

IC Compiler™ II Design Planning User Guide

Version T-2022.03-SP1, April 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

© 2022 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>.

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

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

New in This Release	12
Related Products, Publications, and Trademarks	13
Conventions	13
Customer Support	14
<hr/>	
1. Introduction to Design Planning	15
Design Planning Overview	15
Hierarchical Design Planning Flow	16
Design Partitioning	18
Deciding on the Physical Partitions	18
Design Planning at Multiple Levels of Physical Hierarchy	19
Design Validation During Design Planning	21
<hr/>	
2. Working in Design Planning Mode	24
User Interfaces	24
Starting the Command-Line Interface	25
Exiting the IC Compiler II Tool	26
Entering <code>icc2_shell</code> Commands	26
Interrupting or Terminating Command Processing	27
Getting Information About Commands	27
Displaying Command Help	27
Using Variables	28
Using Application Options	28
Viewing Man Pages	29
Using Tcl Scripts	30
Generating a Tcl Script From Spreadsheet Data	30
Using Setup Files	35
Using the Command Log File	36
Running Tasks in Parallel	36
Monitoring Distributed Tasks	36

3. Splitting Constraints	40
Split Constraints Flow	40
Split Constraints Output Files	42
Split Constraints Example 1	44
Split Constraints Example 2	45
 4. Creating a Floorplan	46
Supported Types of Floorplans	47
Channeled Floorplans	47
Abutted Floorplans	47
Narrow-Channel Floorplans	48
Reading the Verilog Netlist	49
Creating Dense and Sparse Outline Views for Large Designs	49
Creating an Initial Floorplan	51
Creating an L-, T-, or U-Shaped Floorplan	52
Creating a Complex Rectilinear Floorplan	53
Creating a Floorplan Based on a Physical Rules File	54
Adjusting the Floorplan Aspect Ratio	55
Defining Routing Tracks	56
Flipping the First Row in the Floorplan	58
Updating the Floorplanning Without Disturbing Specified Object Types	60
Validating the FinFET Grid	61
Scaling the Die Area	62
Reporting Floorplan Information	65
Reading, Writing, and Copying Floorplan Information	65
Reading in Floorplan Information	66
Reading DEF Files	66
Writing Out Floorplan Files	67
Copying Floorplan Objects From Block to Block	67
Creating and Validating Floorplan Rules for Advanced Technology Nodes	73
Creating Additional Floorplan Rules	73
Floorplan Area Rules	74
Floorplan Enclosure Rules	74
Floorplan Halo Rules	74

Contents

Floorplan Spacing Rules	75
Floorplan Width Rules	75
Floorplan Density Rules	75
Validating Floorplan Rules and Repairing Errors	76
<hr/>	
5. Handling Black Boxes	78
Identifying Potential Black Boxes	79
Creating Black Box References	82
Creating a Black Box Timing Model	84
Black Box Timing Example	85
UPF for Black Box Designs	87
<hr/>	
6. Planning I/Os and Flip-Chip Bumps	88
Validating Library Requirements	90
Creating I/O Rings or Guides	92
Creating Arrays of Bump Cells	94
Creating Rows or Columns of Bump Cells	97
Placing Bump Cells in Predetermined Locations	101
Creating Power I/O Placement Constraints	102
Creating Signal I/O Placement Constraints	105
Assigning I/O Pads to Bumps	110
Automatic Bump Assignment	111
User-Specified Bump Assignment With Matching Types	112
Placing I/Os and Writing Constraints	115
Adding and Placing Physical-Only I/O Cells	116
Inserting Corner Cells	117
Inserting Break Cells	117
Inserting Filler Cells	118
Checking I/O Placement	119
Routing RDL Nets	122
Optimizing RDL Routes	130
Creating RDL Net Shields	132
Creating a Flip-Chip Design With Multiple Levels of Hierarchy	135

