows. For more information about these options, see the man page.

```
create_power_plan_regions
  region_name
  [-core |
   -group_of_macros macro_cells |
   -polygon {polygon_area} |
   -voltage_area voltage_area
   [-exclude_macros macro_cells]
   [-expand {x_offset y_offset}]
   [-jog_threshold threshold]
   [-macro_offset offset]
   [-macro_offset_file file_name]
   [-notch_threshold threshold]
   [-remove_jog_method expand | shrink]
   [-remove_notch_type convex | concave]
```

For example, the following command creates a power plan region based on the core area and excludes two macro cells. This technique is useful in designs where you do not want to create a power and ground grid that overlaps certain macro cells. In this case, the power plan region forms a new power and ground grid region, and the tool creates power and ground on the region.

```
icc_shell> create_power_plan_regions region_1 \
           -exclude_macros {macro1 macro2} -core -expand -35 \
           -notch_threshold 50 -macro_offset 35
```

Retrieving Power Plan Regions

Use the `get_power_plan_regions` command to retrieve a collection of power plan regions. To retrieve the region names, bounding boxes, and boundary points for the power plan regions, use the `get_power_plan_regions` command together with the `get_attribute` command. The following example creates a power plan region named `region_1` and retrieves the name, bounding points, and bounding box for all power plan regions.

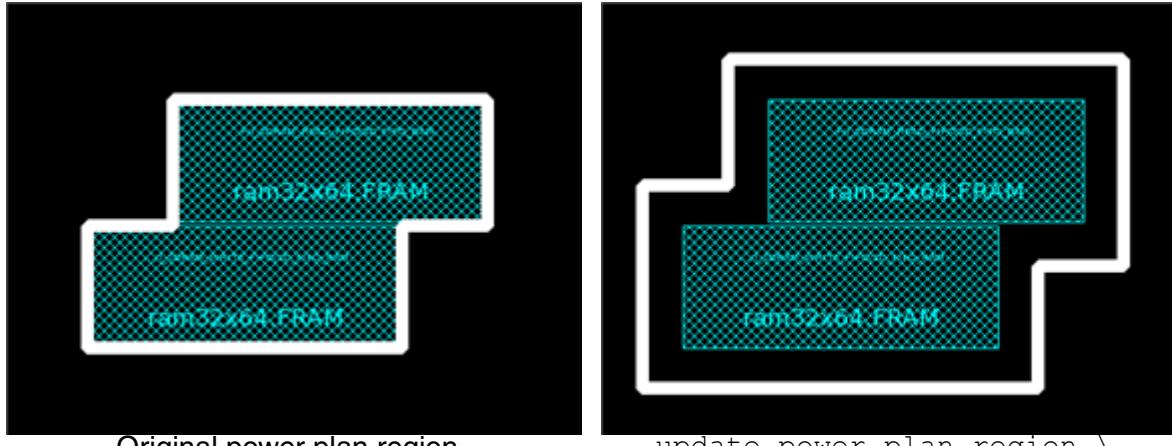
```
icc_shell> create_power_plan_regions region_1 \
           -group_of_macros {U1/RAM U2/RAM}
All blocks have been successfully merged
1
icc_shell> get_power_plan_regions
{region_1}
icc_shell> get_attribute [get_power_plan_regions] name
region_1
icc_shell> get_attribute [get_power_plan_regions] points
{1310.980 1257.350} {1310.980 1135.225} {996.660 1135.225}
...
...
```

Reshaping Power Plan Regions

The `update_power_plan_region` command modifies the boundary of an existing power plan region. You can use the `update_power_plan_region` command after you have defined a power plan region. The syntax of the `update_power_plan_region` command is as follows:

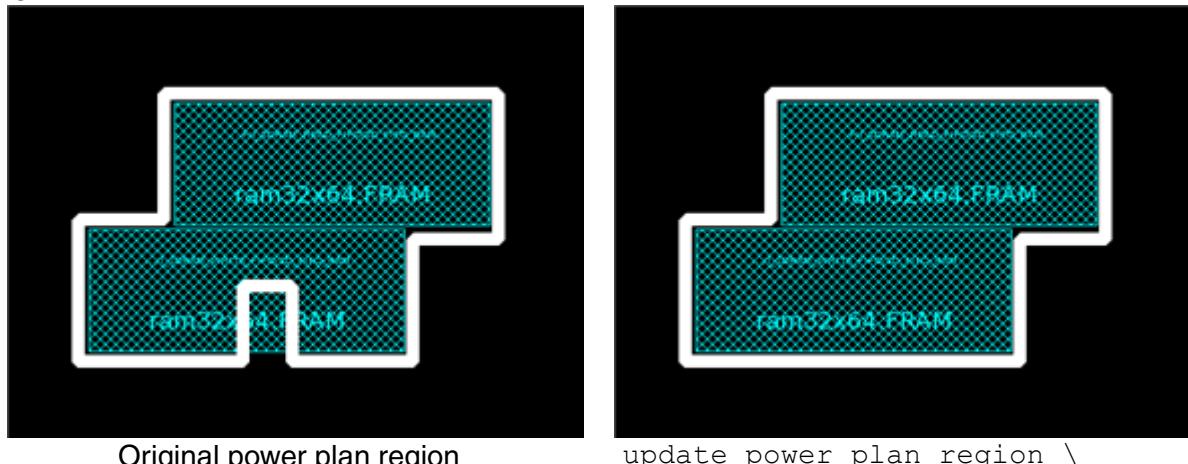
```
update_power_plan_region
  -power_plan_region region_name
  [-polygon {polygon_area}]
  [-expand {x_dist y_dist}]
  [-remove_jog_method expand | shrink]
  [-jog_threshold threshold]
  [-remove_notch_type convex | concave]
  [-notch_threshold threshold]
```

Use the `-polygon` option to create a new boundary for the power plan region by specifying bounding points. Use the `-expand` option to enlarge the power plan region boundary by a specified amount. You can also specify the `-expand` option with a negative distance to shrink the power plan region. [Figure 8-6 on page 8-20](#) shows a power plan region before and after running the `update_power_plan_region` command with the `-expand` option.

Figure 8-6 Power Plan Region Expansion

```
update_power_plan_region \
-power_plan_region region1 \
-expand {20 20}
```

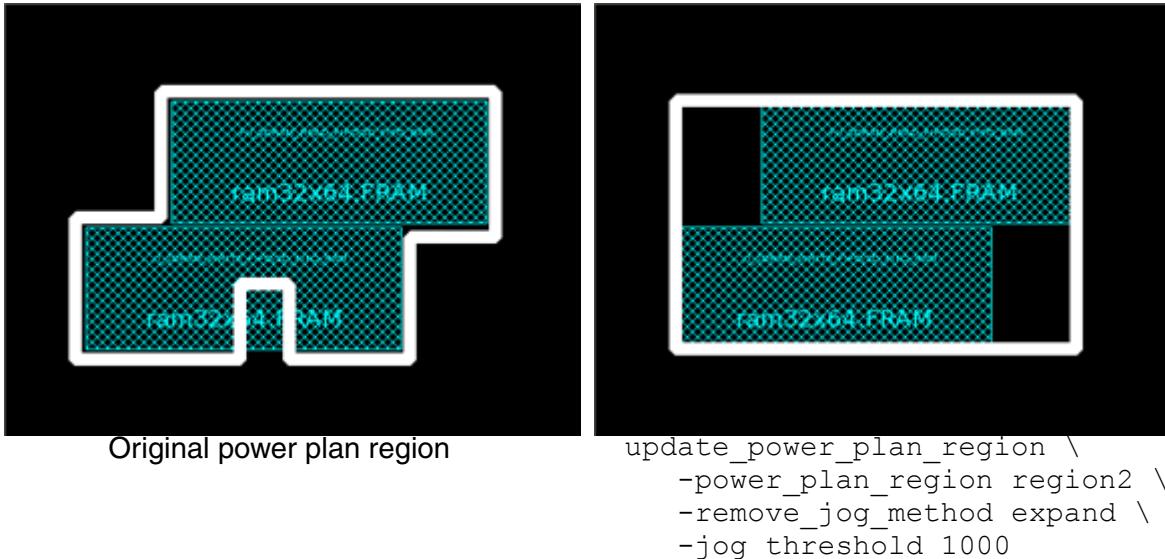
Use the `-remove_notch_type` and `-notch_threshold` options to remove the specified notch type from the power plan region. Specify `convex` to remove notches that extend outside the region; specify `concave` to remove notches that indent into the region. The command removes any `convex` or `concave` notches with a width smaller than the threshold specified by the `-notch_threshold` option. [Figure 8-7](#) shows a power plan region before and after running the `update_power_plan_region` command with the `-remove_notch_type concave` and `-notch_threshold 1000` options.

Figure 8-7 Notch Removal

```
update_power_plan_region \
-power_plan_region region2 \
-remove_notch_type concave \
-notch_threshold 1000
```

Use the `-remove_jog_method` and `-jog_threshold` options to remove jogs in the power plan region. Specify `-remove_jog_method expand` to increase the size of the region; specify `-remove_jog_method shrink` to decrease the size of the region. The command removes any jog edge in the region that is shorter than the threshold specified by the `-jog_threshold` option. [Figure 8-8](#) shows a power plan region before and after running the `update_power_plan_region` command with the `-remove_jog_method expand` and `-jog_threshold 1000` options.

Figure 8-8 Jog Removal



Reading and Writing Power Plan Regions

To write out and read in a power plan region, use the `write_power_plan_regions` and `read_power_plan_regions` commands. The `write_power_plan_regions` command writes an ASCII file containing the coordinates of the power plan region. The following example writes out a power plan region file, displays the contents, and reads the file.

```
icc_shell> write_power_plan_regions -output core.reg
1
icc_shell> sh cat core.reg
Begin Power Plan Region Definition: region_1
345.565 1843.705
1846.165 1843.705
1846.165 345.565
345.565 345.565
End Power Plan Region Definition
icc_shell> read_power_plan_regions core.reg
1
```

Creating the Power Ring Template File

The power ring template file is a text file that describes the metal layer, width, spacing, and offset for the horizontal and vertical segments of the power ring and specifies advanced rules for power ring vias and ring alignment to standard cells. A power ring template contains a `template` header, one or more `side` sections, and an optional `advanced_rule` section.

If you specify unique template names, you can combine multiple power ring templates and power mesh templates in a single template file. The power mesh template syntax is described in “[Creating the Power Mesh Template File](#)” on page 8-27. Both the power ring and power mesh templates support parameters. Using parameterizable templates, you can assign power ring parameters when you implement the power rings and experiment with different configurations without modifying the template. You can use parameters to define width, spacing, and offset for the power rings. When you compile the power rings, you can assign values to the parameters by using the `set_power_ring_strategy` command.

The power ring and power mesh template header uses the following syntax:

```
template : template_name (parameter1, parameter2, ...) {
```

You can also specify a header without parameters by using the following syntax:

```
template : template_name {
```

For more information about using parameters in templates, see “[Creating and Using Parameters in a Template File](#)” on page 8-29.

Creating the Side Section

The `side` section that follows the template header defines the layer, width, spacing and offset for the power ring. You can create one or more side specifications in a single ring template. The `side` section uses the following syntax:

```
# The side section defines the configuration of one or more sides of a
# rectangular or rectilinear ring. You can specify multiple side
# sections. Specify "horizontal" or "vertical" to assign properties
# for all horizontal or vertical sides. Specify side numbers to assign
# properties for individual ring segments. Side number settings
# override horizontal and vertical settings. Side numbering starts with
# 1 from the lowest leftmost edge of the pattern and increments in the
# clockwise direction.

side : horizontal | vertical | {index1, index2, ...} {

    # The layer specification defines the layer used to
    # create the ring segment

    layer : layer_name
```

```
# The width specification defines the ring width of the
# sides. Specify one or more values that correspond to the number
# of nets defined by set_power_ring_strategy. If the number of
# nets in the strategy is greater than the number of width values
# defined here, the last width value is used for any undefined
# values. The unit is microns.

width : {width1, width2, ...}

# The spacing specification defines the spacing between rings for
# multiple nets. Specify "minimum" to use the minimum spacing from
# the technology file for the ring width. Specify one or more
# values corresponding to the number of ring nets defined in the
# ring strategy. The number of spacing values is equal to the
# number of rings minus one. The unit is microns.

spacing : minimum | {spacing1, spacing2, ... }

# The offset specification defines the separation between the
# inner edge of the innermost ring and the boundary of the ring
# contour. The unit is microns.

offset : offset
}
```

Creating the Advanced Rules Section

The `advanced_rule` section enables or disables advanced options during ring creation. The `advanced_rule` section uses the following syntax:

```

# The advanced rule section controls various options for via
# creation and power ring alignment

advanced_rule : on | off {

    # align_std_cell_rail : on enables the tool to align
    # horizontal rings with standard cell rails such that
    # the rings do not block vias on other nets. By default
    # this rule is off

    align_std_cell_rail : on | off

    # corner_bridge : on enables the tool to create via
    # bridges to connect rings that belong to the same
    # net. By default, this option is off

    corner_bridge : on | off

    # honor_advanced_via_rule : on honors via rules
    # specified by the set_preroute_advanced_via_rule
    # command. By default, this option is off

    honor_advanced_via_rule : on | off
}

```

Creating the Power Ring Strategy

The power ring strategy associates the power ring template with the following design characteristics and specifications:

- The list of power and ground nets for the power ring
- An extension specification for the power ring
- The routing periphery for the power ring
- A skip specification for the power ring

In the template-based power planning flow, the details of the power and ground rings are defined within a template file. The template file supports complex, design-independent power structures that can be used across several designs. See “[Creating the Power Ring Template File](#)” on page 8-22 for more information about the power ring template file.

You can define multiple ring strategies for a single design. For multivoltage designs, you might require a different strategy for each voltage area.

The `set_power_ring_strategy` command defines the strategy for the power rings in your design. The syntax of the `set_power_ring_strategy` command is as follows:

```

set_power_ring_strategy
  strategy_name
  -nets nets
  [-core
   | -voltage_areas voltage_areas
   | -polygon {polygon_area}
   | -macros macro_names
   | -power_plan_regions power_plan_region_name]
  [-template template_spec]
  [-extension {extension_spec}]
  [-skip {skip_spec}]

```

Each power ring strategy is identified by a unique strategy name. You can create more than one strategy by running the `set_power_ring_strategy` command multiple times. The power ring periphery is defined by the `-core`, `-voltage_area`, `-polygon`, `-macros`, or `-power_plan_regions` option.

The extension strategy defines which ring segments to extend, the direction they extend, and the distance of the extension. This option is specified by the `-extension` option. The extension specification for the power ring contains the `nets:`, `direction:`, and `sides:` keywords. If you omit one or more net names from the specification, the command extends all the nets defined by the `-nets` option. You can specify the extension direction by including the `direction:` keyword and up to four directions by using the letters L,R,T and B for left, right, top, and bottom respectively. Vertical power ring segments extend only in the top and bottom directions. Horizontal power ring segments extend only in the left and right directions.

You can prevent the tool from creating all segments of the power ring by specifying the `-skip` option of the `set_power_ring_strategy` command. The option omits the specified power ring segments for each power net in the ring. Specify the power ring nets, sides, and layers by using the following syntax:

```

-skip {{nets: net_name} {sides: {side_1 side_2 ...}} \
      {layers: layer_name}}

```

You can specify multiple sides within braces. The lowest leftmost side of the ring periphery is side 1. Side numbers increase as you proceed clockwise around the ring.

Reporting and Removing Power Ring Strategies

You can report existing power ring strategy settings for all defined strategies by using the `report_power_ring_strategy` command. To delete one or more power ring strategies, use the `remove_power_ring_strategy` command. To delete all power ring strategies, use the `-all` option with the `remove_power_ring_strategy` command.

The following example shows the output of the `report_power_ring_strategy` and `remove_power_ring_strategy` commands.

```

icc_shell> report_power_ring_strategy
*****
Report : Power Ring Strategy
Design : floorplan_init
Version: F-2011.09
Date   : Mon Aug 22 16:38:45 2011
*****
Power Ring Strategy  : ring_strategy_1
Net names            : VDD1 VSS1
Template             : ringtemplate.tpl
Routing region       : region1
Net Extension        : -- Nets --  -- Direction --  -- Sides --
Skip Side            : -- Nets --  -- Layer --      -- Sides --
Power Ring Strategy : ring_strategy_2
Net names            : VDD2 VSS2
Template             : ringtemplate.tpl
Routing region       : region2
Net Extension        : -- Nets --  -- Direction --  -- Sides --
Skip Side            : -- Nets --  -- Layer --      -- Sides --
1

icc_shell> remove_power_ring_strategy -all
Power plan strategy ring_strategy_1 is removed.
Power plan strategy ring_strategy_2 is removed.
1

```

Compiling the Power Rings

To add the power rings defined by the `set_power_ring_strategy` command to your design, use the `compile_power_plan` command. The `compile_power_plan` command uses the following syntax:

```

compile_power_plan
  [-ignore_design_rules]
  [-ring]
  [-strategy strategy_name]
  [-undo]
  [-verbose]
  [-write_default_template template_file_name]

```

To create a power ring for a power ring strategy, use the `compile_power_plan -ring -strategy strategy_name` command.

Use the `compile_power_plan -ring -write_default_template ring.tpl` command to write out an example power ring template to a file named `ring.tpl`. You can combine power mesh and power ring templates within a single file if you make sure that each template name is unique.

To compile the power rings using the GUI layout window, choose Preroute > Compile Power Plan. See “[Compiling the Power Mesh](#)” on page 8-37 for more information about using the GUI to compile the power rings in your design.

Creating the Power Mesh Template File

The power mesh template file is a text file that describes the width, spacing, pitch, offset, and other characteristics of the power mesh. The power mesh template also supports advanced rule definitions for strap optimization, strap alignment, strap insertion into channel areas, and other design features. When you commit the power network, IC Compiler creates the power and ground mesh according to the power mesh definition in the template file, and the power strategy set by the `set_power_plan_strategy` command.

Template-based power planning supports parameterizable templates. This feature enables you to define variables as placeholders for specific power mesh parameters and assign values when you create the power mesh in IC Compiler. The parameterizable template simplifies template creation and maintenance by supporting template reuse and multiple templates within the same file. Using parameterizable templates, you can assign values to the parameters, such as width, offset, and spacing, by using the `set_power_plan_strategy` command. Parameterizable templates allow you to easily experiment with different power strap configurations without modifying the template. When you create the power plan, you can assign values to the parameters by using the `set_power_plan_strategy` command.

A power mesh template contains a header, one or more layer sections, and an advanced rule section. A template file can contain multiple power mesh and power ring templates. A template header uses the following syntax:

```
template : template_name (parameter1, parameter2, ...) {
```

You can also specify a header without parameters by using the following syntax:

```
template : template_name {
```

Creating the Layer Section

The layer section defines the metal layer, strap width and other physical details of the strap in the power mesh. You can create one or more layer definitions in a single power mesh template. The layer section of the power mesh template uses the following syntax:

```
# The layer section defines the configuration of a layer in
# the power mesh. You can specify multiple layer sections.

layer : layer_name {
    direction : horizontal | vertical

        # The width specification defines the width of the power strap.
        # Multiple widths correspond to multiple nets. The net order is
        # defined by the strategy. The unit is microns.

    width : {width1, width2, ...}

        # The spacing specification defines spacing between power
        # straps. Multiple spacings correspond to unique distances
        # between multiple nets. The net order is defined by the
        # strategy. The unit is microns.

    spacing : minimum | interleaving | {spacing1, spacing2, ... }

        # The number specification defines the number of straps in the
        # power mesh.

    number : strap_number

        # The pitch specification defines the pitch between nets. The unit
        # is microns.

    pitch : pitch_between_nets

        # The offset_type specification defines how the offset is
        # calculated. offset_type: edge specifies that the offset is the
        # distance between the routing region boundary and the inner edge
        # of the first strap. offset_type: centerline specifies that the
        # offset is the distance between the routing region boundary and
        # the centerline of the first strap.

    offset_type: centerline | edge
```

```

# The offset_start specification defines where the mesh placement
# begins. offset_start: boundary specifies that the placement
# begins at the boundary of the routing area defined in the
# strategy. Points {x y} specify the starting placement coordinate.

offset_start : boundary | {x, y}

# The offset specification defines the offset value. The unit is
# microns.

offset : offset_to_boundary_or_coordinate

# The trim_strap specification defines how power mesh straps are
# trimmed. trim_strap: true trims all wire segments that intersect
# less than two other wires. By default, all straps stop at the
# boundary of the routing area and the tool does not trim straps
# that touch the routing area boundary.

trim_strap : true | false
}

```

You can define multiple width or spacing values for nets defined in the layer section of the strategy. If your strategy defines two or more nets with different widths and pitches, you must define all widths and spacings by using braces, ({ }). For example, if your strategy contains three nets VSS1, VSS2 and VSS3 with widths of 0.3, 0.35, and 0.5, and spacings of 0.2 and 0.25, you must include the following width and spacing definitions in your template file:

```

width : {0.3 0.35 0.5}
spacing : {0.2 0.25}

```

An example of the `set_power_plan_strategy` command that associates nets VSS1, VSS2, and VSS3 with these width and spacing settings is:

```
icc_shell> set_power_plan_strategy -nets {vss1 vss2 vss3}
```

Creating and Using Parameters in a Template File

You can pass parameters from the `set_power_plan_strategy` and `set_power_ring_strategy` commands into the template. To replace a hard-coded value in the template with a parameter, replace the numerical value with a parameter name preceded by the at character (@) and add the parameter name to the template header. Do not use parameters for keyword or Boolean values.

The following mytemplate.tpl template excerpt contains two templates: upper_grid and lower_grid. The p1 and p2 parameters are declared in the header. The p1 parameter replaces the numerical values for pitch for layers ME7 and ME5. The p2 parameter replaces the numerical values for pitch for layer ME8 and ME6.

```

template : upper_grid(p1,p2) {
    layer : ME7 {
        pitch : @p1
        ...
    }
    layer : ME8 {
        pitch : @p2
        ...
    }
}
template : lower_grid(p1,p2) {
    layer : ME5 {
        pitch : @p1
        ...
    }
    layer : ME6 {
        pitch : @p2
        ...
    }
}

```

Using the previous template, the following `set_power_plan_strategy` commands specify the template file, template name and parameter values to create the power plan. The first command specifies the `upper_grid` template and sets parameters p1 and p2 to 3.0 and 9.0 respectively. The second command specifies the `lower_grid` template and sets parameters p1 and p2 to 2.0 and 10.0 respectively.

```

icc_shell> set_power_plan_strategy s1 \
    -template mytemplate.tpl:upper_grid(3.0,9.0)
icc_shell> set_power_plan_strategy s2 \
    -template mytemplate.tpl:lower_grid(2.0,10.0)

```

Creating the Advanced Rules Section

The advanced rule section of the template file controls various power mesh spacing and alignment during mesh insertion with the `compile_power_plan` command. IC Compiler supports the following advanced rules in the template file:

- Optimize routing tracks
- Align strap with standard cell rails
- Insert straps inside channel area
- Honor the maximum distance from a standard cell to a power strap
- Align the straps with power-switch cells
- Honor advanced via rules
- Control stack via creation

The syntax for the advanced rules section is:

```

advanced_rule : on | off {
    optimize_routing_tracks : on | off {
        layer : layer_names # layers to optimize straps
        alignment : true | false # perform alignment
        sizing : true | false # perform automatic sizing
    }
    align_strap_with_stdcell_rail : on | off {
        layer : layer_names # layers to align straps
        align_with_rail : true | false
        put_strap_in_row : true | false
    }

    # Insert extra straps inside the channel area when on
    insert_channel_straps : on | off {
        # Layer name of the extra straps inserted. If not
        # specified, IC Compiler uses the vertical layer from
        # the template
        layer : layer_names

        # Width of the inserted straps, default is minimum
        width : minimum | list_of_floats

        # Spacing between inserted straps, default is minimum
        spacing : minimum | list_of_floats

        # Minimum spacing for channel strap insertion, straps
        # are not inserted into channels narrower than
        # the specified value
        channel_threshold : float

        # Only check if there is strap of the specified layer
        # inside channel. If not, insert straps
        check_one_layer : true | false

        # Consider channels between placement blockages
        # and between placement blockage and macros. By default,
        # only consider channels between macros
        honor_placement_blockage : true | false

        # Consider channels between voltage areas
        # and between voltage areas and placement blockages.
        honor_voltage_area : true | false

        # Space between boundary and macro or placement blockage is
        # considered a channel
        boundary_strap : true | false
    }
}

```

```

honor_max_stdcell_strap_distance : on | off {

    # Insert an extra strap if the distance between std
    # cell area and the closest strap is greater than
    # max_distance
    max_distance : float

    # Layer of the extra strap, default is the highest layer
    # of straps in the template
    layer : layer_name

    # Distance between the extra strap and the nearest strap
    offset : offset_values
}

# Insert extra straps to align with power-switch cells
align_straps_with_power_switch : on | off {

    # Specify the power switch to align. By default, all
    # cells with power-switch type are considered for
    # strap alignment
    power_switch : power_switch_name

    # Layer of the alignment strap. If not specified, the
    # vertical layer defined in layer section is used.
    layer : layer_names

    # Width of the alignment strap in microns. By default,
    # the strap width defined in layer section is used.
    width : float

    # Direction of the alignment strap, the default is
    # vertical
    direction : vertical | horizontal

    # Offset of the alignment strap from the center of the
    # aligned pins on power switch. Default is 0
    offset : offset_value
}

# Honor the advanced via rule set by the
# set_preroute_advanced_via_rule command.
honor_advanced_via_rule : on | off

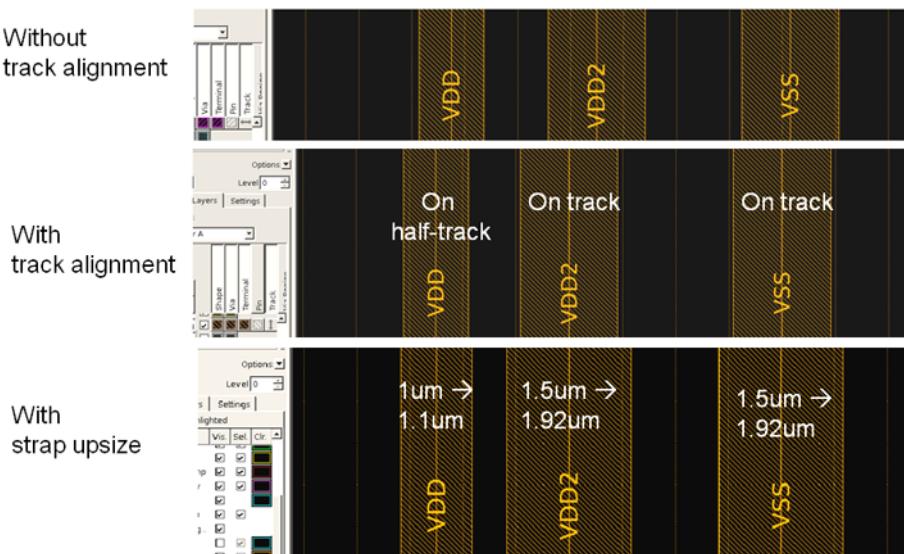
# Define the stacked vias in the power grid. "all" creates vias at
# all strap intersections; "adjacent" creates vias only between
# adjacent layers without stacking vias; "specified" creates vias
# based on the connect_layers specification
stack_vias : all | adjacent | specified {
    connect_layers : {lower_layer upper_layer}
}
}

```

Each advanced rule is associated only with its template. The following figures illustrate different applications of the advanced rules.

[Figure 8-9](#) shows the results after using the `optimize_routing_tracks` rule to control strap width and spacing.

Figure 8-9 Layout Results Using optimize_routing_tracks



The `insert_channel_straps` advanced rule adds extra vertical straps in the channels between hard macros. IC Compiler identifies the channels between hard macros and skips channels that contain synthesized straps or are too narrow for extra vertical straps. IC Compiler inserts extra vertical straps in the center of channels between hard macros. The `channel_threshold` parameter defines the minimum threshold needed to insert straps. You can select the layer and control the width and spacing of the strap. [Figure 8-10](#) shows an example in which an extra strap is inserted by IC Compiler.

Figure 8-10 Strap Insertion

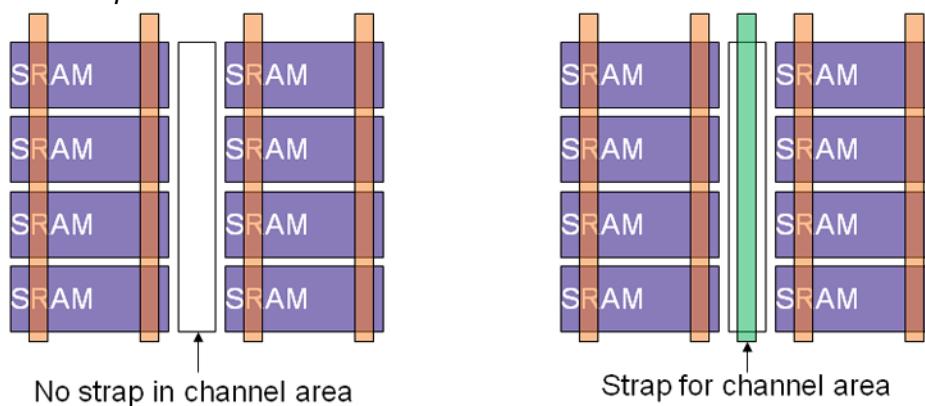
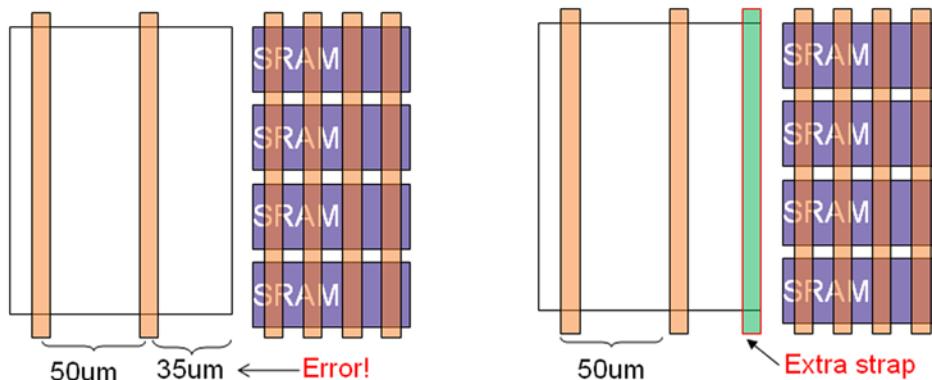


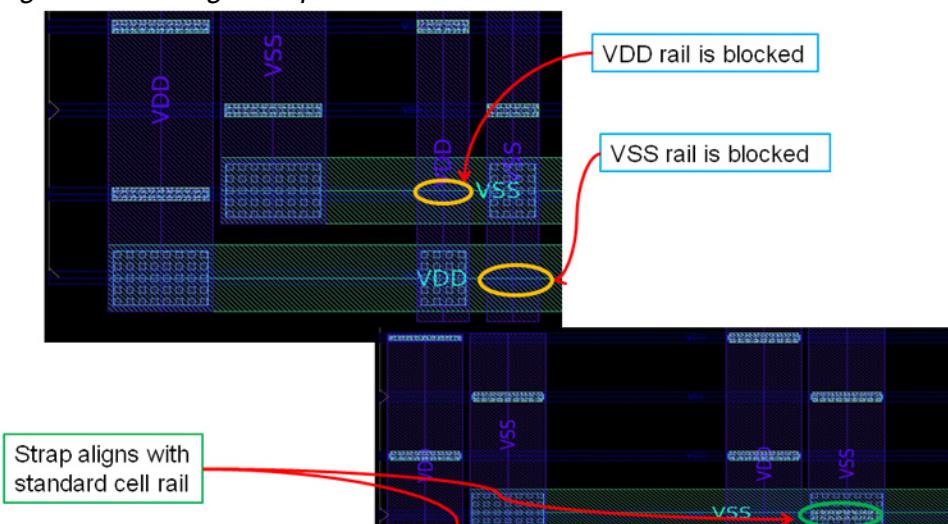
Figure 8-11 shows an example where the `honor_max_stdcell_strap_distance` rule is on and the distance between the standard cell boundary to the power strap determines whether a strap is inserted. In this design, if the distance is greater than 25 microns, a strap is inserted. You can specify the layer used to insert the strap.

Figure 8-11 Channel Minimum Width for Strap Insertion



The `align_strap_with_stdcell_rail` rule controls strap alignment with the horizontal power and ground rails of standard cells on the lowest horizontal metal layer. This rule can also place the straps within the standard cell rails to avoid horizontal power straps overlapping with the horizontal ground standard cell rails. An example is shown in [Figure 8-12](#).

Figure 8-12 Align Straps with Standard Cell Rails



Creating the Power Mesh Strategy

The power mesh strategy associates the power mesh template with the following design characteristics and specifications:

- A list of power and ground nets for the power mesh
- An extension specification for the power mesh
- The routing area for the power mesh
- A blockage specification for the power mesh
- A template file name for the power mesh

In the template-based power planning flow, the details of the power and ground mesh are defined within a template file. The template file supports complex, design-independent power structures that can be used across several designs. See “[Creating the Power Mesh Template File](#)” on page 8-27 for more information about the power mesh template file.

The power mesh strategy defines the routing area and a group of power and ground nets. You can define multiple power mesh and ring strategies for a single design. For multivoltage designs, you might require a different strategy for each voltage area.

The `set_power_plan_strategy` command defines the strategy for the power mesh in your design. The syntax for the `set_power_plan_strategy` command is as follows:

```
set_power_plan_strategy
  strategy_name
  -nets nets
  [-core |
   -voltage_areas voltage_areas |
   -polygon {polygon_area} |
   -macros macro_names |
   -power_plan_regions power_plan_region_name]
  [-template template_spec]
  [-extension {extension_spec}]
  [-blockage {blockage_spec}]
```

Each power mesh strategy is identified by a unique strategy name. You can create multiple strategies by running the `set_power_plan_strategy` command multiple times. The power mesh area is defined by the `-core`, `-voltage_area`, `-polygon`, `-macros`, or `-power_plan_regions` option.

The extension strategy defines which power mesh straps to extend, the direction they extend, and the distance of the extension. This option is specified by the `-extension` option. The extension specification for the power mesh contains the `nets:`, `direction:`, and `stop:` keywords. If you don't specify one or more net names, the command extends all the nets

defined by the `-nets` option. You can specify the extension direction by including the `direction:` keyword and up to four directions by using the letters L,R,T and B for left, right, top, and bottom respectively. By default, power and ground straps extend in all directions.

The extension stop point for the power mesh is defined by the `stop:` keyword and one of the following stop points: `first_target`, `innermost_ring`, `outermost_ring`, `design_boundary`, or a user-specified distance. The `first_target` keyword specifies that the power strap stops at the first wire segment of the same net; the `innermost_ring` keyword specifies that the power strap stops at the innermost ring of the same net. The `outermost_ring` keyword specifies that the strap stops at the outermost ring of the same net, and the `design_boundary` keyword specifies that the strap extends to the boundary of the current design. You can also specify a distance; the strap extends by the specified distance from the boundary of the routing region. If you specify a negative distance, IC Compiler creates a gap between the strap end and the boundary. If you do not specify a stop point by using the `stop:` keyword, the strap is not extended.

The following example defines a power mesh strategy named `strategy1` that extends the `VDD1` and `VDD2` nets in the left and bottom directions to the outermost ring, and extends the `VDD3` net in all directions.

```
icc_shell> set_power_plan_strategy strategy1 \
    -nets {VDD1 VDD2 VDD3} -core -extension { \
    {{nets: "VDD1 VDD2"} {direction: "LB"} {stop: outermost_ring}} \
    {{nets: "VDD3"} {stop:design_boundary}}}
```

The `-blockage` option of the `set_power_plan_strategy` command specifies routing restrictions in the routing area for the given power mesh strategy. You can specify multiple blockages within the braces, with one blockage per group. Blockages are separated by braces and contain the keywords `nets:`, `layers:`, and a destination keyword `voltage_area:`, `macro:`, `polygon:`, `plan_group:`, or `power_plan_region:`. The destination keywords require an associated value. The net names following the `nets:` keyword are a subset of the nets specified by the `-nets` option. If you do not specify the `nets:` keyword, the blockage applies to all the nets in the strategy. The layer names following the `layers:` keyword are the routing layers on which the power and ground straps are blocked. If you do not specify the `layers:` keyword, the blockage applies to all routing layers. Use the `voltage_area:`, `macros:`, `polygon:`, or `plan_group:` keyword to represent the blockage area.

To prevent placing the power and ground mesh over the `VA1` voltage area, specify the following syntax by using the `-blockage` option of the `set_power_plan_strategy` command:

```
-blockage {{nets: "VDD"} {voltage_area: "VA1"}}
```

You can use a hyphen character (-) as a positional placeholder with the `set_power_plan_strategy -nets` command. In the following example, the `compile_power_plan` command ignores any specifications in the top.tpl template file that refer to the second net name in the list. This feature enables template reuse.

```
icc_shell> set_power_plan_strategy strategy1 -core \
           -nets {VDD1 - VDD3} -template top.tpl
```

Reporting and Removing Power Mesh Strategies

You can report existing power mesh strategy settings for all defined power mesh strategies by using the `report_power_plan_strategy` command. To delete one or more power mesh strategies, use the `remove_power_plan_strategy` command. To delete all power mesh strategies, use the `-all` option with the `remove_power_plan_strategy` command.

The following example uses the `report_power_plan_strategy` and `remove_power_plan_strategy` commands.

```
icc_shell> report_power_plan_strategy
*****
Report : Power Plan Strategy
Design : floorplan_init
Version: F-2011.09
Date   : Mon Aug  8 14:11:34 2011
*****
Power Plan Strategy : strategy1
Net names          : VDD
Template           : default.tpl
Routing region     : core area
Net Extension      : -- Nets --    -- Direction --    -- Stop at --
                      VDD            LB                outermost_ring
Blockage          : -- Nets --    -- Area --       -- Layer --
1
icc_shell> remove_power_plan_strategy -all
Power plan strategy strategy1 is removed.
```

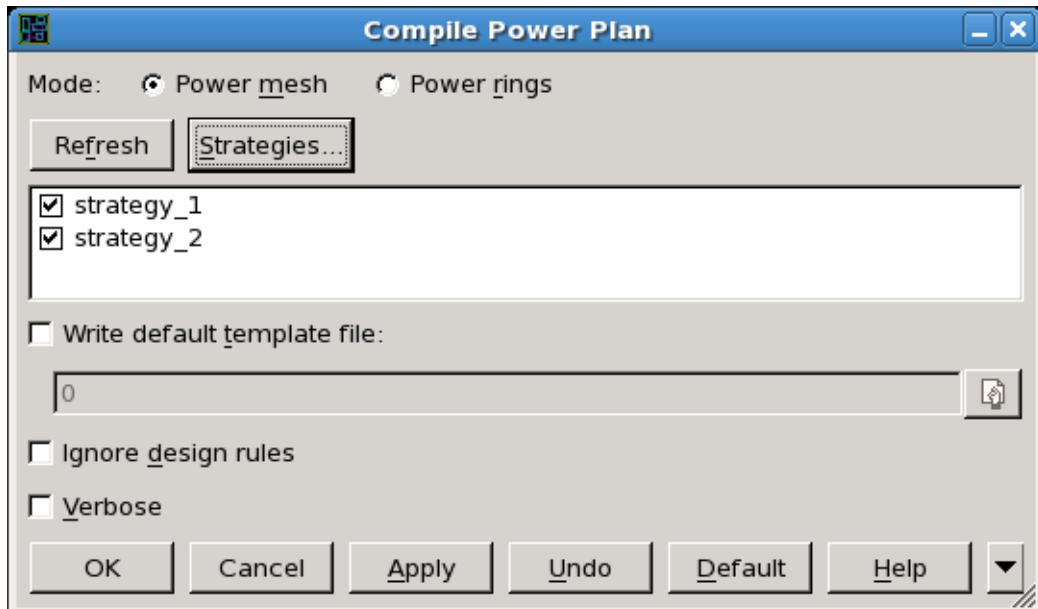
Compiling the Power Mesh

To create the power mesh defined by the `set_power_plan_strategy` command, use the `compile_power_plan` command. Use the `compile_power_plan -write_default_template mesh.tpl` command to write out an example power mesh template to a file named `mesh.tpl`. You can combine power mesh and power ring templates within a single file. The `compile_power_plan` command uses the following syntax:

```
compile_power_plan
[-ignore_design_rules]
[-ring]
[-strategy strategy_name]
[-undo]
[-verbose]
[-write_default_template template_file_name]
```

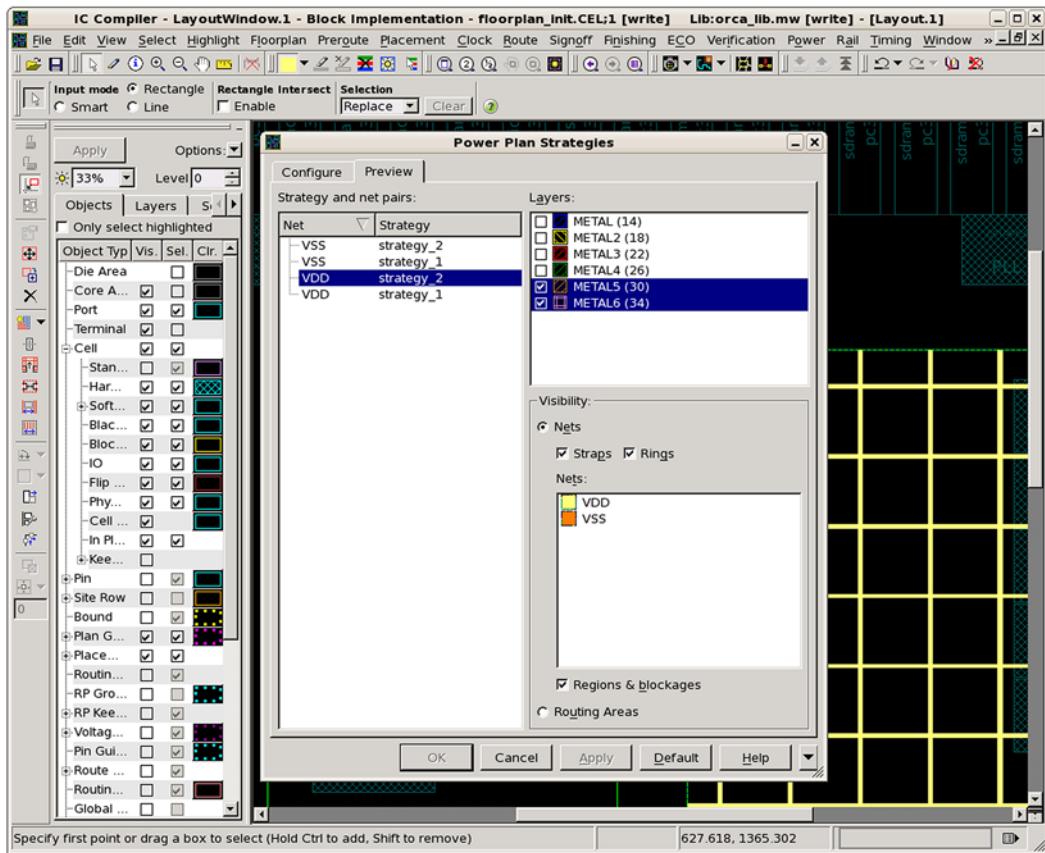
To compile the power mesh using the GUI layout window, choose Preroute > Compile Power Plan. This opens the dialog box shown in [Figure 8-13](#). This GUI dialog is also used to add the power rings to your design. The dialog box lists the strategies defined by using the `set_power_plan_strategy` command.

Figure 8-13 Compile Power Plan Dialog Box



You can preview the power mesh in the GUI before committing it. Click the Strategies button to open the Power Plan Strategies dialog and click the Preview tab. To preview nets, select the Layers to preview, select the Nets option in the Nets section, and select Straps or Rings to preview them. To preview routing areas, select the Routing Areas option. An example power mesh preview is shown in [Figure 8-14 on page 8-39](#).

Figure 8-14 Power Plan Preview



Template-Based Power Plan Examples

This section contains additional examples that show how to create template-based power plans for different power network architectures.

Creating Rings Around Voltage Areas

The following design contains unique rings around three voltage areas. To implement this power plan, do the following steps:

1. Create the ring.tpl template file.

```
template : ring_around_va {
    side : horizontal {
        layer : M3
        width : 2.0
        spacing : minimum
        offset : 1.0
    }
    side : vertical {
        layer : M4
        width : 2.0
        spacing : minimum
        offset : 1.0
    }
}
```

2. Set the power ring strategy.

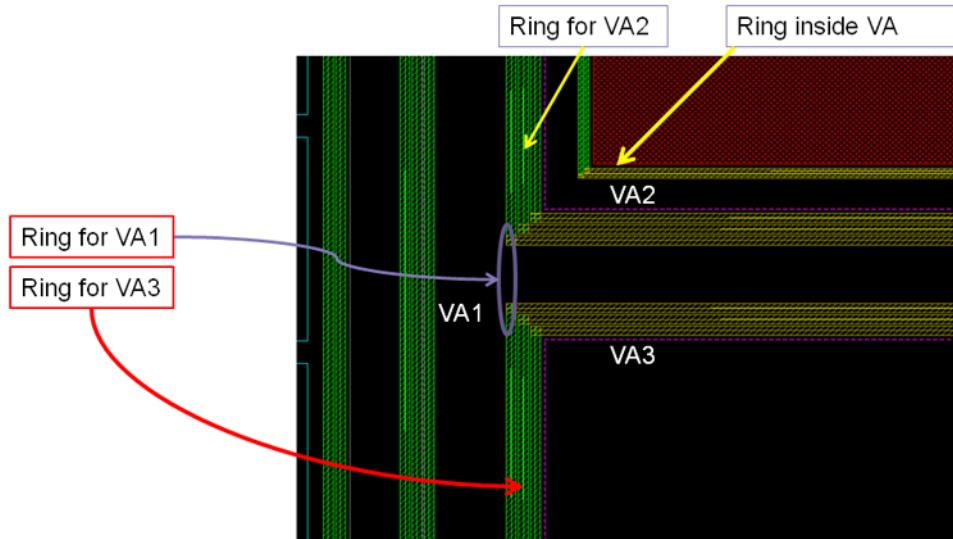
```
icc_shell> set_power_ring_strategy strategy_1 -nets {VDD VSS} \
           -voltage_areas VA1 -template ring.tpl:ring_around_va
icc_shell> set_power_ring_strategy strategy_2 -nets {VDD2 VSS VDD1} \
           -voltage_areas VA2 -template ring.tpl:ring_around_va
icc_shell> set_power_ring_strategy strategy_3 -nets {VDD3 VSS VDD1} \
           -voltage_areas VA3 -template ring.tpl:ring_around_va
```

3. Compile the power rings.

```
icc_shell> compile_power_plan -ring \
           -strategy {strategy_1 strategy_2 strategy_3}
```

Figure 8-15 shows the layout with power rings around three voltage areas.

Figure 8-15 Rings Around Voltage Areas



Aligning Rings With Standard Cell Rails

The following design requires that the power and ground rings align with the standard cell rails. To implement this power plan, do the following steps. Note that this power plan uses parameters for the ring width and ring spacing.