7. Creating a 3DIC Design	138
Preparing Libraries for a 3D IC Design	140
Preparing the Designs for the Active Die	140
Creating the Silicon Interposer Design	141
Reading and Writing Bump Cell and TSV Locations Using CSV Files	143
Creating the Top-Level Design	147
Creating Virtual Blocks	151
Creating the Power and Ground Meshes for the Interposer	152
Routing the Interposer	153
3D IC Glossary of Terms	160
 8. Managing Design Blocks	161
Exploring the Design Hierarchy	162
Creating Module Boundaries	162
Committing Design Blocks	166
Creating Block Placement Abstracts	166
Changing the Block Boundary and Block Origin	168
Moving Objects Between the Top and Block Levels	169
Pushing Objects Down From the Top Level to the Block Level	169
Push Down Object Options	171
Pushing Down Feedthrough Nets	172
Using Via Locations as New Pin Locations During Push Down	173
Writing Pushed Down Objects to the Netlist	174
Popping Objects Up From the Block Level to the Top Level	175
Pop Up Object Options	176
Generating ECO Scripts for Netlist Editing	176
 9. Performing Block Shaping and Macro Placement	180
Setting Macro Constraints	181
Creating Relative Placement Constraints for Macros and Macro Arrays	183
Setting Macro Keepouts	185
Reserving Space for Macros Between Connected Blocks	187
Creating Macro Arrays	187

Contents

Shaping Blocks	188
Creating Block Shaping Constraints	190
Performing Block Shaping With Tcl Constraints	196
Creating Channel Constraints for Block Shaping	201
Creating the Block Grid for Multiply Instantiated Blocks	204
Creating the Initial Macro Placement	207
Identifying Channels Between Objects	209
Application Options for Macro Placement	211
Performing Freeform Macro Placement	214
Integrated Freeform Macro Placement	217
Performing Machine Learning-Based Macro Placement	220
Placing Macros Island Style	224
Placing Macros Away From Block Edges	225
Placing Macros Among Standard Cells	227
Using the Macro Placement Assistant	227
Creating Macro Groups	228
Performing Other Tasks on Macros	232
Performing Timing-Driven and Congestion-Driven Placement	235
<hr/>	
10. Performing Power Planning	237
Creating and Connecting Power Nets	240
Defining PG Via Masters	241
Creating Power and Ground Ring Patterns	242
Creating Power and Ground Mesh Patterns	243
Creating Power and Ground Macro Connections	244
Creating Power and Ground Standard Cell Rails	245
Creating Channel and Alignment Power Straps	246
Creating Complex Composite Patterns	246
Creating Center-Aligned Vias	247
Creating Bridging Straps	249
Creating Via Bridging Straps	251
Creating Tapering Straps	255
Creating a Checkerboard Via Pattern	258
Defining Power Plan Regions	260

Contents

Setting the Power Plan Strategy	263
Strategy Settings for the -pattern Option	265
Strategy Settings for the -blockage Option	265
Strategy Settings for the -extension Option	266
Creating Via Rules Between Different Strategies	266
Instantiating the Power Plan	267
Handling Design Data During Design Planning Flow	268
Improving compile_pg Runtime in Large Designs	270
Application Options for PG Routing	270
Reducing the Memory Size of the Design Database	277
Pattern-Based Power Network Routing Example	277
PG Script Generator	280
Manually Creating Power Straps and Vias	282
Using Secondary PG Constraints	283
Hierarchical Secondary PG Aware Placement Constraints	283
Support for Abstract View	287
Honoring Metal and Via Spacing Rules to Blockages	287
Using PG Pattern Shapes	288
Using Via Matrixes	290
Debugging Vias and PG Straps Removed Due to DRC Violations	293
Merging Shapes in the PG Mesh	295
Inserting Stapling Vias Into an Existing Power Mesh	296
Connecting Via Ladders to Power and Ground	299
Inserting and Connecting Power Switches	300
Setting Resistance Values for Power Switch Cells	306
Trimming the Power and Ground Mesh	306
Checking Power Network Integrity	308
Checking for Missing Vias	308
Checking Power Network Connectivity	310
Validating DRC in the Power Network	312
PG Design Checker Rules	314
Analyzing the Power Plan	314
Performing Distributed Power Network Routing	319
Preparing for Distributed Power Network Routing	319
Performing Distributed Power Network Routing	320