1. Create the ring.tpl template file and include the `align_std_cell_rail` advanced rule setting.

```
template : ring_align_stdcell(w,s) {
    side : horizontal {
        layer : M3
        width : @w
        spacing : @s
        offset : 1.0
    }
    side : vertical {
        layer : M4
        width : @w
        spacing : @s
        offset : 1.0
    }
    advanced_rule : on {
        align_std_cell_rail : on
    }
}
```

2. Set the power ring strategy. The region_1 power plan region is defined outside this procedure.

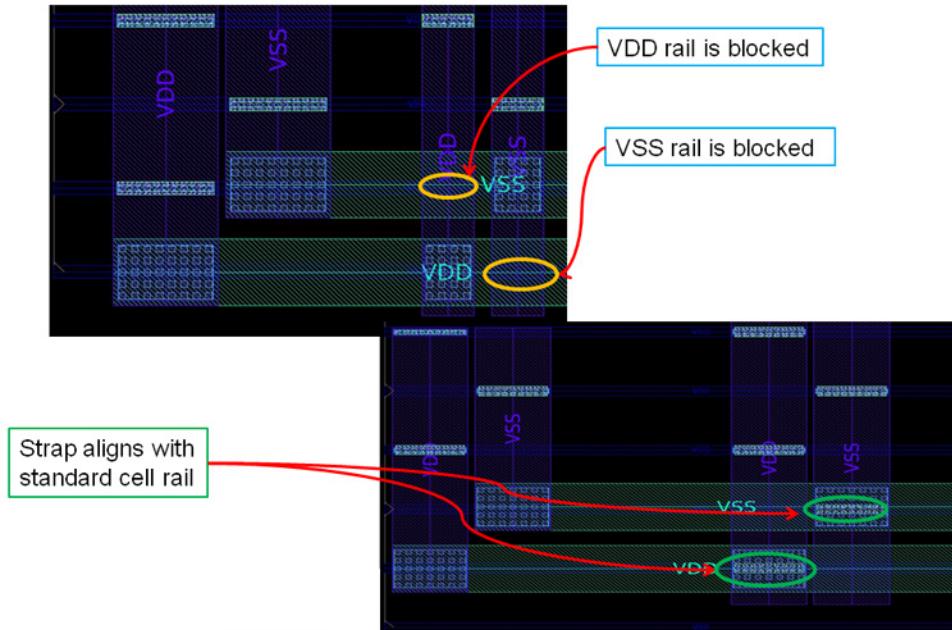
```
icc_shell> set_power_ring_strategy strategy_1 -nets {VDD VSS} \
    -power_plan_regions region_1 \
    -template ring.tpl:ring_align_stdcell(2.0, minimum)
```

3. Compile the power rings.

```
icc_shell> compile_power_plan -ring \
    -strategy strategy_1
```

Figure 8-16 shows the layout with power rings aligned to standard cells.

Figure 8-16 Rings Aligned to Standard Cell Rails



Creating Rings Around a User-Specified Polygon

The following example creates a ring using a user-specified rectilinear pattern. In this example, *coordinates* represents a set of x- and y-coordinates that describe the polygon.

1. Create the ring.tpl ring template file as described in the previous examples.
2. Create the power plan region with the specified coordinates.

```
icc_shell> create_power_plan_regions region_1 \
           -polygon { coordinates }
```

3. Create the power ring strategy.

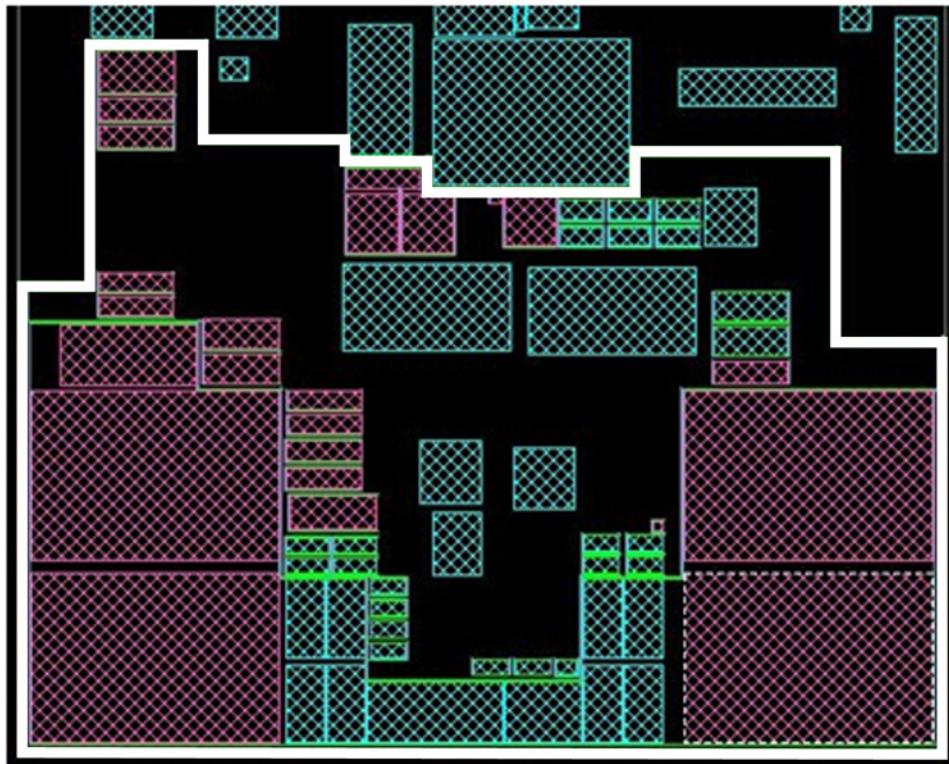
```
icc_shell> set_power_ring_strategy strategy_1 -nets {VDD VSS} \
           -power_plan_regions region_1 -template ring.tpl
```

4. Compile the power rings.

```
icc_shell> compile_power_plan -ring -strategy strategy_1
```

[Figure 8-17](#) shows the layout with a power ring with a shape specified by a complex polygon.

Figure 8-17 Rectilinear Power Ring



Creating Rings Around Macro Cells

The following example creates rings around the Ram1, Ram2, Ram3, and Ram4 macro cells.

1. Create the ring template file as described in the previous examples.
2. Create the power plan strategy.

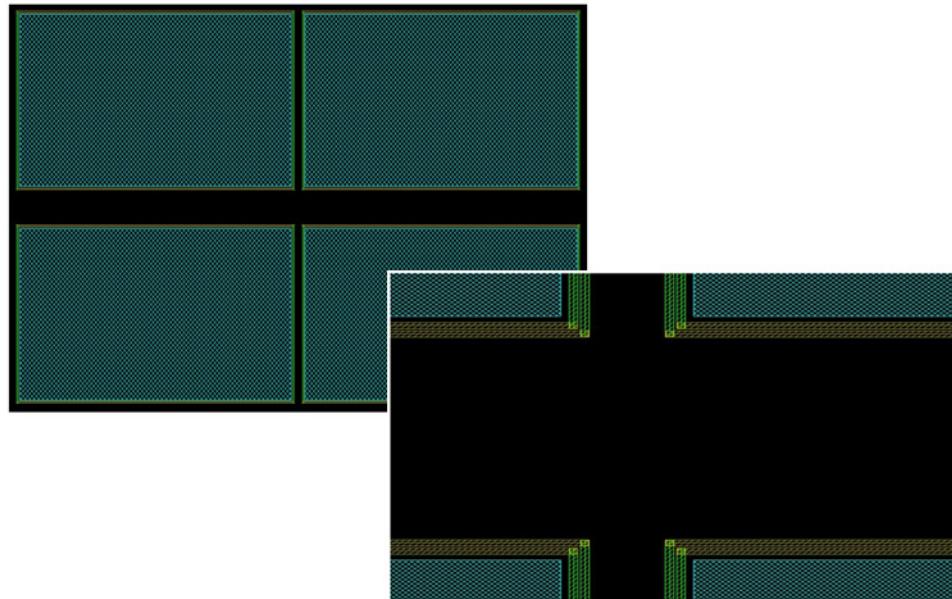
```
icc_shell> set_power_ring_strategy strategy_1 -nets {VDD VSS} \
           -macros {Ram1 Ram2 Ram3 Ram4} -template ring.tpl
```

3. Compile the power rings.

```
icc_shell> compile_power_plan -ring -strategy strategy_1
```

[Figure 8-18](#) shows the layout with power rings around the macro cells.

Figure 8-18 Power Rings Around Macro Cells



Creating Corner Bridging

Template-based power planning can create via bridges at ring corners to improve IR drop. Do the following steps to create power rings with via bridges.

1. Create the ring template file. Note the corner_bridge advanced rule setting.

```
template : via_bridge(w,s) {
    side : horizontal {
        layer : METAL5
        width : @w
        spacing : @s
        offset : 0
    }
    side : vertical {
        layer : METAL6
        width : @w
        spacing : @s
        offset : 0
    }
    advanced_rule : on {
        corner_bridge : on
    }
}
```

2. Create the power plan strategy.

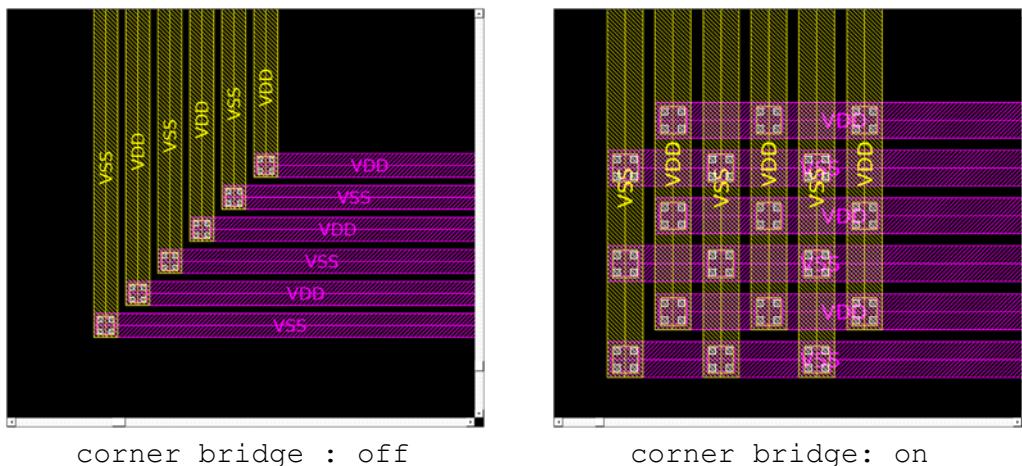
```
icc_shell> set_power_ring_strategy strategy_1 -macros {Macro1} \
-nets {VDD VSS VDD VSS VDD VSS} \
-template via_bridge.tpl:via_bridge(2.0,minimum)
```

3. Compile the power ring.

```
icc_shell> compile_power_plan -ring -strategy strategy_1
```

[Figure 8-19](#) shows a layout with standard corners and with additional via bridges at the corners.

Figure 8-19 Layout Without and With Corner Bridging



Creating Partial Power Rings

The following example creates partial power and ground rings around the specified group of macros. Side numbering begins with side 1 at the leftmost edge at the bottom of the shape. This example excludes sides 1 and 6 from the power ring.

1. Create the template file as shown in the previous examples.
2. Create the power plan region around the group of macros.

```
icc_shell> create_power_plan_regions region_1 \
           -group_of_macros {Ram1 Ram2}
```

3. Create the power ring strategy.

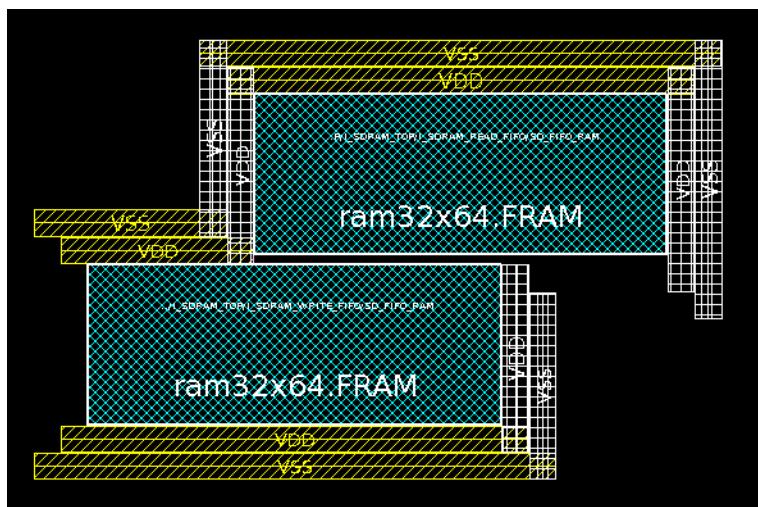
```
icc_shell> set_power_ring_strategy strategy_1 -nets {VDD VSS} \
           -power_plan_regions region1 -template ring.tpl \
           -skip {{nets: VDD VSS} {sides: 1 6}}
```

4. Compile the power rings.

```
icc_shell> compile_power_plan -ring -strategy strategy_1
```

[Figure 8-20](#) shows the layout after you run the previous steps.

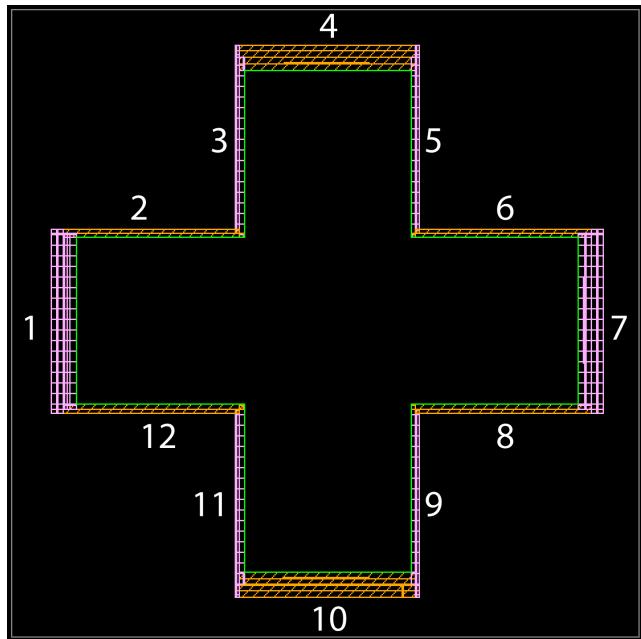
Figure 8-20 Partial Power Ring With Excluded Sides



Creating Power Rings with Varying Segment Widths

The layout in [Figure 8-21](#) contains a cross-shaped power ring with wider power segments on the left, right, top and bottom sides. The power ring is created by defining a power plan region and specifying wider power segments on sides 1, 4, 7, and 10 of the power ring. The sides of the voltage area are labeled for reference.

Figure 8-21 Cross-Shaped Power Ring



To implement the cross-shaped power ring with wide and narrow ring segments,

1. Create the ring.tpl power ring template file and specify wider ring segments on sides 1, 4, 7, and 10.

```
template : wide_narrow_ring(narrow,wide) {
    side : horizontal {
        layer : METAL5
        width : @narrow
        spacing : minimum
        offset : 0
    }
    side : vertical {
        layer : METAL6
        width : @narrow
        spacing : minimum
        offset : 0
    }
}
```

```

    side : {1 7} {
        layer : METAL6
        width : @wide
        spacing : minimum
        offset : 0
    }
    side : {4 10} {
        layer : METAL5
        width : @wide
        spacing : minimum
        offset : 0
    }
}

```

2. Create the power plan region and power ring strategy.

```

icc_shell> create_power_plan_regions region_1 \
-polygon {{900 500} {1300 500} {1300 900} {1700 900} {1700 1300} \
{1300 1300} {1300 1700} {900 1700} {900 1300} {500 1300} \
{500 900} {900 900} {900 500}}

icc_shell> set_power_ring_strategy ring_strategy_1 \
-nets {VDD VSS} -power_plan_regions region_1 \
-template ring.tpl:wide_narrow_ring(2.0,20.0)

```

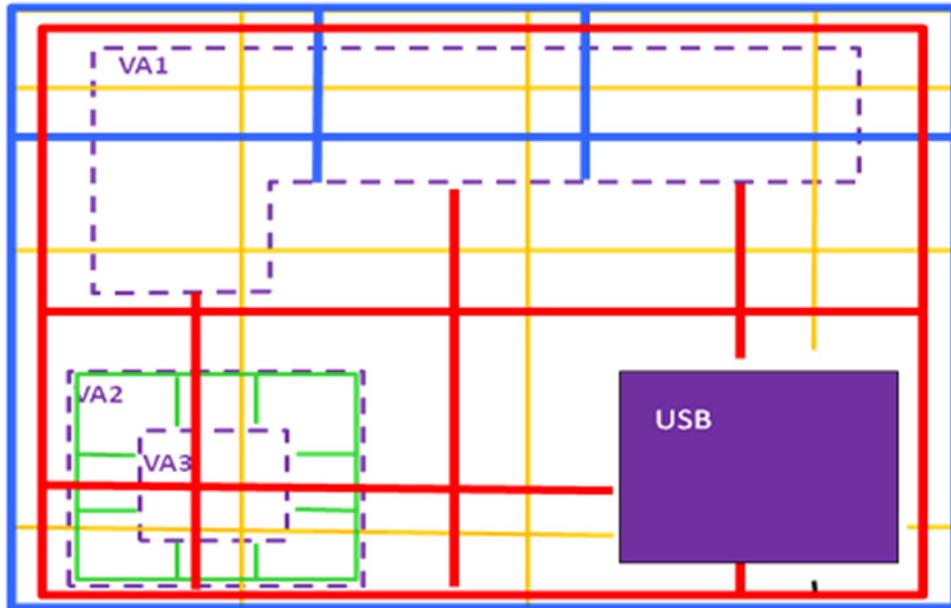
3. Compile the power ring.

```
icc_shell> compile_power_plan -ring -strategy strategy_1
```

Creating a Power Plan Strategy for a Multivoltage Design

[Figure 8-22 on page 8-50](#) represents a multivoltage design that contains four power domains, three voltage areas, and a common ground net VSS. The voltage areas are defined as:

- TOP: Primary power nets VDD and VSS
- VA1: Primary power net VDD1
- VA2: Shutdown voltage area with the primary power net VDD2 and always-on power net VDD
- VA3: Always-on voltage area with the primary power net VDD

Figure 8-22 Multivoltage Design

This design requires three power network strategies:

- VA3 and top shared power network strategy
- VA1 power network strategy
- VA2 power network strategy

To define the strategy, begin by assigning a strategy name. Identify the routing area, net names, extensions, and blockages required for the strategy. In this example, the top-level power and ground net names are VDD and VSS, and the routing area is the core area. The VDD net extends in all directions except the top; the VSS net extends in all directions. The VDD net is blocked by the USB macro cell and the VA1 voltage area. The VSS net is blocked only by the USB macro cell. The strategy definition is as follows:

```
icc_shell> set_power_plan_strategy top_strategy \
    -nets {VDD VSS} -core -blockage { \
        {{nets: "VDD"} {voltage_area: "VA1"}} \
        {macro: USB}} \
    -extension { \
        {{nets: "VDD"} {direction: "LRB"} {stop: outer_ring}} \
        {{nets: "VSS"} {direction: "LRTB"} {stop: outer_ring}}} \
    -template mytemplate.tpl:upper_grid(50,50)
```

The VA1 power network contains one net, VDD1, and uses the VA1 routing area. The VDD1 net extends left, right, and top. The definition for the VA1 strategy is as follows:

```
icc_shell> set_power_plan_strategy s1_strategy \
-nets VDD1 -voltage_areas VA1 \
-extension { \
  {{nets: "VDD1"} {direction: "LRT"} {stop: outer_ring}}} \
-template mytemplate.tpl:upper_grid(20,40)
```

The VA2 power network contains the single VDD2 power net, and uses the VA2 routing area. The VDD2 net does not extend and is blocked by the VA3 voltage area. The definition for the VA2 strategy is as follows:

```
icc_shell> set_power_plan_strategy s2_strategy \
-nets VDD2 -voltage_areas VA2 \
-blockage { \
  {{nets: "VDD2"} {voltage_area: "VA3"}}} \
-template mytemplate.tpl:lower_grid(20,20)
```

Note that each strategy references a unique template block defined within the mytemplate.tpl template file. Using the same template file, you can create a unique power and ground strap configuration for each voltage area by referencing different template blocks and specifying unique template parameters. The template file syntax is discussed in the following section.

Creating Power Straps on Macro Cells

The design shown in [Figure 8-23](#) requires horizontal power straps on all macro cells that have vertical power and ground straps. The number of straps is controlled by the cell height; if the cell height is small than 120 microns, IC Compiler creates one pair of VDD and VSS straps. Otherwise, IC Compiler creates two pairs.

Figure 8-23 Power Straps on Macro Cells



To implement this power plan, create two strategies: one for cells with a height of less than 150 um and one for cells with a height of more than 150 um. Use the following four steps to create these straps:

1. Find all the macro cells with vertical pins, and divide them into two groups.

```
icc_shell> set vertical_pin_macros \
    [filter_collection [all_macro_cells] "(orientation == S) || \
    (orientation == FS) || (orientation == N) || \
    (orientation == FN)"]
icc_shell> set cell100 [get_cells $vertical_pin_macros \
    -filter height<150]
icc_shell> set cell200 [get_cells $vertical_pin_macros \
    -filter height>150]
```

2. Create the 65nm.tpl template file with a parameter for the number of straps.

```
template : macro_strap(num_straps) {
    layer : M5 {
        direction : horizontal
        width : 3.0
        spacing : minimum
        number : @num_straps
        pitch :
        offset_start : boundary
        offset :
        align :
        trim_strap : true
        advanced_rule : off
    }
}
```

3. Set the power plan strategies.

```
icc_shell> set_power_plan_strategy macro_100 \
    -nets "VDDC VSSC" -macros $cell100 \
    -template 65nm.tpl:macro_strap(1)
icc_shell> set_power_plan_strategy macro_200 \
    -nets "VDDC VSSC" -macros $cell200 \
    -template 65nm.tpl:macro_strap(2)
```

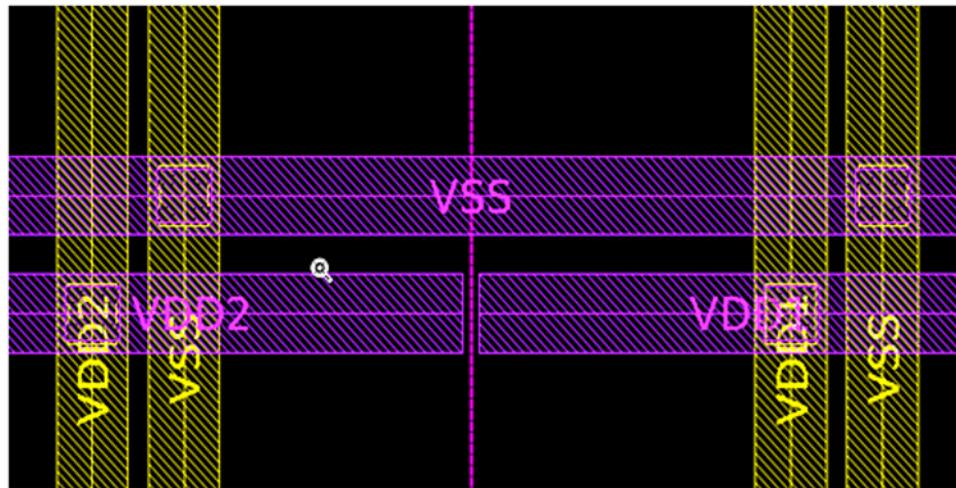
4. Compile the power plan.

```
icc_shell> compile_power_plan -strategy {macro_100 macro_200}
```

Creating a Power Grid in an Abutted Voltage Area

In the design shown in [Figure 8-24 on page 8-53](#), the voltage areas abut and the ground net VSS is shared across voltage area boundaries. However, the power nets VDD1 and VDD2 do not connect and stay on their respective sides; the nets are powered by different supplies. Power nets VDD1 and VDD2 should be spaced by half the minimum spacing at the voltage area borders to avoid a spacing rule violation.

Figure 8-24 Abutted Voltage Areas



The following steps create the power plan for this design:

1. Create the mytemplate.tpl template file.

```
template : va_strap {  
    layer : METAL6 {  
        direction : vertical  
        width : 10  
        spacing : minimum  
        number :  
        pitch : 82  
        offset_start : boundary  
        offset : 7  
        trim_strap : true  
    }  
    layer : METAL7 {  
        direction : horizontal  
        width : 10  
        spacing : minimum  
        number :  
        pitch : 92  
        offset_start : boundary  
        offset : 46  
        trim_strap : true  
    }  
}
```

2. Create a strategy for each voltage area.

```
icc_shell> set_power_plan_strategy s1 -nets {VDD1 VSS} \
-voltage_areas VA1 \
-extension {{nets: VDD1} {direction: L} {stop: -half_space}} \
-template mytemplate.tpl:va_strap
icc_shell> set_power_plan_strategy s2 -nets {VDD2 VSS} \
-voltage_areas VA2 \
-extension {{nets: VDD2} {direction: R} {stop: -half_space}} \
-template mytemplate.tpl:va_strap
```

Note that the expression `{stop: -half_space}` defines a stopping point for the power strap that is one half the minimum spacing distance within the voltage area. If you specify `{stop: half_space}` without the minus sign, IC Compiler extends the strap to one half the minimum spacing beyond the voltage area boundary. The minimum spacing is defined in the technology file.

3. Compile the power plan.

```
icc_shell> compile_power_plan -strategy {s1 s2}
```

Creating Extension Finger Connections

The design shown in [Figure 8-25](#) contains macro cells with an internal power and ground ring. To implement this power network, begin by creating a power and ground ring around the macro cell. The `preroute_instances` command creates only one connection per top, bottom, left, and right side for the VDD net and the VSS net. The rings in the macro cell must connect to the chip-level power and ground ring by using extension fingers.

Figure 8-25 Power Plan

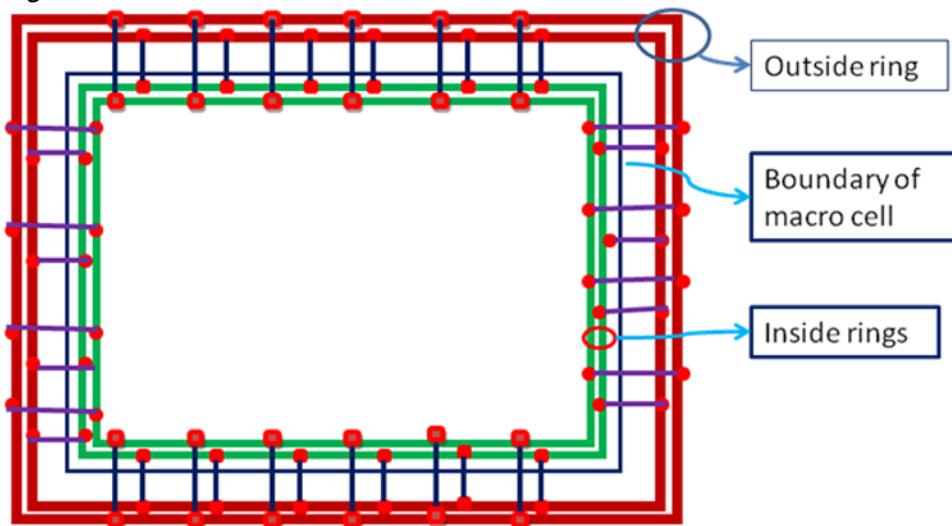
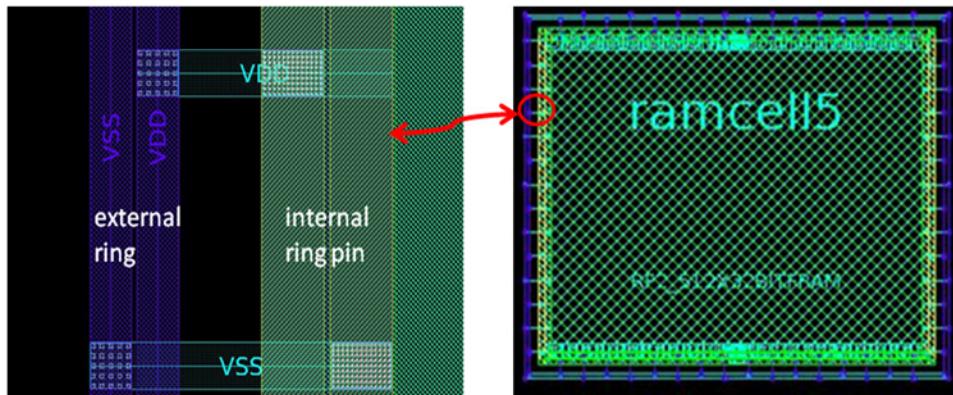


Figure 8-26 External Power Ring

To implement this power plan, define the routing area as the macro cell boundary and create a reduced power plan region for each of the macro cells based on the macro cell boundary. You can extend the power straps in each direction and limit the extension to `first_target`. The following steps create the template-based power plan for this design.

1. Create the `mytemplate.tpl` template file.

```
template : top_two {
    layer : METAL7 {
        direction : horizontal
        width : 1.5
        spacing : interleaving
        number : 10
        pitch :
        offset_start : boundary
        offset :
        trim_strap : true
    }
    layer : METAL8 {
        direction : vertical
        width : 2.0
        spacing : interleaving
        number : 10
        pitch :
        offset_start : boundary
        offset :
        trim_strap : true
    }
}
```

2. Define the power plan region and power plan strategy.

Using the macro as the routing area, create a blockage based on the internal area of macro and extend the power straps to the outer ring.

```
icc_shell> create_power_plan_regions r_in \
    -group_of_macros ramcell15 -expand -12.8
icc_shell> set_power_plan_strategy s_macro \
    -macros ramcell15 -nets {VDD VSS} \
    -blockage {power_plan_region: r_in} \
    -extension {stop: outer_ring} \
    -template mytemplate.tpl:top_two
```

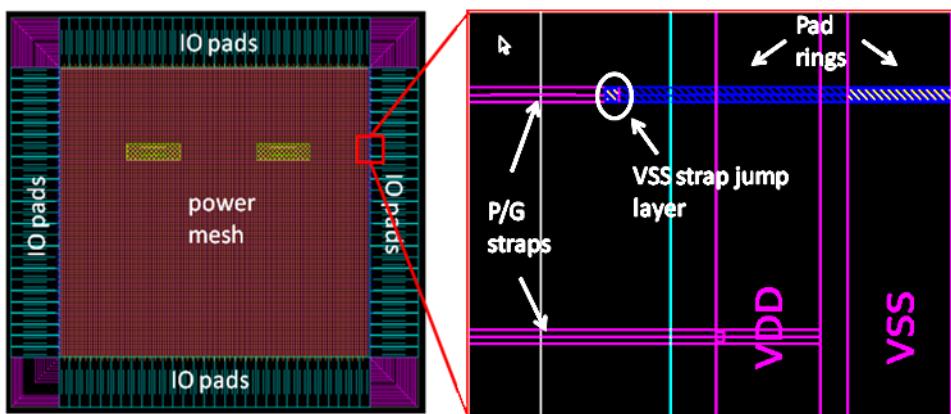
3. Compile the power plan.

```
icc_shell> compile_power_plan -strategy s_macro
```

Creating Pad Ring Connections

The design shown in [Figure 8-27](#) connects power straps to pad rings.

Figure 8-27 I/O Pad Ring



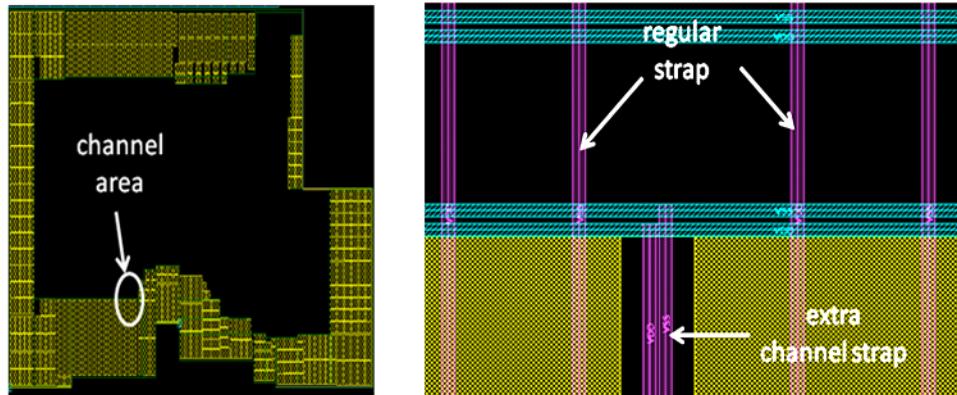
To implement this power plan, define the power plan strategy and compile the power plan as follows:

```
icc_shell> set_power_plan_strategy s_top -core \
    -nets {VDD VSS} -extension {stop: pad_ring}
icc_shell> compile_power_plan -strategy s_top
```

The `-extension {stop: pad_ring}` option extends the VDD and VSS power straps to the pad ring and connects them. When you specify this option, power straps that extend to the pad or core ring can use an adjacent layer for routing to avoid a power or ground strap that is blocking the extension. This applies only when the straps extend outside the core to connect to a core or pad ring.

Inserting Straps into the Channel Area

The design shown in [Figure 8-28 on page 8-57](#) uses power straps within the channel area.

Figure 8-28 Channel Straps

During the design process, channel areas might form between adjacent macros; by default, straps are not inserted into channels if the strap pitch does not place them there. You can change this behavior by using advanced rule support.

To create straps in the channel area, set `advanced_rule` to `on` and set `insert_channel_straps` to `on` in the advanced rules section. Define the layers, strap width, spacing, and minimum channel threshold for the strap. The following template skeleton enables channel straps that are routed using layer M5 with a strap width of 1.89 microns for channels that are 10 or more microns wide.

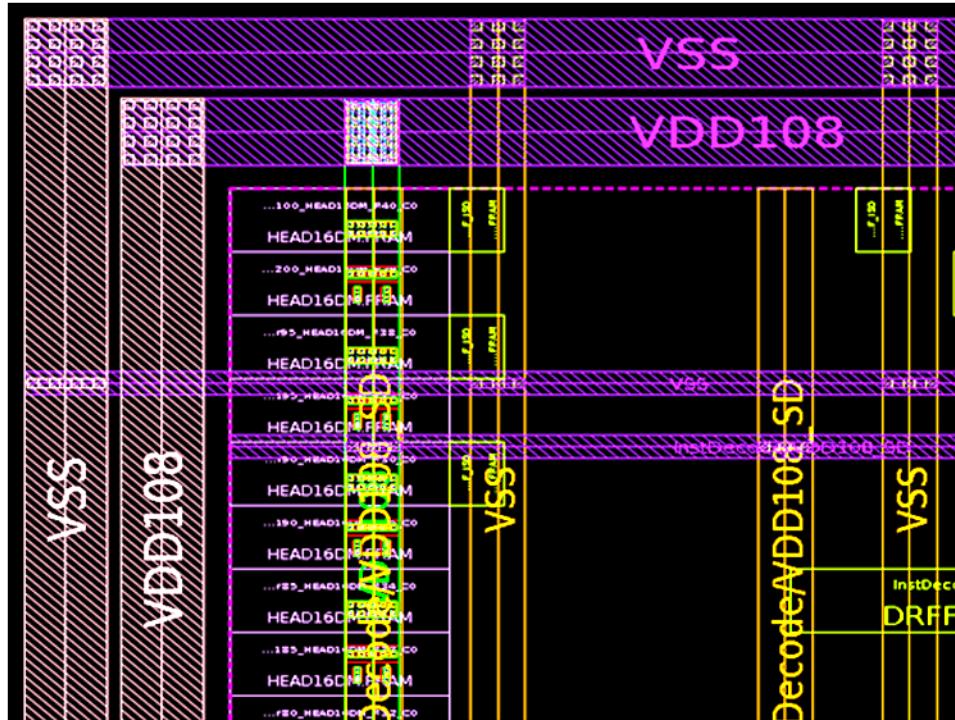
```
template : channel_strap {
    layer : M5 {
        direction : vertical
        # additional directives not shown
    }
    layer : M6 {
        direction : horizontal
        # additional directives not shown
    }

    advanced_rule : on {
        # additional rules not shown
        insert_channel_straps : on {
            layer : {M5}
            width : 1.89
            spacing : minimum
            channel_threshold : 10
        }
    }
}
```

Align Power Straps with Power Switches

The design shown in [Figure 8-29](#) aligns power straps with power switches.

Figure 8-29 Power Straps and Switches



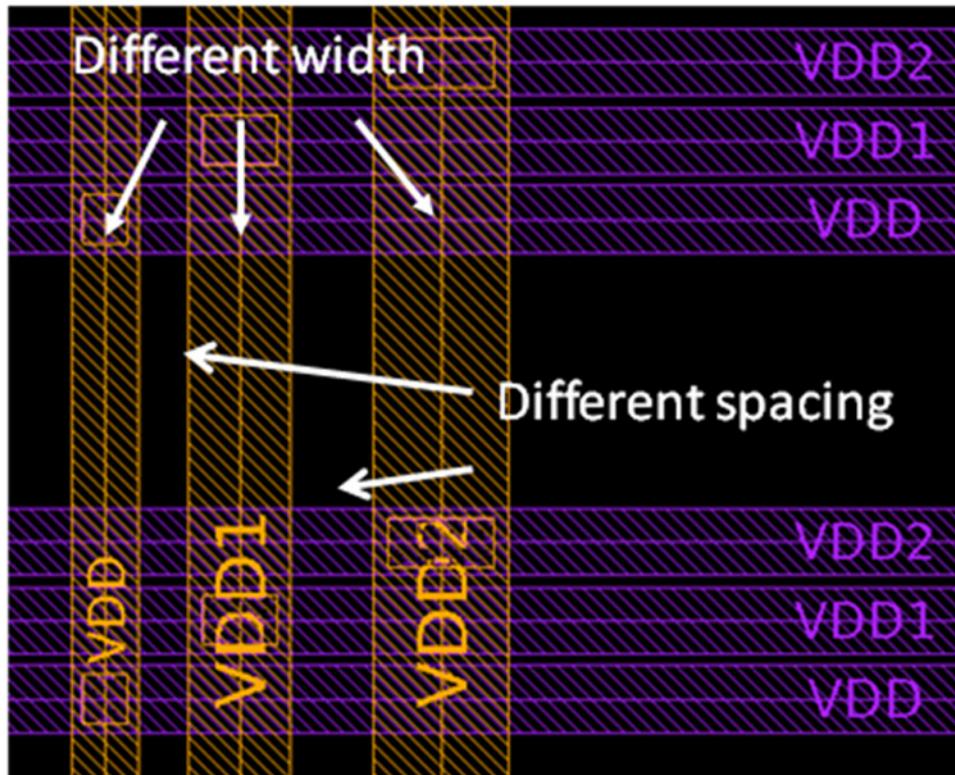
You can align power straps with power switches by setting the `align_straps_with_power_switch` directive in the `advanced_rule` section to `on`. Specify the power-switch name, layer, strap width, and direction for the power strap. The following template skeleton aligns the power straps on layer ME6 with the power switches in the vertical direction.

```
template : align_strap {
    layer : ME7 {
        direction : horizontal
        width : 1.00000
        spacing : minimum
        number : 6
        pitch :
        offset_start : boundary
        offset :
        trim_strap : true
    }
    layer : ME6 {
        direction : vertical
        width : 2.00000
        spacing : minimum
        number : 12
        pitch :
        offset_start : boundary
        offset :
        trim_strap : true
    }
    advanced_rule : on {
        align_straps_with_power_switch : on {
            power_switch : HEAD16DM
            layer : ME6 #strap layer
            width : 2.0 #strap width
            direction : vertical
        }
    }
}
```

Nonuniform Power Straps

The layout shown in [Figure 8-30](#) uses a nonuniform power grid.

Figure 8-30 Nonuniform Power straps



To implement this power plan, create multiple power straps with unique individual spacing and width values. Use a parameterized template and apply different values to the parameters with the `set_power_plan_strategy` command.

To implement this power plan, do the following steps:

1. Create the mytemplate.tpl template file.

```
template : nonuniform(p1,p2,p3,p4,p5) {
    layer : METAL7 {
        direction : horizontal
        width : @p1
        spacing : minimum
        number : 36
        pitch :
        offset_start : boundary
        offset :
        trim_strap : true
    }
    layer : METAL8 {
        direction : vertical
        width : {@p1 @p2 @p3}
        spacing : {@p4 @p5}
        number : 12
        pitch :
        offset_start : boundary
        offset :
        trim_strap : true
    }
}
```

2. Set the power plan strategy by using parameters to represent power strap widths.

```
icc_shell> set_power_plan_strategy s_top \
-nets {VDD VDD1 VDD2} -core \
-template mytemplate.tpl:nonuniform(1.0,2.0,3.0,0.5,1.5)
```

3. Compile the power plan.

```
icc_shell> compile_power_plan -strategy s_top
```

Creating Power-Switch Arrays and Rings for Multithreshold-CMOS Designs

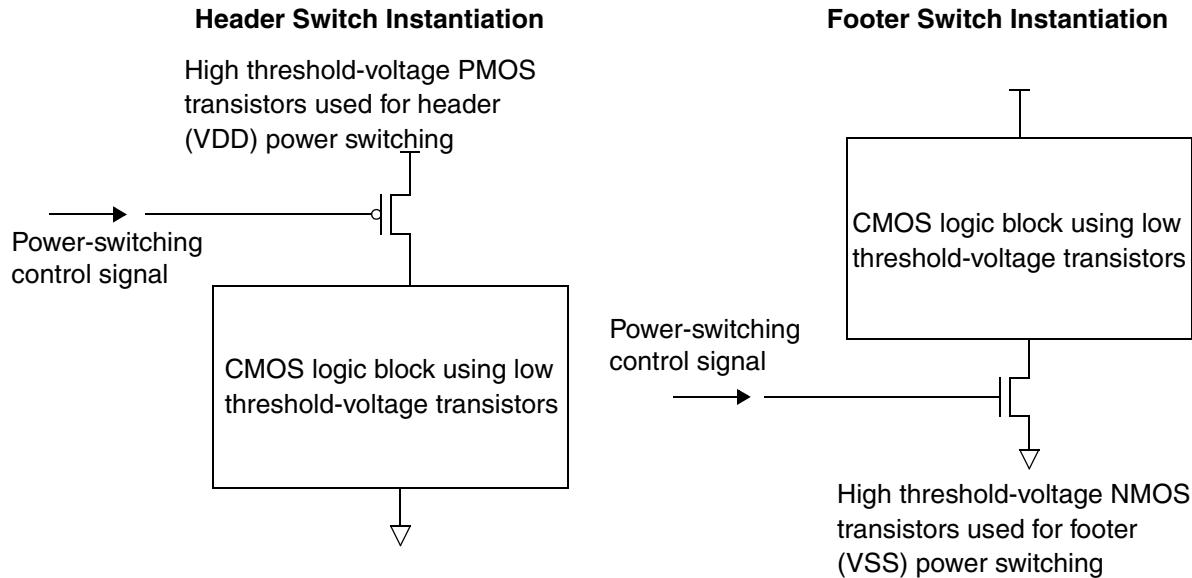
As process technology continues to scale to smaller and smaller geometries, the leakage current of CMOS devices increases exponentially. An effective method of controlling leakage power is to use multithreshold-CMOS technology with shutdown transistors. The following sections illustrate the use of multithreshold-CMOS and power gating to reduce leakage power in CMOS designs. The following sections describe the implementation of a power-switch network in a multivoltage design.

- [Power Switch Overview](#)
 - [Selecting the Proper Power-Switch Strategy](#)
 - [Creating a Power-Switch Array](#)
 - [Creating a Power-Switch Ring](#)
 - [Exploring Different Switch Configurations](#)
 - [Optimizing Power Switches](#)
 - [Connecting Power Switches](#)
-

Power Switch Overview

You can implement power switching in your design during floorplanning to reduce static leakage power to idle circuit blocks. Power switching is a technique that reduces leakage power by cutting the flow of current between VDD and VSS while the circuit block is idle. There are two types of power switches: header switches and footer switches. Header switches with PMOS transistors are placed between VDD and the block power supply pins; footer switches with NMOS transistors are placed between VSS and the block ground pins. In many designs, only one type of switch is used. Switch cells are encapsulated within standard cells.

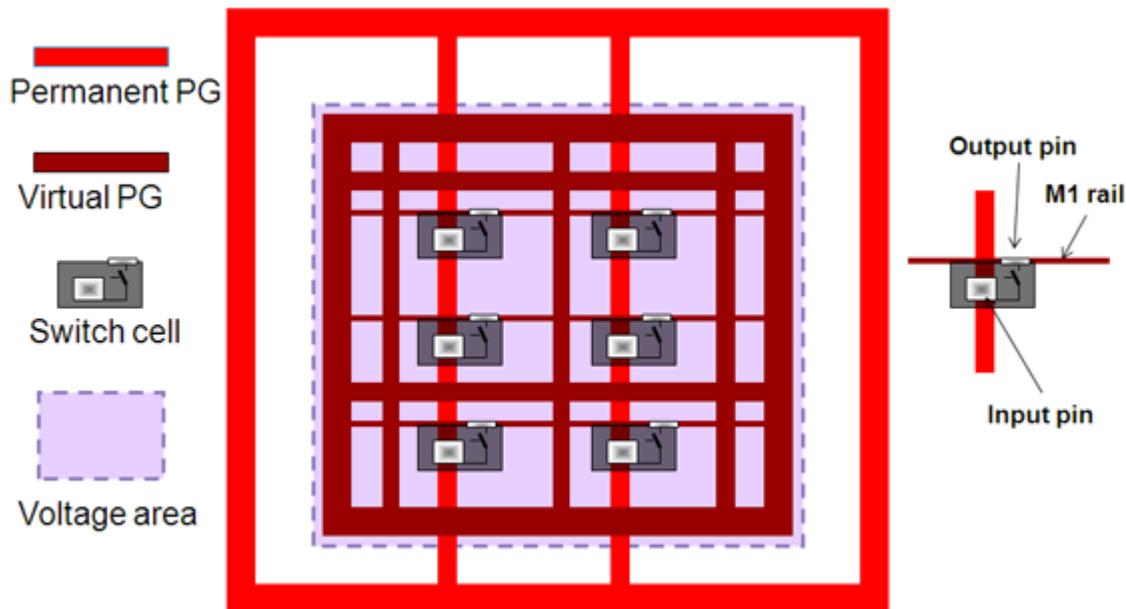
Figure 8-31 Power-Switching Network Transistors



The switching strategy shown in [Figure 8-31](#) is a coarse-grain strategy; this strategy applies power switching to the entire block through a single control. Multiple transistors in parallel drive a common supply net for the block. In a fine-grain strategy, each library cell has its own power switch, allowing fine-grain control over which cells are powered down. The fine-grain approach has better potential for power savings, but requires more area.

[Figure 8-32 on page 8-64](#) shows the layout view of the power-switch array used to shut down power to a circuit block. The power-switch network controls the current flow from the permanent power network to the virtual power network. Permanent power is a constant supply that originates from the top-level circuit pads and virtual power is switched by the power switch.

Figure 8-32 Power Switch and Power Rails



Selecting the Proper Power-Switch Strategy

Although power gating uses relatively large power-switch cells, the actual current available to the switched power network might be less than that available on the permanent power supply rails. The current reduction depends on the dynamic power requirements of the shutdown block, the size of the voltage area, and the number of power-switch cells. The physical placement of the power-switch cells is critical to achieve an acceptable IR distribution within the shutdown block.

Two primary placement styles are used when placing power-switch cells: array-style placement and ring-style placement. In the array-style approach, power-switch cells are placed in a uniform grid within the voltage area. Array-style placement provides high drive current and minimizes resistance, which in turn impacts IR drop. The benefits of an array-style placement include

- Greater control over IR drop distribution
- Better optimization that allows for fewer gating cells than the ring-style method

The shortcomings of the array-style approach include

- Less available cell placement area

- More difficult or impossible implementation with hard macros, as the existing placement must be modified to accommodate power gates inside the voltage area
- Potentially more congestion or increase in area
- Potentially greater power grid complexity

In a ring-style approach, IC Compiler places power-switch cells in a ring that surrounds the shutdown voltage area. The benefits of a ring-style placement include:

- Easier power grid creation for the block
- Easier application to designs that contain hard macros, without the need for modification inside the block. No extra cells or routing is required, with the exception of support for retention, isolation, level shifting, and always-on logic.
- Easier connection of power and daisy chains by using abutment

A major limitation of the ring-style approach is that the IR drop impact might be unacceptable toward the center of the voltage area as the distance increases from the power-switch cells. The severity of the IR drop depends on the voltage area size, shape, and internal dynamic power requirements of the block. For some designs, it might be impossible to implement a ring-style approach without violating IR drop and inrush current requirements.

Creating a Power-Switch Array

You can use the `create_power_switch_array` command to add power switches in a uniform grid array inside the voltage area. After inserting the power switches, the command legalizes the power switches and locks their position by setting the `is_fixed` attribute on each power switch. [Table 8-5](#) describes the `create_power_switch_array` command options. For more information about these options, see the man page.

Table 8-5 create_power_switch_array Command Options

Command option	Description
<code>-lib_cell cellname</code>	Specifies the library cell or power-switch cell to be used.
<code>-bounding_box {left bottom right top}</code>	Specifies the coordinates for the rectangle in which to create the power-switch cell array.
<code>-design hierarchy_name</code>	Specifies the hierarchy level name in which to create the power-switch cell array.
<code>-offset_to_va offsets</code>	Specifies the offset to maintain between the voltage area boundary and the power-switch array.

Table 8-5 create_power_switch_array Command Options (Continued)

Command option	Description
-orientation N W S E FN FE FS FW	Specifies the placement orientation of the switch cells.
-pattern normal staggered	Specifies the instantiation pattern for the switch cells.
-place_pattern <i>pattern_syntax</i>	Specifies the insertion pattern for cells in the array.
-prefix <i>prefix_name</i>	Specifies a name prefix for the inserted switch cells.
-relative_to_voltage_area	Sets the origin of the bounding box coordinates at the lower-left corner of the voltage area.
-respect <i>object_type</i>	Specifies object types that cannot have power-switch cells overlapping the object.
-snap_to_row_and_tile	Snaps the standard cell placement to row and tile. IC Compiler ignores the -orientation option when you specify the -snap_to_row_and_tile option.
-start_column <i>index</i>	Specifies the column index used when creating the instance path name. Use the -start_row <i>index</i> option to specify the row index.
-voltage_area <i>voltage_area</i>	Specifies the voltage area in which to place the switch cells.
-x_increment <i>deltax</i>	Specifies the incremental step in the x-direction for power-switch cell placement. Use the -y_increment <i>deltay</i> option to specify the y-direction incremental step.

The following example creates a power-switch array by using the `create_power_switch_array` command. The command instantiates the `POWER SWITCH1` cell in a staggered pattern within the `V_DOMAIN1` voltage area with a spacing between cells of 120 microns in the y-direction and 120 microns in the x-direction. The command attaches the `V_DOMAIN1_` prefix to each inserted power-switch cell.

```
icc_shell> create_power_switch_array -lib_cell POWER SWITCH1 \
    -voltage_area V_DOMAIN1 -y_increment 120 -x_increment 120 \
    -prefix V_DOMAIN1_ -pattern staggered
```

When you map more than one power-switch cell by using the `map_power_switch` command, you can reference a particular power-switch cell by using the `switch_name:lib_cell_name` syntax with the `-switch_lib_cell` option of the `create_power_switch_ring` command. Note that you can also use this syntax with the `-place_pattern` option to refer to specific power-switch cells.

The following example maps two power-switch cells to the SW switch name and places the cell_2 cell in the power-switch ring.

```
icc_shell> map_power_switch SW -domain VA -lib_cells {cell_1 cell_2}
icc_shell> create_power_switch_array -bounding_box {110 200 250 400} \
    -lib_cell SW:cell_2
```

Creating a Power-Switch Ring

The `create_power_switch_ring` command creates a full or partial ring of switch cells around the voltage areas or macro cells in your design. You can use command options to control the cell density of the switch cell ring, the type of filler cells that are inserted between the power-switch cells, the startpoint, the orientation, and other settings. The command places only the power-switch cells, it does not connect the power, ground, and control pins. The command does not honor the blockages or existing cells. However, IC Compiler issues a warning message if the inserted cells create conflicts. [Table 8-6](#) describes the `create_power_switch_ring` command options. For more information about these options, see the man page.

Table 8-6 create_power_switch_ring Command Options

Command option	Description
<code>-switch_lib_cell cellname</code>	Specifies the library cell to be used as the switch cell.
<code>-area_obj name</code>	Specifies the voltage area or macro around which to create the ring of switch cells.
<code>-check_overlap</code>	Checks whether the inserted cells overlap with other objects.
<code>-continue_pattern</code>	Continues the pattern on adjacent sides.
<code>-density density</code>	Specifies the cell density of the power-switch ring.
<code>-design hierarchy_name</code>	Specifies the hierarchy level name in which to create the power-switch cell array.
<code>-end_point {point}</code>	Specifies the endpoint of the ring.

Table 8-6 create_power_switch_ring Command Options (Continued)

Command option	Description
<code>-filler_lib_cell <i>cell_names</i></code>	Specifies the ordered list of filler cell names.
<code>-inner_corner_lib_cell <i>cell_name</i></code>	Specifies the cell to be placed at the inner corners of the ring. Use the <code>-outer_corner_lib_cell</code> option to specify the cell for the outer corners of the ring.
<code>-inner_corner_orientation N W S E FN FE FS FW</code>	Specifies the orientation of the cells placed at the inner corners of the ring. Use the <code>-outer_corner_orientation</code> option to specify the cell orientation for the outer corners.
<code>-offset <i>offset_values</i></code>	Shrinks or enlarges the boundary of the ring with respect to the voltage area or macro boundary by the specified amount.
<code>-place_pattern <i>pattern_syntax</i></code>	Specifies the insertion pattern for cells on the ring.
<code>-polygon { <i>points</i> }</code>	Specifies a rectilinear polygon around which to insert the power-switch cells.
<code>-prefix <i>prefix_string</i></code>	Specifies the name prefix for the cells inserted by this command.
<code>-respect hard_blockage soft_blockage blockage macro fixed_cells all</code>	Avoids the specified objects during switch-cell placement. You can specify more than one object type.
<code>-same_orientation</code>	Places all power switches in the same orientation.
<code>-snap_to_row_and_tile</code>	Places power-switch cells only in legal locations. You must also specify the <code>-same_orientation</code> option when you use the <code>-snap_to_row_and_tile</code> option. IC Compiler ignores the <code>-switch_orientation</code> option when you specify the <code>-snap_to_row_and_tile</code> option.
<code>-start_point {<i>point</i>}</code>	Specifies the startpoint of the ring.
<code>-switch_num <i>count</i></code>	Specifies the number of power-switch cells to insert into the ring.
<code>-switch_orientation N W S E FN FE FS FW</code>	Specifies the orientation of the switch cells placed on the bottom edge of the ring.

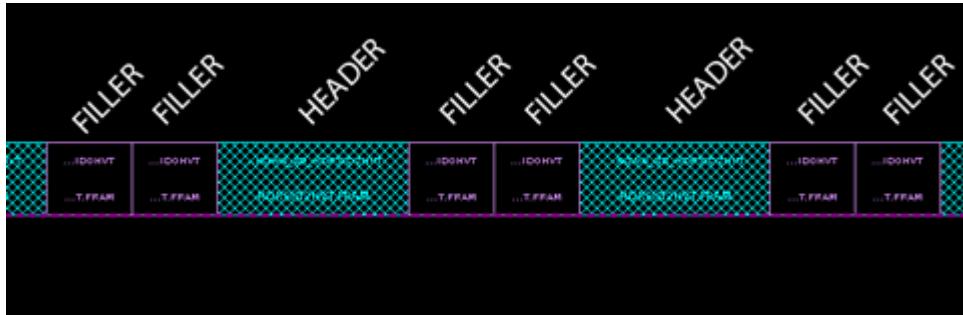
Table 8-6 create_power_switch_ring Command Options (Continued)

Command option	Description
<code>-x_increment delta_x</code>	Specifies the placement pitch in the x-direction in microns.
<code>-y_increment delta_y</code>	Specifies the placement pitch in the y-direction in microns.

You can create a power-switch ring with each power-switch cell in the same orientation by specifying the `-same_orientation` option. Use the `-x_increment` and `-y_increment` options to control the spacing of the power switches in the ring. Use the `-snap_to_row_and_tile` option to force the power switches to legal locations; IC Compiler uses the same orientation for each cell in the ring when you specify this option.

The following example creates a power-switch ring around the DOMAIN1 power domain. The command uses the SWITCH_HEADER power-switch cell and creates a ring pattern using the FILLER and HEADER library cells. [Figure 8-33](#) shows the ring created by using the command in the example.

```
icc_shell> create_power_switch_ring -area_obj DOMAIN1 \
    -switch_lib_cell SWITCH_HEADER \
    -place_pattern { { FILLER HEADER FILLER } * }
```

Figure 8-33 Power-Switch Ring

When you specify more than one power-switch cell by using the `map_power_switch` command, you can reference a particular cell by using the `switch_name:lib_cell_name` syntax with the `-switch_lib_cell` option to the `create_power_switch_ring` command. The following example maps two power-switch cells to the SW switch name and places the cell_2 cell in the power-switch ring.

```
icc_shell> map_power_switch SW -domain VA -lib_cells {cell_1 cell_2}
icc_shell> create_power_switch_ring -area_obj INST \
    -switch_lib_cell SW:cell_2
```

Exploring Different Switch Configurations

There are many possible configurations you can select when creating a power-switch array network; the challenge is to determine the optimal configuration. You can use the `explore_power_switch` command to estimate the IR drop of various power-switch array configurations. The `explore_power_switch` command invokes both power network synthesis and power network analysis and requires power network synthesis constraints set by using the `set_fp_rail_constraints` command.

To perform power-switch exploration, provide the power-switch library cells for insertion, the switch array x- and y-pitches, and the permanent and virtual supply names. The `explore_power_switch` command automatically loops through the configurations you provide and performs the following steps:

- Inserts header or footer arrays based on your specification
- Legalizes placement after power-switch insertion
- Logically connects permanent and virtual PG nets to power switches
- Performs multivoltage power network synthesis and creates a virtual power and ground grid for all voltage areas
- Connects power switches automatically to the global supply
- Preroutes standard cells if the `-create_virtual_rails` option is not specified
- Performs power network analysis on the current configuration by using the `analyze_fp_rail -nets {vdd+vdd_local}` command

The `explore_power_switch` command creates a report containing a sorted list of maximum IR drop values for all configurations. Based on the results of the command, you can select the appropriate power-switch configuration to achieve the desired IR drop for your design.

The following command performs power-switch cell exploration on the current design. The command explores the IR drop in the VA1 voltage area by instantiating the FOOTER8, FOOTER16, and FOOTER32 power-switch cells using the x-spacing values of 25, 50, and 100 microns. The VIRTUAL_VSS virtual ground net and VSS real ground net are used in the analysis. See the man page for a complete list of command options.

```
icc_shell> explore_power_switch \
    -lib_cells {"FOOTER8" "FOOTER16" "FOOTER32"} \
    -x_increment {25 50 100} -voltage_area VA1 \
    -virtual_pg_net VIRTUAL_VSS -real_pg_net VSS
```

Optimizing Power Switches

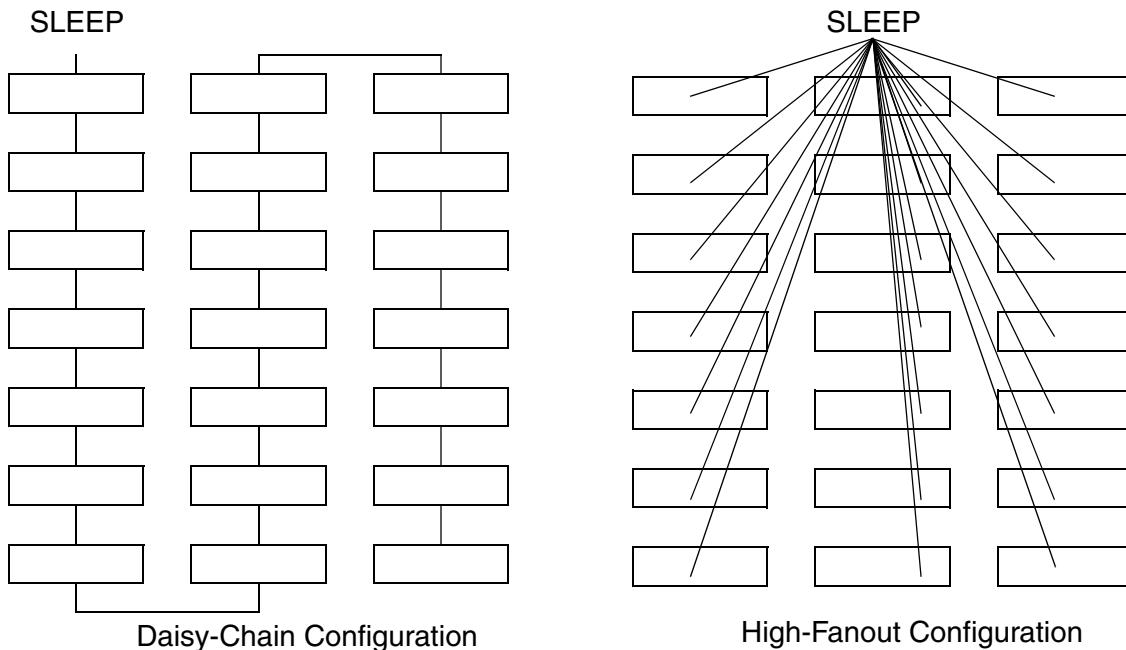
After you insert power switches into a design, you can optimize the power switches by using the `optimize_power_switch` command. The command sizes the power-switch cells to attempt to reduce the IR drop to the target level set by using the `set_mtcmos_pna_strategy` command. The following example uses the `optimize_power_switch` command to optimize the power-switch network to achieve a target IR drop of 120 mV.

```
icc_shell> set_mtcmos_pna_strategy -target_voltage_drop 120
icc_shell> optimize_power_switch -virtual_pg_net VIRTUAL_VDD \
    -real_pg_net VDD
```

Connecting Power Switches

The control pins of the cells within the power-switch cell network must be connected to a master sleep control signal. There are two connection styles: high fanout and daisy chain. The high-fanout connection style uses a single net and buffers to simultaneously power up the switch-cell network. The daisy-chain connection style enables the switch cells in a sequential fashion and results in a lower inrush current. [Figure 8-34](#) shows a schematic view of these two connection schemes.

Figure 8-34 Daisy-Chain and High-Fanout Connection Styles



To connect the sleep control pins on the power-switch cells in daisy-chain mode, use the `connect_power_switch -mode daisy` command. In the following example, the `connect_power_switch` command connects the sleep control pins by using a daisy chain.

```
icc_shell> connect_power_switch -source SLEEP -port_name sleep_port \
    -mode daisy -direction horizontal -verbose -voltage_area { VA1 VA2 }
```

Analyzing the Power Network

You can perform power network analysis on your design to predict the IR drop of different floorplan and power routing configurations. Power network analysis extracts and analyzes the power rail network connected to the net names you specify. You can analyze IR drop and electromigration effects during initial power grid planning while the floorplan is incomplete and the power ports of the standard cells are not yet connected. After detailed placement and power rail routing, you can verify that the size and quantity of the power nets are adequate to power the design. The following section describes the steps to perform power network analysis on your design.

- [Performing Power Network Analysis](#)
- [Creating Connectivity Views](#)
- [Performing Power Analysis With Switching Activity Information](#)

Performing Power Network Analysis

To perform power network analysis on your design, use the `analyze_fp_rail` command or choose Preroute > Analyze Power Network in the GUI. [Table 8-7](#) describes the command options for the `analyze_fp_rail` command.

Table 8-7 analyze_fp_rail Command Options

Command option	Description
<code>-analyze_power</code> ("Analyze power" check box in the GUI)	Runs the power analysis engine on the current design.
<code>-create_virtual_rails_layer</code> ("Create virtual rails" check box in the GUI)	Creates virtual power straps for the standard cell pin connections.

Table 8-7 analyze_fp_rail Command Options (Continued)

Command option	Description
-ignore_blockages ("Ignore blockages" check box in the GUI)	Ignores blockages for virtual pads.
-ignore_conn_view_layers <i>layer</i> ("Ignore Conn view layers" check box in the GUI)	Ignores the specified metal layers in the Conn view.
-nets <i>net_names</i> ("Power Ground nets" text box in the GUI)	Specifies the names of the power or ground nets on which to perform power network analysis.
-output_directory <i>directory_name</i> ("Output" text box in the GUI)	Specifies the directory name in which to save the analysis results file.
-pad_masters <i>master_nets</i> ("Specified pad masters" check box in the GUI)	Specifies the power net name and associated power pad.
-power_budget <i>power</i> ("Power budget" text box in the GUI)	Specifies the total power budget.
-read_default_power_file <i>file_name</i> ("Default" option in the GUI)	Specifies the file name that contains power information in default format.
-read_pad_instance_file <i>file_name</i> ("Instances" option in the GUI)	Specifies the file name that contains pad instance names and optional net names.
-read_pad_master_file <i>file_name</i> ("Masters" option in the GUI)	Specifies the file name that contains pad master names and optional net names.
-read_power_compiler_file <i>file_name</i> ("Power Compiler/..." button in the GUI)	Specifies the file that contains the Power Compiler report.
-read_prime_power_file <i>file_name</i> ("PrimePower/..." button in the GUI)	Specifies the file that contains the PrimePower report.

Table 8-7 analyze_fp_rail Command Options (Continued)

Command option	Description
-top_level_only ("Top level cells only" option in the GUI)	Specifies that only top-level cells are used in the power calculation. Cells inside soft macros are ignored.
-use_pins_as_pads ("Use top level design pins as pads" check box in the GUI)	Treats power pins as power pads during power analysis.
-voltage_supply voltage ("Supply voltage" text box in the GUI)	Specifies the supply voltage for the specified power net.

Creating Connectivity Views

Use the `create_connview` command to invoke the Hercules connectivity engine to create CONN views and current source files for hard macro cells in your design. Power network analysis uses the `create_connview` command options and values that you specify, together with those specified in the technology file for the Milkyway design library, to create a Hercules runset.ev file. The Hercules connectivity engine uses the runset file and creates a SMASH view in the database. After the SMASH view is created, the command uses this data to create a CONN view for the hard macro cells. Power network analysis updates the runset file each time you run the `create_connview` command.

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

Performing Power Analysis With Switching Activity Information

The amount of power consumed by a design depends on the switching activity that occurs within the design. You must provide an accurate model of the switching activity to produce an accurate estimate of the power consumption for the design. Switching activity can be annotated on design nets, ports, and cell pins.

After capturing the switching activity and annotating your design, you can invoke power network analysis by using the `analyze_fp_rail -analyze_power` or `synthesize_fp_rail -analyze_power` command. These commands calculate the dynamic power value of each cell instance by considering the specified switching probabilities and by using the obtained power values for IR drop analysis. For more information, see “[Performing Power Network Analysis for Each Cell Instance](#)” on page 8-76

Annotating Switching Activity

You can annotate the switching activity with one of the following methods:

- Use the `set_switching_activity` command to annotate particular design objects, such as nets, pins, ports, and cell instances, of the current design.

Switching activity information consists of the static probability and the toggle rate. The static probability is the probability that the value of the design object has a logic value 1. It defines the percentage of time the signal is at a high state; the default is 0.5. The toggle rate is the rate at which the design object switches between logic values 0 and 1 within a time period. The following example defines a toggle rate of 0.25 and a static probability of 0.4 for the CLK signal.

```
icc_shell> set_switching_activity -toggle_rate 0.25 \
           -static_probability 0.4 CLK
```

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

- Generate one or more switching activity interchange format (SAIF) files by using RTL or gate-level simulation. Use the `read_saif` command to annotate the switching activity information onto nets, pins, ports, and cells in the current gate-level design.

A SAIF file is typically generated within a simulation flow that contains a testbench. The SAIF file contains the switching activity information organized in a hierarchical fashion, where the hierarchy of the SAIF file reflects the hierarchy of the simulation testbench. The following example uses the `read_saif` command in verbose mode to read the `data.saif` file for the current design. The design appears as `cpu/system_controller` in the SAIF file.

```
icc_shell> read_saif -input data.saif \
           -instance_name cpu/system_controller
```

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

- Use default switching activity.

If no simulation data is available, IC Compiler applies the built-in default toggle rate to analyze the power consumption.

The switching activity of primary inputs and black box outputs cannot be propagated. You can set the default toggle rate (0.1) and default static probability (0.5) for primary inputs and black box outputs by using the `power_default_toggle_rate` and `power_default_static_probability` variables. The default toggle rate value is the value of the `power_default_toggle_rate` variable multiplied by the related clock frequency. The clock can be specified by using the `-clock` option of the `set_switching_activity` command. If no related clock is specified on the net, the clock with the highest frequency is used. The default switching activity applied to these objects is propagated throughout the design for power network analysis.

Performing Power Network Analysis for Each Cell Instance

After capturing the switching activity and annotating your design, you can invoke power network analysis by using the `analyze_fp_rail -analyze_power` or `synthesize_fp_rail -analyze_power` command to analyze the power information from each cell instance in your design based on static probability and toggle rates. These commands calculate dynamic power values for each cell instance by using the switching probabilities and toggle rates that you specify. The power values are used to perform IR drop analysis. Power is not calculated unless you specify the `-analyze_power` option.

In the following example, the voltage drop on the VDD net is analyzed by using a supply voltage of 1.2 volts.

```
icc_shell> analyze_fp_rail -analyze_power -nets { VDD } \
           -voltage_supply 1.2
```

You can use the `-analyze_power` option concurrently with the `-power_budget` and the `-read_default_power_file` options.

Note:

If you use the `-analyze_power` option concurrently with the `-read_default_power_file` option, the power specified in the default power file has a higher priority than that calculated by the `-analyze_power` option.

Viewing the Analysis Results

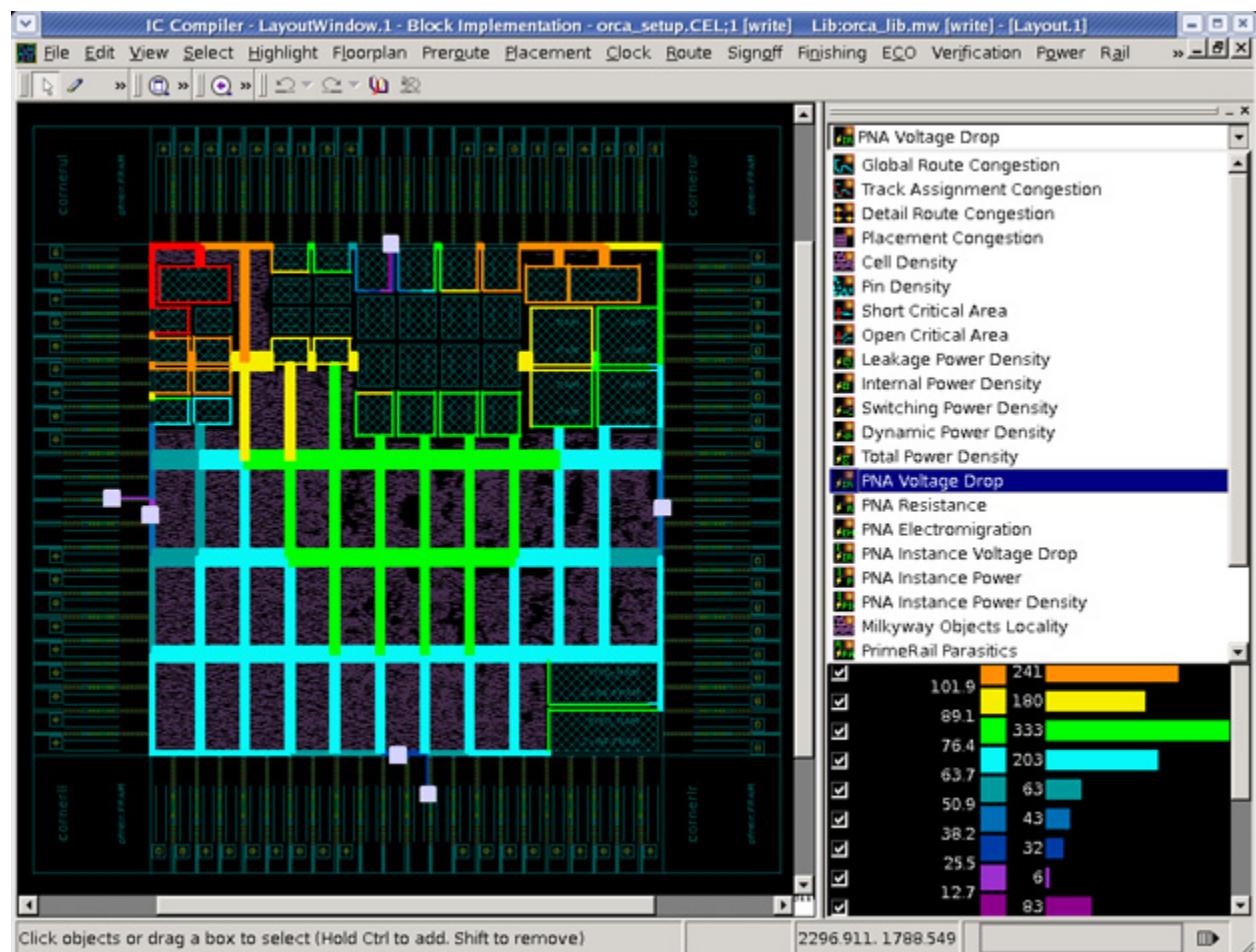
After you complete power and rail analysis, you can check for power-related violations in your design. IC Compiler provides several analysis maps that overlay on the layout view of your design, including a voltage drop map, a resistance map, an electromigration map, an instance voltage drop map, an instance power map, and an instance power density map. The following sections outline the steps to display the different power and rail analysis maps available in IC Compiler:

- [Displaying the Voltage Drop Map](#)
- [Displaying the Resistance Map](#)
- [Displaying the Electromigration Map](#)
- [Displaying the Instance Voltage Drop Map](#)
- [Displaying the Instance Power Map](#)
- [Displaying the Instance Power Density Map](#)

Displaying the Voltage Drop Map

After synthesizing the power network by using the `synthesize_fp_rail` command or by choosing Preroute > Synthesize Power Network in the GUI, you can review the voltage drop for different metal layers and locations in the layout. Choose Preroute > Power Network Voltage Drop Map in the GUI to view the voltage drop map, or use the `load_fp_rail_map` command. IC Compiler reads the results data saved to the `./pna_output` directory, constructs a color-coded voltage drop overlay, and displays the overlay on the layout view. The command also builds a color histogram that displays the distribution of voltage drop values. Figure 8-35 shows an example voltage drop analysis overlay in the IC Compiler GUI.

Figure 8-35 Voltage Drop Map



You can configure the histogram and voltage drop map display by selecting threshold voltages and metal layers in the GUI. To select only certain metal layers for IR drop analysis, select or deselect the check boxes in the layers box in the PNA Voltage Drop panel of the GUI. To enter a maximum or minimum threshold value for display in the histogram and

voltage drop map, select the “Max threshold” or “Min threshold” check box and enter the value in the text box. To limit the voltage drop bins that are displayed, select or deselect the check boxes in the histogram box in the PNA Voltage Drop panel of the GUI. After you change the selection for either threshold or metal layers, click the Apply button to display the updated histogram and voltage drop map.

To use the mouse to examine the voltage drop at individual locations in the layout, click the Query button and move the mouse onto the layout view. As you move the mouse over different points in the layout, the layout window displays the voltage drop at that point in the query panel in the lower-left area of the layout.

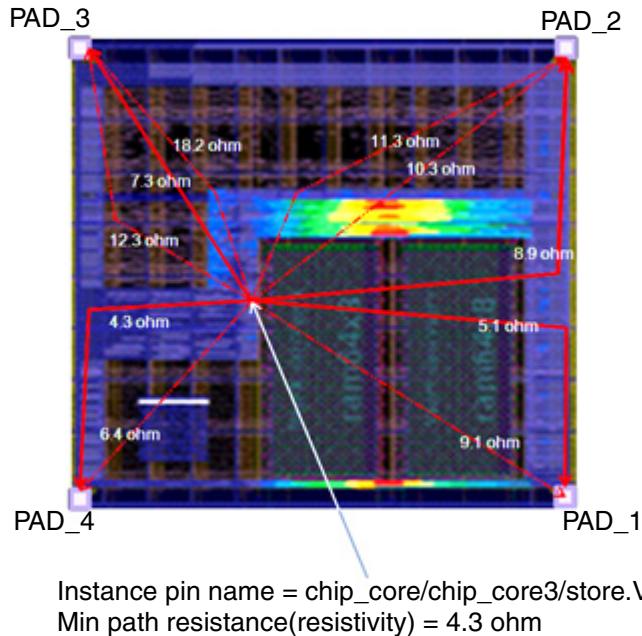
To change the net for analysis, click the Reload button and choose a net name from the Net drop-down box. Click OK to view the results in the GUI.

Displaying the Resistance Map

The resistance map displays a map of resistivity values for the power network in your design. The resistivity value at any instance pin or power grid location is the minimum path resistance value of all independent path resistances to any voltage source location.

[Figure 8-36](#) shows a series of paths to the chip_core/chip_core3/store.VDD pin; the lowest resistivity value of 4.3 represents the resistivity for the node. You can use the resistance map to check for unusually high values of resistivity, which might indicate missing vias or disconnected nets in the power network.

Figure 8-36 Minimum Path Resistance



Use the `load_fp_rail_map -nets net_name -type R` command or choose Preroute > Power Network Resistance Map in the GUI to load the map and histogram. The operation of the GUI for changing the threshold values, histogram, metal layer selection, and query text are similar to that for the voltage drop map; see “[Displaying the Voltage Drop Map](#)” on page 8-77 for information about operating the GUI.

Displaying the Electromigration Map

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 that can cause a functional failure. You can view the electromigration analysis results for your design by using the `load_fp_rail_map -nets net_name -type EM` command or by choosing Preroute > Power Network Electromigration Map in the GUI. The operation of the GUI for changing the threshold values, histogram, metal layer selection, and query text are similar to that for the voltage drop map; see “[Displaying the Voltage Drop Map](#)” on page 8-77 for information about operating the GUI.

Displaying the Instance Voltage Drop Map

The instance voltage drop map displays voltage drop values for each instance in the design. You can view instance voltage drop analysis results for your design by using the `load_fp_rail_map -nets net_name -type InstIR` command or by choosing Preroute > Power Network Instance Voltage Drop Map in the GUI. The operation of the GUI for changing the threshold values, histogram, and query text are similar to that for the voltage drop map; see “[Displaying the Voltage Drop Map](#)” on page 8-77 for information about operating the GUI.

Displaying the Instance Power Map

The instance power map displays power values for each instance in the design. You can view instance power analysis results for your design by using the `load_fp_rail_map -nets net_name -type P` command or by choosing Preroute > Power Network Instance Power Map in the GUI. The operation of the GUI for changing the threshold values, histogram, and query text are similar to that for the voltage drop map; see “[Displaying the Voltage Drop Map](#)” on page 8-77 for information about operating the GUI.

Displaying the Instance Power Density Map

The instance power density map displays power density values for areas of the design. You can view instance power density analysis results for your design by using the `load_fp_rail_map -nets net_name -type PD` command or by choosing Preroute > Power Network Instance Power Density Map in the GUI. The operation of the GUI for

changing the threshold values, histogram, and query text are similar to that for the voltage drop map; see “[Displaying the Voltage Drop Map](#)” on page 8-77 for information about operating the GUI.

9

Performing Clock Planning

This chapter describes how to reduce timing closure iterations by performing clock planning on a top-level design during the early stages of the virtual flat flow, after plan groups are created and before the hierarchy is committed. You can perform clock planning on a specified clock net or on all clock nets in your design.

For hierarchical timing closure, clock planning can also be used to generate realistic clock latency through timing budgeting to fix timing violations.

Clock planning tries to minimize clock skew by running block-level and top-level clock tree synthesis during the early stages of the design flow. Clock planning determines the clock budgets, allocates resources for clock buffers and clock routes, determines optimal clock pin locations for soft macros, and provides an estimate of the block-level insertion delays and skew for each plan group before finalizing the floorplan. Having optimal clock pin locations is a key factor in meeting the final clock skew and insertion delay numbers with the optimal number of buffers.

This chapter includes the following sections:

- [Setting Clock Planning Options](#)
- [Performing Clock Planning Operations](#)
- [Generating Clock Network and Source Latency for Each Clock Pin of Each Plan Group](#)
- [Using Multivoltage Designs in Clock Planning](#)

- [Performing Plan-Group-Aware Clock Tree Synthesis](#)
- [Checking for Electromigration After Clock Tree Synthesis](#)
- [Supporting Abutted Floorplans in Hierarchical Clock Planning](#)

Setting Clock Planning Options

Before you can compile clock trees inside the plan groups and build clock trees at the top level, you must first set different clock planning options such as anchor cell insertion, specifying nets for clock planning, and whether or not to route the clock nets after clock planning. You can also modify the clock tree constraint settings.

To set clock planning options,

1. Choose Clock > Set Clock Plan Options.

The Set Clock Plan Options dialog box opens.

Alternatively, you can use the `set_fp_clock_plan_options` command.

2. Set the options, depending on your requirements.

- Clock Nets – Enter the name of the clock nets on which to do clock planning.
- No Feedthroughs in Plan Groups – Enter a list of plan groups on which you do not want buffers placed. During the top-level clock tree synthesis phase of clock tree planning, no buffers are placed on the plan groups, unless they drive sinks inside those plan groups. This minimizes the creation of feedthroughs.
- Anchor Cell – Enter the name of the cell that is inserted as an anchor cell for all the plan groups. The anchor cell is a clock buffer cell. All plan groups should use the same buffer type. Inverters are not supported. This option is required. If you do not specify the name of the anchor cell, the IC Compiler tool issues an error message.

For each clock interface net (nets that cross plan group boundaries), an anchor cell (driver cell) is inserted for each plan group on every input clock net that crosses a hierarchical block. This partitions the plan-group-level clock subtree from the top-level clock tree.

An isolation cell is also inserted at the top level (for each output clock port on the plan group) to isolate the plan-group-level clock subtree from the top-level clock tree.

The anchor cell is inserted inside the plan group at the center of the mass of flip-flops that are connected to each clock net to isolate the top-level clock net from the clock net that is inside the plan group. The input pin of the anchor cell is driven by the clock net, and the output pin of the anchor cell drives the root clock pin of the block.

- Route Mode – Choose whether or not to route the top-level clock nets after clock planning. Based on the routing information, clock pins are created and assigned a location where the route crosses the plan group boundary.
 - Global route – Select this option to perform global routing on the clock nets.
 - Detailed route – Select this option to perform detail routing on the clock nets.
 - None – Select this option if you do not want to route the clock nets. This is the default.
- Output directory – Enter the name of the directory in which to write out all constraint files, log files, and generated reports from clock planning. The default directory is /cp_output.
- Keep block level tree – Select this option if you want to keep the top-level clock tree buffers inside the plan groups during clock planning. The default is to remove the block-level clock tree when clock planning is complete.

Clock tree planning might, for example, show a large insertion delay and skew value on one of the blocks in your design. By keeping the clock tree buffers inside the plan groups, you can more easily analyze them to determine why clock tree planning shows such a large value.
- Set clock tree options – Select this option to open the Set Clock Tree Options dialog box.

3. Click OK or Apply to set the clock planning options.

Reporting Clock Planning Options

You can get a report of the clock plan options by using the `report_fp_clock_plan_options` command. If no clock plan options have been set, this command reports the default values.

Removing Clock Planning Options

You can remove (reset) the database entries for the clock planning options you set by using the `reset_fp_clock_plan_options` command.

Performing Clock Planning Operations

Clock planning is done during the early stages of the virtual flat flow (after plan groups have been created and before the hierarchy is committed) to allocate clock resources and provide an estimate of the block-level insertion delay and skew for each plan group, before finalizing the floorplan.

You can perform clock planning operations to compile the clock trees inside plan groups and build clock trees at the top level based on the options you have selected in the Set Clock Plan Options dialog box (`set_fp_clock_plan_options` command).

To perform clock planning operations,

1. Choose Clock > Compile Clock Plan.

The Compile Clock Plan dialog box appears.

Alternatively, you can use the `compile_fp_clock_plan` command.

2. Set the options, depending on your requirements.

- Operation Condition – Select the operating conditions (Max, Min, or Min/Max) for top-level clock tree synthesis and optimization. The default is Max.
- Insert Anchor only – If you select this option, the clock planning tool only inserts anchor cells on the input ports of the plan groups, and does not synthesize the clock plan.

By default, the clock planning tool inserts anchor cells on the input ports of the plan groups plan groups, and then synthesizes the clock plan.

3. Select OK or Apply.

During clock planning, the following operations are performed:

- Anchor cells are inserted on the input ports of the plan groups to isolate the clock trees inside the plan groups from the top-level clock tree.
- Fast clock tree synthesis is run inside each plan group to estimate the insertion delay and skew values at the input of the anchor cells.
- The clock tree synthesis results are annotated on floating pins, which are defined on the anchor cells.
- The top-level clock tree is synthesized to the anchor cell floating pins.
- Detail routing is run on the clock interface nets.

Generating Clock Tree Reports

You can use clock skew analysis to generate a skew report for a specified clock (or for all the clocks within a design) before or after routing. You can view the report in a text window or write it to a specified file.

To generate clock tree reports, choose Clock > Report Clock Tree or use the `report_clock_tree` command. By default, the global skew is reported for all the generated clock trees.

Generating Clock Network and Source Latency for Each Clock Pin of Each Plan Group

For hierarchical timing closure, clock planning is also used to generate realistic clock latency through budgeting to fix timing violations. Top-level timing violations result not only from delays on combinational logic between registers, but also from clock skew between launching and capturing registers.

When you perform clock planning operations, it enables the timing budgeter to generate clock network and clock source latencies for each clock pin of each plan group in the design. For more information, see “[Performing Clock Latency Budgeting](#)” on page 12-31.

- 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.

Example:

```
icc_shell> set_clock_latency 2 -max -rise [get_clocks CLK]
```

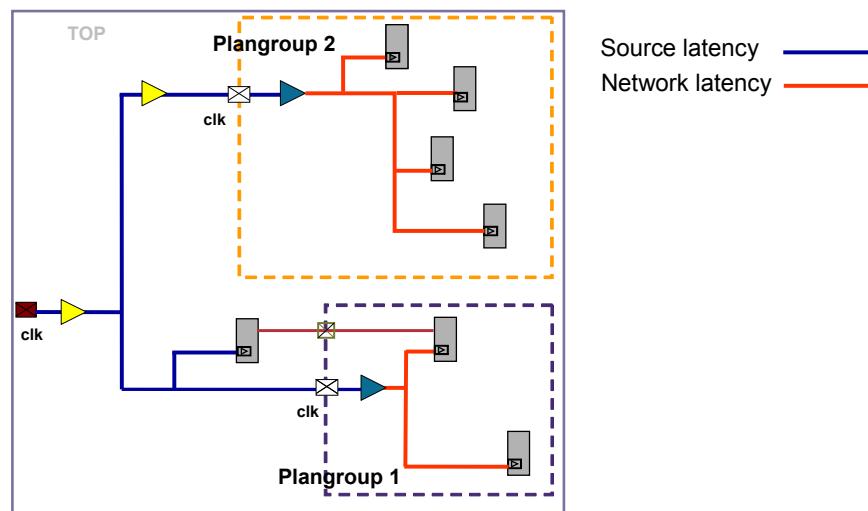
- Clock source latency is the time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point in the design.

Example:

```
icc_shell> set_clock_latency 2 -source -max -rise [get_clocks CLK]
```

[Figure 9-1](#) shows an example of the clock network latency and the clock source latency.

Figure 9-1 Clock Network and Source Latency



Creating a Virtual Clock for I/O Paths

For each plan group, a virtual clock can be created to describe clock latency outside of the plan group

The syntax for defining virtual clocks created for clock latencies outside the plan group is

- Clocks launching flops of input paths:

clock_name_v_in

- Clocks capturing flops of output paths:

clock_name_v_out

Note:

To minimize the number of virtual clocks, only the worst-case clock latency is created for all input and output pins of plan groups launched or captured by a virtual clock.

[Figure 9-2 on page 9-7](#) shows a virtual clock example.

- In plan group 1, the name of the input path is in1.

clk1_v_in launching flop A on the top level has source latencies only.

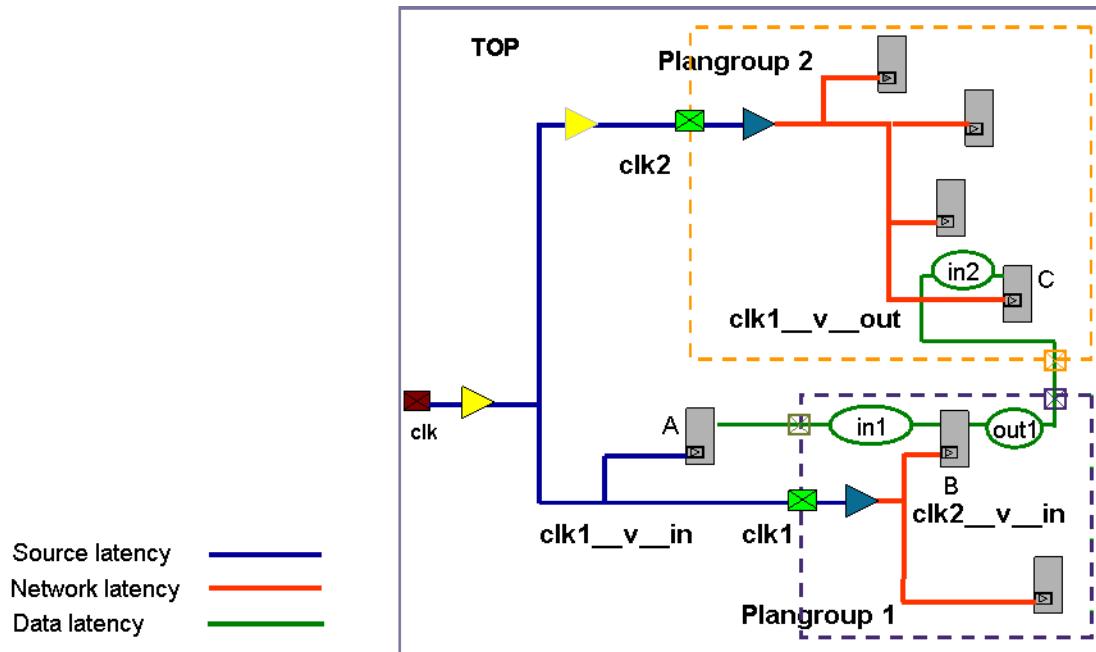
- In plan group 1, the name of the output path is out1.

clk1_v_out (*clk2*) capturing flop C in plan group 2 has both source and network latencies.

- In plan group 2, the name of the input path is in2.

clk2_v_in (*clk1*) launching flop B in plan group 1 has both source and network latencies.

Figure 9-2 Virtual Clock Example



Using Multivoltage Designs in Clock Planning

Clock planning supports multivoltage designs. Designs in multivoltage domains operate at various voltages. Multivoltage domains are connected through level-shifter cells. A level-shifter cell is a special cell that can carry signals across different voltage areas.

Clock planning isolates each plan-group-level clock from the top-level clocks by inserting an anchor cell. However, if you are using multivoltage designs in your clock planning, level-shifter cells should have already been inserted into the voltage area of the plan group.

Interpreting Level-Shifter Cells as Anchor Cells During Clock Planning

If there is a level-shifter cell for an interface clock net (a net that comes from the top-level clock net into a plan group), it is interpreted as an anchor cell. Then, during clock planning, anchor cells are inserted for each interface clock net only if they do not cross voltage areas. This prevents the insertion of buffers and inverters into the wrong voltage areas. Both plan-group-level clocks and voltage areas now isolated from the top-level clocks.

Note:

Feedthroughs created by clock planning might not honor the voltage area constraints. For all designs in multivoltage domains, you should specify the “No Feedthroughs in Plan Groups” option in the Set Clock Plan Options dialog box (`set_fp_clock_plan_options` command).

Performing Plan-Group-Aware Clock Tree Synthesis

You can perform plan-group-aware clock tree synthesis in clock planning. Clock tree synthesis includes the following operations:

- Generates a clock tree that honors the plan groups while inserting buffers into the logic hierarchy tree or, if it exists, into the corresponding physical region. The physical region can be a voltage area, an exclusive move bound, or a plan group.
- Prevents new clock buffers from being placed on top of a plan group unless they drive the entire subtree inside that particular plan group. This results in a minimum of clock feedthroughs, which makes the design easier to manage during partitioning and budgeting.

To perform plan-group-aware clock tree synthesis in clock planning with fully abutted floorplans, set the following variable:

```
icc_shell> set cp_full_abut_cts_region_aware true
```

To control plan-group-aware clock tree synthesis in clock planning, click in the text box next to the “No Feedthroughs in Plan Groups” option in the Set Clock Plan Options dialog box and enter a list of plan groups where clock feedthroughs should not be generated.

Checking for Electromigration After Clock Tree Synthesis

Cells in clock trees typically use more power than other cells because of the high switching rate and the capacitive loads of large fanouts. If the cells of clock networks are physically clustered or clumped together in the chip layout, excessive local currents through the power supply rails can cause electromigration problems. To help prevent such problems, use the `set_clock_cell_spacing` command, which ensures a minimum distance between clock cells.

After clock tree synthesis has been completed, you can check for excessive local power usage with the `report_tile_power` command. In the command, you specify the name of the power or ground net to be analyzed, the supply voltage, the maximum allowable current in the rails connected to standard cells, and the layer name of the rail if it is not the first metal layer. For example,

```
icc_shell> report_tile_power -net VSS -current_budget 0.08 \
-voltage_supply 0.90 -guard_band_ratio 0.95
```

This command requests a chip power analysis of the VSS (ground) net, with the supply voltage at 0.90 volts and a maximum rail current of 0.08 mA. An optional guard band ratio further restricts the allowable maximum current to 95 percent of the nominal limit.

To determine whether this power constraint is met, IC Compiler divides the standard-cell area into units called power tiles. Each power tile consists of a set of standard cells supplied by a segment of a power rail, extending from one via to another via, or from a via to the end of the row. It then calculates the power consumed by all cells in the tile and compares this value with the specified maximum current, subject to adjustment for the number of vias associated with the rail segment and the guard band setting.