11. Performing Global Planning	321
Topology Interconnect Planning	321
Creating Supernet Bundles	323
Creating Net Estimation Rules for Topology Plans	323
Creating Topology Plans	324
Creating Topology Plans From the Command Line	324
Creating Topology Plans in the GUI	326
Setting the Current Topology Plan	326
Reviewing Topology Plans	327
Refining Topology Plans	329
Optimizing Topology Plans	330
Reporting Topology Plan Repeaters	336
Performing What-If Analysis on the Topology Plan	338
Setting the Current Topology Plan	339
Propagating Topology Plans	339
Writing Topology Plans in Tcl Format	340
Creating Topology Groups	341
Creating Topology Groups From the Command Line	341
Creating Topology Groups in the GUI	342
Setting the Current Topology Group	343
Removing Topology Groups	343
Branching and Ripping Support for Bundles	344
Bidirectional Support for Bundles	345
Creating Routing Corridors	346
Creating Global Routes Within Routing Corridors	348
Adding Repeater Cells	349
Pushing Down Repeater Cells	350
12. Performing Clock Trunk Planning	351
Setting Clock Trunk Planning Constraints	354
Performing Clock Trunk Planning for Multiply Instantiated Blocks	355
Performing Block-Level Clock Trunk Planning	355
Checking the Feasibility of Performing Clock Trunk Planning	361
Performing Clock Trunk Planning for Hierarchical Designs With Abstracts	363
Setting Clock Trunk Planning Pin Constraints	364
Removing Clock Tree Planning Pin Constraints	366

Reporting Clock Tree Planning Pin Constraints	367
Performing Top-Level Clock Trunk Planning	367
Performing Top-Level Clock Trunk Planning Push-Down Flow	369
Performing QoR Analysis	370
Managing Timing Constraints for Clock Feedthrough Paths	371
Performing Incremental Clock Trunk Planning	372
Performing Clock Trunk Analysis in the GUI	373
<hr/>	
13. Performing Pin Assignment	375
Creating Pin Blockages	377
Precedence Rules and Conflict Resolution for Pin Constraints	378
Creating Block Pin Constraints	379
Creating Individual Pin Constraints	381
Creating Bundles and Bundle Constraints	383
Bundle Constraint Example	386
Setting Pin Constraints	387
Topological Map	388
Topological Constraint Examples	389
Specifying Routing Topology Using Side Constraints	389
Forcing a Connection Through a Block	390
Using Net Grouping in Topological Constraints	391
Specifying Side, Offset, and Layer Topological Constraints	391
Specifying a Connection Without Feedthroughs	392
Allowing Mixed Feedthroughs on Abutted and Narrow Channel Blocks ..	393
Specifying a Partial Path	394
Specifying Topological Constraints for Multiply Instantiated Blocks ..	395
Feedthrough Control	396
Physical Pin Constraints	398
Physical Pin Constraint Example	399
Pin Spacing Control	399
Block Pin Constraints	400
Setting Pin Placement Priority on Nets and Bundles	401
Performing Global Routing for Pin Placement	402
Creating a Channel Congestion Map	403
Checking the Design Before Pin Placement	404
Placing Pins	406

Contents

Application Options for Pin Placement	408
Removing Pins and Terminals	413
Performing Incremental Bundle Pin Placement	413
Controlling Feedthrough Sharing in Multiply Instantiated Blocks	414
Checking Pin Placement	414
Checking Pin Alignment	418
Saving Output From <code>check_pin_placement</code>	420
Enabling Various Pin Placement Checks	420
Checking Pin Placement on Multiply Instantiated Blocks	423
Checking Feedthroughs	424
Reporting and Writing Pin Constraints	425
Reporting Pin Constraints	426
Writing Pin Constraints	426
Writing Out Feedthrough Information as Tcl Commands	428
Creating Topological Pin Constraints in the GUI	429
Reporting Estimated Wire Length	432
<hr/>	
14. Performing Timing Budgeting	433
Creating Block Timing Abstracts	434
Performing Virtual In-Place Optimization	437
Performing Incremental Virtual In-Place Optimization	443
Applying Manual Budget Constraints	444
Constraining Layers, Delays, and Partial Routes for Timing Estimation	445
Updating Budget Information	447
Writing Out Budget Information	453
Budget Shells	455