For each tile that exceeds the maximum allowable power, the command reports the total power used in the tile, the total power budget for the tile, and the lower-left and upper-right coordinates of the tile rectangle. For example,

```
icc_shell> report_tile_power -net VSS -current_budget 0.08 \
-voltage_supply 0.90 -guard_band_ratio 0.95
...
Net : VSS
Supply Voltage : 0.900
Current Budget : 0.076
Standard Cell Rail Layer : METAL1
Power (mW)      Power Budget (mW)      Tile Bbox
-----
1.639e-01    1.368e-01      {{430.280 392.535} {482.300 399.915}}
1.639e-01    1.368e-01      {{639.370 392.535} {691.390 399.915}}
1.574e-01    1.368e-01      {{430.280 392.845} {482.300 400.225}}
1.710e-01    1.368e-01      {{639.370 392.845} {691.390 400.225}}
1.513e-01    1.368e-01      {{586.840 407.295} {639.370 414.675}}
1.575e-01    1.368e-01      {{586.840 407.605} {639.370 414.985}}
9.361e-02    6.840e-02      {{149.735 422.055} {731.350 429.435}}
9.342e-02    6.840e-02      {{150.790 422.365} {731.350 429.745}}
7.616e-02    6.840e-02      {{691.390 702.495} {731.350 709.875}}
7.254e-02    6.840e-02      {{691.390 702.805} {731.350 710.185}}
-----
Number of power tiles exceeding budget : 10
```

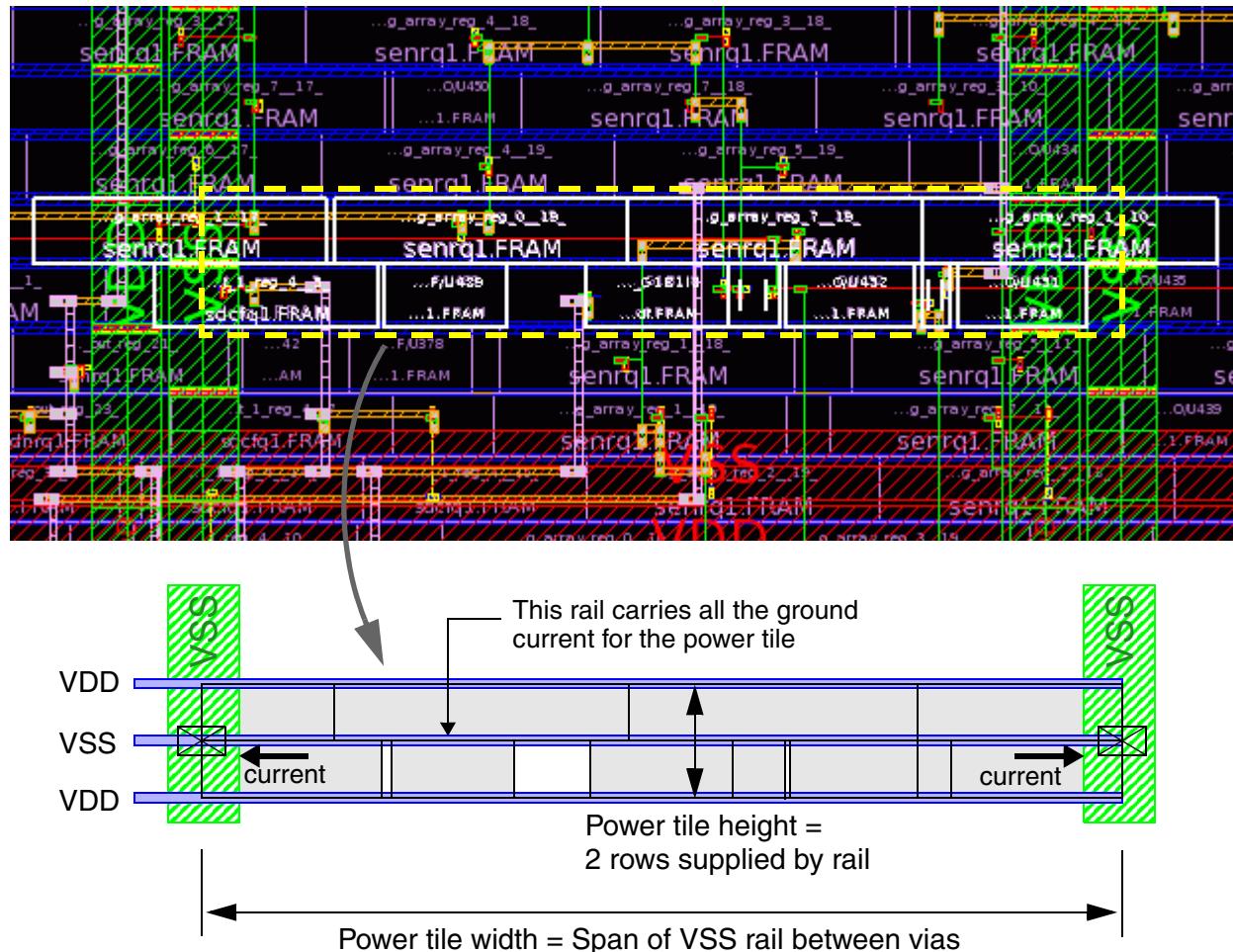
The two different power budget values shown in the table represent the budgets for single-via and two-via power tiles.

For more information about the `report_tile_power` command and its options, see the man page for the command.

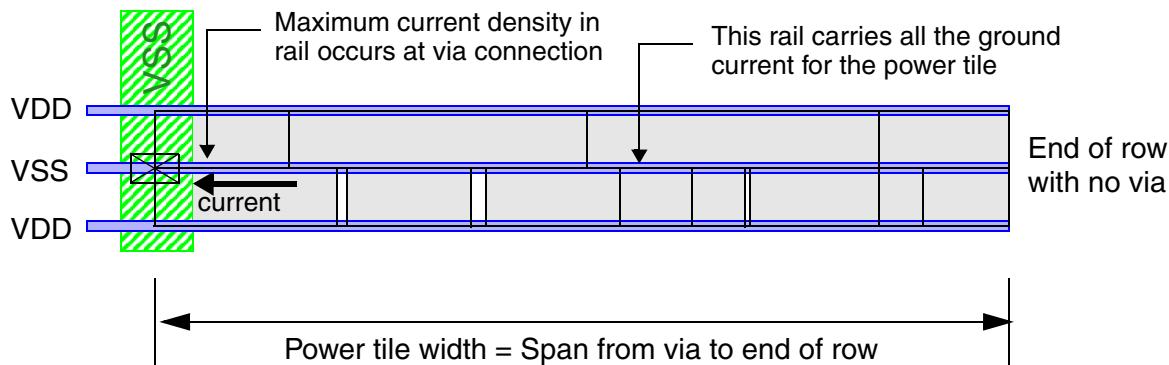
Power Tiles

A power tile typically consists of two rows of standard cells supplied by a single power or ground rail extending between two vias, as shown in [Figure 9-3](#). In this example, the ground current is shared between two via connections, one at each end of the power tile. The presence of two vias, one at each end of the tile, increases the total current that the rail can handle.

Figure 9-3 Power Tile With a Via at Each End



A power tile can also consist of the standard cells extending from a via to the end of the row, without a via at the other end of the tile, as shown in [Figure 9-4 on page 9-11](#). In that case, the rail must be able to handle all of the current for the power tile where it connects to the single via.

Figure 9-4 Power Tile With a Single Via

Power Budget Calculation

To determine whether the maximum current density constraint is met for the rail serving the power tile, IC Compiler compares the total power of the cells in the tile with this value:

$$P_T = I_B V_R N_V G$$

Where P_T is the maximum allowable power of the tile, V_R is the rail voltage, I_B is the budgeted maximum current for the rail, N_V is the number of via connections to the power rail in the tile (typically 2), and G is an optional guard band ratio specified in the `report_tile_power` command. The default guard band ratio is 1.0.

For example, suppose that you specify the analysis for ground rails as follows:

```
icc_shell> report_tile_power -net VSS -current_budget 0.08 \
-voltage_supply 0.90 -guard_band_ratio 0.95
```

For a power tile with a via at each end, the maximum allowable power of the tile is:

$$P_T = (0.08 \text{ mA}) \times (0.90 \text{ volts}) \times (2) \times (0.95) = 0.1368 \text{ mW}$$

For a power tile with a via at just one end, the maximum allowable power of the tile is:

$$P_T = (0.08 \text{ mA}) \times (0.90 \text{ volts}) \times (1) \times (0.95) = 0.0684 \text{ mW}$$

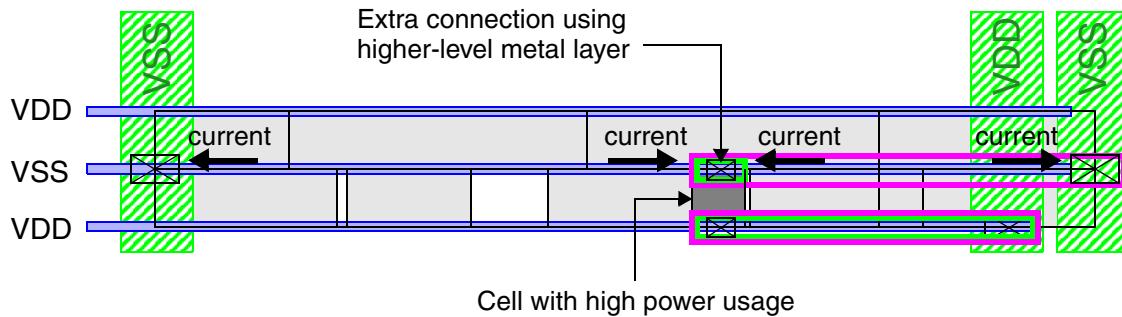
When you execute the `report_tile_power` command, IC Compiler divides the standard cell rows into power tiles according to the type of power rail (VDD or VSS) and the locations of vias on those rails. Then it calculates the total power of each tile by adding up the power

used by each standard cell within the tile. If a standard cell extends beyond the end of the tile, only the power corresponding to the fraction of the cell's area within the tile counts toward the total power of the tile.

Extra Power Straps on Higher Layers

Some power network synthesis flows apply additional via connections to high-power cells using upper-layer metal layers. For example, in [Figure 9-5](#), extra rail connections using the METAL3 layer provide additional power and ground pathways for the current drawn by one high-power cell. These additional pathways reduce the amount of current that must be carried by the bottom-layer metal rail, easing any electromigration issues for that rail.

Figure 9-5 Power Tile With Three Vias



By default, the `report_tile_power` command does not consider the presence of higher-level power supply connections. To consider them in the analysis, use the `-honor_extra_strap_layer` option of the `report_tile_power` command. In that case, IC Compiler looks for additional power connections in higher layers, and if present, increases the number of vias used to calculate the total tile budget. In this example, there are three vias connected to the VSS rail in the power tile, providing four pathways for current flow out of the rail, so the total power budget is calculated as follows:

$$P_T = (0.08 \text{ mA}) \times (0.90 \text{ volts}) \times (4) \times (0.95) = 0.2736 \text{ mW}$$

To save runtime, use the `-honor_extra_strap_layer` option only if the design contains extra power straps using higher layers.

Supporting Abutted Floorplans in Hierarchical Clock Planning

You can perform hierarchical clock planning by using the `compile_fp_clock_plan` command on designs with fully abutted floorplans. This avoids possible DRC violations on nets going through different plan groups.

From a placed design, set the following variable to perform hierarchical clock planning on fully abutted floorplans:

```
icc_shell> set cp_in_full_abut_mode true
```

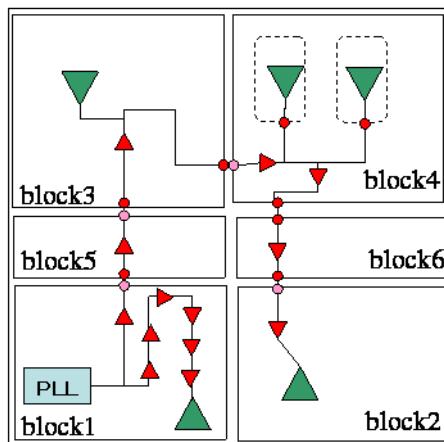
Run the `compile_fp_clock_plan` command to compile the clock trees inside plan groups and build clock trees at the top level.

Note:

Clock tree planning resolves possible DRC violations on nets going through different plan groups in the design, because the ideal clock buffer location is far from the plan group that is associated with the logical hierarchy in which the buffer is added.

[Figure 9-6](#) shows an example of a fully abutted floorplan.

Figure 9-6 Fully Abutted Floorplan



10

Performing In-Place Timing Optimization

In-place timing optimization is used to improve the timing of a given design, and in particular, to meet the timing constraints on the design. It is an iterative process based on virtual routing. If you run in-place optimization after global routing, the tool deletes the global routes and optimizes the design based on the virtual route timing.

This chapter includes the following sections:

- [Running In-Place Timing Optimization](#)
- [Running Trace Mode In-Place Optimization](#)
- [Using In-Place Optimization With Multivoltage Designs](#)
- [Performing In-Place Optimizations Based on Pin Locations](#)
- [Virtual In-Place Optimization](#)

The optimization process starts with the most significant changes on multiple violation paths that will quickly improve overall timing. Next, the scope is narrowed to focus on individual paths. Netlist changes are committed only if the timing improves. The output is a legally placed netlist that is plan group aware.

Three types of optimizations are performed: timing improvement, area recovery, and fixing timing design rule violations. These optimizations preserve the netlist's logical hierarchy, and the physical locations of most cells change as little as possible.

Many methods for improving timing are used, including

- Inserting buffers and inverters (legal locations are found automatically)
In the case of inverters, it is guaranteed that the polarity of the signals is preserved
- Increasing or decreasing cell size (If the cell size is increased, the larger cell is automatically adjusted to ensure that all cells are placed in legal locations.)
- Moving cells
Similar to cell sizing, the cells are placed in legal locations

Running In-Place Timing Optimization

To perform in-place timing optimization,

1. Choose Timing > In Place Optimization.

The In Place Optimization dialog box appears.

2. Set the options in the GUI depending on your requirements.

- Add buffers for feedthrough nets only

Enable this option to only add buffers for feedthrough nets. Optimization is not performed and timing is not updated. Using this option minimizes design changes in the late stages of design planning. The default is disabled.

You must perform pin placement by using the `place_fp_pins` command before running in-place optimization. See “[Performing Pin Assignment on Soft Macros and Plan Groups](#)” on page 11-7 for more information.

The tool automatically selects medium-sized buffers and inserts them near each pin (port) location determined by the `place_fp_pins` command. During the optimization phases, in-place optimization might resize or move these buffers. The buffers are added in the same hierarchy as the plan groups.

- Perform high fanout synthesis optimization only

Enable this option to perform only high-fanout synthesis (HFS) on the design. You can combine this option with the “Don’t add any new cells at top level” option when implementing fully abutted or narrow channel designs.

The default is disabled.

Use the `optimize_fp_timing -hfs_only` command to perform normal high-fanout synthesis. For incremental high-fanout synthesis, use the `set_ahfs_options -incremental true` command, before running the `optimize_fp_timing -hfs_only` command. Note the following limitations for the regular and incremental high-fanout synthesis flows:

- Regular high-fanout synthesis is commonly used for the normal flow
- Incremental high-fanout synthesis is required for the exploration flow to reduce runtime
- Regular high-fanout synthesis in the exploration flow is not well suited for analyzing the clumped placement of pre-existing buffers
- Incremental high-fanout synthesis does not analyze preexisting buffers and their placement
- Incremental high-fanout synthesis is optional for the regular high-fanout synthesis flow

These guidelines are based on the following QoR metrics: worst negative slack (WNS), total negative slack (TNS), design rule check (DRC), and runtime.

- Effort

You can specify how much effort is used to minimize the worst negative slack in the design. If you select an effort level of high, more effort is expended to improve the timing of the design, resulting in more CPU time. In-place optimization stops when it finds the worst negative slack in the design cannot be further improved. The output is a legally placed netlist.

The default effort is medium.

- Fix design rules violations

Enable this option to fix design rules. The default is disabled. Design rule violations, such as maximum transition, maximum capacitance, and maximum fanout, are fixed by use of buffer insertion, gate sizing, and automatic high-fanout net synthesis for handling medium- and high-fanout nets.

When you run in-place optimization at the virtual route stage, only obvious design rule violations are fixed because wire locations and interconnect are estimated at this stage since timing analysis cannot be completely accurate and runtime is shorter. After you finish global routing, optimization takes longer to run, but the results are based on more accurate timing information.

- Enable area recovery

Enable this option to direct the in-place optimization engine to invoke additional operations to try to reduce the cell area without causing timing violations. The area is optimized by removing cells and decreasing the size of the cells on noncritical timing paths. This allows more space for optimization on critical paths.

The default is disabled.

Note:

Area recovery is an operation that tries to reduce the total area of cells used in the design. Area recovery does not increase timing delays on the critical paths, but some paths might see an increase in timing delay if the path is nonviolating. If the path has positive slack, area recovery might reduce this slack to zero. Area recovery does not cause nonviolating paths (paths with nonnegative slack) to become violating paths.

Area recovery, however, is a difficult and expensive operation, resulting in longer runtime but with a smaller area being used. Designs with high utilization benefit from area recovery but with a cost that results in higher runtimes.

- Keep global routes

Use this option if you want to preserve the global routes after timing optimization. By default, the tool removes global routes from the design.

- Don't add any new cells at top level

Enable this option to prevent new cell insertion at the top level of the design during in-place optimization. You can combine this option with the “Perform high fanout synthesis optimization only” option when implementing fully abutted or narrow-channel designs.

By default, new cells can be inserted at the top level of the design.

- Report quality of results

Enable this option to get reports on the worst negative slack (WNS) and total negative slack (TNS) of the design, the utilization percentage of each plan group and top level, the number of buffers added, and the number of cells sized for each plan group and top level.

By default, the command does not report the quality of results.

3. Click OK or Apply.

Alternatively, you can use the `optimize_fp_timing` command.

Running Trace Mode In-Place Optimization

Use the `optimize_fp_timing` command or choose Timing > In-Place Optimization in the GUI to perform in-place optimization. To improve capacity and runtime, you can run the in-place optimization in trace mode by setting the `set_fp_trace_mode` command. In this mode, the tool optimizes all top-level and interface paths or interface timing paths only.

[Table 10-1](#) describes the `set_fp_trace_mode` command options. For more information about these options, see the man page.

Table 10-1 set_fp_trace_mode Command Options

Command option	Description
<code>-include_top_logic</code>	Includes the top-level logic in the trace mode timing netlist. By default, trace mode considers only interface logic.
<code>-verbose</code>	Prints additional informational messages.

Note:

To check if a design is in trace mode, use the `get_fp_trace_mode` command.

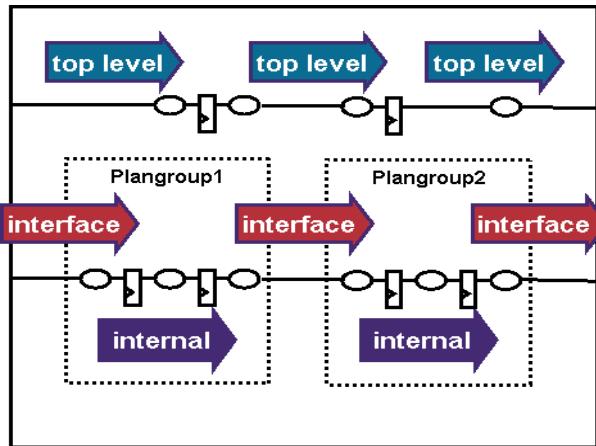
In trace mode, the tool considers only the top-level and interface paths even though the entire design is loaded.

When trace mode timing is used in a flat design with plan groups defined for soft macros, trace mode timing selectively filters paths. This is done by tracing only the nets and cells of timing paths at the top level which cross plan group boundaries. Accordingly, the `report_clock_timing` command reports only the timing violations on those paths. By using trace mode timing analysis, you can detect and fix these timing violations in the early phases of the design flow. This enables faster timing analysis by focusing on interface nets.

During the trace mode in-place optimization stage, only the interface logic or interface logic plus top-level logic is optimized. This results in good QoR and might improve runtime.

[Figure 10-1](#) illustrates the top-level, internal, and interface paths for trace mode timing.

Figure 10-1 Top-Level, Interface, and Internal Paths for Trace Mode Timing



Reporting Trace Mode Status

You can report the current option settings for trace mode by using the `report_fp_trace_mode_options` command. The command shows whether top-level and interface logic are included in the timing optimization. The following example shows a report generated by the `report_fp_trace_mode_options` command.

```
icc_shell> report_fp_trace_mode_options
Trace mode is set with following options:
  interface  : included
  top        : not included
  verbose    : off
1
```

Removing the Trace Mode

To remove trace mode timing and return to the normal mode for timing analysis, use the `end_fp_trace_mode` command. This command removes the trace mode design from memory. After the next command that accesses the netlist of the design is used, it triggers a reload of the entire design netlist.

Using In-Place Optimization With Multivoltage Designs

The `optimize_fp_timing` command detects multivoltage settings and performs multivoltage-aware optimization. By following predefined multivoltage design rules, in-place optimization improves timing by swapping cell instances. The command replaces cell instances with cells of different sizes from within the same power domain and inserts buffers that obey multivoltage design rules.

IC Compiler handles always-on nets and honors power guides created by the `set_power_guide` command during feedthrough buffer optimization. If a feedthrough net has a power guide setting, IC Compiler uses the appropriate buffers to satisfy the setting. IC Compiler uses normal buffering on nets that do not have a power guide setting.

Performing In-Place Optimizations Based on Pin Locations

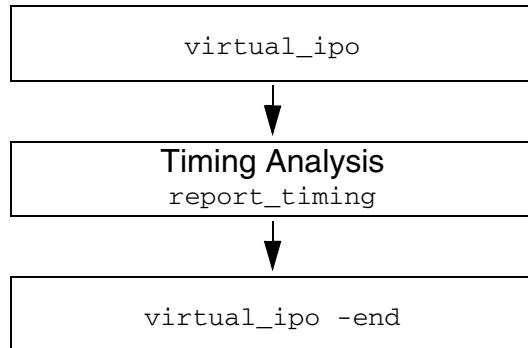
You can use the `optimize_fp_timing` command after pin placement. Timing optimization understands pin locations decided by the previous pin placement command when performing timing optimization. It chooses medium-sized buffers and inserts them near each pin (port) location determined by the `place_fp_pins` command.

Virtual In-Place Optimization

The `virtual_ipo` command performs virtual in-place optimization on the design and predicts timing after physical synthesis. The command estimates the timing of the design by using analytical models and does not change the placement or the netlist. As a result, the `virtual_ipo` command can complete faster than the `optimize_fp_timing` command. The output from the `virtual_ipo` command contains estimated and annotated delays of cells and nets along paths that violate timing. When you run the exploration flow, you should first add buffers on feedthrough nets by using the `optimize_fp_timing -feedthrough_buffering_only` command before you run the `virtual_ipo` command.

To remove timing data after performing virtual in-place optimization, use the `virtual_ipo -end` command, which removes all back-annotation data inserted during the previous virtual in-place optimization run. This operation is required after you have performed timing closure using virtual in-place optimization, before proceeding to physical optimization as shown in [Figure 10-2 on page 10-8](#).

Figure 10-2 Virtual In-Place Optimization Flow



For more information about virtual in-place optimization, see [“Virtual In-Place Optimization” in Chapter 6](#).

11

Performing Routing-Based Pin Assignment

You can create top-level soft macro pins and plan group pins in hierarchical designs, constrain the pins during pin creation, edit the soft macro pins, add feedthroughs, and analyze the quality of the pin assignment results. IC Compiler supports concurrent pad and pin assignment.

This chapter includes the following sections:

- [Setting Pin Assignment Constraints](#)
- [Performing Pin Assignment on Soft Macros and Plan Groups](#)
- [Adjusting the Position of Placed Pins](#)
- [Adding Feedthrough Pins](#)
- [Adding Feedthrough Pins on Multivoltage Designs](#)
- [Performing Pin Placement With Multiply Instantiated Modules](#)
- [Using Pin Guides for Pin Placement](#)
- [Checking Pin Assignment and Pin Alignment](#)

Setting Pin Assignment Constraints

Prior to assigning pins, you can set constraints to specify the preferred locations and layers for the pins in the soft macros and plan groups in your design. Global pin assignment constraints apply to the entire design or the blocks you specify. Physical constraints define specific parameters for individual soft macro or plan group pins. The following sections describe the global pin assignment constraints and physical pin assignment constraints available in IC Compiler.

Specifying Global Pin Assignment Constraints

You can set constraints on pins before performing pin assignment with the `set_fp_pin_constraints` command.

```
set_fp_pin_constraints
  [-allow_feedthroughs off | on]
  [-allowed_layers layers]
  [-block_level]
  [-bus_ordering lsb_to_msb | msb_to_lsb | scrambled | 
    consistent_wirelengths]
  [-corner_keepout_num_wiretracks wiretracks_number]
  [-corner_keepout_percent_side keepout_percentage]
  [-exclude_clock_feedthroughs off | on]
  [-exclude_feedthroughs nets]
  [-exclude_hfn_feedthroughs hfn_number]
  [-exclude_network]
  [-exclude_scan_chain_net_feedthroughs off | on]
  [-exclude_sides side_numbers]
  [-feedthrough_map_file file_name]
  [-hard_constraints off | spacing | location | layer]
  [-incremental off | on]
  [-keep_buses_together off | on]
  [-nets nets | -exclude_nets nets | -pins pins]
  [-no_stacking stacking_allowed | pg_pins_only | signal_pins_only | all]
  [-pin_preroute_spacing preroute_spacing_number]
  [-pin_spacing pin_spacing_number]
  [-read_feedthrough_map off | on]
  [-reserved_channel_nets nets]
  [-reserved_channel_threshold channel_size]
  [-scramble_skip skip_number]
  [-use_physical_constraints off | on]
  [blocks]
```

Pin constraints set with the `set_fp_pin_constraints` command are honored during pin assignment on soft macros and plan groups. Pin assignment constraints are saved in the Milkyway database.

The following command sets a pin placement constraint that allows the creation of feedthrough ports on plan groups and soft macros. Bus bits are grouped together.

```
icc_shell> set_fp_pin_constraints -allow_feedthroughs on \
           -keep_buses_together on
```

Specifying Constraints for Special Conditions

You can use options of the `set_fp_pin_constraints` command to handle special conditions in your design, such as narrow channels for special nets, bus bit ordering, and detecting clock nets.

Reserving Narrow Channels for Special Nets

The `set_fp_pin_constraints -reserved_channel_threshold` command specifies the maximum spacing between plan groups and soft macros for clock nets and other user-specified nets. This feature is useful for floorplans that contain only narrow channels between plan groups.

By default, the channel size is 0.0 and the command does not reserve a channel for special nets. If you set a channel size greater than 0.0, any space between adjacent plan group edges that are separated by less than this amount is considered a channel reserved for special nets. You should set the `-allow_feedthroughs` option to `off` to exclude feedthroughs on these special nets. Generally, feedthroughs are allowed for other nets.

During global routing, special nets can route freely through any space between the plan groups. If the size of the space you specified is less than or equal to the threshold, other nets are prevented from routing parallel to the direction of the channel. However, any net that has a pin in the channel must enter the channel, regardless of whether it is a special net. For example, the channel might contain pins because there are standard cells or small macros in the channel area or there are pins on a block at the very edge of a plan group. In such cases, the routing to these targets must enter the channel.

When this option is used during pin assignment, pins are discouraged on blocks that face the channel unless they are perfectly aligned across the channel or are pins that belong to special nets. The tool routes nonspecial nets by using other routing paths, or by creating feedthrough pins.

Note:

Two voltage areas that face each other do not generate a reserved channel for special nets. However, if your floorplan consists entirely of plan groups with narrow channels between them and each plan group is a voltage area, you should allow voltage area feedthroughs by using the `set_fp_voltage_area_constraints -allow_feedthroughs true` command. Otherwise, both feedthroughs and channel routing are restricted.

You can use the `set_fp_pin_constraints -reserved_channel_nets` command to specify a collection of nets which are permitted in the reserved channels. By default, this is the set of all nets that are marked as clock nets in the Milkyway database.

If you use the `-reserved_channel_nets` option, the collection you specify must include all nets to be routed in the reserved channels. These nets are implicitly marked as “exclude feedthroughs” because they are routed through the channels. This option overrides any previous reserved net specification.

Keeping Bus Bits Together

If you specify the `set_fp_pin_constraints -keep_buses_together` on command, the plan-group-aware router automatically detects buses based on user-defined bus groups in the Milkyway database. The tool groups bus pins together during pin assignment.

Reserving Space for Clock Pins

Before you set pin assignment constraints, you can ensure that there are optimal locations reserved for clock net pins by using the `mark_clock_tree -clock_net` command. To assign clock nets, the plan-group-aware router checks the net type in the Milkyway database for nets with the “clock” net type. The command routes nets with the “clock” net type before routing the remaining signal nets. If no clocks are detected, the command issues a warning message indicating that there are no clock nets and requests that you run the `mark_clock_tree -clock_net` command to mark the clock nets in the Milkyway database.

Specifying Pin Physical Design Constraints

You can use the `set_pin_physical_constraints` command to set constraints on individual pins or nets in a block, soft macro, or plan group.

```
set_pin_physical_constraints
  objects |
  -pin_name pin_name [-cell cell_name] [-nets nets]
  [-depth pin_depth]
  [-exclude_sides exclude_side_numbers]
  [-layers layers]
  [-location point]
  [-off_edge center | location | auto]
  [-offset offset_distance]
  [-order order_number]
  [-pin_spacing spacing]
  [-side side_number]
  [-width pin_width]
```

If a conflict arises between the individual pin constraints and the global pin constraints, the individual pin constraints have higher priority. You can combine the `-nets` and `-cell` options in the same command line. The constraints are stored in the Milkyway database. For more information about the command options, see the `set_pin_physical_constraints` man page.

The following example sets individual pin placement constraints on the `datain_1` and `dataout_1` pins for the current design. The pins can be placed on layers metal3 and metal4. The `datain_1` pin is restricted to side 1, which is the bottommost left side of the module. The `dataout_1` pin is restricted to side 3.

```
icc_shell> set_pin_physical_constraints -pin_name { datain_1 } \
    -layers { metal3 metal4 } -side 1
icc_shell> set_pin_physical_constraints -pin_name { dataout_1 } \
    -layers { metal3 metal4 } -side 3
```

You can define constraints for specific plan group and soft macro pins in your design by combining the `-cell` and `-pin_name` options of the `set_pin_physical_constraints` command. The following example applies pin constraints to the `datain_1` and `dataout_1` pins of the `mod1` plan group. If you do not specify the `-cell` or `-nets` options, the constraints are applied to the top-level pins.

```
icc_shell> set_pin_physical_constraints -pin_name { datain_1 } \
    -cell mod1 -layers { metal3 metal4 } -side 1
icc_shell> set_pin_physical_constraints -pin_name { dataout_1 } \
    -cell mod1 -layers { metal3 metal4 } -side 3
```

You can load a file containing `set_pin_physical_constraints` commands and apply the constraints to a specific block by using the `read_pin_pad_physical_constraints` command. The following example reads the `pinconstraints.tcl` file and applies the constraints to the `block1` plan group.

```
icc_shell> read_pin_pad_physical_constraints \
    -cell block1 pinconstraints.tcl
```

Ignoring Existing Pin Physical Constraints

By default, pin assignment honors any preexisting pin physical constraints set by using the `set_pin_physical_constraints` command or the `read_pin_pad_physical_constraints` command. To ignore preexisting physical constraints on pins, use the `set_fp_pin_constraints` command with the `-use_physical_constraints off` option as follows:

```
icc_shell> set_fp_pin_constraints -use_physical_constraints off
```

Reporting Pin Assignment Constraints

You can use the `report_fp_pin_constraints` command to display the pin assignment constraints that are set for specified soft macros, plan groups, or ports. These constraints control both pin assignment and pin placement.

Use the following command syntax to report pin constraints for each soft macro and plan group in the design. The following example shows the output from the command for a design containing a plan group.

```
icc_shell> report_fp_pin_constraints
report_fp_pin_constraints
-----
Pin assignment options for plan group "U1":
Min layer for PG pins is metal2
Max layer for PG pins is metal6
Number of wire tracks between pins: 1
Number of wire tracks between preroutes and pins: 3
Allow overlapping pins on different layers
Soft constraint for pin-pin spacing
Soft constraint for pin location
Soft constraint for pin layer
No pins at the corners for 5 wiretracks
Create feedthrough ports
Consider internal placement.
Don't create EQ pins
Don't retain movable pins
Physical constraints are in effect.

INFO: Global pin assignment options (for all blocks):
Create pins for tied high/low signal ports
Don't keep bus bits together
readFeedthroughMap on
inputFeedthroughMapFileName feedthroughs.txt
1
```

Specify the name of the soft macro or plan group on the command line to report pin constraints for a specific soft macro or plan group.

```
icc_shell> report_fp_pin_constraints BLOCK1
```

Specify the `-block_level` option to report pin constraints for only the top level of the current design.

```
icc_shell> report_fp_pin_constraints -block_level
```

You can use the `report_fp_pin_constraints` command to troubleshoot problems with pin placement in your design. For example, if the tool is not placing some soft macro or plan group pins, you should ensure that the pins are not excluded from placement. The tool skips

pin placement for pins that are connected to a net that is excluded by using the `set_fp_pin_constraints -exclude_nets` command. The following report shows that pins connected to nets `datain_01`, `dataout_02`, `dataout_03`, `data_internal1_06`, `data_internal2_04`, and `data_internal2_05` are excluded from placement by the `place_fp_pins` command.

```
icc_shell> report_fp_pin_constraints
-----
Pin assignment options for plan group "PG1":
Min layer for PG pins is metal2
Max layer for PG pins is metal8
... more messages ...
List of 6 nets excluded from pin placement:
  datain_01 data_internal2_04
  dataout_02 data_internal2_05
  dataout_03 data_internal1_06
```

Performing Pin Assignment on Soft Macros and Plan Groups

After defining pin constraints by using the `set_fp_pin_constraints` and `set_pin_physical_constraints` commands, you can use pin assignment to assign pins at the top level of the design, or to assign pins on soft macros and plan groups.

For top-level pin assignment, the tool considers the top-level connections to plan groups, macros, and pad locations when it determines where to assign the pins. For block-level pin assignment, the tool considers the cell placement inside the soft macro or plan group when it assigns pin locations. The tool minimizes the wire lengths from the pins to the internal connections within the soft macro or plan group. Use block-level pin assignment in a bottom-up design flow.

IC Compiler supports pin assignment on designs that contain mixed pads and ports. Terminals are placed directly on top of and on the same layer as pad cells and bump cells, and on soft macro pins that are internally connected to an I/O cell pin. This reduces the RC delay between the primary signal I/O and the I/O cell to zero. When the I/O cell is located within a soft macro child cell, a soft macro pin is placed on top of the I/O cell.

If multiple I/O cell pins are assigned to the same pin connection, the tool creates electrically equivalent terminals at the corresponding I/O cell pin locations. Except for bump cell pins, an error occurs if the port to I/O cell pin connection is not one-to-one. Terminal placement honors any specified physical pin constraints.

The pin assignment results are stored at the child level within the cell reference. If you open a child cell after pin assignment and then close it without saving it, the pins are also discarded.

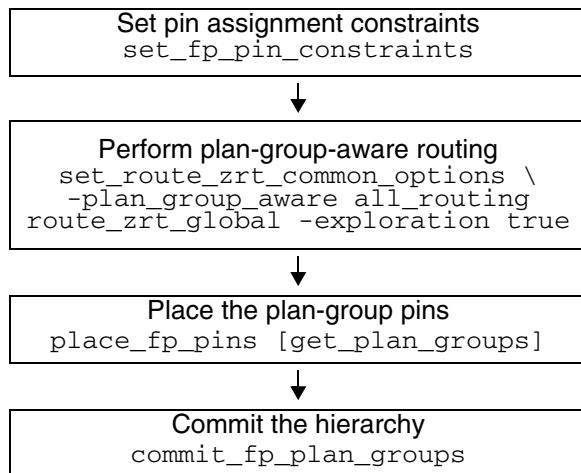
Performing Plan-Group-Aware Global Routing

During pin placement, IC Compiler performs global routing to determine the best pin locations for each plan group and soft macro. You can set global routing options by using the `set_fp_pin_constraints` and `set_route_zrt_common_options` commands.

By default, the `place_fp_pins` command performs global routing during pin placement. You can run global routing and pin placement as separate steps by running the `route_zrt_global` command, followed by the `place_fp_pins -use_existing_routing` command. The `-use_existing_routing` option directs the command to perform pin placement on plan groups by using the global routes from the previous run of `route_zrt_global`.

[Figure 11-1](#) shows the pin placement flow where you set global routing constraints and perform global routing as separate steps.

Figure 11-1 Pin Placement Flow With Global Routing



Set the Zroute common route option to perform plan-group-aware routing by using the `set_route_zrt_common_options` command as follows.

```
icc_shell> set_route_zrt_common_options -plan_group_aware all_routing
```

If you specify the `-plan_group_aware` common route option, the Zroute global router becomes plan-group-aware. The Zroute global router reads the plan group definitions and honors the pin placement constraints placed on the plan groups. The `all_routing` keyword routes all the nets in the design and preserves the hierarchy and pin constraints for the pin placement flow.

If you specify the `set_route_zrt_common_options -plan_group_aware off` command, the Zroute global router ignores the plan groups and routes the design as flat. The default is off.

Use the `route_zrt_global` command to perform plan-group-aware global routing as follows.

```
icc_shell> route_zrt_global -exploration true
```

Improving the Quality of the Global Routing for Pin Assignment

Apply the following routing guidelines to create a more accurate pin assignment and increase the effective usage of channel routing resources.

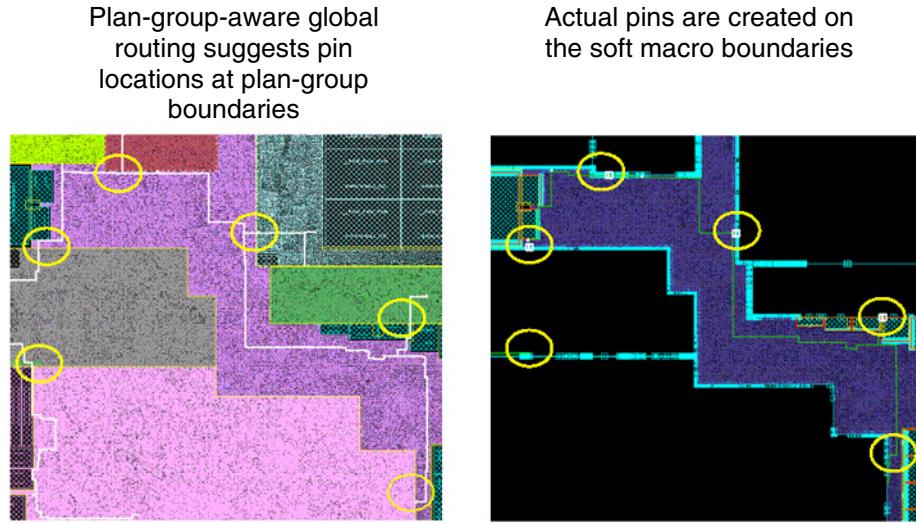
- Route nets completely inside the plan group if they exist only in the context of the modules associated with the plan group hierarchy.
- Avoid routing nets without a connection to the logic modules associated with a plan group over that plan group, unless feedthrough creation is allowed.
- Ensure that interface nets are routed across the plan group boundary at least as many times as the number of ports on those nets for the associated logic module.

The router also tries to minimize the number of routes across the plan group boundary, so that they exceed the number of ports only when necessary.

- Minimize jogs in the channels between plan groups.
- Route clock nets first, followed by signal nets.

[Figure 11-2 on page 11-10](#) shows an example of plan-group-aware global routing and pin assignment. The picture on the left shows pins placed at the boundaries where the routes intersect the plan group boundaries, as shown by the areas that are circled. The picture on the right shows the actual pins created at the boundaries of the soft macros, as shown by the areas that are circled. These are the exact locations where the routing crosses those boundaries.

Figure 11-2 Plan-Group-Aware Global Routing and Pin Assignment



Improving the Performance of the Global Router

To increase the routing speed of the plan-group-aware global router, you can set the `-plan_group_aware` Zroute common route option to `top_level_routing_only` during floorplanning. This setting prevents subsequent `route_zrt_global` and `place_fp_pins` commands from routing nets that are entirely contained within the plan groups. The following command directs the router to perform global routing only at the top level of the design.

```
icc_shell> set_route_zrt_common_options \
    -plan_group_aware top_level_routing_only
```

Another technique to increase the speed of the global router is to use fast exploration mode by specifying the `-exploration true` option to the `route_zrt_global` command. The global router runs in a lower-effort mode that requires less processing time. Use the congestion map to identify routing congestion hotspots.

```
icc_shell> route_zrt_global -exploration true
```

Placing Soft Macro and Plan Group Pins

The `place_fp_pins` command performs pin placement on the soft macros and plan groups in your design.

```
place_fp_pins
  [[-block_level] [-effort low | high] |
   -use_existing_routing]
  [-include_flip_chip_style_connections]
  [-retain_bus_naming]
  [-verbose]
  [-create_voltage_area_feedthroughs]
  [-minimum_feedthrough_length route_length]
  [soft_macros_or_plan_groups]
```

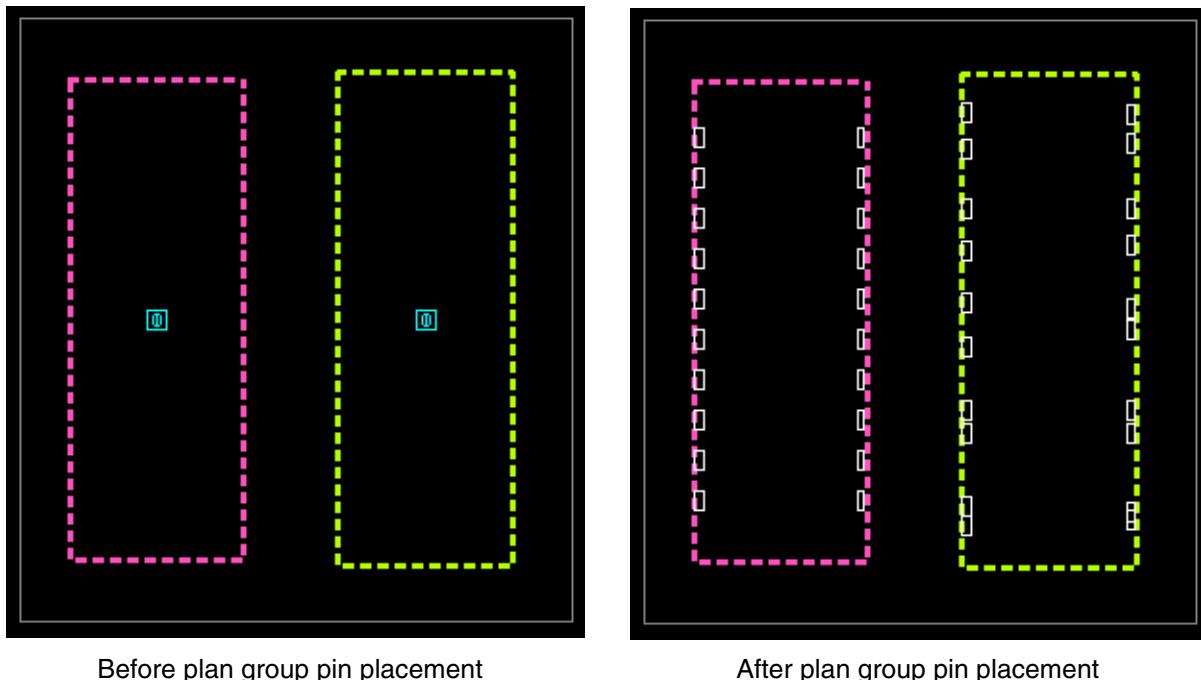
You can include plan group, black box, and soft macro reference names on a single command line and perform pin placement in a single iteration by using the `place_fp_pins` command. After performing pin placement, you can change the pin constraints and perform an iterative pin placement.

The following example places the pins for the pg1 and pg2 plan groups. The `-effort high` option specifies that the `place_fp_pins` command performs global routing in exploration mode before placing the pins. By default, flyline routing is used for pin placement.

```
icc_shell> place_fp_pins -effort high [get_cells {pg1 pg2}]
```

[Figure 11-3](#) shows a layout before and after pin placement by using the `place_fp_pins` command.

Figure 11-3 Pin Placement on Plan Groups



Performing Block-Level Pin Placement

You can assign pin locations for the top level of the current design by specifying the `-block_level` option as shown in the following command.

```
icc_shell> place_fp_pins -block_level
```

Block-level pin assignment considers the cell placement, internal connections, and pin constraints inside the soft macro cell.

Committing the Hierarchy

After finalizing the floorplan, you can commit the physical hierarchy by using the `commit_fp_plan_groups` command to convert the plan groups in your design to soft macros. The command also places pins physically on the soft macro. The placement of the pins is based on the constraints you set by using the `set_fp_pin_constraints` command and the `set_pin_physical_constraints` command, as well as the results of global routing.

For more information about committing the hierarchy, see “[Converting Plan Groups to Soft Macros](#)” on page 13-2.

During commit hierarchy, the `commit_fp_plan_groups` command converts plan groups to soft macros by performing the following actions:

- Converts the design to a two-level hierarchical design (the design netlist is partitioned to blocks and to the top level).
- Creates CEL views for each soft macro.
- Creates nets in the soft macros.
- Pushes cell rows (placement tiles) into the soft macro CEL views.
- Pushes power straps, ground straps, and standard cell preroutes down into each soft macro CEL view.
- Pushes placement blockages down into each soft macro.
- Creates pin placement based on the routing results.

Note:

Commit hierarchy can take longer to run if power and ground straps and standard cell preroutes are pushed down because each soft macro might require repair.

Plan groups that extend beyond the core boundary might have problems with pin and feedthrough placement. Make sure that plan group boundaries are within the core boundary.

The `commit_fp_plan_groups` command reports the plan groups converted to soft macros and the number of pins converted for each plan group. The command also checks the Milkyway database for any nets with the “clock” net type and assigns the pin slots for the clock nets first. If no clocks are detected by pin placement during commit hierarchy, a warning message is issued saying that no clock nets were found.

Adjusting the Position of Placed Pins

After placing soft macro and plan group pins by using the `place_fp_pins` command, you can further adjust the placement of the pins by aligning the pins with other objects or removing overlapping pins on different metal layers. The following sections describe the commands used to align pins and remove overlaps.