About This User Guide

The Synopsys IC Compiler II tool provides a complete netlist-to-GDSII 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 II implementation and integration flow. For more information about the IC Compiler II tool, see the following companion volumes:

- *Library Manager User Guide*
- *IC Compiler II Implementation User Guide*
- *IC Compiler II Design Planning User Guide*
- *IC Compiler II Data Model User Guide*
- *IC Compiler II Timing Analysis User Guide*

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

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

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

This preface includes the following sections:

- [New in This Release](#)
- [Related Products, Publications, and Trademarks](#)
- [Conventions](#)
- [Customer Support](#)

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the IC Compiler II Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the IC Compiler II tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

<https://solvnetplus.synopsys.com>

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

- Design Compiler®
- IC Validator
- PrimeTime® Suite
- StarRC™

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>
Purple	<ul style="list-style-type: none"> • Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as <code>include_enclosing = true false</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>
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.

Convention	Description
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy .
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

The SolvNetPlus site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNetPlus site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNetPlus site, go to the following address:

<https://solvnetplus.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 sign up for an account.

If you need help using the SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to <https://solvnetplus.synopsys.com>.

1

Introduction to Design Planning

The IC Compiler II tool is designed for efficient design planning, placement, routing, and analysis of very large designs. You can use the design planning and chip-level analysis capabilities to perform initial floorplanning and feasibility analysis.

For more information, see the following topics:

- [Design Planning Overview](#)
 - [Hierarchical Design Planning Flow](#)
 - [Design Planning at Multiple Levels of Physical Hierarchy](#)
 - [Design Validation During Design Planning](#)
-

Design Planning Overview

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 large designs, design planning helps you to “divide and conquer” the implementation process by partitioning the design into smaller, more manageable pieces for more efficient processing.

Floorplanning is the process of partitioning logical blocks into physical blocks, sizing and placing the blocks, performing a floorplan-level placement of macros and standard cells, and creating a power plan. The goal of floorplanning is to increase the efficiency of downstream physical design steps to enable a robust, optimized design. To generate accurate budget estimates for the physical blocks, you generate timing estimates to guide timing budget allocation between the blocks and top-level cells in the design.

Floorplanning can also be an iterative process that reshapes blocks, creates a new cell placement, reallocates timing budgets, and rechecks top-level timing until an optimal floorplan is created.

An effective floorplan helps to achieve timing closure by placing blocks to minimize critical path length and reduce routing congestion. The challenge is to create a floorplan with good area efficiency and a small footprint, while leaving sufficient area for routing.

Similarly, design planning supports pattern-based power planning, including low-power design techniques that can be used in multivoltage designs. Using pattern-based power

planning, you can create different voltage areas within the design and define a power plan strategy for each voltage area. The tool creates the power and ground mesh based on your specification, and it connects the macros and standard cells to the mesh. You can quickly iterate on different power plan strategies to investigate different implementations and select the optimal power plan.

The IC Compiler II tool supports complete hierarchical design planning for both channeled and abutted layout styles. For more information, see [Supported Types of Floorplans](#).

Design planning in the IC Compiler II tool provides the following features for developing hierarchical and flat designs, such as

- Multi-Level Physical Hierarchy Floorplanning
- Advanced abstraction management for processing large designs
- Guided floorplanning with a graphical Task Manager
- Data flow analysis for macro management and placement assistance
- Fast placement of macros and standard cells
- Pattern-based power planning
- Power network description using UPF
- Virtual in-place timing optimization
- Pin placement and bus bundling
- Timing budgeting

Hierarchical Design Planning Flow

The hierarchical design planning flow provides an efficient approach for managing large designs. By dividing the design into multiple blocks, different design teams can work on different blocks in parallel, from RTL through physical implementation. Working with smaller blocks and using multiply instantiated 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 in a flat form.
- You anticipate problems that might delay the delivery of some blocks and might cause the schedule to slip. A robust hierarchical methodology accommodates late

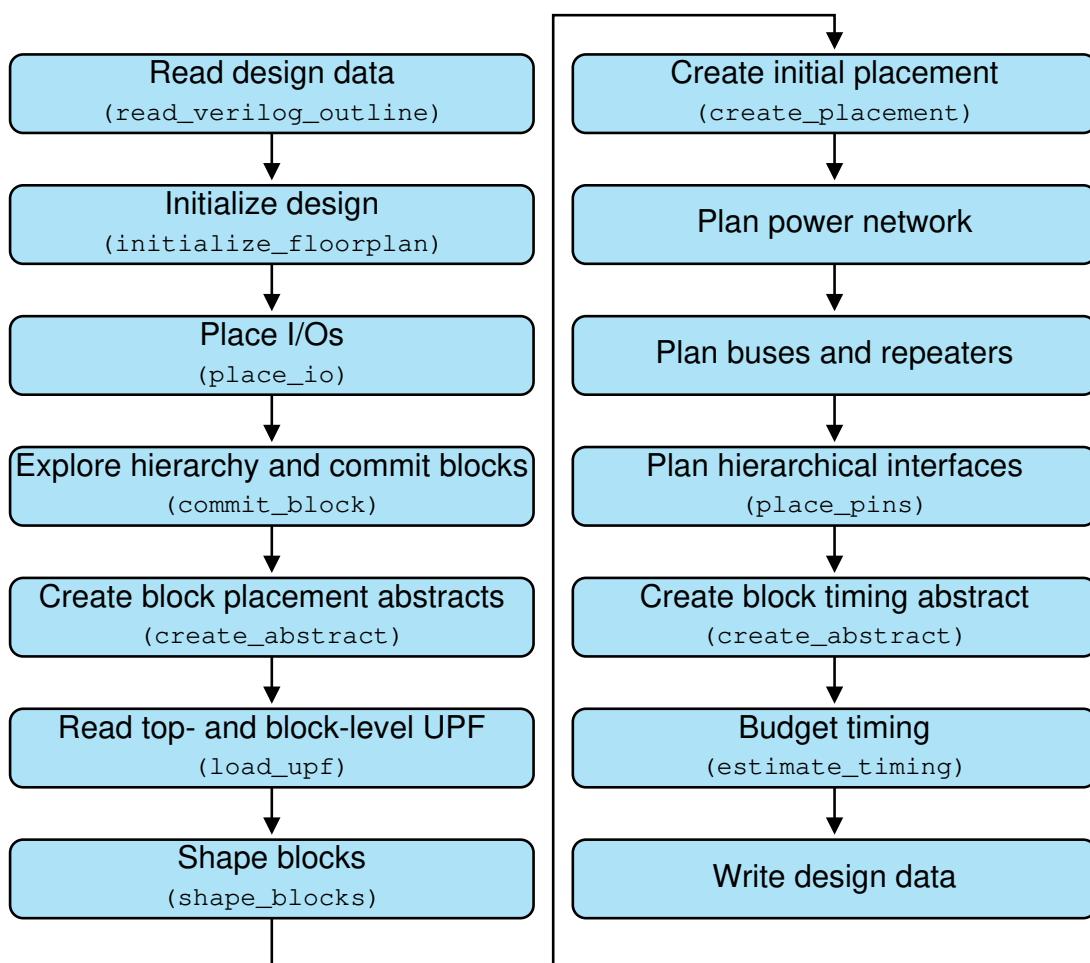
design changes to individual blocks while maintaining minimal disruption to the design schedule.

- The design contains hard intellectual property (IP) macros such as RAMs, or the design was previously implemented and can be converted and reused.

After the initial design netlist is generated in Design Compiler topographical mode, you can use the hierarchical methodology for design planning in the IC Compiler II tool. 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.

The flow to implement a hierarchical design plan is shown in [Figure 1](#).

Figure 1 Hierarchical Design Planning Flow



See Also

- [Design Partitioning](#)
- [Deciding on the Physical Partitions](#)

Design Partitioning

After reading in the netlist and initializing the floorplan, you can determine the physical partitioning for your hierarchical design project. When deciding on the physical partitions, consider the following factors:

- Size

Partition your design with blocks of similar size. Small blocks should be grouped and large blocks should be divided when appropriate.

- Function

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.

- Common hierarchy with 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 the IC Compiler II tool.