Aligning Soft Macro Pins

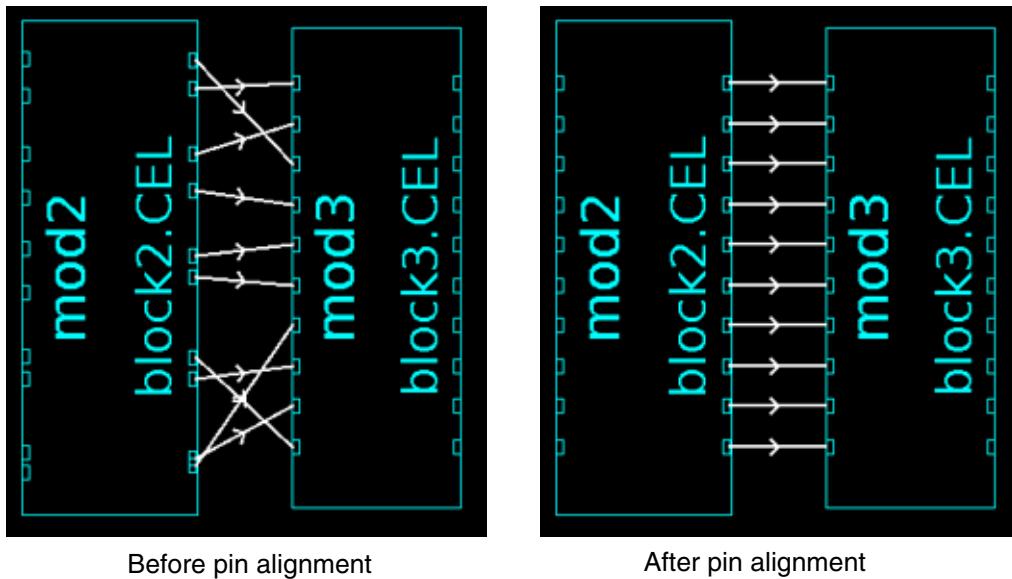
You can align a set of soft macro pins relative to the pins on a reference macro by using the `align_fp_pins` command.

```
align_fp_pins
[-align_with_child_hm_pins]
[-change_layer_width]
[-direction {center | left | right | top | bottom}]
[-fixed]
[-order_type {low_to_high | high_to_low | net_connection}]
[-propagate_single_pins]
[-reference object]
objects
```

The following example uses the `align_fp_pins` command to improve the alignment of the mod2 soft macro pins with respect to the mod3 soft macro.

```
icc_shell> align_fp_pins -reference mod3 [get_pins "mod2/*"]
```

[Figure 11-4 on page 11-14](#) shows the mod2 and mod3 soft macros before and after pin alignment.

Figure 11-4 Pin Alignment on Soft Macros

Ordering Pins Alphabetically

You can place pins in alphabetical order by using the `sort_fp_pins` command. Specify the `-reverse` option to reverse the sort order and order the pins in reverse alphabetical order. The following example sorts the dataout pins on the mod2 module in reverse alphabetical order.

```
icc_shell> sort_fp_pins -reverse [get_pins {mod2/dataout*}]
```

Removing Soft Macro Pin Overlaps

You can remove pin overlaps on soft macros and at the top level of the design by using the `remove_fp_pin_overlaps` command.

```
remove_fp_pin_overlaps
[-cells cells]
[-pins pins]
```

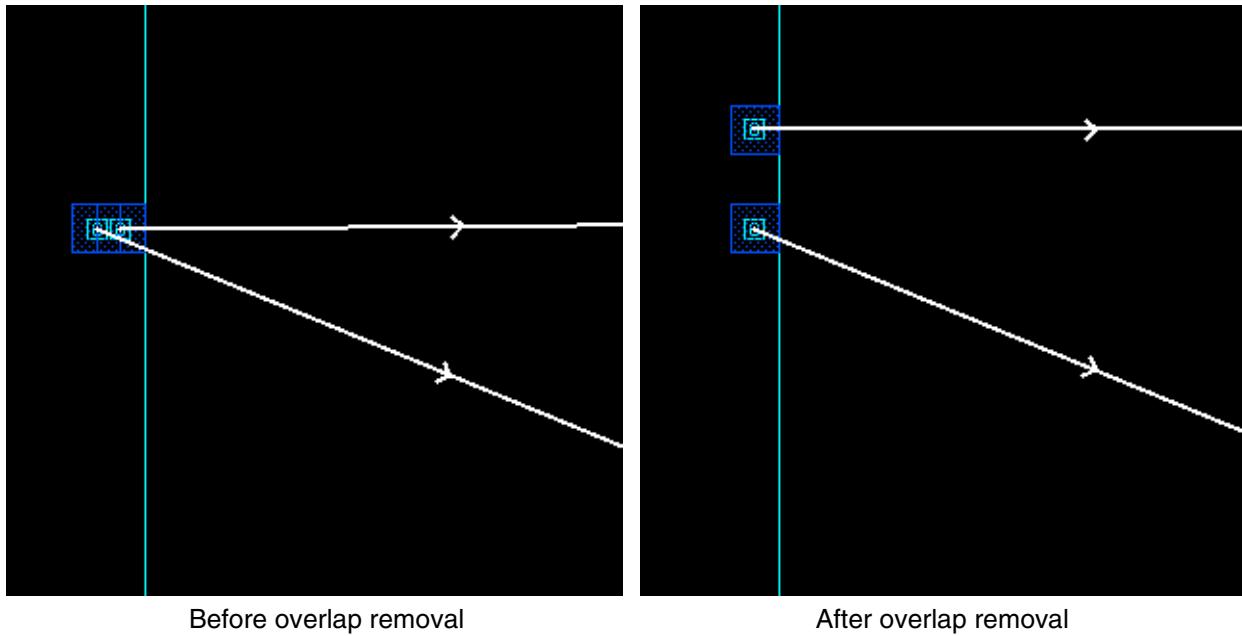
The command moves the pins to remove the overlap without creating a spacing violation. Fixed pins are not moved by the `remove_fp_pin_overlaps` command.

The following command removes overlaps on pins for the mod2 module.

```
icc_shell> remove_fp_pin_overlaps -pins [get_pins "mod2/*"]
```

Figure 11-5 shows two pins before and after overlap removal by using the `remove_fp_pin_overlaps` command.

Figure 11-5 Overlap Removal



Note:

Pin overlap removal might fail if insufficient routing tracks are available or the pin placement is overconstrained. Look for error messages in the console window after performing this operation.

Adding Feedthrough Pins

IC Compiler inserts feedthrough pins into plan groups and soft macros based on the constraints and topology you define for your design.

Excluding Feedthroughs on Clock Nets

To exclude feedthrough generation on clock nets, specify the `set_fp_pin_constraints -exclude_clock_feedthroughs` command or enable the “Exclude feedthroughs on clock nets” option in the Add Pin Assignment Constraints dialog box in the GUI.

Specifying Net and Feedthrough Topology

You can provide user-specified feedthrough topology information to the router to control pin placement. Save the net and feedthrough topology information to a file named `feedthroughMapIn`, or specify the name of the feedthrough map file by using the `-feedthrough_map_file` option to the `set_fp_pin_constraints` command. The feedthrough topology information describes how the tool should route a feedthrough net, which blocks the feedthrough net passes through, and the feedthrough pin location and metal layer used for the connection. IC Compiler can also generate a mapping file that describes the net and feedthrough topology for the design. You can edit the generated file and read in the updated file to perform subsequent iteration in feedthrough placement, or create your own feedthrough map file. The router implements the net and feedthrough topology described in the feedthrough map file.

The general format of the feedthrough map file is:

```

Net_name1:
{Feedthrough block_name block_name ...}
{NoFeedthrough block_name block_name ...}
...

Net_name2:
{object_type object_name properties} {object_type object_name properties}
{object_type object_name properties} {object_type object_name properties}
...
{Net_name3, Net_name4, Net_name5};
{object_type object_name properties} {object_type object_name properties}
{object_type object_name properties} {object_type object_name properties}
...
Signal*:
{object_type object_name properties} {object_type object_name properties}
{object_type object_name properties} {object_type object_name properties}

RegPort[*]:
{object_type object_name properties} {object_type object_name properties}
{object_type object_name properties} {object_type object_name properties}

```

In the format example, `block_name` is the name of a plan group, soft macro, or black box in your design, and `object_type` can be Block, Pin, or Port. The `Feedthrough` and `NoFeedthrough` keywords allow or prevent feedthrough routing through specified plan groups, soft macros, and black boxes in your design. `object_name` is a plan group, soft macro or black box name, a top-level port name, a port name, or a pin name for an I/O pad. You can use an asterisk (*) as a wildcard character to create a feedthrough definition for more than one net at a time. You can also specify multiple comma-separated net names within braces (`{Net_name3, Net_name4, Net_name5}`) for the same feedthrough definition.

The optional *properties* data contains additional information for each feedthrough. [Table 11-1](#) lists the supported property keywords and datatypes.

Table 11-1 Supported Feedthrough Object Properties

Keyword	Datatype
side	integer
offset	float
layers	list of layers

Note that the `offset` keyword cannot be used when you specify multiple nets by using wildcards or multiple net names within braces. The following example defines the side, offset and layers for the feedthrough in `block2`.

```
{Block block2 side 2 offset 100 layers {M3}}
```

The `Feedthrough` and `NoFeedthrough` keywords specify which plan groups, soft macros, or black boxes the feedthrough net can pass. The feedthrough net can route through any of the blocks specified by the `Feedthrough` keyword and cannot route through any of the blocks specified by the `NoFeedthrough` keyword. To force a feedthrough on a block, specify the `Block` keyword.

For example, the following feedthrough map file specifies that, for the `DATA[0]` signal, feedthroughs can be created on blocks A and B and cannot be created on blocks C and D.

```
DATA[0] :
{Feedthrough A B}
(NoFeedthrough C D)
```

If the feedthrough map file contains a duplicate specification for a net, the tool uses the last constraint in the file for that net. Note that nets that are not listed in the feedthrough map file obey the current pin assignment constraints set by using the `set_fp_pin_constraints` command. In addition, any feedthrough specification in the feedthrough map file takes precedence over the `set_fp_pin_constraints` constraint settings.

You can provide only a partial specification for a feedthrough net. You are not required to specify all connections along the route. Use the `Block`, `Pin`, or `Port` keywords to specify only a portion of the feedthrough route. IC Compiler creates feedthrough pins and completes the route, even if the connections are not specified in the feedthrough map file.

[Figure 11-6, Figure 11-7, and Figure 11-8 on page 11-19](#) show example topologies and the control files needed to produce the feedthrough topology.

Figure 11-6 Feedthrough Topology Example 1

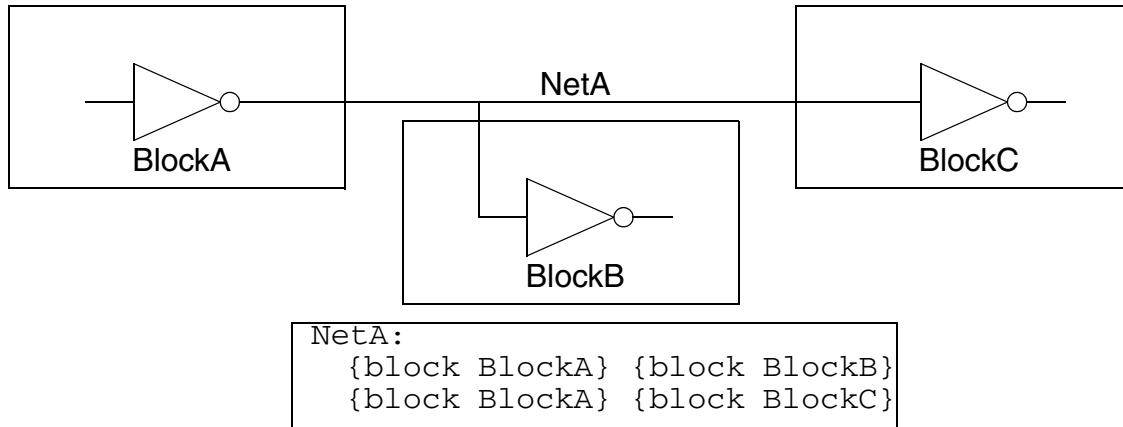


Figure 11-7 Feedthrough Topology Example 2

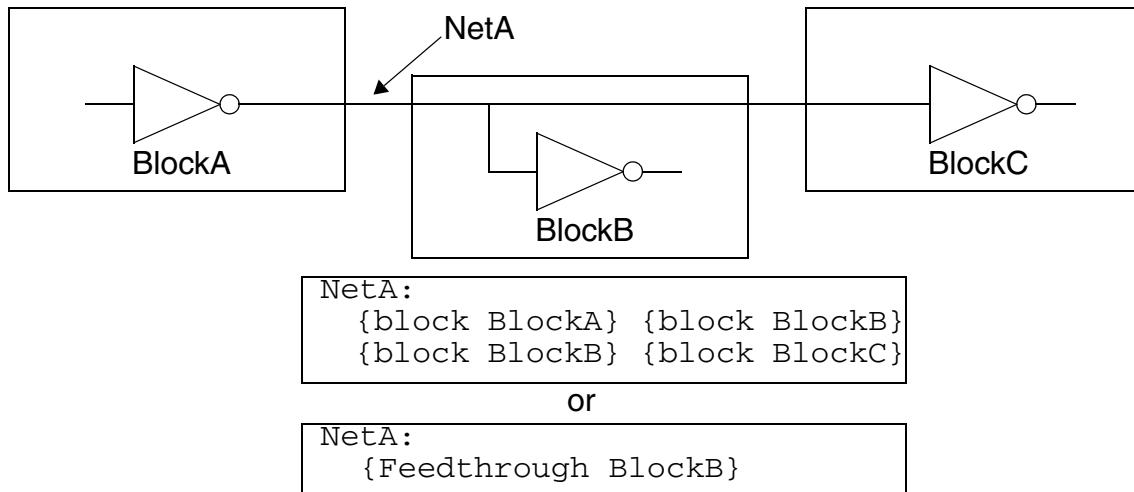
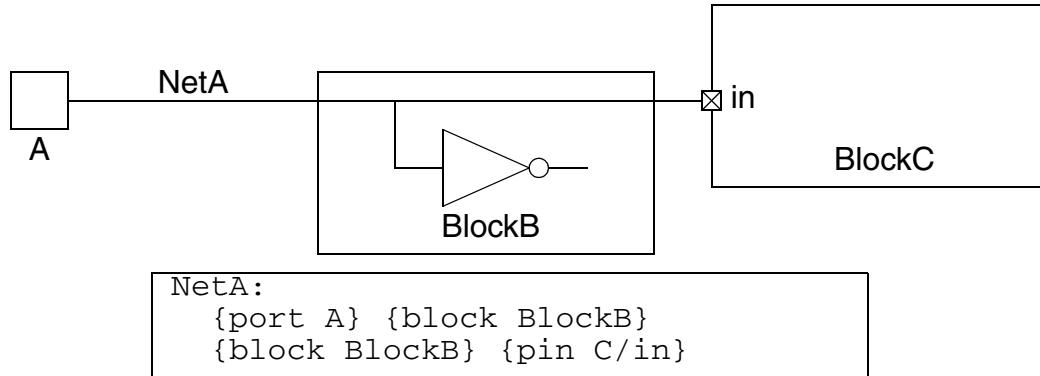


Figure 11-8 Feedthrough Topology Example 3

Note that you should specify a mapping file that does not over constrain the tool, which might degrade the QoR. Also, do not mix the `Feedthrough` and `NoFeedthrough` keywords with the `Block`, `Port`, and `Pin` keywords within the same net specification.

Use the following command to specify a mapping file named `feedthroughs.txt` that contains the desired feedthrough configuration for the current design.

```
icc_shell> set_fp_pin_constraints -allow_feedthroughs on \
      -read_feedthrough_map on -feedthrough_map_file feedthroughs.txt
```

When you run the preceding command, the `place_fp_pins` command reads the feedthrough constraints from the file named `feedthroughs.txt` in the current working directory.

Reporting Feedthrough Topology

To write out the feedthrough topology for the current design, use the `report_fp_feedthroughs` command.

```
report_fp_feedthroughs
  [-block_type {soft_macros plan_groups voltage_areas}]
  [-connectivity_detail_file file_name]
  [-feedthrough_map_file file_name]
  [-output_feedthroughs {nets ports}]
  [-preview_feedthroughs]
  [-write_connectivity_detail]
  [-write_feedthrough_map {blocks side offset layers}]
```

The following example writes feedthrough mapping information containing block, side, offset and layer constraints for plan groups and soft macros to the `ftoutput.txt` file.

```
icc_shell> report_fp_feedthroughs \
    -write_feedthrough_map {blocks side offset layer} \
    -block_type {plan_groups soft_macros} \
    -feedthrough_map_file ftoutput.txt
```

To generate a table that lists the connectivity details for each feedthrough, use the `report_fp_feedthroughs -write_connectivity_detail -block_type block_type` command. By default, the command writes out a file named `connectivityDetailOut`. You can specify the name of the output file by using the `-connectivity_detail_file` option. The specified file lists the feedthrough driver, feedthrough load, feedthrough path, and number of logical and physical connections for each feedthrough. The following `report_fp_feedthroughs` command writes out the feedthrough connectivity details for the soft macros in the design:

```
icc_shell> report_fp_feedthroughs -block_type soft_macros \
    -write_connectivity_detail \
    -connectivity_detail_file connectivity.txt
icc_shell> sh cat connectivity.txt
```

NAME	NODE(LOG)	DRIVER	LOAD(s)	CATEGORY	NODE(PHYS)	PATH
Clk	2	Dp	Alu	CUT	2	Dp>Alu
PSW[0]	1	Dp	Alu, Alu	CUT	2	Dp>Alu
PSW[2]	1	Dp	Alu	CUT	2	Dp>Alu
n939	2	Alu	Dp	CUT	2	Alu>Dp
Alu_Carry	2	Alu	Dp	CUT	2	Alu>Dp
Alu_Neg	2	Alu	Dp	CUT	2	Alu>Dp
Oprnd_A[2]	2	Dp	Alu	CUT	2	Dp>Alu
Oprnd_A[1]	2	Dp	Alu	CUT	2	Dp>Alu
Oprnd_A[13]	2	Dp	Alu	FEEDTHRU	2	Dp>Alu
Oprnd_B[5]	2	Dp	Alu	CUT	2	Dp>Alu
Oprnd_B[14]	2	Dp	Alu	FEEDTHRU	2	Dp>Alu
Op_Result[15]	2	Alu	Dp	FEEDTHRU	2	Alu>Dp
Op_Result[5]	2	Alu	Dp	CUT	2	Alu>Dp

Previewing Feedthrough Pins

You can analyze the routing pattern generated by the plan-group-aware router to review any updated pin locations after global routing and preview feedthrough pin locations. The global routing is analyzed with respect to routes crossing the boundaries of the plan groups or voltage areas. This information can be used to output a list of feedthrough nets on plan groups or voltage areas.

You can also use the GUI to preview feedthrough pins before you create them. See [“Performing Pin and Feedthrough Analysis” on page 11-21](#) for more information.

Use the `report_fp_feedthroughs -preview_feedthroughs` command to create a file that lists the nets that are pending feedthrough creation. The following example writes the list of pending feedthroughs for the plan groups in the design to the `feedthrough_nets` file.

```
icc_shell> report_fp_feedthroughs -preview_feedthroughs \
           -output_feedthroughs nets -block_type plan_groups

icc_shell> sh cat feedthrough_nets
  data_internal1_06 mod2
  data_internal1_07 mod2
  data_internal1_08 mod2
  data_internal1_09 mod2
  data_internal1_10 mod2
```

Creating Feedthrough Pins

After defining the feedthrough map for your design with the feedthrough map file, enable the tool to read the feedthrough map file with the `set_fp_pin_constraints -read_feedthrough_map` command. Route your design with the `route_zrt_global` command and insert the feedthrough pins with the `place_fp_pins` command. The `place_fp_pins` command creates feedthroughs on plan groups, soft macros, and voltage areas. For more information about creating feedthroughs in voltage areas of your design, see “[Adding Feedthrough Pins on Multivoltage Designs](#)” on page 11-23.

To create feedthrough pins on the plan groups or soft macros in your design, use the `place_fp_pins` command. In the following example, the `place_fp_pins` command creates feedthroughs on the mod2 plan group.

```
icc_shell> place_fp_pins { mod2 }
```

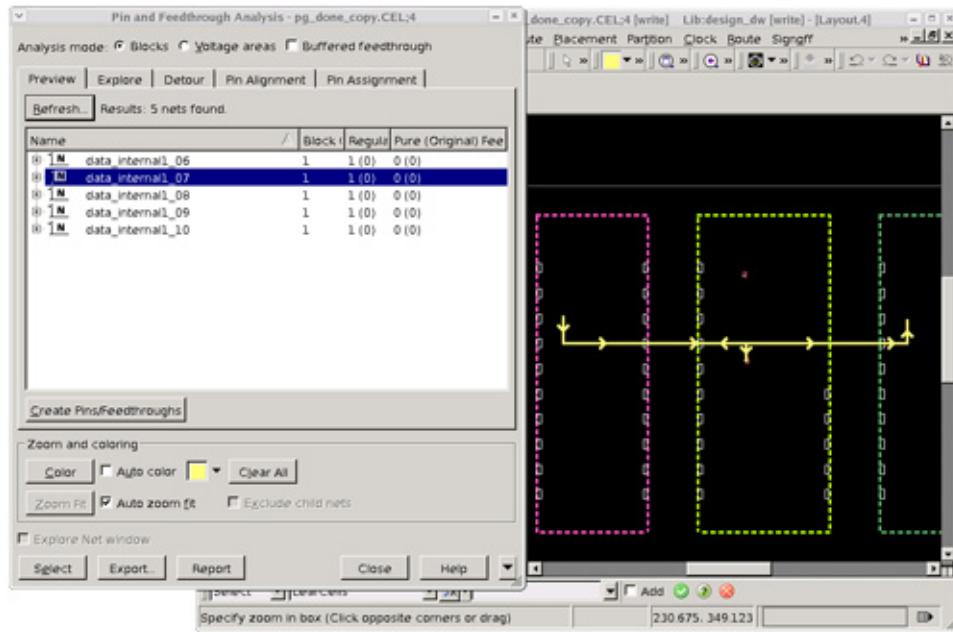
Note:

Pin and feedthrough routing is highly dependent on both pin constraints and the quality of the global routing.

If your design has terminals or I/O cells that overlap plan groups, you must use the `place_fp_pins -include_flip_chip_style_connections` command to properly create feedthroughs for all terminals.

Performing Pin and Feedthrough Analysis

You can use the GUI to examine individual feedthrough nets before and after the tool creates the feedthrough pins. To preview feedthrough nets, choose Pin Assignment > Pin and Feedthrough Analysis in the GUI, click the Preview tab and click the Refresh button to load the list of pending feedthroughs. Click individual net names to highlight the feedthrough in the layout window. [Figure 11-9](#) shows the GUI highlighting a feedthrough net.

Figure 11-9 Feedthrough Analysis GUI

After feedthrough creation, you can perform the following types of analysis on the pins in your design.

- Explore top-level nets and nets that cross between plan groups or soft macros. Click the “Explore” tab in the GUI to access this function.
- Perform detour analysis and rank nets by detour severity. Click the “Detour” tab in the GUI to access this function.
- Check pin alignment on soft macro pins. Click the “Pin Alignment” tab in the GUI to access this function.
- Perform pin assignment checks and view the results in the layout. Click the “Pin Assignment” tab in the GUI to access this function.

Removing Feedthrough Ports, Nets, and Buffering

Use the `remove_fp_feedthroughs` command to remove feedthroughs from your design.

```
remove_fp_feedthroughs
[-blocks blocks]
[-include {buffered original}]
[-nets nets]
[-voltage_areas voltage_areas]
```

You can remove the feedthrough ports and nets that exist in your design, including those created by other tools, from all soft macros, plan groups, and voltage areas. You can also remove feedthrough ports, nets, and buffers by net name or block name.

For pure feedthroughs, the tool removes all ports and the child-level net. For regular feedthroughs, one port remains as well as the child-level net.

When the tool removes a feedthrough port, the tool also deletes all the pins of that port. The tool reconnects the nets for which feedthroughs are removed at the parent level of the logic hierarchy.

Adding Feedthrough Pins on Multivoltage Designs

Multivoltage designs require special consideration when inserting feedthroughs. The following sections describe the commands needed to create feedthroughs on plan groups and voltage areas. The commands and flow depend on whether the voltage area coincides with a plan group.

Setting Voltage Area Feedthrough Constraints

You can set voltage area feedthrough constraints by using the `set_fp_voltage_area_constraints` command.

```
set_fp_voltage_area_constraints
  [-allow_feedthroughs true | false]
  [-create_feedthrough_module true | false]
  [-exclude_feedthroughs nets]
  [voltage_areas]
```

The voltage area constraints set by using the `set_fp_voltage_area_constraints` command are saved in the Milkyway database,

Creating Voltage Area Feedthroughs

The `create_voltage_area_feedthroughs` command creates feedthroughs in voltage areas based on the constraints set by using the `set_fp_voltage_area_constraints` command.

```
create_voltage_area_feedthroughs
  [-finalize_feedthroughs]
  [-include_flip_chip_style_connections]
  [-minimum_feedthrough_length route_length]
  [-output_feedthroughs {nets ports}]
  [-preview_feedthroughs]
  [-retain_bus_naming]
```

The command uses the routing information from the global router to insert feedthroughs at the points where a net crosses the voltage area boundary.

Note:

If your design has terminals or I/O cells that overlap voltage areas, you must use the `create_voltage_area_feedthroughs -include_flip_chip_style_connections` command to properly create feedthroughs for all terminals.

Creating Feedthroughs Only on Voltage Areas

To create feedthroughs only on voltage areas, enable feedthrough creation by specifying the `set_fp_pin_constraints -allow_feedthroughs on` command. Perform exploration-mode global routing by specifying the `route_zrt_global -exploration true` command. Add the feedthroughs to the voltage areas by running the `create_voltage_area_feedthroughs` command with the `-finalize_feedthroughs`, `-preview_feedthroughs`, or `-output_feedthroughs` option.

```
icc_shell> set_fp_pin_constraints -allow_feedthroughs on
icc_shell> route_zrt_global -exploration true
icc_shell> create_voltage_area_feedthroughs -finalize_feedthroughs
```

Creating Feedthroughs on Voltage Areas Coincident with Plan Groups

To create feedthroughs on voltage areas that are coincident with plan groups, enable feedthrough creation by specifying the `set_fp_pin_constraints -allow_feedthroughs on` command. To place the plan group pins and create the feedthroughs, specify the `place_fp_pins -effort high` command.

```
icc_shell> set_fp_pin_constraints -allow_feedthroughs on
icc_shell> place_fp_pins -effort high
```

Creating Feedthroughs on Voltage Areas Not Coincident with Plan Groups

To create feedthroughs on voltage areas that are not coincident with plan groups, you must enable feedthroughs on both plan groups and voltage areas. Enable feedthrough creation on plan groups by specifying the `set_fp_pin_constraints -allow_feedthroughs` on command. Enable feedthrough creation on voltage areas by specifying the `set_fp_voltage_area_constraints -allow_feedthroughs true` command. To place the plan group pins and create the feedthroughs, specify the `place_fp_pins -effort high` command.

You can specify the `-include_flip_chip_style_connections` and `-retain_bus_naming` options together with the `-create_voltage_area_feedthroughs` option to the `place_fp_pins` command. The `-include_flip_chip_style_connections` and `-retain_bus_naming` options are honored during feedthrough creation in voltage areas.

```
icc_shell> set_fp_pin_constraints -allow_feedthroughs on
icc_shell> set_fp_voltage_area_constraints \
    -allow_feedthroughs true
icc_shell> place_fp_pins -effort high \
    -create_voltage_area_feedthroughs [get_plan_groups]
```

Reporting Voltage Area Constraints

Use the `report_fp_voltage_area_constraints` command to display the options used to control feedthrough creation on the voltage areas in your design. The following example reports the existing voltage area feedthrough constraints for the current design.

```
icc_shell> report_fp_voltage_area_constraints
Voltage Area           Feedthroughs
-----
val                   Allowed
DEFAULT_VA            Not Allowed
```

Removing Voltage Area Constraints

Use the `remove_fp_voltage_area_constraints` command to reset the feedthrough constraints for the specified voltage areas to their default values.

Performing Pin Placement With Multiply Instantiated Modules

The pin placement flow supports designs with multiply instantiated modules. The following sections describe the special considerations for pin placement with multiply instantiated modules. For information about determining which modules in your design are multiply instantiated modules, see “[Identifying Multiply Instantiated Modules](#)” on page 6-63.

Setting Pin Assignment Constraints for Multiply Instantiated Modules

Use the `set_fp_pin_constraints` command or choose Pin Assignment > Pin Constraints in the GUI to apply pin constraints to multiply instantiated modules. For more information, see “[Specifying Global Pin Assignment Constraints](#)” on page 11-2. The same pin assignment constraints used in the pin placement flow are assigned on all instances of a multiply instantiated module set to avoid conflicts and potential problems during plan-group-aware routing, and pin placement during commit hierarchy. If an instance of a multiply instantiated module set has different pin assignment constraint settings, a warning message is issued during the `set_fp_pin_constraints` step.

Note:

No feedthroughs are allowed for multiply instantiated modules.

Performing Plan-Group-Aware Routing for Multiply Instantiated Modules

The plan-group-aware router performs global routing for multiply instantiated modules. (See “[Previewing Feedthrough Pins](#)” on page 11-20.) You can analyze the results for each plan group based on wire length, timing, and congestion reports. Based on this analysis, you should be able to determine which instance to use as the master instance for the multiply instantiated module.

The plan-group-aware router automatically turns off feedthroughs on multiply instantiated modules.

Note:

If any multiply instantiated or unique modules are instantiated as CEL views in the design and pin assignment has not been performed on them, the plan-group-aware router issues an error message.

Selecting a Master Instance

Before committing a multiply instantiated module instance, you must first choose a master instance by using the `select_mim_master_instance` command. The command designates an instance as the “master” instance among a set of multiply instantiated modules. All instances of the multiply instantiated module use the same specified master instance. The master instance information is saved as a plan group property in the Milkyway database.

The following command selects the `block1` instance as the master instance.

```
icc_shell> select_mim_master_instance [get_cells {block1}]
```

Note:

If you do not select a master instance before committing the hierarchy, the `commit_fp_plan_groups` command issues a warning and randomly chooses the first multiply instantiated module instance as the master instance.

Committing a Master Multiply Instantiated Module Plan Group

The `commit_fp_plan_groups` command converts plan groups to soft macros. Plan groups for multiply instantiated module instances are also converted by this command. The master instance plan group is converted to a soft macro, and the instances from the same multiply instantiated module are converted to instances of the master soft macro.

Pin placement occurs during commit hierarchy for multiply instantiated module instances based on the pin locations determined by the `place_fp_pins` command for the chosen master instance and the plan-group-aware routing results. Pin placement supports multiply instantiated module instances mirrored along both the x- and y-axes.

Using Pin Guides for Pin Placement

Pin guides create a boundary for pin placement. After you create a pin guide, pins for the specified blocks can only be placed within the pin guide.

Creating Pin Guides

You can create rectangular or rectilinear pin guide shapes by using the `create_pin_guide` command.

```
create_pin_guide
  -bbox bounding_box_rect |
  -boundary rectilinear_boundary
  [-exclusive]
  [-name pin_guide_name]
  [-parents soft_macro_or_plan_group]
  objects
```

You can create pin guides on soft macros, plan groups, or the current top-level cell to control the placement of pins or ports on soft macros or plan groups. The tool creates pin guides based on net names or pins names. The `place_fp_pins` command determines the best location for the pins constrained by the pin guide.

Note:

Pin guides must intersect the module boundary in one contiguous area. Multiple intersections imply that you want a feedthrough on the net. If there are two or more external connections for the net, it is interpreted as a “real” feedthrough, that is, one of the ports is the original port on the block, and the other is the new port.

The routing for the nets that have ports associated with the pin guides is constrained to cross the corresponding soft macro or plan group boundaries through the interior of the pin guide. As a result, pin assignment honors the pin guide regions and places pins within the corresponding plan group or soft macro edge segment. If the pin guide overlaps more than one edge of the soft macro, the router can choose to go through any edge segment that overlaps that pin guide.

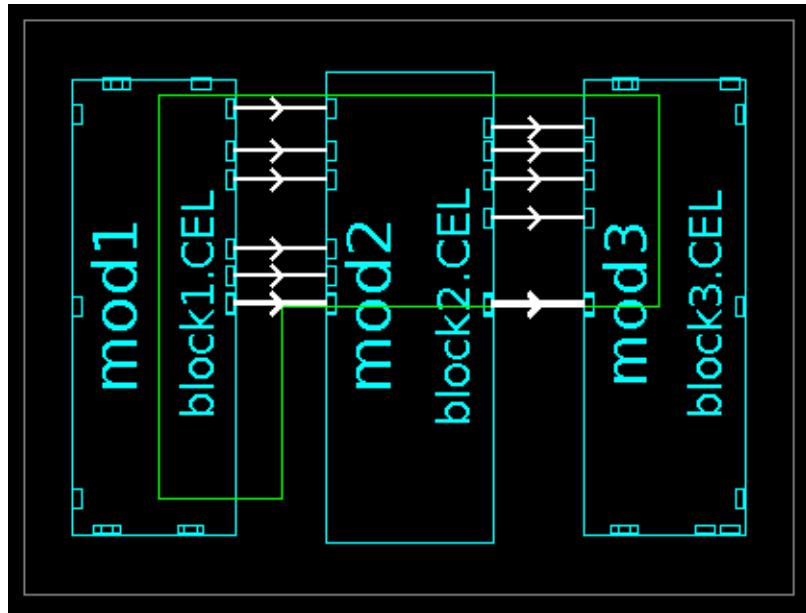
The pin guide is a polygon associated with a group of ports or pins and the overlapping plan group or soft macro. Only one pin guide can be associated with any given pin on a soft macro or plan group. If the pin guide does not overlap any soft macro or plan group, or if it does not have any ports associated with it, it is disregarded during routing and pin assignment.

The following command creates a rectilinear pin guide that constrains the pins connected to the nets “`data_internal*`”. The pin guide intersects three plan groups: `mod1`, `mod2`, and `mod3`. The coordinates `{260 250}` `{292 250}` `{292 300}` `{390 300}` `{390 355}` `{260 355}` and `{260 250}` outline the pin guide.

```
icc_shell> create_pin_guide -name {pg1} \
-boundary {{260 250} {292 250} {292 300} {390 300} {390 355} \
{260 355} {260 250}} -parents {mod1 mod2 mod3} \
[get_nets data_internal*]
```

[Figure 11-10 on page 11-29](#) shows the design with pins placed by using a pin guide created by the previous `create_pin_guide` command. The pin guide rectilinear region is highlighted.

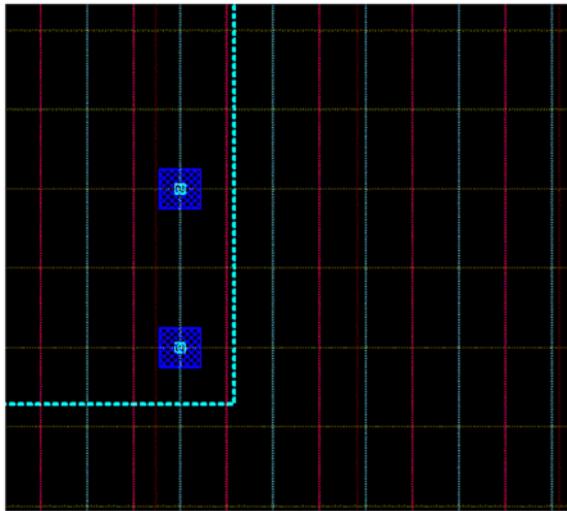
Figure 11-10 Pin Placement With a Pin Guide



Using Pin Guides for Block-Level Off-Edge Pin Placement

You can restrict the placement of top-level pins in your design by using the `create_pin_guide` and `place_fp_pins -block_level` commands. Pin placement is restricted to the boundary of the pin guide you define. You can use the pin guide to create a region for off-edge pins; the tool places pins on the wire track. [Figure 11-11 on page 11-30](#) shows an example of an off-edge pin.

Figure 11-11 Region Constraints on Off-Edge Pins



Pins placed inside the pin guide
on the wire track

You can use the `set_fp_pin_constraints -allowed_layers layers` command to restrict off-edge pin placement to a specified set of metal layers. The `layers` argument is a collection of metal layers ordered consecutively.

Creating Pin Guides for Feedthroughs

You can specify the locations of feedthroughs on a soft macro or plan group by creating pin guides for the feedthroughs.

Note:

The feedthrough guide controls only where the pins are placed on the block boundaries.
It does not control the routing inside the block.

If a pin guide shape passes through a soft macro or plan group, and if the feedthrough net does not already touch the soft macro or plan group, a “pure” feedthrough is created. A “pure” feedthrough has no internal connections with the block it passes through. Conversely, if the feedthrough net has one or more cell instances within the block (soft macro or plan group) that the net is connected to, one side of the feedthrough is counted as the original port and the other side is a “regular” (connected) feedthrough. To ensure that the connectivity of the feedthrough is clear, the pin guide shape should touch the port, plan group, or soft macro that is connected to the net.

To generate feedthroughs that honor pin guides,

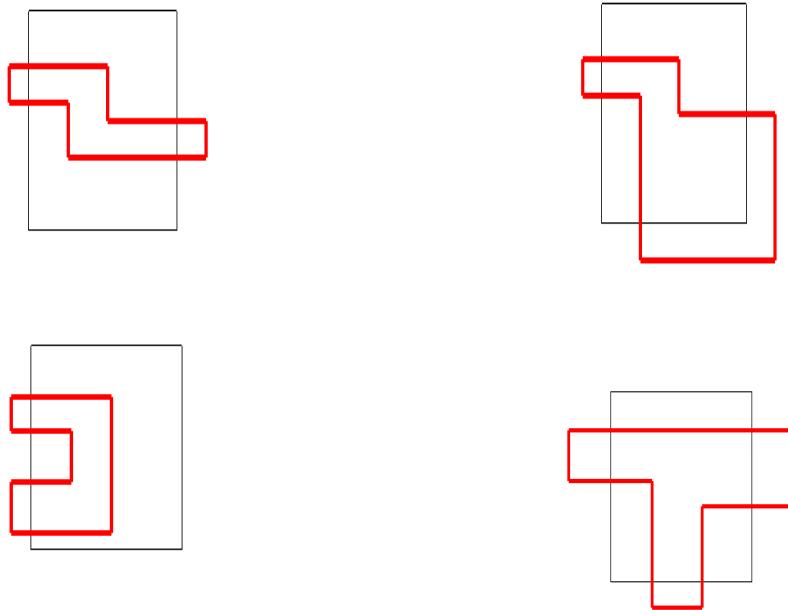
- Enable feedthrough creation by running the `set_fp_pin_constraints -allow_feedthroughs` on command
- Add the feedthrough block to the list of parent objects by specifying the `create_pin_guide -parents object` command when creating the pin guide.

Feedthrough guides must meet the following criteria:

- A feedthrough pin guide on a block must have at least two connections on the net outside the block. If it does not, the feedthrough pin guide is ignored and a warning message is issued.
- A feedthrough pin guide must intersect the block boundary in at least two disjoint areas.

IC Compiler supports the rectilinear feedthrough guide shapes shown in [Figure 11-12](#).

Figure 11-12 Rectilinear Feedthrough Pin Guide Shapes



Reporting Pin Guide Settings

You can report the current pin guide settings by using the `report_pin_guides` command. Specify the `-pins` option to get detailed information about pin guides associated with specified pins or specify the `-nets` option to get detailed information about pin guides associated with specified nets.

Checking for Off-Edge Pin Placement

You can ensure that all pins are on an edge by using the `check_fp_pin_assignment -off_edge` command. You can check for pins that are not inside the associated pin guide by using the `check_fp_pin_assignment -outside_pin_guide` command.

Checking Pin Assignment and Pin Alignment

You can analyze and evaluate the quality of the pin placement results by checking the placement and alignment of soft macros and plan group pins in the design. You can refine the pin assignment based on the results.

Checking the Pin Assignment

You can check or verify the placement of soft macro or plan group pins in a design by using the `check_fp_pin_assignment` command.

```
check_fp_pin_assignment
  [-block_level [-net_routing_constraints]]
  [-layers]
  [-macro_type soft_macros | hard_macros | all]
  [-match_pin_shapes]
  [-missing]
  [-nets net_collection]
  [-no_stacking]
  [-off_edge]
  [-outside_pin_guide]
  [-pin_preroute_spacing]
  [-pin_spacing]
  [-pin_type signal_pins | pg_pins | all]
  [-shorts]
  [-single_pin connected | unconnected | all]
  [-wiretrack [-preferred_wiretrack_only] [-allow_half_wiretrack]]
  [objects]
```

The following example checks the pin assignment in the current design for pin spacing violations by using the `check_fp_pin_assignment -pin_spacing` command.

```
icc_shell> check_fp_pin_assignment -pin_spacing
*****
Report: check_fp_pin_assignment
Design: top
Version: E-2010.12
Date: Tue Oct 5 16:34:29 2010
*****
----- Start Of Pin Spacing Checks -----
Error: Pin to pin spacing constraints violated in soft macro "mod2":
  1:pin "dataout_02" connected to net "data_internal2_02"
    BBOX: (351.0500 251.7500) (351.1500 251.8500)
  2:pin "dataout_01" connected to net "data_internal2_01"
    BBOX: (351.1000 251.7500) (351.2000 251.8500)
----- End Of Pin Spacing Checks -----
{mod2/dataout_01 mod2/dataout_02}
```

Checking the Pin Alignment

You can check the pin alignment in your design by using the `check_fp_pin_alignment` command.

```
check_fp_pin_alignment
[-detour]
[-nets net_names]
[-report_nets]
[-tolerance value]
```

You can view a list of nets with pins that are not optimally aligned by pin assignment. The list displays the total number of two-connection nets among the specified nets that have at least one of their two connections inside a soft macro. The command also reports the number and percentage of misaligned nets among the two-connection nets.

The following example lists pins in the design that have a detour tolerance greater than 10 percent. The following pin detour report displays the nets where the bounding box of the standard cell macro pins on the net is smaller than the bounding box of all standard cells and soft macros on the net.

```
icc_shell> check_fp_pin_alignment -report_nets -detour -tolerance 10
*****
Report: check_fp_pin_alignment detour check
Design: test
Version: F-2011.09
Date: Thu Jun 30 17:44:11 2011
*****
3 pin detours.
***** Nets with pin detour, tolerance = 0.100000 *****
data_internal1_09
data_internal1_02
data_internal1_07
*****
```

12

Performing Timing Budgeting

During the design planning stage, timing budgeting is an important step in achieving timing closure in a physically hierarchical design. The timing budgeting determines the corresponding timing boundary constraints for each top-level soft macro or plan group (block) in a design. If the timing boundary constraints for each block are met when they are implemented, the top-level timing constraints are satisfied.

Timing budgeting distributes positive and negative slack between blocks and then generates timing constraints in the Synopsys Design Constraints (SDC) format for block-level implementation.

Timing budgeting distributes the timing constraints from the top level to the block level, creating a new constraint set for each of the top-level blocks in the design. Budgeting also allocates slack between blocks for timing paths crossing the block boundaries. Timing budgeting propagates the timing constraints downward one hierarchical level at a time. Timing budgeting also propagates timing exceptions to block-level constraint sets.

You can use the timing budgeting feature in virtual flat mode to generate constraints for plan groups.

During timing budgeting, you can also create quick timing models for soft macros and plan groups and load them into your design.

This chapter includes the following sections:

- [Timing Budgeting Prerequisites](#)
- [Performing Pre-Budget Timing Analysis](#)
- [Running the Timing Budgeter](#)
- [Generating a Quick Timing Model From a CEL View](#)
- [Performing Timing Budgeting On Plan Groups](#)
- [Budgeting With Multiply Instantiated Modules](#)
- [Performing Budgeting on Multiple Scenarios](#)
- [Performing Distributed Timing Budgeting for Multiple Scenarios](#)
- [Performing Hierarchical Signal Integrity Budgeting](#)
- [Performing Post-Budget Timing Analysis](#)
- [Performing Clock Latency Budgeting](#)

Timing Budgeting Prerequisites

Before a cell can be budgeted, it has to meet the following requirements:

- It must be floorplanned with all cells legally placed. The closer the floorplan is to the final layout, the better the budgeting results.

Note:

A globally routed design, while not required, results in budgets that are more accurate.

- The top-level timing constraints must be loaded.
- The timing information available must be sufficient to successfully execute the timing analyzer on the cell to be budgeted. This requires complete timing views for all top-level soft and hard macros.

You can run budgeting on a design that does not have timing models for all soft macros. If a soft macro does not have a timing model, the paths that interface with that macro are automatically marked as false paths.

- The timing information available must be sufficient to successfully execute the timing analyzer on the cell to be budgeted. This requires complete timing views for all top-level soft and hard macros.

You can run budgeting on a design that does not have timing models for all soft macros. If a soft macro does not have a timing model, the paths that interface with that macro are automatically marked as false paths.

- Run a timing report on your top-level design before performing timing budgeting. If design errors are reported during the timing report, budgeting cannot be run successfully.
- Timing paths should have reasonably small slack values.

Budgeting is effective only when your design is close to meeting timing. It would be very difficult to meet timing if, for example, the worst negative slack was 10 nanoseconds and the clock period was also 10 nanoseconds. Timing is considered reasonable, however, if the worst negative slack is 10 percent or less but no more than 20 percent of the clock period. If it is more than 20 percent of the clock period, you should carefully review the results of the path timing report to determine the cause of the timing violations.

The most common cause of incomplete or missing constraints in the output timing boundary files is that no timing path passes through the port for which constraints are either incomplete or missing. This might be intended if false path specifications are loaded. However, it might be caused unintentionally, for example, by incomplete or incorrect timing views or by dangling nets.

Performing Pre-Budget Timing Analysis

You can perform pre-budget timing analysis on your design. The timing analysis is applied to blocks (plan group or soft macro) or the top-level design. Pre-budgeting timing analysis allows you to handle early checking of the timing rules and constraints set in the Synopsys Design Constraints (SDC) file before your design is submitted to the budgeting process.

By providing feedback on design and timing information, the timing checking analysis tool can help you determine the feasibility of your design and its timing constraints.

You can generate a report file that contains additional design information for validating the budgets assigned to each block in your design. You can use this report to analyze why certain budgets are assigned to each block port and to debug the budgets generated by the IC Compiler design planning timing budgeter.

The design information in the report file includes, but is not limited to, clocks in the design, timing constraints in the design, timing exceptions in the design, delay distribution on the worst timing path through the block ports, pins tied high or low, high negative slack on the timing path, and so on.

To generate a pre-budgeting timing analysis report file,

1. Choose **Timing > Floorplan Timing Environment Check**.

The Timing Environment Analysis Options dialog box appears.

Alternatively, you can use the `check_fp_timing_environment` command.

2. Select the Budgeting Analysis tab. This is the default.
3. Enter the hierarchical cell name or plan group name and the pin names on which to apply the budgeting analysis.
4. Select the options to use for performing budgeting analysis. The options you specify determine the type of information that is written to the output report file.

You can show reports on:

- Block pin statistics – For each block, prints the number of hierarchical pins that can and cannot be budgeted and the total number of pins on the block. A hierarchical pin is budgeted if timing constraints or exceptions are propagated to it. The default is on.
- Unbudgetable pins – Reports why pins cannot be budgeted. The default is on. A hierarchical pin might not be budgeted for the following reasons:

The pin is not connected to a net.

The pin is not connected to a top-level net, but is connected to an internal net.

The pin is not connected to an internal net, but is connected to a top-level net.

Only fixed delay exists on one side of the pin. (There is nothing to budget, although there is an input or output delay constraint on these pins.)

Timing paths through pins are not constrained.

The pin is constrained by a user-defined budget.

The pin is connected to a power or ground net, or it has a `set_case_analysis` constraint on it.

The pin is either an input or output clock port.

- Unconstrained pins – For each block to be budgeted, lists the pin names that are not constrained. A pin is unconstrained if no timing path goes through it. The default is on.
- Exception pins – For each block to be budgeted, lists the pin names that have timing exceptions propagated to it. The exceptions are `set_false_path`, `set_multicycle_path`, `set_min_delay`, or `set_max_delay`. The default is on.
- Static logic pins – For each block to be budgeted, lists the pins that are set to a static logic state as a result of a case analysis statement. It reports if a block pin is tied high (1) or low (0) by a power or ground net in the logical netlist. It also reports if a block pin is set to a specific state by `set_case_analysis`, `set_logic_one`, or `set_logic_zero` constraints. The default is on.