Deciding on the Physical Partitions

The IC Compiler II tool 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, to examine the logic design hierarchy, and to display information about the hierarchical cells and logic blocks in your design. You can select the hierarchical cells, leaf cells, or other objects in layout or schematic views. The viewer provides a tree browser to help in understanding the design hierarchy, and information about the blocks in the design such as the utilization, number of standard cells, and so on.

For more information, see [Exploring the Design Hierarchy](#).

- Committing Blocks

After you decide on an initial partitioning, you can commit the blocks. Blocks are committed early in the floorplanning flow, and abstract views are used to generate an initial floorplan and timing budgets for the design.

For more information about committing blocks, see [Committing Design Blocks](#).

Design Planning at Multiple Levels of Physical Hierarchy

Large, complex SoC designs require hierarchical layout methodologies capable of managing multiple levels of physical hierarchy at the same time. Many traditional design tools -- including physical planning, place and route, and other tools -- are limited to two levels of physical hierarchy: top and block. The IC Compiler II tool provides comprehensive support for designs with multiple levels of physical hierarchy, resulting in shorter time to results, better QoR, and higher productivity for physical design teams. Use the `set_editability` command to enable or disable specific blocks and design levels of hierarchy for planning.

The IC Compiler II tool provides support in several areas to accommodate designs with multiple levels of physical hierarchy:

Data Model

The data model in the IC Compiler II tool has built-in support for multiple levels of physical hierarchy. Native physical hierarchy support provides significant advantages for multi-level physical hierarchy planning and implementation. When performing block shaping, placement, routing, timing, and other steps, the tool can quickly access the specific data relative to physical hierarchy needed to perform the function.

Block Shaping

In a complex design with multiple levels of physical hierarchy, the block shaper needs to know the target area for each sub-chip, the aspect ratio constraints required by hard macro children, and any interconnect that exists at the sibling-to-sibling, parent-to-child, and child-to-parent interfaces. For multi-voltage designs, the block shaper needs the target locations for voltage areas. These requirements add additional constraints for the shaper to manage. For multi-level physical hierarchy planning, block shaping constraints on lower level sub-chips must be propagated to the top level; these constraints take the form of block shaping constraints on parent sub-chips. To improve performance, the shaper does not need the full netlist content that exists within each sub-chip or block.

The IC Compiler II data model provides block shaping with the specific data required to accomplish these goals. For multi-voltage designs, the tool reads UPF and saves the power intent at the sub-chip level. The tool retrieves data from the data model to calculate targets based on natural design utilization or retrieves user-defined attributes that specify design targets.

Cell and Macro Placement

After block shaping, the cell and macro placement function sees a global view of the interconnect paths and data flow at the physical hierarchy boundaries and connectivity to macro cells. With this information, the tool places macros for each sub-chip at each level of hierarchy. Because the tool understands the relative location requirements of interconnect paths at the boundaries at all levels, sufficient resources at the adjacent sub-chip edges are reserved to accommodate interconnect paths. The placer anticipates the needs of hierarchical pin placement and places macros where interconnect paths do not require significant buffering to drive signals across macros.

The placer models the external environment at the boundaries of both child and parent sub-chips by considering sub-chip shapes, locations, and the global macro placements. Using this information, the placer creates cell placement jobs for each sub-chip at each level of hierarchy. By delegating sub-chip placement across multiple processes, the tool minimizes turnaround time while maximizing the use of compute resources.

Power Planning

For power planning, the IC Compiler II tool provides an innovative pattern-based methodology. Patterns describing construction rules -- widths, layers, and pitches required to form rings and meshes -- are applied to different areas of the floorplan such as voltage areas, groups of macros, and so on. Strategies associate single patterns or multiple patterns with areas. Given these strategy definitions, the IC Compiler II tool characterizes the power plan and automatically generates definitions of strategies for sub-chips at all levels. A complete power plan is generated in a distributed manner. Because the characterized strategies are written in terms of objects at each sub-chip level, power plans can be easily re-created to accommodate floorplan changes at any level.