- Number of pin connections – Reports the number of pin connections from the startpoint and the endpoint of the timing paths going through one block to other blocks, through top-level standard cells, through pad cells, through top-level ports, and through ports on the plan group.

Enter the number of pin connections to display for each type of net connection. The default is 1.

- Delay violating cells – Lists the top-level cells of interface paths that have cell delays greater than the specified percentage of the captured clock period. It traverses all the critical paths per clock domain through the block pin.

The *percent* argument is required and must be greater than or equal to zero.

- Report to file – Prints the report to an output file in a single-line format. Enter the name of the output file. The default is off.
- Output in table format – Formats the report in a table. If the report is in a table, it is easier to read, but it might be difficult to parse using automated scripts. The default is off.

5. Select the Timer and Bottleneck Analysis tab to generate timing and bottleneck reports.

You can show timing and bottleneck reports on:

- Zero wire delay – Performs zero wire delay analysis and reports the negative-slack paths that have slack less than the negative percentage of the captured clock period. This enables timing analysis in a special mode in which the wire delays are set to zero. The default is off.

Specify a slack limit. The slack limit is the threshold percentage of the end clock period for each negative-slack path. A timing path is printed in the output report file if its slack is less than the negative percentage of the path's captured clock period.

- Virtual IPO – Reports timing analysis results based on virtual in-place optimization. The default timing report from this option uses a slack threshold of zero. The default is off.

Specify a slack limit for the timing paths in the virtual in-place optimization timing report. Only paths that have slack less than the slack percentage of the capture clock period are reported.

- Bottleneck cells – Reports bottleneck cells. It lists the cells in the design that contribute to multiple timing violations. It lists the leaf cell, its reference, and the number of paths through the leaf cell that have timing violations. Based on this report, you can check the fanin and fanout logic to determine the possible cause of the timing bottleneck. The default is off.
- Run Virtual IPO – Disable this option if you do not want virtual in-place optimization to run before reporting bottleneck cells. The default is to run virtual in-place optimization.

- Slack limit – You can specify the slack limit for reporting bottleneck cells only. Only timing paths having slack less than the slack limit are explored to locate bottleneck cells. The default is 0.
- Maximum number of cells – You can specify the maximum number of bottleneck cells to report. This allows you to limit the number of bottleneck cells reported for a design. The default is 20.
- Maximum number of paths – You can optionally specify a maximum number of paths to report.

6. Click OK.

The following information is printed in the report file for the whole design:

- Negative-slack paths based on zero wire delay
- Bottleneck cells
- Timing report based on virtual in-place optimization

The remainder of the report is divided by soft macro or plan group. All the timing environment information related to one soft macro or plan group is printed together.

Running the Timing Budgeter

You can run the timing budgeter to perform proportional timing budgeting (the default) or advanced timing budgeting based on virtual in-place optimization for plan groups or soft macros in the design.

Proportional budgeting allocates positive and negative slack based on the actual physical circuits in each path. It also allocates slack on the path based on the delay on the path portions within individual blocks. Timing budgets are distributed proportionally to the worst or best case delay that is actually present for each path (per clock domain) inside the top-level soft macros. Portions of interface paths which are hierarchically in the top level are considered fixed delays just as hard macros are considered as fixed delays. This means that all slack for interface paths is budgeted solely to blocks.

After slack is allocated between blocks, the design is processed to generate a timing constraints file in the Synopsys Design Constraints (SDC) format for each block that has a timing model.

To run the timing budgeter,

1. Choose Timing > Allocate Budgets.

The Allocate Budgets dialog box appears.

Alternatively, you can use the `allocate_fp_budgets` command.

2. Set the options, depending on your requirements.

- Hierarchical cells – Select this option to specify a list of hierarchical cell names for budgeting. By default, all plan groups and soft macros in the current design are budgeted.
- Black box cells – Select this option if you want a list of black box cells to be budgeted with black box timing models. The default is off.

Enter the names of the black box cells in the text box. Black box cells are not budgeted unless they are included in this text box. Black box cells that have timing models and are omitted from the text box are treated as hard macros.

- Fixed delay objects – Select this option to specify that certain objects that are implemented have fixed delays that will not change in subsequent optimizations. The list of objects includes plan groups, soft macro cells, hard macro cells, black boxes, hierarchical cells for plan groups, pins of soft macros with hierarchical models, and pins of hierarchical cells for plan groups.

The delay allocated to the timing paths passing through these pins during budgeting is equal to the current delay.

Select the Plan groups option if you want a list of cells to be treated as hard macros for budgeting purposes. The tool skips slack allocation for cells designated as fixed delay cells. Timing budget that is usually distributed to the fixed delay cell is distributed to other soft macros on the same timing path. The default is off.

- Output file name format specification – Enter the directory and naming style for the output SDC constraints file. The syntax of the string specifies the directory to write the SDC files and the type of names to output. If you specify `outputdir/i.sdc`, IC Compiler writes instance names. The string `outputdir/m.sdc` requests the tool to write reference names. One consolidated SDC file is written for designs that contain multiply instantiated modules. See the man page for more information.
- Interblock logic – Select this option to fix top-level timing budget values during budgeting. If you deselect this option, the tool performs proportional timing budgeting at the top level as well as plan groups and soft macros.
- Exploration mode – Select this option to perform budgeting in exploration mode. See “[Performing Fast Time to Budget Analysis](#)” on page 12-8 for more information about this option. The default is off and the tool performs budgeting in the normal floorplanning flow.
- Incremental budgeting – Select this option to perform budgeting in incremental mode. During incremental budgeting, the timing budgeter retrieves block implementation information from designs with hierarchical models and budgets the delays using this

information. The block-level slack of the worst timing paths through the block pins is considered. The tool tightens budgets for pins associated with positive slack numbers, and relaxes budgets for pins with negative slack numbers.

Note:

Use this option only on designs with interface logic models that were created with the `create_ilm`, `create_ilm_models`, or `create_block_abstraction` commands. Designs that contain interface logic models commands contain information about the block-level slack of the worst path through the hierarchical pins. If the design contains plan groups or if the ILMs in the design do not have pins with negative slack information, incremental budgeting issues an error message and stops.

- Split long lines – Select this option if you want the tool to split long SDC command lines in the budget file by breaking the line and inserting the Tcl continuation character (\) at the end of the line. The default is on.
- Write constraints for partial netlists – The `allocate_fp_budgets` command does not create partial constraint files by default. Specify the `-print_partial_constraints` option to generate partial constraint files for blocks with a partial netlist. Partial netlists are commonly used in the on-demand-loading flow, the trace mode flow, and the hierarchical flow using ILMs or block abstraction models. The partial constraint file for blocks with a partial netlist is incomplete and contains only constraints for the portion of the block netlist at the top level.
- Enable QTM model generation – Select this option to enable quick timing model generation for the budgeted plan groups or soft macros. (This option is equivalent to the `create_qtm_model` command.) The delay arcs and the constraint arcs that are created in the quick timing models represent the budgets on the plan groups or soft macros.

You can load the quick timing models for the soft macros or plan groups after they are committed to soft macros. You can also generate a top-level timing report to check the QoR of your budgets and identify potential problems before running optimization on the individual blocks.

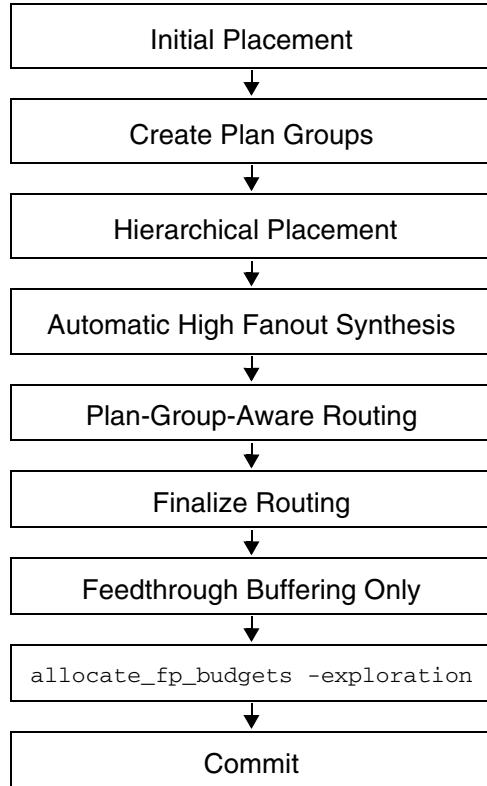
- Output QTM directory – Enter the name of the directory into which the quick timing model files are written after the timing budgeting is finished. The default is the current directory.

Performing Fast Time to Budget Analysis

You can perform timing budgeting on an early design version using the `-exploration` option to the `allocate_fp_budgets` command. Using this option, `allocate_fp_budgets` adjusts input and output delay and load values for block constraints. This option eliminates the need for an optimized netlist as a prerequisite for timing budget exploration.

[Figure 12-1](#) shows a typical design planning flow for timing budget exploration.

Figure 12-1 Timing Budget Exploration Flow



Checking the QoR of the Timing Budgets

After you have loaded the quick timing models into your design, you can check the QoR of the timing budgets before running optimization for each individual block by using the `report_timing` command to create a top-level timing report to see if there are any paths with negative slack.

If the current budgets are either overconstrained or underconstrained, the timing report shows such paths as negative or positive slack. To check the results for any negative slack paths in the timing report, run a timing report for that path on a saved design with ILM or block abstraction models.

Zero slack budgeted paths are reported in the timing report with a slack of zero. Because the quick timing represents the timing of the blocks if they met the generated SDC budgets, the top-level timing should converge to a slack of zero.

Generating a Quick Timing Model From a CEL View

You can generate quick timing models for the current design or the soft macro cells inside the current design. The quick timing models are created based on the clock information provided in the given SDC files. The delay information in the quick timing models is based on the real delays in the design.

To generate quick timing models,

1. Choose Timing > Generate QTM Model.

The Generated QTM Model dialog box appears.

Alternatively, you can use the `generate_qtm_model` command.

2. Set the options, depending on your requirements.

- Soft macro cells or current design – Select the soft macro cells, and specify a list of soft macro names from which to create quick timing models. By default, a quick timing model is created for the current design.
- Cell name or SDC file name – Specify the list of cell names for the soft macros in the current design for which you want to create quick timing models.

Specify the SDC files which contain clock information for the design or the soft macros. When creating quick timing models for the soft macros inside the current design, use this option together with the “soft macro cells” option and specify a list of cell names and SDC file names with this option to provide one SDC file for each soft macro cell.

- Output QTM directory – Specify the directory to write the quick timing model files. By default, the tool writes the files to the current working directory.

3. Click OK or Apply.

Generating a Quick Timing Model for the Partitions of Large Designs

Large designs that might not fit as flat designs in IC Compiler are automatically partitioned and floorplanned. You can create a quick timing model for the partitions of large designs. The input to the quick timing model that is generated is the CEL view of the partition. The quick timing model, along with the top-level design, should fit in a design that is floorplanned in IC Compiler. The quick timing model timing generation for all CEL view partitions can occur in parallel.

The generated quick timing model output for the partitions of large designs have the following:

- Interface ports of the output quick timing model

The interface ports of the output quick timing model are based on the ports of the original CEL view. The direction of the quick timing model ports is the same as the direction of the ports in the Milkyway design.

- Sequential delay arcs (clock to output delay arcs)

If the worst slack path through the output port does not start at the input port of the design, a sequential delay arc is created at the output port.

- Combinational delay arcs (input to output delay arcs)

If the worst slack path through the output port starts at the input port of the design, a combinational delay arc is created.

- Setup and hold arcs (clock to input constraint arcs)

The worst slack paths for maximum and minimum delay mode are used to create setup and hold arcs.

- Load information about input pins and drive information about output pins

The load value on the input port is the sum of all the pin and wire capacitances that are connected to the input port. The driving cell of the output port is the cell that directly drives the net connected to the output port.

Quick Timing Model Command Summary

[Table 12-1](#) summarizes the commands in the quick timing model flow.

Table 12-1 IC Compiler Quick Timing Model Command Summary

To do this action	Use this command
Begin defining a quick timing model.	<code>create_qtm_model</code>
Create a constraint arc.	<code>create_qtm_constraint_arc</code>
Create a delay arc for a quick timing model.	<code>create_qtm_delay_arc</code>
Create a drive type in a quick timing model description.	<code>create_qtm_drive_type</code>
Create a generated clock for the model.	<code>create_qtm_generated_clock</code>

Table 12-1 IC Compiler Quick Timing Model Command Summary(Continued)

To do this action	Use this command
Create a load type for a quick timing model description.	<code>create_qtm_load_type</code>
Create a path type in the quick timing model.	<code>create_qtm_path_type</code>
Create a quick timing model clock.	<code>create_qtm_clock</code>
Create a quick timing model port.	<code>create_qtm_port</code>
Create a quick timing model from a CEL view.	<code>generate_qtm_model</code>
Report model data.	<code>report_qtm_model</code>
Save the quick timing model.	<code>save_qtm_model</code>
Set a global parameter.	<code>set_qtm_global_parameter</code>
Set drive on a port.	<code>set_qtm_port_drive</code>
Set load on ports.	<code>set_qtm_port_load</code>
Set various technology parameters.	<code>set_qtm_technology</code>
Write out the quick timing model.	<code>write_qtm_model</code>

Performing Timing Budgeting On Plan Groups

If you have a design with plan groups, the top-level design contains the complete design netlist. The `allocate_fp_budgets` command generates the SDC constraints and attributes for each individual plan group. These constraints and attributes are transferred to the individual soft macro blocks when the hierarchy is committed. See “[Converting Plan Groups to Soft Macros](#)” in Chapter 13.

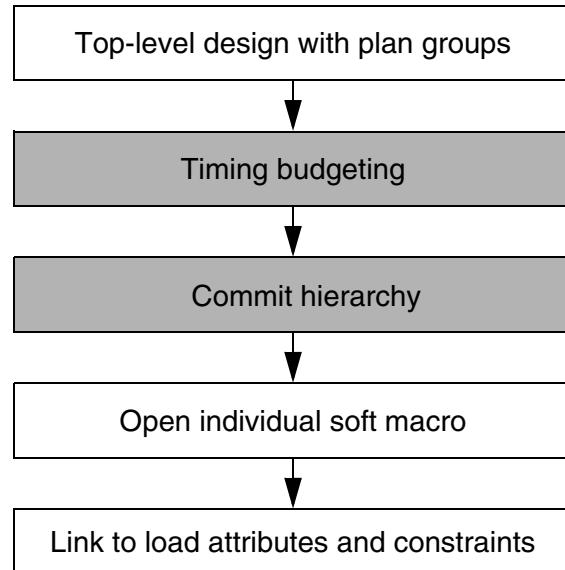
Note:

The automatic transfer of budgeted timing SDC constraints and attributes works on designs with single or multiple scenarios.

Timing budgeting on plan groups runs with full-chip timing. It honors the pin locations and the feedthrough nets assigned to the plan groups.

[Figure 12-2](#) shows the flow for performing timing budgeting on plan groups.

Figure 12-2 Performing Timing Budgeting on Plan Groups



Note:

After committing the hierarchy using the `commit_fp_plan_groups` command, the created soft macros do not have timing models. At this point, you might encounter a few warning messages. These messages do not affect any physical design operations, such as refining the pin assignment.

If you want to perform an early timing check at the top level after you commit the hierarchy, do the following:

1. Save the soft macro CEL views by using the `save_mw_cel -hierarchy` command.
2. Close the soft macro CEL views by using the `close_mw_cel` command, which leaves open only the CEL view for the top-level design.
3. Generate hierarchical models for the soft macros from the top level by using the `create_ilm_models` or `create_block_abstraction` command.
4. Run a top-level timing report using the hierarchical models.

This top-level timing report uses virtual-route based timing only, as there is no global route or parasitic information for the hierarchical models.

Budgeting With Multiply Instantiated Modules

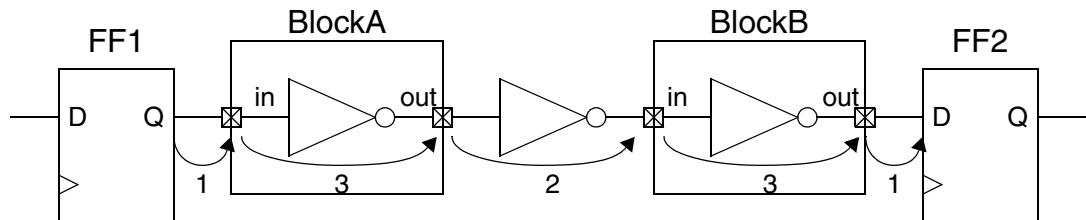
When multiply instantiated modules are present in the design, you must consider how the constraints are treated for each module. When budgeting according to the master names of the multiply instantiated modules, you can generate a single SDC constraint file, which contains a single set of constraints for each individual block of the multiply instantiated modules. A single SDC constraint file allows you to run a single optimization only one time. You can also create multiple constraint files, one for each soft macro. Multiple constraint files are generated if you budget according to instance names.

To generate a single master constraints file, use the `allocate_fp_budgets -file_format_spec output_dir/m.sdc` command, which generates one master SDC file for each set of multiply instantiated modules. To generate a separate constraints file for each multiply instantiated module, use the `allocate_fp_budgets -file_format_spec output_dir/i.sdc` command.

Conflicts might occur when budgeting according to instance names. In this case, the strictest constraint is used if a conflict occurs.

[Figure 12-3](#) shows a design example containing multiply instantiated modules.

Figure 12-3 Multiply Instantiated Module Design



In this example the clock period is 10 ns and the budgets are allocated along the timing path as shown in the figure. If BlockA and BlockB are independent, the set of corresponding input and output constraints for each block are:

BlockA:

```
set_input_delay 1 [get_ports in]
set_output_delay 6 [get_ports out]
```

BlockB:

```
set_input_delay 6 [get_ports in]
set_output_delay 1 [get_ports out]
```

If BlockA and BlockB are multiply instantiated modules, the delays cannot be correctly combined by choosing the strictest value for each constraint. The combination of the delays would result in the following constraint, which is impossible to meet for timing closure:

```
set_input_delay 6 [get_ports in]
set_output_delay 6 [get_ports out]
```

Instead, the budget values for the two multiply instantiated blocks are considered and the delay combination resulting in the strictest budget is kept. For the design example, either of the following constraint sets produces the correct result:

```
set_input_delay 1 [get_ports in]
set_output_delay 6 [get_ports out]
```

or

```
set_input_delay 6 [get_ports in]
set_output_delay 1 [get_ports out]
```

If the connectivity of the blocks gives a different budget for each block, the strictest budgeting constraint is created. Given this set of constraints:

BlockA:

```
set_input_delay 2 [get_ports in]
set_output_delay 6 [get_ports out]
```

BlockB:

```
set_input_delay 6 [get_ports in]
set_output_delay 1 [get_ports out]
```

The constraint defining the strictest budget is kept as shown in the following example.

```
set_input_delay 2 [get_ports in]
set_output_delay 6 [get_ports out]
```

This approach is used in designs with plan groups and hierarchical models. For multicorner-multimode designs, constraints for each scenario are resolved independently for each active scenario.

Table 12-2 shows a list of conventions used when writing constraints to a combined SDC file.

Table 12-2 Combined SDC File Constraint Guidelines and Examples

Command	Description and examples		
create_clock	Clocks might connect to different pins in different instances. Also different clocks might connect to the same pin, in different instances. These conflicts are resolved by creating all the clocks that connect to a specific pin in any instance on that pin in the Master. This might result in multiple clock definitions on the same pin, but the timer can use all of them.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
create_clock -name clk [get_ports {clk}]	create_clock -name clk [get_ports {clk}]	create_clock -name clk [get_ports {clk}]	
create_clock -name clk [get_ports {clk1}]	create_clock -name clk [get_ports {clk2}]	create_clock -name clk [get_ports {clk1 clk2}]	
create_clock -name clk1 [get_ports {clk}]	create_clock -name clk2 [get_ports {clk}]	create_clock -name clk1 -add [get_ports {clk}] create_clock -name clk2 -add [get_ports {clk}]	
create_clock -name clk [get_ports {clk}]	create_clock -name clk [get_pins {U1/A}]	create_clock -name clk list [[get_ports {clk}] [get_pins {U1/A}]]	
create_clock <exception_clock>	Virtual clocks are generated to model exceptions. If an exception affects one or more of the multiply instantiated modules, the corresponding virtual clock is created.		
create_generated_clock	Clocks generated in different instances that have unique names are treated the same as they are in individual SDC files. If generated clocks are created with the same name have any conflicting attributes (options <code>-source</code> , <code>-multiply_by</code> , <code>-divide_by</code> , <code>-invert</code> , <code>-master_clock</code>), a warning is issued and a clock is selected for output to the SDC file.		

Table 12-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
<code>create_generated_clock -name clk_g1 -source U1/A -multiply_by 2 U2/ B</code>	<code>create_generated_clock -name clk_g2 -source U1/C -multiply_by 2 U2/ D</code>	<code>create_generated_clock -name clk_g1 -source U1/A -multiply_by 2 U2/ B create_generated_clock -name clk_g2 -source U1/C -multiply_by 2 U2/ D</code>
<code>create_generated_clock -name clk_g -source U1/ A -multiply_by 2 U2/B</code>	<code>create_generated_clock -name clk_g -source U1/ C -multiply_by 2 U2/D</code>	<code>create_generated_clock -name clk_g -source U1/ A -multiply_by 2 U2/B</code>
set_clock_uncertainty	The uncertainty set on clocks is valid for the entire design and is the same for all multiply instantiated modules. If uncertainty is defined on ports or pins, the worst-case uncertainty value is used in the merged SDC.	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
<code>set_clock_uncertainty 0.5 [get_clocks clk]</code>	<code>set_clock_uncertainty 0.5 [get_clocks clk]</code>	<code>set_clock_uncertainty 0.5 [get_clocks clk]</code>
<code>set_clock_uncertainty 0.5 [get_ports clk1]</code>	<code>set_clock_uncertainty 0.6 [get_ports clk1]</code>	<code>set_clock_uncertainty 0.6 [get_ports clk1]</code>
set_clock_latency	The largest value among all latency values in the different SDC files for the <code>-max</code> option for the same clock. Similarly, the smallest value is chosen for the <code>-min</code> option.	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
<code>set_clock_latency -max 0.2 [get_clocks {clk}]</code>	<code>set_clock_latency -max 0.3 [get_clocks {clk}]</code>	<code>set_clock_latency -max 0.3 [get_clocks {clk}]</code>
<code>set_clock_latency -min 0.1 [get_clocks {clk}]</code>	<code>set_clock_latency -min 0.2 [get_clocks {clk}]</code>	<code>set_clock_latency -min 0.1 [get_clocks {clk}]</code>
set_clock_transition	The same command is written for all multiply instantiated modules in the combined SDC file.	

Table 12-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples		
set_propagated_clock	The same command is written for all multiply instantiated modules in the combined SDC file.		
set_input_delay	Two <code>set_input_delay</code> commands are generated in the combined SDC file if the <code>set_input_delay</code> statement for the same pin refers to a given clock in one instance and a different clock in another instance. If the delay values on the same port with respect to the given clock are different for different instances, the value defining the strictest budget is selected.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<pre>set_input_delay 2.3 -clock [get_clocks {clk1}] -max [get_ports {in}]</pre>	<pre>set_input_delay 5.8 -clock [get_clocks {clk2}] -max [get_ports {in}]</pre>	<pre>set_input_delay 2.3 -clock [get_clocks {clk1}] -max [get_ports {in}] set_input_delay 5.8 -add_delay -clock [get_clocks {clk2}] -max [get_ports {in}]</pre>	
set_output_delay	Two <code>set_output_delay</code> commands are generated in the combined SDC file if the <code>set_output_delay</code> statement for the same pin refers to a given clock in one instance and a different clock in another instance. If the delay values on the same port with respect to the given clock are different for different instances, the value defining the strictest budget is selected.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<pre>set_output_delay 2.3 -clock [get_clocks {clk1}] -max [get_ports {in}]</pre>	<pre>set_output_delay 5.8 -clock [get_clocks {clk2}] -max [get_ports {in}]</pre>	<pre>set_output_delay 2.3 -clock [get_clocks {clk1}] -max [get_ports {in}] set_output_delay 5.8 -clock [get_clocks {clk2}] -add_delay -max [get_ports {in}]</pre>	
set_false_path	If the <code>set_false_path</code> exception refers only to clocks, pins or ports outside the multiply instantiated modules, it is modeled by using virtual clocks. If some of the pins are inside the multiply instantiated modules, a one-to-one correspondence is required for all blocks. Otherwise the <code>set_false_path</code> constraint is dropped from the combined SDC file.		

Table 12-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
set_false_path -from [get_clocks {clk}]	set_false_path -from [get_clocks {clk}]	set_false_path -from [get_clocks {clk}]
set_false_path -from [get_clocks {clk_virtual1}]	set_false_path -from [get_clocks {clk_virtual2}]	set_false_path -from [get_clocks {clk_virtual1}] set_false_path -from [get_clocks {clk_virtual2}]
set_false_path -from [get_pins {U1/A}]	set_false_path -from [get_pins {U1/A}]	set_false_path -from [get_pins {U1/A}]
set_false_path -from [get_pins {U1/A}]	set_false_path -from [get_pins {U2/A}]	no output for the exception
set_multicycle_path	If the <code>set_multicycle_path</code> exception refers only to clocks, pins, and ports outside the multiply instantiated modules, it is modeled by using virtual clocks. If all of the pins are inside multiply instantiated modules, a <code>set_multicycle_path</code> statement is output in the combined SDC file if there is a one-to-one correspondence for all the multiply instantiated modules.	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
set_multicycle_path -from [get_clocks {clk}]	set_multicycle_path -from [get_clocks {clk}]	set_multicycle_path -from [get_clocks {clk}]
set_multicycle_path -from [get_clocks {clk_virtual1}]	set_multicycle_path -from [get_clocks {clk_virtual2}]	set_multicycle_path -from [get_clocks {clk_virtual1}]
set_multicycle_path -from [get_clocks {clk_virtual2}] set_multicycle_path -through [get_pins {U1/A}]	set_multicycle_path -through [get_pins {U1/A}]	set_multicycle_path -through [get_pins {U1/A}]
set_multicycle_path -through [get_pins {U1/A}]	set_multicycle_path -through [get_pins {U2/A}]	No <code>set_multicycle_path</code> statement.

Table 12-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
set_min_delay	If the <code>set_min_delay</code> exception refers only to clocks, pins, and ports outside the multiply instantiated modules, it is modeled by using virtual clocks. If all of the pins are inside multiply instantiated modules, a <code>set_min_delay</code> statement is output in the combined SDC file if there is a one-to-one correspondence for all the multiply instantiated modules.	
set_max_delay	If the <code>set_max_delay</code> exception refers only to clocks, pins, and ports outside the multiply instantiated modules, it is modeled by using virtual clocks. If all of the pins are inside multiply instantiated modules, a <code>set_max_delay</code> statement is output in the combined SDC file if there is a one-to-one correspondence for all the multiply instantiated modules.	
set_drive	If different <code>set_drive</code> values exist for the same port in different instances, largest resistance value is used for the <code>-max</code> option. The smallest resistance value is used for the <code>-min</code> option. The options <code>-rise</code> and <code>-fall</code> are treated separately.	
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>
<code>set_drive 2 -max [get_ports {in}]</code>	<code>set_drive 3 -max [get_ports {in}]</code>	<code>set_drive 3 -max [get_ports {in}]</code>
<code>set_drive -min [get_ports {in}]</code>	<code>set_drive 3 -min [get_ports {in}]</code>	<code>set_drive 2 -min [get_ports {in}]</code>
set_driving_cell	The library cell with the largest drive resistance to output is selected if there are different library cell and pin paths for a given port in different instances. If the <code>set_driving_cell</code> command for one port has the same library cell and pin for different instances, but the <code>input_transition_rise</code> or the <code>input_transition_fall</code> numbers are different, the largest number for <code>-max</code> and the smallest number for <code>-min</code> is output.	
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>
<code>set_driving_cell -lib_cell INVX2 -pin Z [get_ports {in}]</code>	<code>set_driving_cell -lib_cell INVX4 -pin Z [get_ports {in}]</code>	<code>set_driving_cell -lib_cell INVX2 -pin Z [get_ports {in}]</code>
<code>set_driving_cell -lib_cell INVX2 -pin Z -input_transition_rise 0.5 -max [get_ports {in}]</code>	<code>set_driving_cell -lib_cell INVX2 -pin Z -input_transition_rise 0.7 -max [get_ports {in}]</code>	<code>set_driving_cell -lib_cell INVX2 -pin Z -input_transition_rise 0.7 -max [get_ports {in}]</code>

Table 12-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples		
set_input_transition	The largest value for <code>-max</code> conditions and the smallest value for <code>-min</code> conditions are chosen if different input transition numbers exist for the same port in different instances. The <code>-rise</code> and <code>-fall</code> options are treated separately.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<code>set_input_transition -max 0.5 [get_ports {in}]</code>	<code>set_input_transition -max 0.7 [get_ports {in}]</code>	<code>set_input_transition -max 0.7 [get_ports {in}]</code>	
<code>set_input_transition -min 0.5 [get_ports {in}]</code>	<code>set_input_transition -min 0.7 [get_ports {in}]</code>	<code>set_input_transition -min 0.5 [get_ports {in}]</code>	
set_load -pin_load	The largest value for <code>-max</code> and the smallest value for <code>-min</code> are chosen if the <code>set_load -pin_load</code> statements for the same port on different instances have different values.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<code>set_load -pin_load 10 -max [get_ports {out}]</code>	<code>set_load -pin_load 15 -max [get_ports {out}]</code>	<code>set_load -pin_load 15 -max [get_ports {out}]</code>	
<code>set_load -pin_load 10 -min [get_ports {out}]</code>	<code>set_load -pin_load 15 -min [get_ports {out}]</code>	<code>set_load -pin_load 10 -min [get_ports {out}]</code>	
set_load -wire_load	The largest value for <code>-max</code> and the smallest value for <code>-min</code> are chosen if the <code>set_load -wire_load</code> statements for the same port on different instances have different values.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<code>set_load -wire_load 10 -max [get_ports {out}]</code>	<code>set_load -wire_load 15 -max [get_ports {out}]</code>	<code>set_load -wire_load 15 -max [get_ports {out}]</code>	
<code>set_load -wire_load 10 -min [get_ports {out}]</code>	<code>set_load -wire_load 15 -min [get_ports {out}]</code>	<code>set_load -wire_load 10 -min [get_ports {out}]</code>	
set_port_fanout_number	The largest value is output if the <code>set_port_fanout_number</code> for the same port on different instances have different values.		

Table 12-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
set_port_fanout_number 10 [get_ports {out}]	set_port_fanout_number 20 [get_ports {out}]	set_port_fanout_number 20 [get_ports {out}]
set_logic_zero	A one-to-one correspondence is required for all the blocks if some of the pins with <code>set_logic_zero</code> statements are inside the multiply instantiated modules. The <code>set_logic_zero</code> constraint is dropped from the combined SDC file otherwise. The same is true for <code>set_logic_zero</code> statements on ports.	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
set_logic_zero [get_ports {in}]	set_logic_zero [get_ports {in}]	set_logic_zero [get_ports {in}]
set_logic_zero [get_ports {in}]	no set_logic_zero statement for port in	no set_logic_zero statement for port in
set_logic_one	A one-to-one correspondence is required for all the blocks if some of the pins with <code>set_logic_one</code> statements are inside the multiply instantiated modules. The <code>set_logic_one</code> constraint is dropped from the combined SDC file otherwise. The same is true for <code>set_logic_one</code> statements on ports.	
set_case_analysis	A one-to-one correspondence is required for all blocks in the pin name and case analysis value if the pins are inside multiply instantiated modules. Otherwise the <code>set_case_analysis</code> constraint is dropped from the combined SDC file. The same is true for <code>set_case_analysis</code> statements on the ports.	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
set_case_analysis 0 [get_ports {in}]	set_case_analysis 0 [get_ports {in}]	set_case_analysis 0 [get_ports {in}]
set_case_analysis 0 [get_ports {in}]	set_case_analysis 1 [get_ports {in}]	No set_case_analysis statement generated
set_case_analysis 0 [get_ports {in}]	No set_case_analysis statement for port in	No set_case_analysis statement generated
set_operating_conditions	Operating conditions must match exactly for all multiply instantiated module instances for the generation of <code>set_operating_conditions</code> statements.	

Table 12-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
set_operating_conditions -max BCCOM	set_operating_conditions -max BCCOM	set_operating_conditions -max BCCOM
set_operating_conditions -max BCCOM	set_operating_conditions -max WCCOM	No set_operating_conditions statement generated.
set_disable_timing		The list of disabled cells, pins or ports in the current design match exactly for all multiply instantiated module instances for the generation of set_disable_timing statements.
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
set_disable_timing -from A -to Z [get_cells {U1}]	set_disable_timing -from A -to Z [get_cells {U1}]	set_disable_timing -from A -to Z [get_cells {U1}]
set_disable_timing -from A -to Z [get_cells {U1}]	set_disable_timing -from B -to Z [get_cells {U1}]	No set_disable_timing statement generated
set_disable_timing -from A -to Z [get_cells {U1}]	No corresponding set_disable_timing statement.	No set_disable_timing statement generated
set_clock_gating_check		A set_clock_gating_check statement is written to the combined SDC file if it is present in at least one of the SDC files generated for individual instances. If two different set_clock_gating_check statements are applied to equivalent objects in the two SDC files, the strictest constraint is written.
<u>Module1.sdc</u>	<u>Module2.sdc</u>	<u>Combined.sdc</u>
set_clock_gating_check -setup 0.5 [get_cells {U1}]	set_clock_gating_check -setup 0.5 [get_cells {U1}]	set_clock_gating_check -setup 0.5 [get_cells {U1}]
set_clock_gating_check -setup 0.5 [get_cells {U1}]	set_clock_gating_check -setup 0.7 [get_cells {U2}]	set_clock_gating_check -setup 0.5 [get_cells {U1}] set_clock_gating_check -setup 0.7 [get_cells {U2}]

Table 12-2 Combined SDC File Constraint Guidelines and Examples (Continued)

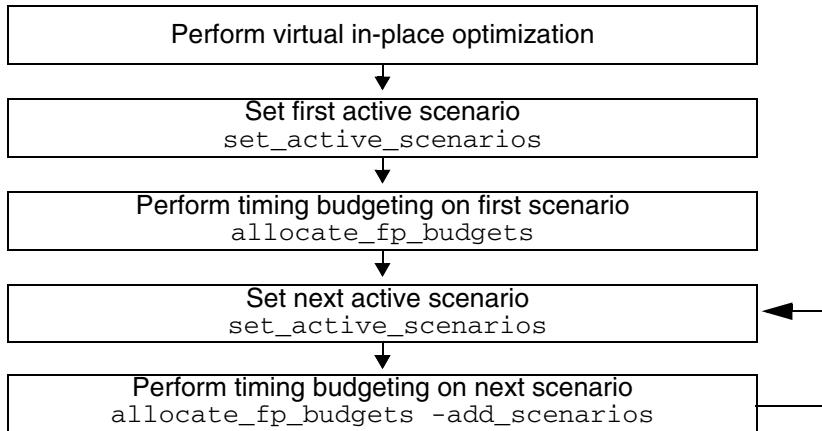
Command	Description and examples		
<code>set_clock_gating_check -setup 0.5 [get_cells {U1}]</code>	<code>set_clock_gating_check -setup 0.7 [get_cells {U1}]</code>	<code>set_clock_gating_check -setup 0.7 [get_cells {U1}]</code>	
set_max_time_borrow	The worst case maximum time borrow value is written to the merged SDC file.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<code>set_max_time_borrow 0.5 {latch1}</code>		<code>set_max_time_borrow 0.5 {latch1}</code>	
<code>set_max_time_borrow 0.5 {latch1}</code>	<code>set_max_time_borrow 0.6 {latch1}</code>	<code>set_max_time_borrow 0.5 {latch1}</code>	
set_max_transition	The worst case maximum transition time value is written to the merged SDC file.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<code>set_max_capacitance 1.0 [get_ports port1]</code>	<code>set_max_capacitance 1.5 [get_ports port1]</code>	<code>set_max_capacitance 1.0 [get_ports port1]</code>	
set_min_capacitance	The worst case set_min_capacitance value is written to the merged SDC file.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<code>set_min_capacitance 0.3 [get_ports port1]</code>	<code>set_min_capacitance 0.4 [get_ports port1]</code>	<code>set_min_capacitance 0.4 [get_ports port1]</code>	
set_max_fanout	The worst case set_max_fanout value is written to the merged SDC file.		
<i>Module1.sdc</i>	<i>Module2.sdc</i>	<i>Combined.sdc</i>	
<code>set_max_fanout 10 [get_ports port1]</code>	<code>set_max_fanout 15 [get_ports port1]</code>	<code>set_max_fanout 10 [get_ports port1]</code>	

Performing Budgeting on Multiple Scenarios

You can perform timing budgeting and generate constraints for each scenario for designs that contain multiple scenarios. To minimize system resources, you can use the get dominant scenario timing budgeting flow and activate only the dominant scenario.

[Figure 12-4](#) illustrates the get dominant scenario timing budgeting flow. For large designs, this flow requires less system resource than activating all scenarios at the same time.

Figure 12-4 Get Dominant Scenario Timing Budgeting Flow



IC Compiler budgets all scenarios in two phases in the get dominant scenarios timing budgeting flow. The tool first performs timing budgeting by applying the constraints for the dominant scenario. Next, the tool budgets other scenarios when you specify the `-add_scenarios` option and combines the constraints. This flow makes it possible to generate constraints for all scenarios without making all scenarios active at the same time, reducing the system resource requirements necessary for performing timing budgeting.

Note the following when using this flow:

- If you specify the `-add_scenarios` option without first performing an initial timing budgeting run, the results are undefined.
- If you budget the same scenario twice by using two consecutive `-add_scenarios` options, the results are undefined.

Performing Distributed Timing Budgeting for Multiple Scenarios

You can perform distributed timing budgeting to increase the number of budgeting scenarios and decrease the overall runtime for timing budgeting. Distributed timing budgeting enables you to run each scenario independently on a different machine, reducing the memory requirement for timing budgeting.

Specify the `-host_options name`, `-work_directory dirname`, `-pre_open_cell_script prescriptname`, and `-post_open_cell_script postscriptname` options to enable distributed timing budgeting. The `-host_options` option specifies that IC Compiler creates a task for each scenario and combines the results after the tasks complete. The `-work_directory` option specifies the directory name for the distributed budgeting run. The default directory is `./TB_work_dir`. The `-pre_open_cell_script` option specifies the script file name that is run before opening the cell in the distributed process. The `-post_open_cell_script` options specifies the script file name that is run immediately after the distributed process opens the cell.

To use distributed timing budgeting, you must have a distributed processing system configured in your compute environment. The following example uses the `set_host_options` command to enable four distributed processes using the Oracle Grid Engine and uses the `allocate_fp_budgets` command to perform distributed timing budgeting.

```
icc_shell> set_host_options -name my_grd_host_options \
    -pool grd -submit_options {-P bnormal -l arch=glinux, \
    cputype=amd64,mem_free=4G,mem_avail=4G -l \
    qsc=f -cwd}

icc_shell> allocate_fp_budgets \
    -host_options my_grd_host_options \
    -pre_open_cell_script pre.tcl \
    -post_open_cell_script post.tcl
```

Note the following when performing distributed budgeting:

- Some implementation commands are disabled and should not be included in the pre-open-cell and post-open-cell scripts
- Relative paths are not supported in the pre-open-cell and post-open-cell scripts, use absolute paths instead
- The design being budgeted must be a multicorner-multimode design

Performing Hierarchical Signal Integrity Budgeting

Timing budgeting can consider crosstalk effects across soft macro boundaries when generating SDC constraints. Hierarchical signal integrity budgeting can extract effective drive strength for input pins and coupling capacitance for interface nets. The timing budgeter can retrieve crosstalk analysis information from a hierarchical signal integrity module and then write out this information to block pin attributes. Bringing the top-level crosstalk information into the block level during budgeting makes it easier for block-level crosstalk analysis to consider the crosstalk effects at the top level.

Before running timing budgeting, set the `enable_hier_si` variable to true to enable signal integrity budgeting and take the crosstalk timing effects across soft macro boundaries into account.

```
icc_shell> set enable_hier_si true
```

Run the timing budgeting command.

```
icc_shell> allocate_fp_budgets
```

IC Compiler processes the hierarchical signal integrity information during budgeting and writes out this information about the block's pin attributes.

The total effective drive strength is written out on block pins to represent the top-level aggressor drive strength, and the total coupling capacitance is written out on block pins to represent the total coupling capacitance in the block CEL view.

Block-Level Hierarchical Signal Integrity Flow

To run the block-level hierarchical signal integrity flow, do the following:

1. Enable the hierarchical signal integrity flow.

```
set enable_hier_si true
```

2. Define the signal integrity options used for analysis or optimization.

```
set_si_options -static_noise true -delta_delay true
```

The `-static_noise true` option reduces static noise.

The `-delta_delay true` option optimizes the timing with crosstalk delay.

3. Perform routing and postroute optimization.

```
route_opt -xtalk_reduction
```

IC Compiler performs crosstalk prevention during global routing and track assignment, and crosstalk analysis and optimization during the postroute optimization phase.

4. Create a Milkyway signal-integrity-aware hierarchical model.

To create a signal-integrity-aware interface logic model, use the following command:

```
icc_shell> create_ilm -include_xtalk
```

The `-include_xtalk` option stores crosstalk information, such as boundary net aggressor data, effective net resistance, total capacitance, and coupling capacitance, with the interface logic model.

To create a signal-integrity-aware block abstraction model, use the following commands:

```
icc_shell> create_block_abstraction
icc_shell> save_mw_cel
```

Top-Level Hierarchical Signal Integrity Flow

To run the top-level hierarchical signal integrity flow, do the following:

1. Enable the hierarchical signal integrity flow.

```
set enable_hier_si true
```

2. Define the signal integrity options used for analysis or optimization.

```
set_si_options -static_noise true -delta_delay true
```

The `-static_noise true` option reduces static noise.

The `-delta_delay true` option optimizes the timing with crosstalk delay.

3. Perform routing and postroute optimization.

```
route_opt -xtalk_reduction
```

IC Compiler performs crosstalk prevention during global routing and track assignment, and crosstalk analysis and optimization during the postroute optimization phase.

Performing Post-Budget Timing Analysis

After performing timing budgeting on a design, you can perform post-budget timing analysis to generate a report containing budgeted and actual delays through a hierarchical block.

Note:

This command must be run during the same IC Compiler timing budgeting session in which the budgets were created.

To perform post-budget timing analysis,

1. Choose Timing > Floorplan Budgeting Report.

The Budgeting Report window appears.

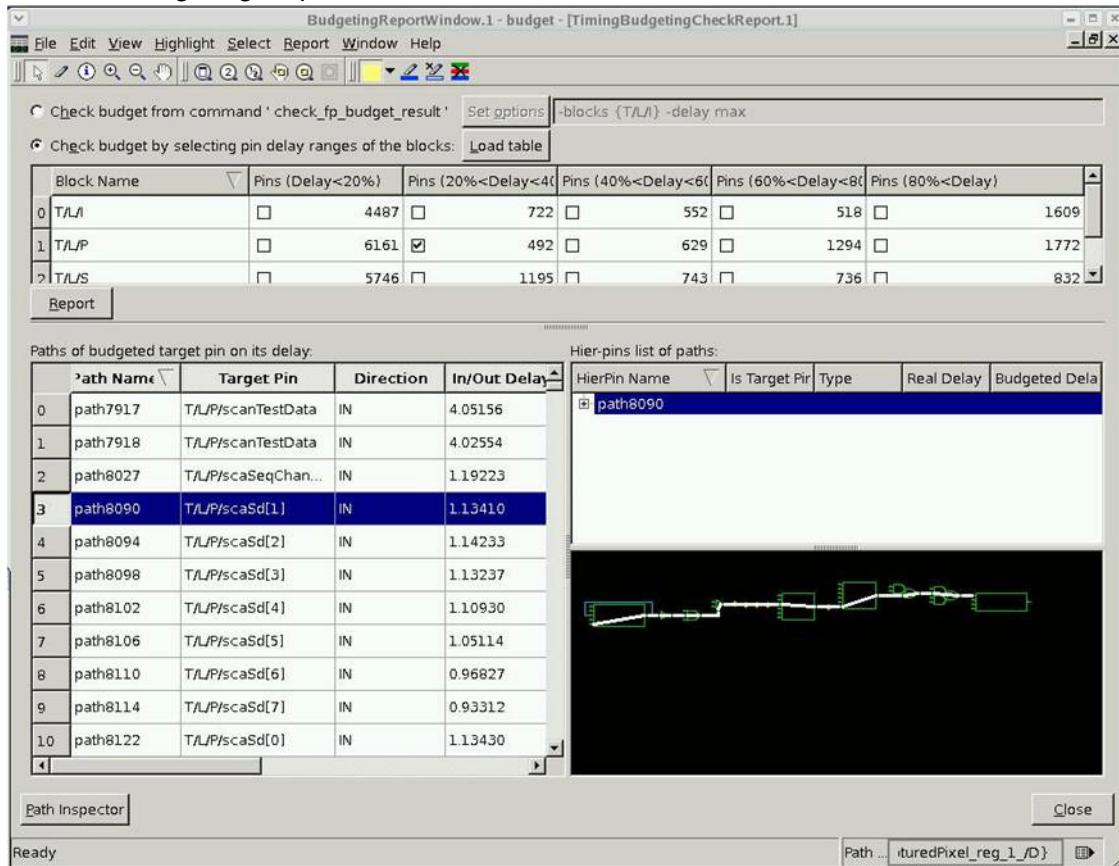
Alternatively, you can use the `check_fp_budget_result` command to generate a report file.

2. Select “Check budget from command `check_fp_budget_result`” and enter the `-blocks`, `-pins`, or `-file_name` options from the `check_fp_budget_result` command. You can also select command options by clicking the “set option” button and entering the `-blocks`, `-pins` and file name options in the pop-up dialog. A list of paths is displayed based on the options you specify.

Alternately, select “Check budget by selecting pin delay ranges of the blocks” and click Load Table. The table is populated with Block Name and a count of pins within each delay category. Select a check box and click Report to populate the Paths of budgeted target pin on its delay table with the list of paths.

3. Select a Path Name from the list and click Path Inspector. A graphical representation of the path is displayed as shown in [Figure 12-5 on page 12-30](#).

Figure 12-5 Budgeting Report Window



A timing path starts at a startpoint and ends at an endpoint. Timing paths are divided into segments. For each timing path segment, the report prints the actual and budgeted delay or the fixed delay values of the segment. You can use this information to analyze the process of generating budgets for your design.

You can also generate a text or HTML format post-budget timing analysis report using the `check_fp_budget_result` command. For each pin on a block, the report lists the worst case timing paths per endpoint clock domain that pass through the pin. For each path segment on the timing path, the report lists the budgeted and actual delays for the path segment.

To generate a plain text format report, specify the `-file_name reportfile` option. The following example writes a plain text format report to the `report.rpt` file.

```
icc_shell> check_fp_budget_result -file_name report.rpt
Starting check_fp_budget_result ...
1
```

To generate an HTML format report, specify the `-file_name reportfile -html -html_dir dirname` options. The following example writes a plain text format report to the `report.rpt` file, writes an HTML format report to the `htmlreport` directory, and lists the contents of the `htmlreport` directory. IC Compiler generates an HTML file for each plan group in the design. To view the HTML format report, open the `htmlreport/index.html` file in a browser.

```
icc_shell> check_fp_budget_result -html -html_dir htmlreport \
           -file_name report.rpt
Starting check_fp_budget_result ...
1

icc_shell> sh ls htmlreport
I_ORCA_TOP_I_BLENDER_0.html
I_ORCA_TOP_I_BLENDER_1.html
I_ORCA_TOP_I_BLENDER_2.html
index.html
```

Performing Clock Latency Budgeting

You can perform clock latency budgeting. This budgeting feature uses information from files built by clock planning to include clock latencies associated with real clock tree branches that are external to the block being budgeted. Clock planning can create clock budgets for each plan group inside the design to specify the source and network latencies whether or not the clock has a sink inside the plan group.

The `allocate_fp_budgets` command creates virtual clocks to capture these latencies from clock planning. This budgeting feature also generates real latency values in budgets for physical clock ports created by clock planning on a block. The clock ports are created by the clock planning features. These latencies are associated with real physical ports.

Creating Budgets to Reflect Real Clock Latencies

After clock planning, you can create budgets to reflect real clock latencies. This consists of:

- Creating real clock latency values on clock ports for the block.
- Creating input and output virtual clocks for interface path endpoints that are partially outside the block.

The syntax for defining virtual clocks created for latencies outside the block is

`clock_name_v_in`

and

`clock_name_v_out`

Clocks launching (capturing) flip-flops on the top level have source latencies only.

Clocks launching (capturing) flip-flops in blocks have both source and network latencies.

Clock latency budgeting is on by default if timing budgeting (`allocate_fp_budgets` command) is run after clock planning.

You can enable (turn on) clock planning based latency values by setting the following variable in the `icc_shell` to `true` (the default) before performing timing budgeting:

```
set virtual_clocks_from_cp true
```

To disable (turn off) clock planning based latency values, set the following variable in the `icc_shell` to `false`.

```
set virtual_clocks_from_cp false
```

You can include latencies during budgeting for synthesized clock trees which were not created during clock planning by setting the following variable to a value of `true` before running the timing budgeter. The default is `false`.

```
icc_shell> set synthesized_clocks true
```

To disregard clock tree synthesis-based latency values, set the `synthesized_clocks` variable to `false`.

```
icc_shell> set synthesized_clocks false
```

After you create budgets to reflect real clock latencies, check for the following:

- Ensure every real input clock port created by clock planning for a block budget has calculated source and network latency statements in the block budget.
- Ensure latency statements in the budgets reflect the actual clock network delays in the design.
- Ensure every input port `set_input_delay` and `set_output_delay` is referenced to a virtual clock that has the latency for the launch or capture clock associated with the path through the port.
- Check latencies in budgets against clock planning latency files.

Note:

Clock planning creates a source latency file and a network latency file for each clock port in the `tcp_output` directory.

- Check that latencies in budgets are only for clock ports defined in the block budget file.
- Check latencies in clock planning latency files against the clock skew report.

13

Committing the Physical Hierarchy

This chapter describes how to commit the physical hierarchy after finalizing the floorplan by converting plan groups into soft macros. Committing the hierarchy creates a new level of physical hierarchy in the virtual flat design by creating CEL views for selected plan groups. After committing the physical hierarchy, you can also uncommit the physical hierarchy by converting the soft macros back into plan groups.

In addition, this section describes how to propagate top-level preroutes into soft macros and recover all pushed-down objects in child cells to the top level. It also describes how to commit plan groups with associated UPF power domains.

This chapter includes the following sections:

- [Converting Plan Groups to Soft Macros](#)
- [Pushing Physical Objects Down to the Soft Macro Level](#)
- [Pushing Physical Objects Up to the Top Level](#)
- [Handling 45-Degree Redistribution Layer Routing](#)
- [Committing the Hierarchy of Plan Groups With Power Domains](#)
- [Converting Soft Macros to Plan Groups](#)

Converting Plan Groups to Soft Macros

You can convert selected plan groups or all plan groups with exclusive placement constraints into soft macro CEL views.

Before converting plan groups to soft macros, your design should meet the following requirements:

- Exclusive plan groups, standard cells, and hard macros are legally placed.
- Traditional pin assignment or pin placement flow is completed.
- In-place optimization, timing analysis, and timing budgeting are completed.

To convert plan groups to soft macros,

1. Choose Partition > Commit Plan Groups.

The Commit Plan Groups dialog box appears.

2. Set the options, depending on your requirements.

- Target plan groups – Specify whether to commit all plan groups or specified plan groups. The default is all.
- Push down power and ground straps into the child cell – Select the option if you want power and ground straps copied (pushed) down into the newly created soft macro CEL views. The power and ground straps are removed from the top level. The default is off.
- Commit to a new top cell – Select this option to commit the plan groups to a new top-level design in the Milkyway design library. The default is off.
- Top cell name – Enter the name of the new top-level design.

3. Click Apply or OK.

Alternatively, you can use the `commit_fp_plan_groups` command.

Committing the physical hierarchy does the following:

- Converts the virtual flat design to a two-level hierarchical design.
- Removes standard cells and hard macros belonging to a plan group from the top-level design and copies them down to the soft macro that is created for the plan group.
- Creates nets in the soft macros based on the hierarchy preservation data.

- Pushes placement blockages and route guides down to the soft macros.
- (Optional) Pushes power and ground straps and standard cell preroutes down into each soft macro.

After the conversion from plan groups to soft macros is complete, all the standard cell instances of the plan groups become child cell instances of the corresponding converted soft macro cells and are deleted from the top-level design. New corresponding cell instances of the new soft macro cells are created in the top-level design and reside in the same location as the old plan groups.

Utilization of the new soft macros is the same as the utilization of the corresponding plan groups before running the `commit_fp_plan_groups` command. The default utilization is based on the sum of the area of standard cells and hard macros in the cell divided by the area in the core area bounding box of the cell.

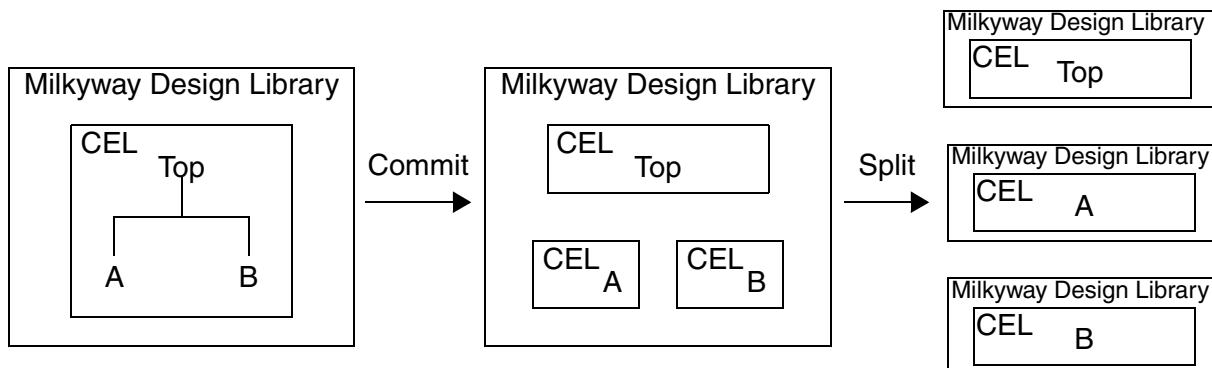
Splitting the Soft Macros

When you commit the soft macros, IC Compiler creates a CEL view for each of the blocks in the same Milkyway design library. To allow different designers to work on different blocks without interference, you can use the `split_mw_lib` command to split the top-level Milkyway design into new Milkyway design libraries for each block and the top-level design.

After splitting, the new Milkyway design libraries inherit all the information, including the technology file and the reference library path, from the original Milkyway design.

[Figure 13-1](#) shows the splitting of soft macros to create separate design libraries.

Figure 13-1 Creating Separate Design Libraries



After block-level implementation, you can link the CEL views for the blocks in different design libraries to the top-level design by using the `set_mw_lib_reference` command.

Pushing Physical Objects Down to the Soft Macro Level

Physical objects such as routing, route guides, blockages, cells, and cell rows, can be pushed down from the top level to the soft macro level by using the `push_down_fp_objects` command. You can control several options during the push-down operation, including the layers to be copied, whether objects are moved or copied, if overlap is checked, whether new pins are created, whether blockages are created, if a boundary region is created, and other options.

[Table 13-1](#) describes the `push_down_fp_objects` command options. For more information about these options, see the man page.

Table 13-1 push_down_fp_objects Command Options

Command option	Description
<code>-allow_feedthroughs</code> ("Allow feedthroughs" check box in the GUI)	Creates new ports for feedthrough routes on signal and clock nets, and creates a new net at the top level for the split net.
<code>-connect_objects_to_child_nets</code> ("Connect copy down objects to their nets" check box in the GUI)	Connects pushed down objects to their nets.
<code>-create_pins</code> ("Create pins" check box in the GUI)	Creates pins during copy down routing.
<code>-freeze_push_down_nets</code> ("Freeze push down nets" check box in the GUI)	Prevents modification of push-down nets during the push-down process and during optimization.
<code>-include_shapes all with_net_id without_net_id</code> ("Shapes" check box and group in the GUI)	Specifies the types of shapes to push down.
<code>-layers {layer_list}</code> ("Limit pushed down objects to specified layers" box in the GUI)	Limits the pushed down objects to the specified layers.
<code>-margin {lx by rx ty}</code> ("Margin around soft macro boundary" check box in the GUI)	Pushes down any object inside the specified margin and pushes down any object that overlaps the cell boundary.
<code>-object_type {list_of_types}</code> ("Types of objects to be pushed down" check boxes in the GUI)	Specifies the types of objects to be pushed down. Legal values are one or more of the following: <code>routing</code> , <code>route_guides</code> , <code>blockages</code> , <code>cells</code> , <code>rows</code> , and <code>shapes</code> .

Table 13-1 push_down_fp_objects Command Options (Continued)

Command option	Description
-objects <i>object_list</i> ("Push down specified objects" check box and text box in the GUI)	Restricts the push-down operation to the specified set of objects. By default, all objects of the type specified by the <code>-object_type</code> option are pushed down.
-overlap_checking on off ("Check objects overlaps" check box in the GUI)	Checks routing overlaps before pushing down routing objects.
-partial_overlap ("Include partially overlapped cells" check box in the GUI)	Pushes down cells that partially overlap the soft macro boundary.
-push_down_type push_down cut_down copy_down ("Push down mode" option in the GUI)	Specifies how the tool processes push-down objects.
-remove_routing ("Remove routing" check box in the GUI)	Removes the route connections between the pushed down cells.
-retain_bus_naming ("Retain bus naming" check box in the GUI)	Retains the bus naming convention, such as "[number]", for the newly created child feedthrough ports and child feedthrough nets.
-row_offset_x <i>offset</i> ("Horizontal" box in "Offset between soft macro core and rows" area in the GUI)	Specifies the horizontal offset in microns between the soft macro core and rows.
-row_offset_y <i>offset</i> ("Vertical" box in "Offset between soft macro core and rows" area in the GUI)	Specifies the vertical offset in microns between the soft macro core and rows.
-to_object_type blockage original_type ("Type of pushed down objects" options in the GUI)	Specifies the object type for objects after they are pushed down to the child level. You must also specify the <code>-object_type routing</code> option or the <code>-object_type cells</code> option, and specify the <code>-push_down_type copy_down</code> option.
-wire_net_type {list_of_net_types} ("Net type:" check boxes in the GUI)	Specifies the types of nets to push down. Legal values are one or more of the following: <code>pg</code> , <code>clock</code> , and <code>signal</code> .

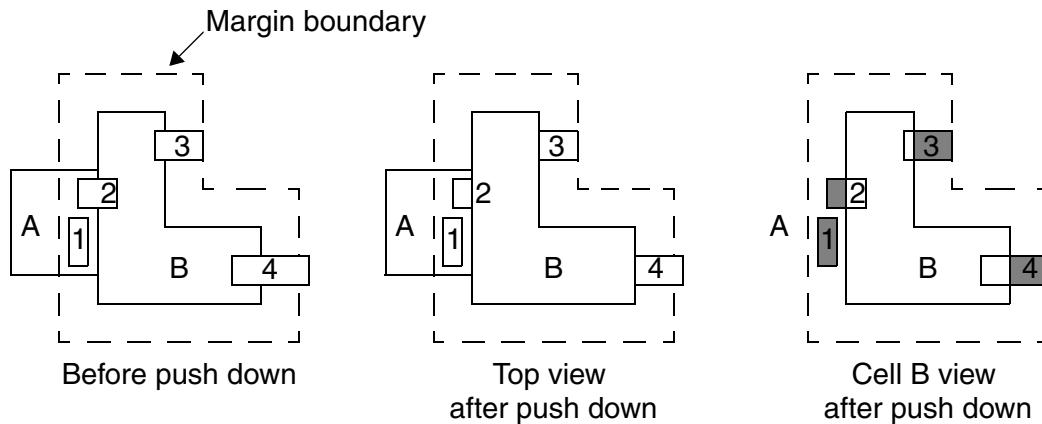
Using the Margin Option for Pushing Down Objects

[Figure 13-2](#) shows the usage of the margin option to push-down objects based on an offset from the cell boundary. The following `push_down_fp_objects` command treats routing objects based on the overlap with the cell B boundary and margin boundary.

```
icc_shell> push_down_fp_objects -object_type {routing} \
    -margin {10, 10, 10, 10} [get_cells B]
```

Routing objects that overlap the boundary of cell B are pushed into cell B as routing, and objects within the margin are pushed down as route guides.

Figure 13-2 Using Margin to Control Routing Push Down



Pushing Down Routing in Multiply Instantiated Modules

Routes are pushed down differently in designs that contain multiply instantiated modules. Only routes that overlap master instances when processing power and ground nets, or clock and signal nets without ambiguous connection, are pushed down. The tool checks the routing alignment between the master instance and nonmaster instances and issues a warning message if alignment is not maintained. For power and ground nets, gaps are placed between the top-level power and ground straps and the PG pins for multiply instantiated module instances that are not aligned with the master instance and are not surrounded by power and ground rings. These gaps allow power and ground routing to connect the PG pins at a later time.

Pushing Physical Objects Up to the Top Level

Physical objects that were previously pushed down into a soft macro can be pushed back up to the top level by using the `push_up_fp_objects` command. Note that when performing a consecutive push down and push up of objects, the objects are pushed up to the top level, not to their original location in the hierarchy.

[Table 13-2](#) describes the `push_up_fp_objects` command options. For more information about these options, see the man page.

Table 13-2 push_up_fp_objects Command Options

Command option	Description
<code>-child_objects child_objects</code> ("Collection of child objects" check box in the GUI)	Restricts the push-up operation to the specified set of objects. By default, all objects of the type specified by the <code>-object_type</code> option are pushed up.
<code>-ignore_not_push_down_nets</code> ("Ignore non push-down nets" check box in the GUI)	Ignores nets in the soft macro that were not pushed down when you specify the <code>-object_type routing</code> option.
<code>-include pushed_down_objects_only all</code> ("All objects" and "Pushed down objects only" check boxes in the GUI)	Specifies whether the command pushes up all objects or only pushes up objects that were pushed down.
<code>-include_shapes all with_net_id without_net_id</code> ("Shapes" check box and group in the GUI)	Specifies the types of shapes to push up.
<code>-layers {layer_list}</code> ("Limit pushed up objects to specified layers" box in the GUI)	Limits the pushed up objects to the specified layers.
<code>-object_type {list_of_types}</code> ("Types of objects to be pushed up" check boxes in the GUI)	Specifies the types of objects to be pushed up. Legal values are one or more of the following: <code>pins</code> , <code>routing</code> , <code>route_guides</code> , <code>blockages</code> , <code>cells</code> , <code>rows</code> , and <code>shapes</code> .
<code>-push_up_type push_up copy_up</code> ("Push up mode" options in the GUI)	Specifies whether the tool creates a copy of the pushed up object or deletes the object from the soft macro.
<code>-remove_routing</code> ("Remove routing between pushed up cells" check box in the GUI)	Removes the route connections between the pushed up cells.

Table 13-2 push_up_fp_objects Command Options (Continued)

Command option	Description
<code>-to_object_type routing pins blockage original_type ("Types of objects after push up from soft macro options in the GUI")</code>	Specifies the object type for objects after they are pushed up to the top level.
<code>-top_level_interface_nets net_objects</code>	Specifies a collection of top-level interface nets for which routing, pin, or cell objects are pushed up.

Handling 45-Degree Redistribution Layer Routing

The `push_up_fp_objects` and `push_down_fp_objects` commands can support 45-degree redistribution layer routing. During push down, the 45-degree routes are split at the intersecting points between the soft macro boundaries and where the 45-degree routes are added. Square-shaped pins are created at the intersection points during push-down; during push up, these pins are deleted at the soft macro boundaries.

During push up, the 45-degree routes are restored to the original state before push down. (Wires are merged at the intersection points between the soft macro boundaries and the routes.)

If a net has no ambiguous connection to a soft macro and a feedthrough is not allowed, routing that is collinear with a block boundary is pushed down.

Committing the Hierarchy of Plan Groups With Power Domains

The `commit_fp_plan_groups` command supports plan groups that have a Unified Power Format (UPF) power domain associated with them. The power domain must be defined completely within the hierarchical cell instance of the plan group that is being converted to a soft macro. For more information, see the Multivoltage Design Flow chapter in the *IC Compiler Implementation User Guide*.

During the commit hierarchy process, the top-level power domains associated with the plan groups are pushed down into a new child cell. This also makes it possible for the top-level design's UPF constraints to be properly transferred and maintained. Embedded voltage areas and power domains associated with those voltage areas are also pushed into the child cell.

Near the end of the commit process, after the top-level hierarchical cell instance is replaced with a top-level child cell instance, all corresponding power domains are removed from the top-level cell, along with any voltage areas associated with the pushed down power domains. This is done to avoid the duplication of power domains across the hierarchy. Any top-level voltage area that was coincident with the plan group boundary, is also removed from the top-level design and treated as the default voltage area within the child cell instance.

Note:

If a power domain is not completely defined within the hierarchical cell instance, in other words it is larger than the plan groups associated with them, it means they might contain UPF elements that are outside the plan group and therefore, they cannot be committed.

Converting Soft Macros to Plan Groups

After plan groups are committed into soft macros, you can uncommit the physical hierarchy by converting the soft macros back into plan groups. The physical hierarchy is flattened into the parent.

Physical objects from the soft macro are pushed up to their relative positions in the top CEL view. Any physical objects that are connected to power and ground nets are connected to their appropriate nets in the top-level design, and any objects that have properties set in the soft macro have the same properties set in the top-level design.

To uncommit the physical hierarchy,

1. Choose Partition > Uncommit Soft Macros.

The Uncommit Soft Macros dialog box appears.

2. Choose whether to convert all soft macros or selected soft macros to plan groups.

3. Set the remaining options, depending on your requirements.

- Preroutes to push up – Select the All option (the default) to push up all routing, including both power and ground, and detail routing. Select the “Power and Ground” option if you want only power and ground straps pushed up to the top-level design.
- Remove feedthrough ports on soft macros – Select this option to remove the feedthroughs from selected soft macros. All feedthrough nets and ports with name suffixes of *_pft and *_rft are removed.

4. Click OK or Apply.

Alternatively, you can use the `uncommit_fp_soft_macros` command.

When you click the OK or Apply button in the dialog box, the following physical objects are pushed up to the top-level design from soft macros:

- Standard cells
- Straps on power and ground nets
- Vias on power and ground nets
- Placement blockages, including soft placement blockages and the internal padding around the boundary and cell rows
- Route guides

Keep the following points in mind when you uncommit the hierarchy:

- The uncommit process does not remove the soft macros that were present before running the `commit_fp_plan_groups` command.
- Any routing done on the soft macros is deleted after you uncommit the hierarchy (except for the power and ground preroutes).

During the uncommit process when the soft macros are converted back into plan groups, the `uncommit_fp_soft_macros` command automatically propagates the child level Unified Power Format (UPF) constraints and the Milkyway and Design Compiler information from the soft macro to the top level of the design.

During the uncommit hierarchy process, the `uncommit_fp_soft_macros` command causes the following actions when it propagates the UPF constraints to the top level.

- Locates all power domains and their corresponding voltage area geometries and instantiates them in the top-level design, according to how they were instantiated in the soft macro. In the case of the topmost power domain in the soft macro, which has a default voltage area, the voltage area is instantiated in the top-level design with the same name as its corresponding power domain and with a geometry that is coincident with the plan group boundary.
- Reconnects the newly instantiated top-cell power domains to existing Milkyway objects.

A

Using the Flip-Chip Flow

This appendix describes how to use the flip-chip flow in IC Compiler. You can use hierarchical or virtual flat implementation flows for designs with flip-chip structures. The IC Compiler flip-chip interface can create both 45-degree and 90-degree redistribution layer routing.

The following two flip-chip design flows are supported in IC Compiler:

- Package-driven, also referred to as a bump-driven flow, where flip-chip I/O driver locations are optimally determined by package defined bump cell locations
- Die-driven, also referred to as an IC-driven flow, where individual bump cell locations are optimally determined by the I/O driver placement

The IC Compiler flip-chip interface supports two design styles: single I/O driver ring and multiple I/O driver ring. IC Compiler places the flip-chip drivers in a ring formation along the periphery of the chip.

In a flip-chip design, a bump cell represents a piece of metal or solder bump that forms an electrical contact to the chip package. Similar to processing silicon wafers for wire bonding, openings are made on the wafer to expose the metal contact points. Solder bumps are formed on these metal contact points. The chip's power, ground, and signal I/Os are available through the solder bumps that are formed on the top side of the wafer during the final processing step. The solder bumps on the chip make the physical connections to the package substrate. The chip is connected to external circuitry (a circuit board, another chip,

or a wafer) by flipping it over so that the solder bumps connect to metal pads to complete the interconnect. This is in contrast to wire bonding in which the chip is mounted upright and wires are used to connect the chip pads to external circuitry.

The flip-chip driver is the I/O circuitry that drives or receives signals from or to the chip through the bump. There are signal bump cells, which are connected to signal and clock nets, and power and ground bump cells, which are connected to power and ground straps. The locations of solder bumps on the chip are determined by the placement of the bump cells. (The I/O driver cells are placed independently.) You can place bump cells on top of the I/O drivers or anywhere within the core area. You can also place standard cells under bump cells.

This appendix includes the following sections:

- [Flip-Chip Implementation Flows](#)
- [Preparing the Library](#)
- [Creating an Initial Floorplan](#)
- [Using a Die-Driven Flip-Chip Flow](#)
- [Using a Package-Driven Flip-Chip Flow](#)
- [Routing Flip-Chip Bumps](#)
- [Using Flip-Chip Structures in Cover Macros](#)
- [Summary of the Flip-Chip Commands](#)

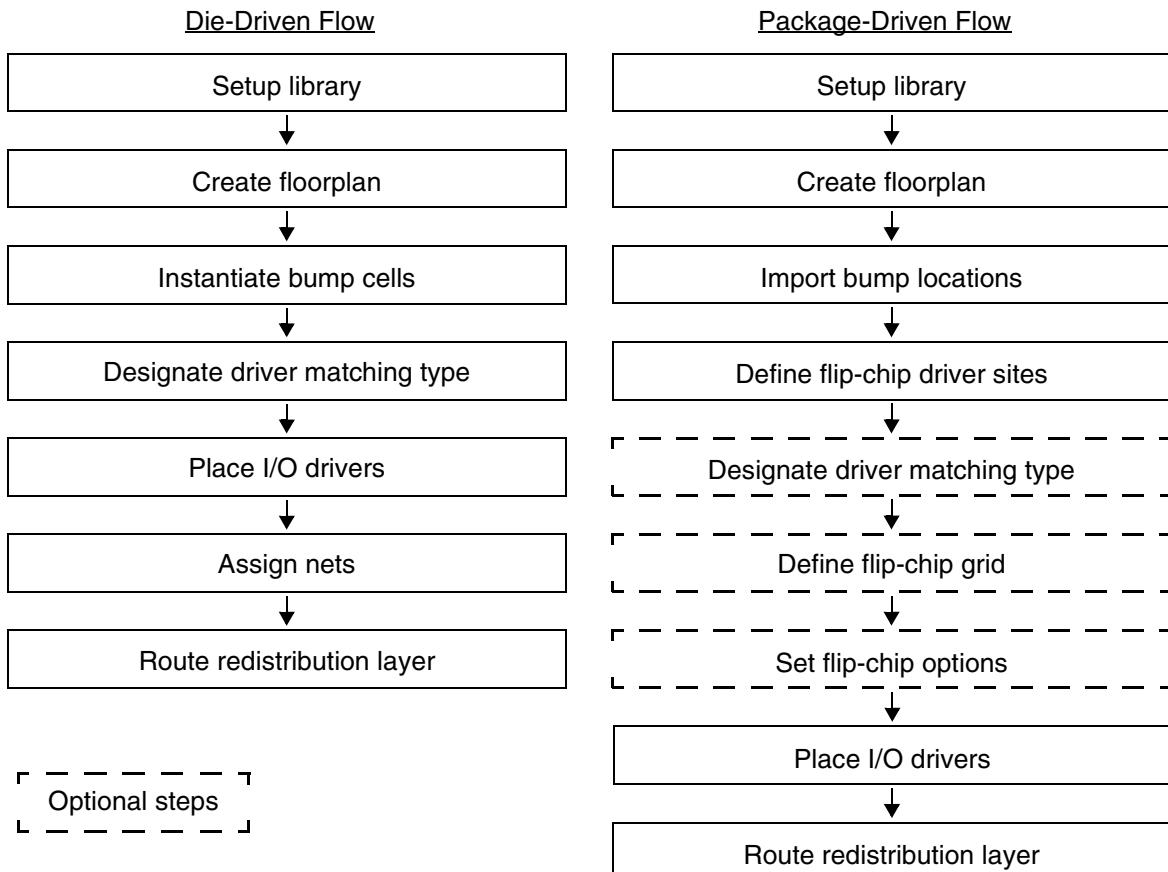
Flip-Chip Implementation Flows

IC Compiler supports two primary flows for flip-chip implementation: the die-driven flow and the package-driven flow. In the die-driven flow, the bump locations are dependent on the die and determine the bump locations for the IC package. Bump cells are not instantiated nor connected to flip-chip I/O drivers as inputs in the Verilog netlist. The optimal bump connections are determined by the placement of the flip-chip drivers and bump patterns.

In the package-driven flow, the IC package dictates the bump locations for the IC design. All of the bump cells are instantiated and connected to flip-chip drivers as inputs within the Verilog netlist. The physical locations for the bump cells are predefined by the package designer. The optimal flip-chip I/O driver locations are determined during flip-chip driver placement based on the predefined bump locations.

[Figure A-1](#) shows the high-level die-driven and package-driven flow diagrams. The following sections describe the steps in both flows.

Figure A-1 Flip-Chip High-Level Flows



Preparing the Library

IC Compiler recognizes flip-chip I/O drivers and bump cells only when they are assigned the correct cell type and port type properties in the Milkyway design library. This section describes the library preparation steps for the flip-chip I/O drivers and bump cells. Library preparation must be done in the Milkyway Environment tool by using Milkyway Scheme commands. Information about the Milkyway Environment tool is available on SolvNet.

I/O Drivers

To prepare the I/O drivers, open the design in the Milkyway Environment tool. Select Scheme mode and perform the following operations:

- Set the cell type as `IO Pad` by using the `cmMarkCellType` command.

```
cmMarkCellType
setFormField "Mark Cell Type" "Cell Type" "pad"
formOK "Mark Cell Type"
```

- Set the PAD port as `flipchip` by using the `dbSetCellPortTypes` command.

```
dbSetCellPortTypes "libname" "cellname" '(
  ("PAD" "inout" "flipchip")
) #f
```

- Set the power and ground pins as `flipchip` by using the `dbSetCellPortTypes` Milkyway command.

```
dbSetCellPortTypes "libname" "cellname" '(
  ("VDD" "inout" "Power" "flipchip")
  ("VSS" "inout" "Ground" "flipchip")
) #f
```

Bump Cells

You can maintain bump cells in the same Milkyway design library that you use for the I/O driver cells. To prepare the bump cells, open the design in the Milkyway Environment tool. Select Scheme mode and perform the following operations:

- Mark the bump cell as a flip-chip bump by using the `cmMarkCellType` command.

```
cmMarkCellType
setFormField mark_cell_type library_name lib
setFormField mark_cell_type cell_name cell
setFormField mark_cell_type cell_type "flip chip pad(bump)"
formOK mark_cell_type
```

- Set the port type.

```
dbSetCellPortTypes "/path/to/library" "BUMPCellName" '(  
    ("BUMP" "inout" "FlipChip" "Universal"  
) #f
```

- Create bump terminals (pins) on a redistribution routing layer by doing the following:
 1. Create a .lib file with a terminal name and a corresponding .db file. Without the .lib file, all bump cells are treated as physical-only cells.
 2. Locate the layer for the octagonal-shaped passivation opening in the flip-chip bump cell.
 3. Create a polygon in the CEL view by using the `geAddPolygon` command or choose Milkyway: Create > Polygon. Query the polygon to get a list of vertices.
 4. Enter the name or number of the layer on which you want to create the polygon, select the L45 routing option, and then add all the vertices.

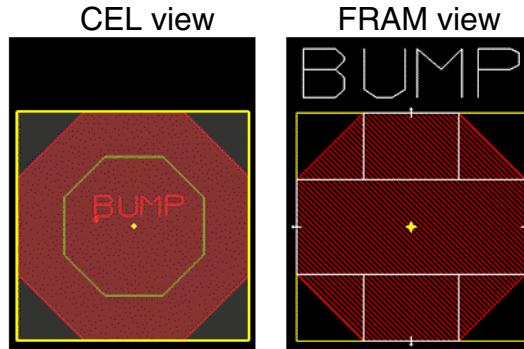
You should now have an octagonal piece of metal in the bump CEL view.

5. Add text to the layer by using the `geAddText` command. Use the text layer that matches the newly created terminal.
6. Create a FRAM view by using the `auExtractBlockPinVia` command.

The CEL and FRAM view should match the terminal name that you created in the .lib file.

[Figure A-2](#) shows the CEL and FRAM views of a bump cell.

Figure A-2 Bump CEL and FRAM Views



Creating an Initial Floorplan

Use the `create_floorplan` command to create an initial floorplan for your design, based on the input control parameters such as target utilization, aspect ratio, core size, row number, chip boundary, and wire tracks. You can also use the `read_def` command to import a previously saved floorplan.

Using a Die-Driven Flip-Chip Flow

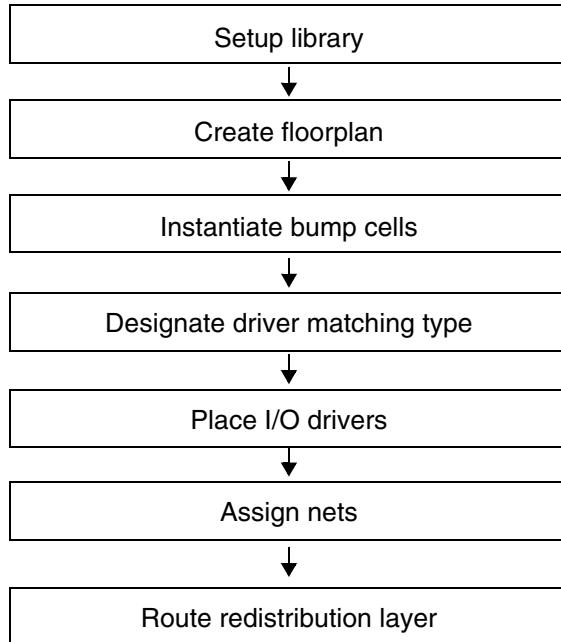
In a die-driven flow, the IC design dictates the flip-chip bump cell locations.

- The bump cells are not instantiated, nor are they connected to flip-chip I/O drivers in the Verilog netlist.
- The connections between the I/O drivers and individual bump cell locations are determined by the placement locations of the flip-chip I/O drivers and the bump pattern placement.
- The connectivity between the I/O ports and the I/O drivers can be output in the Verilog netlist.

The following topics are covered in this section:

- [Instantiating Bump Cells and Designating Matching Types](#)
- [Designating Matching Types for Bump Patterns](#)
- [Placing Flip-Chip I/O Drivers](#)
- [Assigning Nets From Bumps to I/O Drivers](#)

[Figure A-3 on page A-7](#) shows the die-driven flip-chip flow.

Figure A-3 Die-Driven Flip-Chip Flow

Instantiating Bump Cells and Designating Matching Types

After creating the floorplan in the die-driven flip-chip flow, you can instantiate the flip-chip bump patterns and prepare the bump cells for assignment by designating matching types. In the die-driven flip-chip flow, IC Compiler places bump cells in a ring configuration or an array configuration. The following sections describe the steps to create bump patterns and designate matching types.

Creating a Pattern of Bump Cells in a Ring Configuration

You can instantiate a specified number of bump cells and place them in your design in a ring configuration by using the `place_flip_chip_ring` command:

```

place_flip_chip_ring
  -physical_lib_cell {cell_name}
  -prefix prefix_string
  -number num_bump
  -bump_spacing spacing
  -ring_number count
  -ring_spacing spacing
  -boundary {x1 y1 x2 y2}
  [-left_orientation N | W | S | E | FN | FS | FW | FE]
  [-stagger_offset offset]
  [-extra_spacing extra_spacing]
  [-num_extra_spacing num_extra]
  
```

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

Specify the name of the bump reference cell by using the `-physical_lib_cell {cell_name}` option; you can specify only one library cell. The `-prefix prefix_string` option specifies the prefix string used to create the cell name. For example, the argument `-prefix "BUMP_"` specifies that the new bump cells are named BUMP_1, BUMP_2, and so on. The `-number bump_count` option specifies the number of bump cells to place. The `place_flip_chip_ring` command places bump cells until the specified number of bumps is placed or until all the flip-chip bump locations in each specified ring are allocated.

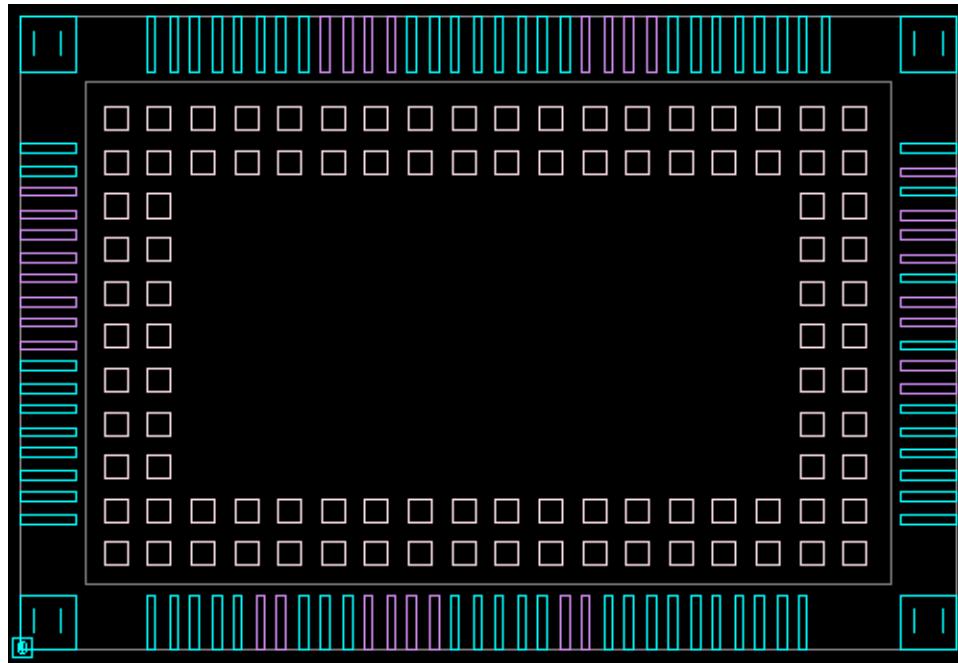
Specify the spacing between bump cells by using the `-bump_spacing spacing` option; specify the spacing between rings by using the `-ring_spacing spacing` option. Both spacing values are specified in microns. The `-ring_number count` option specifies the number of rings to create. The `-boundary {x1 y1 x2 y2}` option specifies the bounding box for the outermost ring.

In the following example, IC Compiler creates and places 100 flip-chip bump cells using the PAD80B library cell in a two-ring configuration. The spacing between the adjacent bump cells and the adjacent ring cells is 70 microns, and the outermost ring occupies the rectangle bounded by the coordinates {290 290} and {2940 1934}.

```
icc_shell> place_flip_chip_ring -physical_lib_cell {PAD80B} \
    -prefix "BUMP_" -bump_spacing 70 -ring_number 2 \
    -ring_spacing 70 -number 100 -boundary "290 290 2940 1934"
```

[Figure A-4 on page A-9](#) shows the layout after running the previous command.

Figure A-4 Placing Bump Cells in a Dual Ring Configuration



Creating an Array of Bump Cells

You can create a two-dimensional array pattern of flip-chip bumps by using the `place_flip_chip_array` command:

```
place_flip_chip_array
  -physical_lib_cell {phy_lib_cels}
  -prefix prefix
  -start_point {x y}
  -number num_bump
  -delta {x y}
  -repeat {columns rows}
  [-orientation N | W | S | E | FN | FE | FS | FW ]
  [-cell_origin lower_left | center ]
```

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

Define the bump cell array by specifying the starting point and the x- and y-pitch for each bump cell in the pattern. The `place_flip_chip_array` command can be used to place power and ground bump cells in the chip core. The `-start_point {x y}` option defines the starting point of the lower leftmost bump cell. The `-delta {x y}` option sets the horizontal and vertical pitch for each bump cell in the array. The `-repeat {columns rows}` option

specifies the number of columns and rows in the array. The `place_flip_chip_array` command places bump cells into the array pattern until the specified number of bumps is placed, or until the array pattern is full.

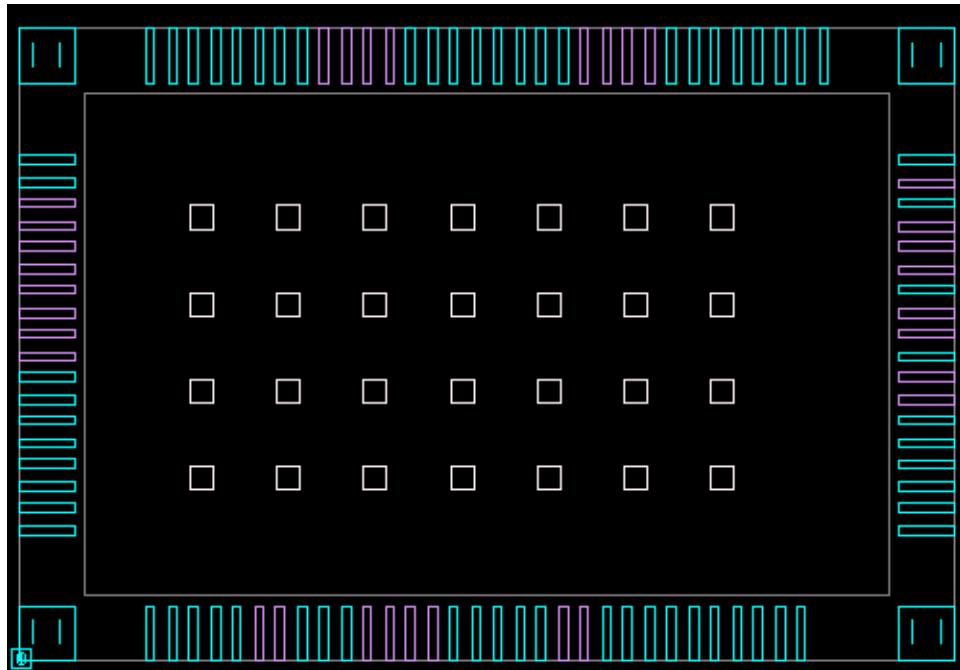
The `-physical_lib_cell`, `-prefix`, and `-number` options operate in the same way as in the `place_flip_chip_ring` command. For more information about these options see “[Creating a Pattern of Bump Cells in a Ring Configuration](#)” on page A-7.

In the following example, the `place_flip_chip_array` command creates 28 flip-chip bump cells and places them in a 7 by 4 array pattern, starting at the lower-left x- and y-coordinate of {590 590}. The bump cells are named with the `VDD_CORE_` prefix. The bump cells are placed by using a 300 micron center-to-center pitch in both the horizontal and vertical directions.

```
icc_shell> place_flip_chip_array -physical_lib_cell {PAD80B_P} \
    -prefix "VDD_CORE_" -start_point {590 590} -number 28 \
    -delta {300 300} -repeat {7 4}
```

[Figure A-5](#) shows the layout after running the previous command.

Figure A-5 Placing Bump Cells in a 7 by 4 Array Pattern



Designating Matching Types for Bump Patterns

Before routing the flip-chip drivers to the flip-chip nets, you can associate flip-chip drivers and bump cells by designating a matching type. The matching type for a cell has two primary functions: provide support for driver-to-bump pairing and provide support for driver placement. IC Compiler assigns flip-chip drivers to flip-chip bump cells that have the same matching type.

Use the `set_matching_type -matching_type name` command to designate bump pattern matching types for the flip-chip bump cells, I/O drivers, and specified pins or I/O ports.

Specify the matching type by using the name argument; the matching type is associated with a flip-chip driver or a flip-chip bump cell. You can run this command multiple times; the new matching type setting overrides the existing matching type setting. The matching type assigned to a driver can also be used to guide driver placement.

The following example assigns the matching type “signal” to all signal bump cells and I/O drivers in the design that have a reference cell name beginning with `BUMP_`.

```
icc_shell> set_matching_type \
           -matching_type "signal" [get_cells "BUMP_*"]
```

Use the `report_matching_type` command to display the matching type assigned to a collection of driver and bump cells.

```
icc_shell> report_matching_type [get_cells "BUMP_*"]
*****
Report : matching type
Design : flip_chip
Version: E-2010.12-SP1
*****
-----
Begin to report cell matching type:
cell      matching type
BUMP_21   signal
BUMP_0    signal
BUMP_22   signal
BUMP_10   signal
BUMP_11   signal
BUMP_23   signal
```

Use the following command syntax to reset the matching type and allow the driver to match any bump cell that does not have an assigned matching type.

```
icc_shell> set_matching_type -matching_type "" [get_cells "BUMP_*"]
```

Placing Flip-Chip I/O Drivers

After instantiating the bump cells in the die-driven flow, place the I/O drivers and assign matching types in preparation for connecting the drivers to the bump cells. I/O drivers can be placed in a ring, array, or strip configuration; the flip-chip grid defines the legal placement locations for the I/O drivers. The following sections define the procedure for defining the flip-chip placement grid, placing flip-chip I/O drivers, and assigning matching types.

Defining the Location for Flip-Chip Drivers

The `set_flip_chip_driver_array`, `set_flip_chip_driver_ring`, and `set_flip_chip_driver_strip` commands define the legal locations of the driver cells, whereas the `place_flip_chip_drivers` command actually places the drivers in the defined legal locations.

Note:

The `set_flip_chip_driver_strip` command is preferred over the `set_flip_chip_driver_ring` and `set_flip_chip_driver_array` commands because it does not require you to specify the maximum driver size. The `set_flip_chip_driver_strip` command can handle drivers of different widths.

The `set_flip_chip_driver_strip` command defines placement locations for flip-chip drivers in rows or columns within a specified bounding box; the command syntax is

```
set_flip_chip_driver_strip
  -bbox {{llx lly} {urx ury}}
  [-driver_orientation N | W | S | E | FN | FS | FW | FE]
  [-alignment left | top | right | bottom]
  [-matching_type matching_type_list]
  [-min_spacing spacing]
  [-name strip_name]
```

If you specify the `-matching_type` option, bump cells with the same matching type only are assigned to the drivers. You can rotate or mirror the driver placement by specifying the `-driver_orientation` option and align the drivers to the left, right, bottom or top sides of the bounding box by specifying the `-alignment` option. For more information about these options, see the man page.

The following example creates two flip-chip driver strips. The command assigns the "signal" matching type to the drivers and rotates the drivers by 180 degrees.

```
icc_shell> set_flip_chip_driver_strip -bbox {{0 0} {200 1000}} \
  -driver_orientation S -matching_type {"signal"}
icc_shell> set_flip_chip_driver_strip -bbox {{300 0} {500 1000}} \
  -driver_orientation S -matching_type {"signal"}
```

The `set_flip_chip_driver_ring` command defines the legal locations for flip-chip driver cells in a ring configuration; the command syntax is

```
set_flip_chip_driver_ring
  -matching_type {matching_type_list}
  -max_driver_size {width height}
  -max_driver_num_per_ring number
  -min_driver_spacing spacing
  -ring_number count
  -ring_spacing spacing
  -outer_ring boundary_box
  [-orientation N | W | S | E | FN | FS | FW | FE]
  [-num_extra_spacing count]
  [-extra_spacing spacing]
  [-stagger_drivers]
  [-name ring_name]
```

Specify which drivers to place in a particular ring by using the `-matching_type` option; the command places only the drivers with the specified matching types into the ring. Specify the maximum width and height of the drivers by using the `-max_driver_size` option. Specify the maximum number of drivers per ring by using the `-max_driver_num_per_ring` option. Set the spacing between driver cells and the spacing between adjacent rings by using the `-min_driver_spacing` and `-ring_spacing` options respectively. Specify the number of rings by using the `-ring_number` option; specify the bounding box that defines the extent of the outer ring by using the `-outer_ring` option.

The following example defines a peripheral ring of flip-chip drivers that meets the following constraints:

- Driver is assigned a "blue" or "green" matching type
- Driver is less than 20 um in width
- Driver is less than 100 um in height

In addition, the command places the flip-chip drivers in two rings, with the outermost ring bounded by the coordinates {100 100} and {900 900} and a spacing of 15.0 microns between the two rings. Each ring can contain at most 150 drivers with a minimum spacing of 1.2 microns between two adjacent drivers of the same ring.

```
icc_shell> set_flip_chip_driver_ring -ring_spacing 15.0 \
  -matching_type {"blue" "green"} -max_driver_size {20 100} \
  -max_driver_num_per_ring 150 -min_driver_spacing 1.2 \
  -ring_number 2 -outer_ring {{100 100} {900 900}}
```

The `set_flip_chip_driver_array` command defines the legal locations for flip-chip driver cells within an array configuration; the command syntax is

```
set_flip_chip_driver_array
  -matching_type {matching_type_list}
  -max_driver_size {width height}
  -start_point {x y}
  -dimension {column row}
  -delta {x y}
  [-orientation N | W | S | E | FN | FS | FW | FE]
  [-name array_name]
```

Restrict the placement of drivers to a particular array by using the `-matching_type` option. This technique can be used for multivoltage driver placement. Define the size of the largest driver by specifying the `-max_driver_size` option. Specify the driver placement grid by using the `-delta` and `-dimension` options; the `-delta` option specifies the pitch between driver cells and the `-dimension` option specifies the number of columns and rows in the driver array. Define the starting point at the lower-left corner of the driver array by specifying the `-start_point` option.