Pin Placement

With block shapes formed, macros placed, and power routed, pin placement retrieves interface data from all levels and invokes the global router to determine the optimal location to place hierarchical pins. The global router recognizes physical boundaries at all levels to ensure efficient use of resources at hierarchical pin interfaces. Pins are aligned across multiple levels when possible. Like all IC Compiler II operations, the global router comprehends multiply instantiated blocks (MIBs) and creates routes compliant with each MIB instantiation. To place pins for MIBs, the pin placement algorithm determines the best pin placement that works for all instances, ensuring that the pin placement on each instance is identical. Additionally, pin placement creates feedthroughs for all sub-chips, including MIBs, throughout the hierarchy. The global router creates feedthroughs across MIBs, determines feedthrough reuse, and connects unused feedthroughs to power or ground as required.

Timing Budgeting

The IC Compiler II tool estimates the timing at hierarchical interfaces and creates timing budgets for sub-chips. The timing budgeter in IC Compiler II creates timing constraints for

all child interface pins within the full chip, the parent and child interfaces for mid-level sub-chips and the primary pins at lowest level sub-chips. The entire design can proceed with placement and optimization concurrently and in a distributed manner.

To examine critical timing paths in the layout or perform other design planning tasks, you can interactively view, analyze, and manually edit any level of the design in a full-chip context. You can choose to view top-level only or multiple levels of hierarchy.

When viewing multiple levels, interactive routing is performed as if the design is flat. At completion, routes are pushed into children and hierarchical pins are automatically added.

Design Validation During Design Planning

As you proceed through the design planning flow, you can use the `check_design` command to validate your design and ensure that it is ready for the next design stage. The command performs one or more checks on your design, based on the arguments you specify. To use the `check_design` command to check your design, specify the command and the `-checks` option followed by one or more check keywords. The following example performs the `dp_pre_floorplan` checks:

```
icc2_shell> check_design -checks {dp_pre_floorplan}
```

The list of keywords related to design planning for the `-checks` option is shown in the following list.

- `dp_pre_floorplan`
 - Checks that the technology file information is correct
 - Checks that the layer directions are set
 - Checks that the design contains both horizontal and vertical layers
- `dp_pre_create_placement_abstract`
 - Checks that the constraint mapping file specifies a UPF file for all blocks
 - Checks that the Verilog files are in place for the outline view for blocks
- `dp_pre_block_shaping`
 - Checks that the target utilization or area exists for all black boxes
 - Checks that there is at least one block or voltage area to shape
 - Checks that the core area and block boundaries are valid

- Checks that the block grid is set if the design contains multiply instantiated blocks (MIBs)
- Performs the multivoltage checks related to the `shape_blocks` command
- `dp_pre_macro_placement`
 - Checks that block shapes do not overlap
 - Checks that blocks and voltage areas are inside the parent boundaries
 - Performs basic placement constraint checking, such as overlapping or out-of-bounds relative locations constraints, macro cell width and height that are an even multiple of the FinFET grid, and macro cell orientation
 - Checks that each block has a logical hierarchy
 - Checks block, voltage area, exclusive movebound, and hard movebound utilization
 - Performs the multivoltage checks related to the `create_placement` command
- `dp_pre_power_insertion`
 - Checks that the preferred routing direction is set for the routing layers
 - Checks that the tracks exist for routing layers
- `dp_pre_pin_placement`
 - Performs pin constraint checks, including topological constraints, individual constraints, constraints propagated from individual pins, bundle pin constraints, block pin constraints, size-related constraints, pin guide and pin blockage constraints, routing guide constraints, and routing blockage constraints
 - Checks that the routing direction of each layer in a topological constraint is consistent with the block edge constraints
 - Checks for consistent edge directions in topological constraint pairs
 - Checks for off-edge pins that are part of a feedthrough constraint
 - Checks for possible overlaps with existing internal block objects such as route blockages, pin blockages, fixed pins, preroutes, and so on
 - Runs the checks performed by the `check_mib_for_pin_placement` command

- `dp_pre_push_down`
 - Runs the checks performed by the `check_mib_alignment` command
- `dp_pre_create_timing_abstract`
 - Checks that the constraint mapping file specifies SDC files for all blocks
- `dp_pre_timing_estimation` (timing abstract checks)
 - Checks that the top and block levels have defined modes and corners
 - Checks that the top-level modes and corners have corresponding block-level modes and corners
 - Checks that the top level contains at least one clock, at least one scenario using a nonestimated corner, and the corner has parasitic parameters
- `dp_pre_timing_estimation` (placement abstract checks)
 - Performs the `pre_create_timing_abstract` checks
 - Checks whether placement abstracts exist
- `dp_pre_budgeting`
 - Checks that the `estimated_corner` is available
 - Performs the same checks as `dp_pre_timing_estimation`
- `dp_floorplan_rules`
 - Checks the segment parity rule
 - Checks the macro spacing rule
 - Runs the checks performed by the `check_finfet_grid` command

2

Working in Design Planning Mode

The IC Compiler II tool supports a design planning mode that only enables functionality, commands, and operations related to design planning tasks and restricts tool usage in other areas.