The following command defines a flip-chip driver array of 2 columns and 20 rows, starting at coordinate {0 100} on left edge of the die. The spacing between drivers is 60 microns, and the largest flip-chip driver allowed is 180 microns in width by 60 microns in height:

```
icc_shell> set_flip_chip_driver_array -max_driver_size {180 60} \
  -start_point {0 100} -dimension {2 20} -delta {0 60} \
  -matching_type {"blue" "green"}
```

Placing Flip-Chip Drivers

To place the flip-chip drivers, use the `place_flip_chip_drivers` command:

```
place_flip_chip_drivers
  [-matching_types {matching_type_list}]
  [-pin_alignment]
  [-snap_to_grid]
  [-swapping_only]
```

For information about the command options, see the `place_flip_chip_drivers` man page.

The `place_flip_chip_drivers` command places drivers with the `io_pad` and `flip_chip_driver` cell types into legal placement locations specified by the `set_flip_chip_driver_array`, `set_flip_chip_driver_ring`, or `set_flip_chip_driver_strip` command. IC Compiler places the drivers and minimizes wire length; you can use the `set_flip_chip_options` command to assign different weights to nets connecting drivers to bump cells and for nets connecting drivers to soft macros.

Place only drivers with a particular matching type by specifying the `-matching_types` option. Minimize jogs in the route between the driver cell and the bump cell by specifying the `-pin_alignment` option; IC Compiler attempts to align the driver center with the bump center. Snap the lower-left corner of the driver to the flip-chip grid by specifying the `-snap_to_grid` option. The flip-chip grid is defined by the `set_flip_chip_grid` command. Use the `-swapping_only` option to optimize the existing driver placement by swapping drivers of the same size.

Designating a Matching Type for Flip-Chip Drivers

Similar to flip-chip bump cells, you can designate a matching type for a flip-chip driver or flip-chip driver pin by using the `set_matching_type` command. Designating a matching type on a driver pin associates the driver with a bump cell with the same matching type for better driver placement. See “[Designating Matching Types for Bump Patterns](#)” on page A-11 for more information about matching types for flip-chip bump cells. The following example sets the matching type “power” for the driver cell `pad1`.

```
icc_shell> set_matching_type -matching_type {"power"} pad1
```

The next example sets the matching type “ground” for the pin named `PAD` on the driver `ppad2`.

```
icc_shell> set_matching_type -matching_type {"ground"} ppad2/PAD
```

Assigning Nets From Bumps to I/O Drivers

You can perform automatic net assignment by using the `assign_flip_chip_nets` command:

```
assign_flip_chip_nets
[-matching_type matching_type_list]
[-prefix prefix]
[-uniquify num_to_uniquify]
[-multiple_terminal_pins pin_name_or_collection]
[-terminal_layers layer_name_or_collection]
[-terminal_names terminal_name_list]
[-eco]
```

Automatic net assignment logically connects flip-chip I/O drivers and bump cells. This process is required because there is no logical connection between driver cells and bump cells in the Verilog netlist in the die-driven flow. After logical connections are established, you can route the bump nets between the flip-chip I/O drivers and bump cells.

The `-matching_type` option defines the list of matching types for the drivers and bumps to be routed. Based on the current driver placement, the command finds the nearest unassigned bump with the same matching type and assigns a logical net between the driver

and bump cell. The `assign_flip_chip_nets` command matches flip-chip bump cells to drivers by using the shortest wire length method. The bump-to-driver connection net assignments are updated in the Milkyway database for the die-driven flow.

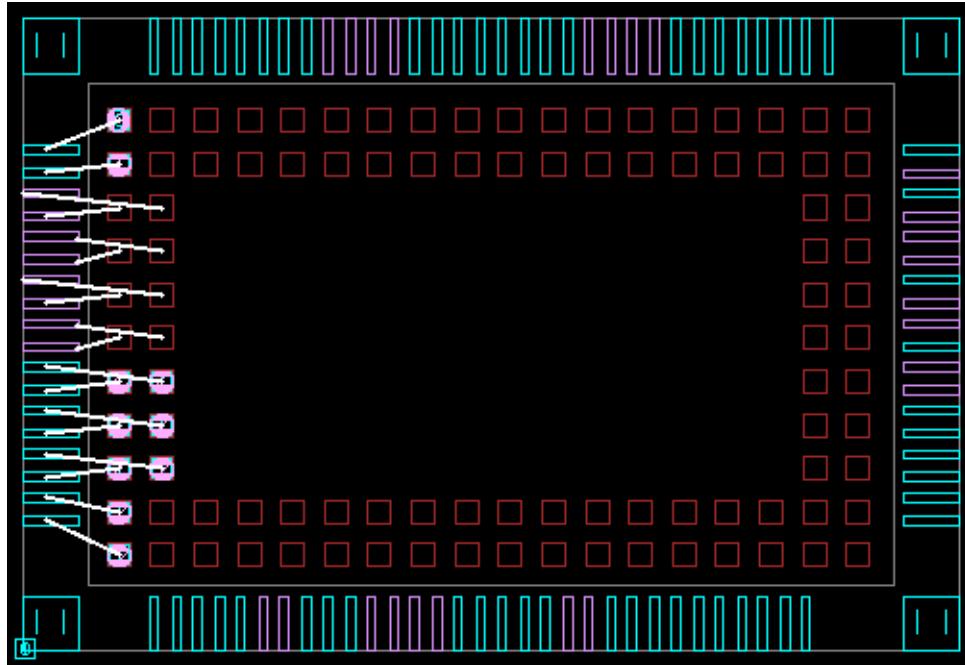
IC Compiler processes the automatic net assignment between drivers and bump cells based on the assigned matching types set by using the `set_matching_type` command. Only drivers and bump cells of an identical type are matched. Unspecified matching types are not processed if you use the `-matching_type` option.

The following example performs automatic net assignment between the drivers and bump cells assigned the matching type "west". The `display_flip_chip_route_flylines` command displays in the GUI the flylines between the driver and bump cells.

```
icc_shell> assign_flip_chip_nets -matching_type {"west"}
icc_shell> display_flip_chip_route_flylines
```

The results of the previous command are shown in [Figure A-6](#).

Figure A-6 Flip-Chip Net Assignment



Automatic net assignment can handle various design scenarios, such as a driver cell with multiple flip-chip pins, or a flip-chip pin with multiple terminals distributed far from each other. These requirements occur primarily on VDD, VSS, and control pins of macro drivers. Net assignment can also handle multiple pad pins per I/O driver, such as multiple bump cells connected to a single I/O driver. For multiple terminal pins, net assignment can assign one bump cell for each terminal. Net assignment can also handle multiple personality matching between drivers and bump cells, such as "power" drivers to "power" bump cells and "signal"

drivers to "signal" bump cells. If the ports of a group of drivers are connected by the same net, such as a VDD net, net assignment can create unique nets and assign a one-to-one connection between each bump and each flip-chip driver port. Net assignment can also create a many-to-one connection between a single bump and multiple flip-chip driver ports.

The following command performs a multiple-terminal-aware net assignment:

```
icc_shell> assign_flip_chip_nets -matching_type "east" \
    -multiple_terminal_pins DI/*
```

The following example performs one-to-one driver-to-bump matching. The `assign_flip_chip_nets` command supports multiple uniquify modes; for more information see the man page.

```
icc_shell> assign_flip_chip_nets -uniquify 1
```

Merging Multiple Flip-Chip Nets

You can merge multiple flip-chip nets into a single net by using the `merge_flip_chip_nets` command:

```
merge_flip_chip_nets
    -from from_nets
    -to to_net
    [-update_routing]
```

The `merge_flip_chip_nets` command disconnects all pins, I/O ports, and terminals from a specified collection of flat nets and reconnects them to the newly merged net. Use the `-from` option to specify the collection of nets to merge; the `-to` option specifies the new name for the merged net. You should use the `-update_routing` option if you use the `merge_flip_chip_nets` command after flip-chip redistribution layer routing.

The following command merges all nets that begin with the string VDD, such as VDD, VDD_1, VDD_2, and VDD_3, into the single VDD net:

```
icc_shell> merge_flip_chip_nets -from [get_flat_nets VDD* -all] \
    -to VDD -update_routing
```

Reporting Crossover Nets

To report crossover nets, use the `report_flip_chip_flyline_cross` command. The `report_flip_chip_flyline_cross` command checks crossover net assignment and determines if the flyline between a driver cell and a bump cell crosses any other flylines between other driver cells and bump cells. If one net consists of multiple drivers and bump cells, each net segment is tested against other driver cell and bump cell flylines.

The report generated by the `report_flip_chip_flyline_cross` command contains a list of net pairs that cross each other; [Example A-1](#) shows an example report.

Example A-1 report_flip_chip_flyline_cross Example

```
icc_shell> report_flip_chip_flyline_cross
*****
```

```
Report : flyline cross
```

```
Design : chip1
```

```
Version: E-2010.12
```

```
*****
```

The following net pairs cross each other
 net1 net2

REG_ADDR_1 [5]	REG_ADDR_1 [4]
TE1_ON_3	TE2_ON_3
DATA_2 [28]	REG_DATA_2 [29]
REG_DATA_3 [30]	IDLE_3
REG_ADDR_3 [1]	REG_ADDR_3 [2]
REG_DATA_2 [9]	REG_DATA_2 [8]
R_TMU0_ON_2	R_TMU1_ON_2
PE_CLK_4	RESET_4
FE_CLK_4	BUSY_4

Total 9 crossing!

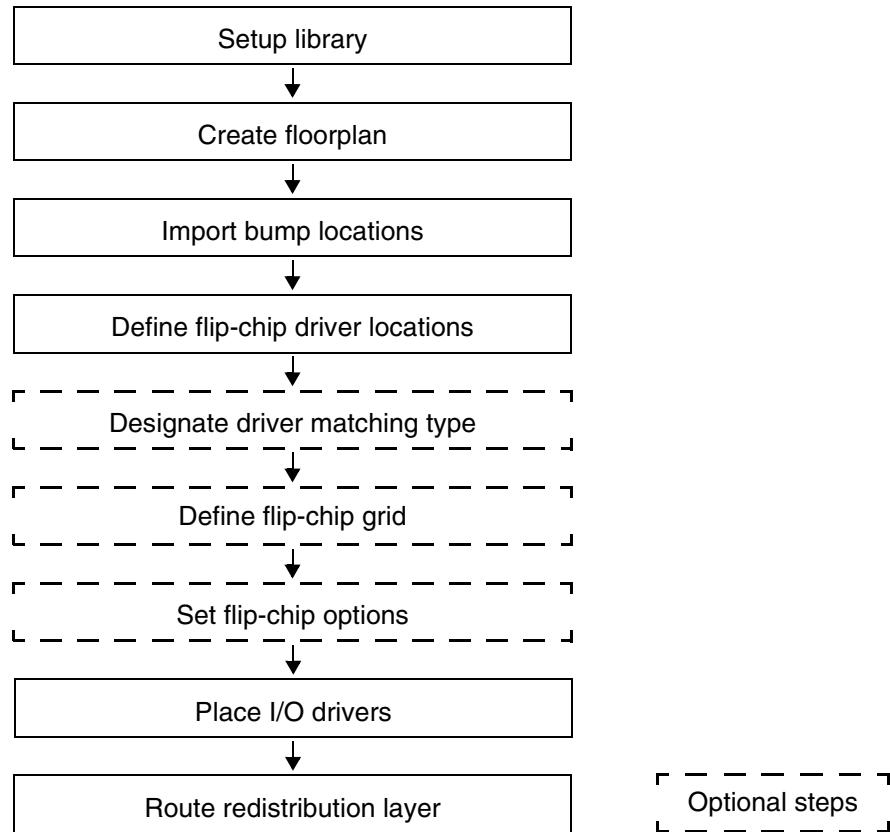
Using a Package-Driven Flip-Chip Flow

In a package-driven flow, the bump cells are imported and placed at physical locations predefined by the flip-chip package designer. A design processed in the package-driven flow has the following properties:

- The bump cells locations are defined in Design Exchange Format (DEF) or Advanced Input Format (AIF) files.
- The bump cells instances, I/O driver instances, and the net connectivity between the bump cells and I/O drivers are defined in the Verilog netlist.
- The I/O drivers are placed as close as possible to their associated bump cells.
- After the bump cells and I/O drivers are placed and the connectivity is defined, the redistribution layer routes are created to connect the bump cells to the I/O drivers.

[Figure A-7](#) shows the package-driven flip-chip flow.

Figure A-7 Package-Driven Flip-Chip Flow



Importing Bump Locations

You can import bump locations in DEF or AIF format. To import them in DEF format, use the `read_def` command as described in [“Reading DEF Files” on page 2-32](#). To import bump locations in AIF format, use the `read_aif` command:

```

read_aif
  [-pad_to_ref_lib {{ pad_type ref_name N | S | E | W } { ... }}]
  [-ignore_assign_nets]
  file_name
  
```

In a package-driven flow, bump cells are imported and placed in predefined physical locations per the requirements of the flip-chip package. The `read_aif` command reads the bump locations from an AIF file. Use the `-pad_to_ref_lib` option to remap the cell

reference in the AIF file. You can also change the orientation of the cell by specifying the `S`, `E`, or `W` keyword. To place the bump cells without connecting them, specify the `-ignore_assign_nets` option.

[Example A-2](#) shows an AIF file that specifies bump cell locations for a design. Note that the bump cell location coordinate {0 0} in an AIF file references the center of the die, while the coordinate {0 0} in IC Compiler references the lower-left die boundary.

Example A-2 Example AIF File

```
[DATABASE]
TYPE=AIF
VERSION=2.0
UNITS=UM

[DIE]
NAME=design
WIDTH=3230.000000
HEIGHT=2183.400000

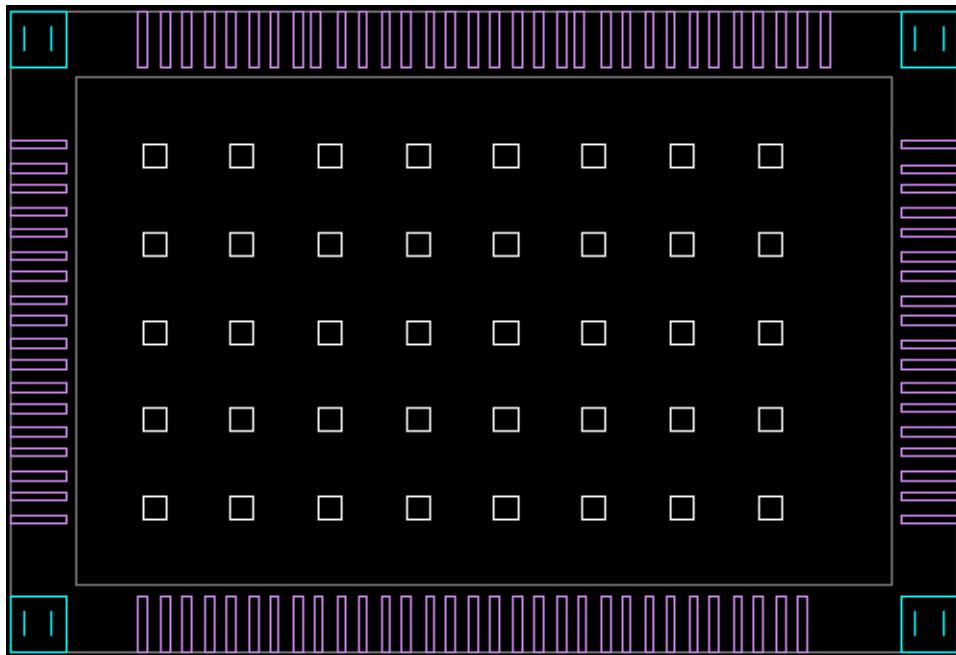
[PADS]
PAD1 = POLY 16.570,40.000 40.000,16.570 40.000,-16.570
      16.570,-40.000 -16.570,-40.000 -40.000,-16.570
      -40.000,16.570 -16.570,40.000 16.570,40.000
PAD2 = SQUARE 40
PAD3 = RECT 60 20

[NETLIST]

;Netname Pad# Type Pad_X Pad_Y Ball#
net[00] PAD_0 PAD1 -1125.0000 -601.7000
net[01] PAD_1 PAD1 -825.0000 -601.7000
net[02] PAD_2 PAD1 -525.0000 -601.7000
net[03] PAD_3 PAD1 -225.0000 -601.7000
net[04] PAD_4 PAD1 75.0000 -601.7000
net[05] PAD_5 PAD1 375.0000 -601.7000
net[06] PAD_6 PAD1 675.0000 -601.7000
net[07] PAD_7 PAD1 975.0000 -601.7000
net[08] PAD_8 PAD1 -1125.0000 -301.7000
net[09] PAD_9 PAD1 -825.0000 -301.7000
net[10] PAD_10 PAD1 -525.0000 -301.7000
...
net[28] PAD_28 PAD2 -525.0000 -1.7000
net[29] PAD_29 PAD2 -225.0000 -1.7000
...
net[38] PAD_38 PAD3 675.0000 598.3000
net[39] PAD_39 PAD3 975.0000 598.3000
```

Figure A-8 shows the layout after importing the AIF file shown in [Example A-2](#).

Figure A-8 Imported Bump Cells



Saving Bump Cell Information

You can write bump cell information, including the locations and net connections, to a DEF, AIF, or Tcl file by using the `write_def`, `write_aif`, or `write_floorplan` command respectively. Specify the name of the output file as an argument to the `write_aif` command; there are no other command options. The following example writes the current bump cell net connection, cell, and orientation information to the `design.aif` file.

```
icc_shell> write_aif design.aif
```

For more information about writing the floorplan information to a DEF or Tcl file, see [“Writing the Floorplan File” on page 2-29](#).

Defining Flip-Chip Driver Locations

You must define the legal locations for flip-chip driver placement. See [“Defining the Location for Flip-Chip Drivers” on page A-12](#) for information about defining the legal locations by using the `set_flip_chip_driver_strip`, `set_flip_chip_driver_ring`, and `set_flip_chip_driver_array` commands.

Designating the Driver Matching Type

I/O drivers and bump cells are associated by driver type; IC Compiler assigns connections between individual I/O drivers and bump cells based on matching type. Use the `set_matching_type` command to designate matching types for both I/O drivers and bump cells as described in “[Designating a Matching Type for Flip-Chip Drivers](#)” on page A-15.

Defining the Flip-Chip Placement Grid

You can define a flip-chip driver placement grid with the `set_flip_chip_grid` command:

```
set_flip_chip_grid
  -grid_origin {x y}
  -x_step x
  -y_step y
```

Note that the flip-chip placement grid is not the same as the standard cell row and column grid. Use the `set_flip_chip_grid` command to define the flip-chip placement grid by setting the minimum array pitch for flip-chip driver placement. Set the origin of the grid by specifying the `-grid_origin` option; set the horizontal and vertical grid spacing by specifying the `-x_step` and `-y_step` options.

The following example specifies a flip-chip grid with a grid origin at (0,0) and a grid spacing of 0.1 microns in both the x- and y-directions.

```
icc_shell> set_flip_chip_grid -grid_origin {0 0} \
  -x_step 0.1 -y_step 0.1
```

Setting Options for Flip-Chip Driver Placement

You can set flip-chip driver placement options by using the `set_flip_chip_options` command:

```
set_flip_chip_options
  [-disable_driver_placement]
  [-package_driven]
  [-one_softmacro_per_island]
  [-flip_chip_net_weight weight]
  [-softmacro_net_weight weight]
  [-signal_net_weight weight]
  [-guardband_width {x y}]
  [-multiple width | height | both]
```

Enable placement of flip-chip drivers based on fixed-bump location constraints by specifying the `-package_driven` option. When multiple width or height flip-chip drivers are placed, you can specify the `-multiple` option to allow the driver to consume more than one slot. See the man page for additional options.

Placing Flip-Chip Drivers

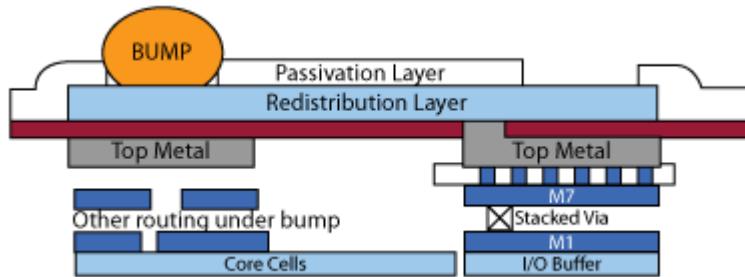
To place the flip-chip drivers, use the `place_flip_chip_drivers` command. The command places drivers with the “IO Pad” cell type into legal placement locations specified by the `set_flip_chip_driver_array`, `set_flip_chip_driver_ring`, or `set_flip_chip_driver_strip` command. IC Compiler places the drivers and minimizes wire length; you can use the `set_flip_chip_options` command to assign different weights for nets connecting drivers to bump cells and for nets connecting drivers to soft macros. For more information about the `place_flip_chip_drivers` command, see “[Placing Flip-Chip Drivers](#)” on page A-14.

Routing Flip-Chip Bumps

Before you can perform detail routing on your design, you must route the bump nets. Bump net routing, also referred to as redistribution layer routing, redistributes the wire-bonding pads to the bump pads without changing the placement of the I/O pads. Typically, bump nets are routed on the top metal layer of the die.

[Figure A-9](#) shows a cross section of redistribution layer routing.

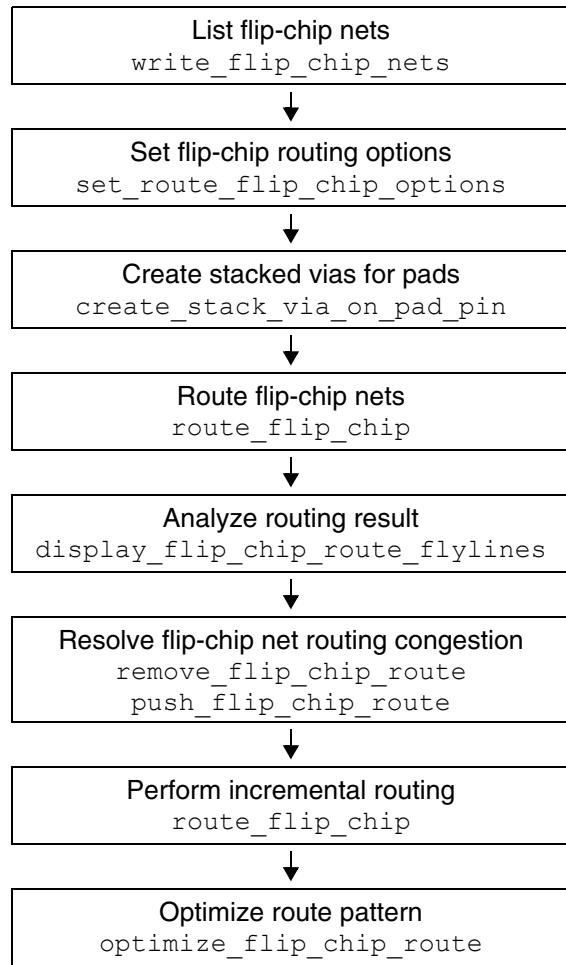
Figure A-9 Cross Section Showing Redistribution Layer Routing



Note:

For peripheral I/O flip-chip design styles, single-layer routing on the top metal layer is preferred. In addition, single-layer routing on the top metal layer minus one level (dual-layer routing) is also allowed when necessary.

[Figure A-10](#) on page A-24 describes the flow for performing flip-chip routing.

Figure A-10 Flip-Chip Routing Flow

Writing Flip-Chip Nets to a File

You can create a file containing the list of flip-chip nets in the current design by using the `write_flip_chip_nets` command. Use the `-file_name` option to specify the output file name. The list of flip-chip nets can be used to create a list of flip-chip nets to be routed by the `route_flip_chip` command.

The following command writes the flip-chip nets to the `bump_net_list` file.

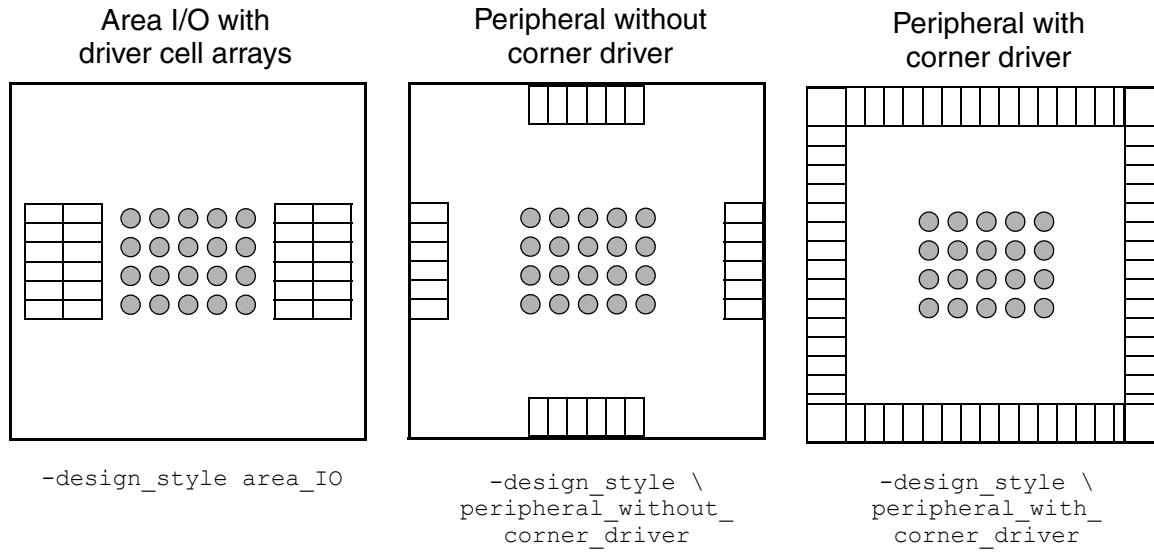
```
icc_shell> write_flip_chip_nets -file_name bump_net_list
Dumping successfully.
```

Setting Flip-Chip Routing Options

Before routing the flip-chip nets between the driver cells and bump cells in your design, you can use the `set_route_flip_chip_options` command to constrain the flip-chip router.

Use the `-design_style` option to specify the type of flip-chip driver and pad configuration for your design. The three supported flip-chip driver and pad configurations are shown in [Figure A-11](#).

Figure A-11 Flip-Chip Pad Layout Configurations



Define the flip-chip routing rules and options by specifying the `-layer_spacing`, `-layer_width`, and `-rule_name` options. Define the minimum length of the first wire segment connecting to the driver or bump cell by specifying the `-min_access_edge_length` option. Save a list of unrouted flip-chip nets to a file by specifying the `-output_unrouted_nets` option.

Creating Stacked Vias on Pad Pins

The `create_stack_via_on_pad_pin` command creates stacked vias between two specified design layers.

```
create_stack_via_on_pad_pin
  -from_metal layer
  -to_metal layer
  [-remove_existing_stack_via true | false]
  [-route_type user_enter | signal_route]
  [-nets collection_of_nets | -nets_in_file file_name]
  [-all_driver_pins true | false]
  [-terminal_names {list_of_terminal_names}]
```

Stacked vias connect the pad pins on the redistribution layer to lower layers of metal in your design. Define the layers to be connected by specifying the `-from_metal` and `-to_metal` options. When creating the stacked via, IC Compiler determines the size of the stacked via based on the nondefault width routing rule and shape of the pad pin. IC Compiler finds the largest pin shape for placing the via that does not cause a DRC violation. You must create stacked vias before routing the flip-chip nets by using the `route_flip_chip` command.

The following command creates stacked vias between the M6 layer and the M8 layer.

```
icc_shell> create_stack_via_on_pad_pin -from_metal M6 -to_metal M8
Created stack vias on 99 pad pin(s).
```

Routing Flip-Chip Nets

You can route the flip-chip nets by using the `route_flip_chip` command:

```
route_flip_chip
  -routing_layer layer
  [-nets collection_of_nets | -nets_in_file nets_file]
  [-route_by_input_net_order]
  [-45_degree]
  [-terminal_names list_of_terminal_names]
```

Use the `route_flip_chip` command after defining routing constraints and creating stacked vias. Define the layer used for the flip-chip routing layer by specifying the `-routing_layer` option. To allow 45-degree flip-chip routing, specify the `-45_degree` option. During 45-degree redistribution layer routing, a flip-chip route near a bump cell or a pad pin might result in an acute angle DRC violation. You can control the wire length of the wire segment that enters the bump cell or leaves the pad pin by using the `set_route_flip_chip_options -min_access_edge_length` command. Changing the minimum access edge length increases the chance of successful net splitting.

In some flip-chip designs, I/O pads or flip-chip drivers contain electrically equivalent pins that must have separate routes to the same or different bump cells. Electrically equivalent pins are supported in redistribution layer routing in the flip-chip flow. Using this feature, you can do the following:

- Create separate routes to the same bump cell or to different bump cells for specified flip-chip drivers or I/O pads with electrically equivalent pins; this is also referred to as multiple-terminal net connections
- Choose a specific electrically equivalent pin for routing
- Perform routing on multiple electrically equivalent pins for power and ground nets to reduce IR drop

You can use the `-terminal_names` option to selectively route from the specified driver pins to the connected bump cells. The logic connections between the electrically equivalent pins and the bump cells must first be established by using the `assign_flip_chip_nets` command.

Each electrically equivalent pin requires its own stacked via. Use the `create_stack_via_on_pad_pin -terminal_names` command to create stacked vias on the flip-chip I/O drivers from the pad pin to the top metal layer for flip-chip routing.

Analyzing Flip-Chip Routing Results

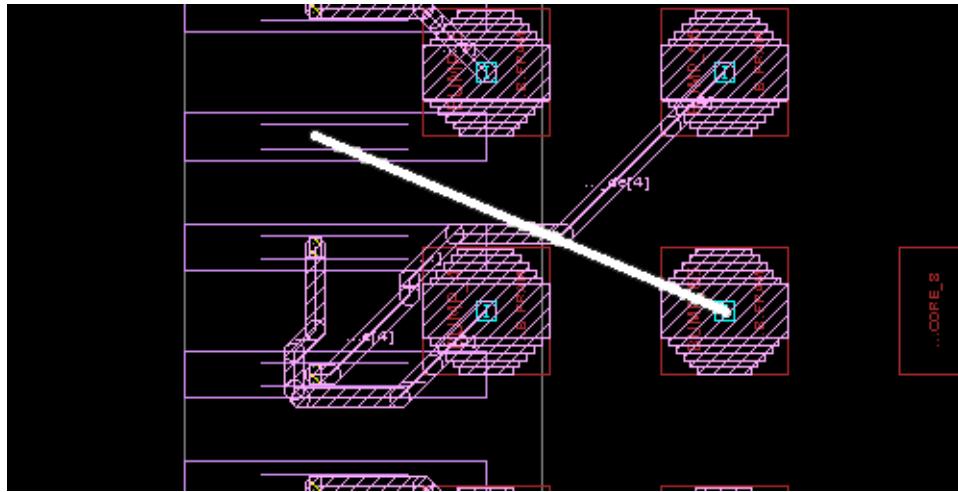
You can visually inspect your design for crossed flip-chip nets by using the `display_flip_chip_route_flylines` command:

```
display_flip_chip_route_flylines
  [-nets {collection_of_nets} | -nets_in_file nets_file]
  [-open_nets [-output_open_nets open_net_file]]
```

The `display_flip_chip_route_flylines` command displays a flyline between the driver and bump cell for each flip-chip net that could not be routed. The following command displays flylines for all unrouted flip-chip nets; [Figure A-12 on page A-28](#) shows the layout after running the command.

```
icc_shell> display_flip_chip_route_flylines -open_nets
```

Figure A-12 Flyline Showing Open Flip-Chip Net



You can also display a list of flip-chip flylines that are crossed by using the `report_flip_chip_flyline_cross` command; for more information see “[Reporting Crossover Nets](#)” on page [A-17](#).

Resolving Flip-Chip Net Routing Congestion

You can use the `remove_flip_chip_route` command to delete the flip-chip routes between driver and pad cell for all flip-chip nets, or for specified nets only. The `remove_flip_chip_route` command uses the following syntax:

```
remove_flip_chip_route
  [-nets net_names | -nets_in_file nets_file]
  [-width width]
  [-contact]
```

The `remove_flip_chip_route` command reduces flip-chip net routing congestion by removing the specified flip-chip routes from the design. To specify the flip-chip routes to remove, use the `-nets` option. You can also save the net names to a file and specify the `-nets_in_file file_name` option to remove the routes associated with those nets. If you do not specify any options, the command removes all flip-chip routes.

The following command deletes the flip-chip route between the pad cell and driver cell for nets VSS_1 and VSS_2.

```
icc_shell> remove_flip_chip_route -nets {vss_1 vss_2}
```

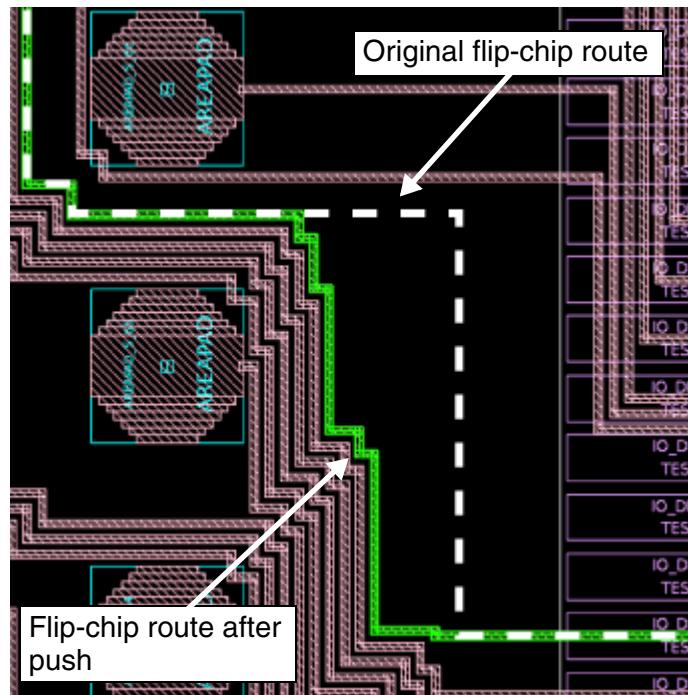
You can use the `push_flip_chip_route` command to make small adjustments to individual flip-chip routes:

```
push_flip_chip_route
  -nets collection_of_nets | -nets_in_file nets_file
  [-layer tech_layer_name]
  [-direction up | down | left | right]
  [-net_push | -neighbor_push]
  [-sweep_range sweep_range]
  [-bounding_box {{llx lly} {urx ury}}]
```

By moving flip-chip routes, you can provide more space for routing, reduce flip-chip routing congestion, and improve the quality of the flip-chip routing.

The `push_flip_chip_route` command pushes flip-chip nets up, down, left, or right as specified by the `-direction` option. The `-net_push` option moves the net until the net is blocked by a routing obstruction or the net meets the bounding box defined by the `-bounding_box` option. [Figure A-13](#) shows the layout result after running the `push_flip_chip_route -direction down -net_push -nets {net1}` command; the dashed line represents the original flip-chip net route.

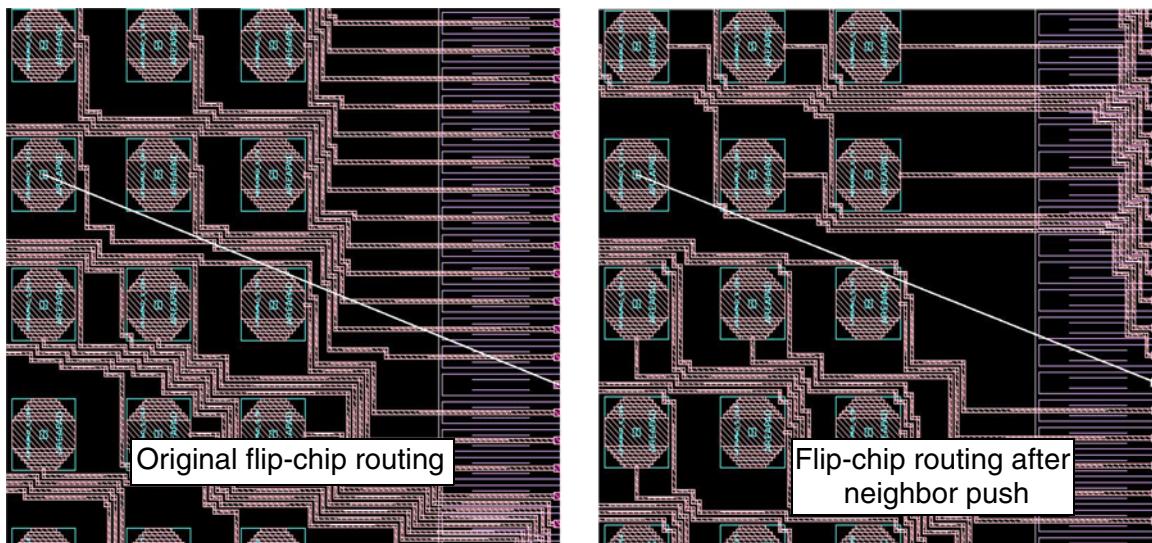
Figure A-13 Flip-Chip Route Pushed in Down Direction



The `-neighbor_push` option opens a pathway for a target net by pushing away neighboring wires. In this mode, the `-direction` option is not used and the `-sweep_range N` option specifies the number of nearby wires to be pushed away on each side of the net pathway. For example, if you specify `-neighbor_push -sweep_range 5 -nets {net1}`, the command clears a pathway for net1 by pushing away the five nearest routes on each side of the pathway.

If the `-sweep_range` option is not specified with `-neighbor_push`, the default of 10 is used, which affects up to 10 routes on each side of the pathway. [Figure A-14](#) shows an example of this type of route push.

Figure A-14 Flip-Chip Routes Pushed by Using Neighbor Push



Performing Incremental Routing and Route Optimization

After performing the initial flip-chip routing and changing or removing individual routes, you can perform incremental flip-chip routing by rerunning the `route_flip_chip` command. You can select which nets to route by specifying the `-nets` or `-nets_in_file` option.

You can improve the quality of the flip-chip routing by using the `optimize_flip_chip_route` command:

```
optimize_flip_chip_route
  -layer tech_layer_name
  [-nets collection_of_nets | -nets_in_file nets_file]
  [-change_route_type user_enter | signal_route]
  [-split_net]
```

The `optimize_flip_chip_route` command optimizes the routing pattern by removing unnecessary jogs. Jogs might be formed after running the `push_flip_chip_route` command or after performing a manual edit of the route. The `optimize_flip_chip_route` command can also split a flip-chip net into two wires of equal length; this can reduce the IR drop on power and ground nets without violating maximum net width DRC rules.

Use the `-layer` option to specify the metal layer on which to optimize the flip-chip wires. Specify the `-change_route_type` option to redefine the route type assigned to each flip-chip route; specify `-change_route_type user_enter` to change the type to User Enter or specify `-change_route_type signal_route` to change the type to Signal Route. Use the `-split_net` option to split a routed 45-degree net into two thinner nets that are routed in parallel.

Using Flip-Chip Structures in Cover Macros

IC Compiler supports flip-chip bump and cover cells in the logical netlist. You can describe the logical connectivity between the flip-chip bump pads and cover cells and the I/O drivers in the Verilog netlist.

You can change the flip-chip bump and cover cells from physical-only cells to become part of the logical netlist by setting the following variable to `true`.

```
icc_shell> set_app_var disable_bump_cover_as_physical_only true
```

Setting this variable to a value of `false` treats bump and cover cells as physical-only.

Summary of the Flip-Chip Commands

[Table A-1](#) provides a summary of the flip-chip commands. For more information, see the appropriate man pages.

Table A-1 Summary of Flip-Chip Commands

Command	Description
<code>assign_flip_chip_nets</code>	Creates or reconnects nets between flip-chip I/O drivers to bump cells.
<code>create_stack_via_on_pad_pin</code>	Creates stacked vias for the I/O drivers from the pad layer to the top metal layer for flip-chip routing.

Table A-1 Summary of Flip-Chip Commands (Continued)

Command	Description
display_flip_chip_route_flylines	Displays the flylines for the specified flip-chip nets in the layout window of the IC Compiler GUI. You can view unrouted nets and identify any nets that are creating bottlenecks.
expand_flip_chip_cell_locations	Proportionally expands or shrinks the relative locations of the selected flip-chip bump or driver cells.
merge_flip_chip_nets	Disconnects pins, I/O ports, and terminals from a specified collection of flat nets and reconnects them to the newly merged net.
optimize_flip_chip_route	Improves and enhances the flip-chip routing patterns by running L-shape and Z-shape optimizations on the flip-chip nets. The command routes the flip-chip nets on a single metal layer.
place_flip_chip_array	Creates the specified number of flip-chip bump cells and places them in a two-dimensional array pattern.
place_flip_chip_ring	Creates the specified number of flip-chip bump cells and places them in a ring configuration.
push_flip_chip_route	Pushes the specified flip-chip net routes in a specified direction or away from the specified nets.
read_aif	Reads the flip-chip bump locations from an Advanced Input Format file.
remove_flip_chip_route	Removes the specified flip-chip wires, paths, and contacts to resolve routing congestion.
report_flip_chip_bump_attributes	Reports the attributes set on the flip-chip bump cells for flip-chip routing.
report_flip_chip_driver_bump	Prints a table of bump cells and corresponding nets and drivers.
report_flip_chip_flyline_cross	Generates a report that lists flip-chip net pairs that cross.

Table A-1 Summary of Flip-Chip Commands (Continued)

Command	Description
<code>route_flip_chip</code>	Performs flip-chip redistribution layer routing to connect flip-chip I/O drivers and the bump cells.
<code>set_flip_chip_bump_attributes</code>	Sets attributes on flip-chip bump cells for flip-chip routing.
<code>set_flip_chip_cell_site</code>	Modifies flip-chip driver cell site properties to set different flip-chip driver legal location constraints.
<code>set_flip_chip_driver_array</code>	Defines the legal locations for flip-chip driver cells to be placed in an array configuration.
<code>set_flip_chip_driver_island</code>	Defines the flip-chip cell sites that form the legal locations for the flip-chip drivers to be placed in an island style. This command cannot be used together with the <code>place_flip_chip_drivers</code> command.
<code>set_flip_chip_driver_ring</code>	Defines the legal locations for flip-chip driver cells to be placed in a ring configuration.
<code>set_flip_chip_driver_strip</code>	Defines the legal locations for flip-chip driver cells to be placed in a strip configuration.
<code>set_flip_chip_grid</code>	Creates equally-spaced grid points for flip-chip driver placement.
<code>set_flip_chip_options</code>	Sets general flip-chip driver placement options.
<code>set_matching_type</code>	Assigns the specified matching type to the specified I/O drivers, bump cells or pins.
<code>set_route_flip_chip_options</code>	Defines the flip-chip routing options.
<code>update_flip_chip_pin_locations</code>	Updates the flip-chip bump I/O pin locations for timing analysis.
<code>write_flip_chip_nets</code>	Writes all bump to pad net names into a file.

Index

A

acceptable overflow criteria 3-14

B

black boxes

- create FRAM view 4-8
- flattening existing instances 4-9
- managing the properties for 4-8
- read Verilog netlist 4-8
- sizing by gate equivalence 4-6

block level designs, performing simultaneous placement and pin assignment 6-10

C

CEL view, working with soft macros 13-9

clock

- latency 9-5, 12-31

clock planning

- performing plan group-aware clock tree synthesis 9-8
- using multivoltage designs 9-7

clock skew analysis 9-4

commands

- copy_floorplan 2-33
- create_plan_groups 7-4
- create_qtm_model 4-11

create_voltage_area 6-59
estimate_fp_area 3-8, 3-15, 3-25
flatten_fp_black_boxes 4-9
get_fp_wirelength 6-52
get_voltage_areas 6-62
optimize_fp_timing 10-5
pack_fp_macro_in_area 6-55
place_fp_pins 10-2
read_def 2-32
read_floorplan 2-33
read_saif 8-75
remove_on_demand_netlist_data 5-10
remove_plan_groups 7-4
report_fp_placement 6-49
report_qtm_model 4-12
set_keepout_margin 3-11
set_qtm_global_parameter 4-11
set_qtm_technology 4-11
set_switching_activity 8-75
shape_fp_blocks 7-22
shape_fp_blocks multiple instantiated modules
shaping plan groups in (shape_fp_blocks) 6-65
update_voltage_area 6-62
write_qtm_model 4-12
conventions for documentation 1-xxii
copy_floorplan command 2-33
creating

pin guides 11-27
customer support 1-xxiii

D

DEF file
reading 2-32

E

Estimate Area GUI
floorplan control options 3-9
using 3-8
using power network control options 3-15
using search control options 3-14
estimate_fp_area options
high_utilization_bound 3-19
pre_route_post_pns 3-24
run_pns_script 3-24
save_as_cel 3-24

F

floorplan control options
increase spacing in the core area 3-13
maintain I/O pad alignment 3-12
recognizing blockages 3-10
select sizing type 3-9
specify core area boundaries 3-10
floorplan file, reading 2-33

G

get_voltage_areas command 6-62

H

hard macros
creating a user-defined array 6-27
manually adjusting 6-56
packing into an area 6-55

hierarchical model
flow for creating 5-4
instantiating and using at the top level 5-12
overview 5-2
viewing in GUI 5-12

I

in-place optimization
pin placement 10-7
in-place optimization, running 10-2

L

level shifters 9-7

M

MinChip
using multivoltage designs 3-25
MinChip technology
performing steps inside 3-3
preparing the design
blockages over hard macros 3-8
edge blockages 3-7
filler cells 3-8
floorplan 3-4
hard macro packing 3-6
placement blockages 3-7
power network synthesis scripts 3-4
preroute standard cell scripts 3-5
relative hard macro placement 3-5
sliver size 3-6
multiple instantiated modules
placement of 6-63
multivoltage designs
using in clock planning 9-7
using in MinChip 3-25

O

objects

placing and shaping in design core 7-18
optimizing pins for block level designs 6-10
overview of hierarchical models 5-2

P

packing hard macros 6-55
physical data
 copying 2-33
physical hierarchy
 commit 13-1
 uncommit 13-9
physical objects, connected to power and ground 13-9
pin assignment
 analyzing quality of 11-32
 results, analyzing 11-1
pin guides
 creating 11-27
pin overlaps, removing 11-14
pin placement flow, detecting buses 11-4
pin placement, in-place optimization 10-7
placement
 measuring QoR 6-48
plan group-aware routing, detecting buses 11-4
plan groups
 creating 7-2
 defining the shape of 7-3
 exclusive 7-2
 removing 7-4
 uncommit hierarchy 13-1
pre-budget timing analysis 12-3
propagate preroutes 13-1
pushed up
 physical objects 13-10
pushed up, physical objects 13-9

Q

quick timing model for black boxes
 command summary 12-11
 using Tcl commands to create 4-11

R

read_def command 2-32
read_floorplan command 2-33
reading
 DEF file 2-32
real clock latencies, creating budgets for 12-31

S

search control options
 acceptable overflow 3-14
 auto routability 3-14
shaping objects 7-19
sliver_size option, using for estimate_fp_area 3-6
sliver_size parameter 3-11
soft macros
 discarded pins 11-7
 properties set in CEL view 13-9
 uncommit physical hierarchy 13-1
SolvNet
 accessing 1-xxiii
 documentation 1-xxi
 Download Center 1-xx
switching activity, annotating 8-75

T

timing budgeting
 creating real clock latencies 12-31
 performing in design planning 12-1
 pre-budget timing analysis 12-3
prerequisites 12-2
running 12-6

U

update_voltage_area command 6-62

planning location and shape of 6-57

removing commands

remove_voltage_area 6-62

supporting physical boundary scenarios 6-61

updating 6-62

V

voltage areas