The following topics describe how to use the IC Compiler II tool:

- [User Interfaces](#)
 - [Entering `icc2_shell` Commands](#)
 - [Using Variables](#)
 - [Using Application Options](#)
 - [Viewing Man Pages](#)
 - [Using Tcl Scripts](#)
 - [Using Setup Files](#)
 - [Using the Command Log File](#)
 - [Running Tasks in Parallel](#)
 - [Monitoring Distributed Tasks](#)
-

User Interfaces

The IC Compiler II tool operates in the X windows environment on UNIX or Linux. It provides a flexible working environment with both a shell command-line interface and a graphical user interface (GUI). The shell command-line interface, `icc2_shell`, is always available during a IC Compiler II session. You can start or exit a session in either `icc2_shell` or the GUI, and you can open or close the GUI at any time during a session.

The IC Compiler II tool uses the tool command language (Tcl), which is used in many applications in the EDA industry. Using Tcl, you can extend the `icc2_shell` command language by writing reusable procedures and scripts (see the *Using Tcl With Synopsys Tools* manual).

The following topics describe how to start and exit the tool using the command-line interface.

- [Starting the Command-Line Interface](#)
- [Exiting the IC Compiler II Tool](#)

For information about using the GUI, see the *IC Compiler II Graphical User Interface User Guide*.

Starting the Command-Line Interface

The `icc2_shell` command-line interface is a text-only environment in which you enter commands at the command-line prompt. It is typically used for scripts, batch mode, and push-button operations.

To start `icc2_shell` in design planning mode,

1. Ensure the path to the bin directory is included in your PATH variable.
2. Enter the `icc2_shell` command in a UNIX or Linux shell and specify the `-dp_mode` option.

```
% icc2_shell -dp_mode
```

You can include other options on the command line when you start `icc2_shell`. For example, you can use

- `-f script_file_name` to execute a script
- `-x command` to execute an `icc2_shell` command
- `-output_log_file file_name` to create a log file of your session
- `-h` to display a list of the available options (without starting `icc2_shell`)

At startup, `icc2_shell` performs the following tasks:

1. Creates a command log file.
2. Reads and executes the setup files.
3. Executes any script files or commands specified by the `-f` and `-x` options, respectively, on the command line.
4. Displays the program header and `icc2_shell>` prompt in the shell in which you started `icc2_shell`.

See Also

- [Using the Command Log File](#)
- [Using Setup Files](#)
- [Using Tcl Scripts](#)

Exiting the IC Compiler II Tool

You can end the session and exit the IC Compiler II tool at any time. To exit the tool, use the `exit` or `quit` command.

Note:

When you exit the tool from the command line, the tool exits without saving the open blocks.

Entering `icc2_shell` Commands

You interact with the IC Compiler II tool by using `icc2_shell` commands, which are based on the tool command language (Tcl) and include certain command extensions needed to implement specific IC Compiler II functionality. The IC Compiler II command language provides capabilities similar to UNIX command shells, including variables, conditional execution of commands, and control flow commands. You can

- Enter individual commands interactively at the `icc2_shell>` prompt in `icc2_shell`
- Enter individual commands interactively on the console command line in the GUI
- Run one or more Tcl scripts, which are text files that contain `icc2_shell` commands (see [Using Tcl Scripts](#))

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

If you need to reuse a command from the output for a command-line interface, you can copy and paste the portion by selecting it, moving the pointer to the `icc2_shell` command line, and clicking with the middle mouse button.

When you run a command, the IC Compiler II tool echoes the command output (including processing messages and any warnings or error messages) in `icc2_shell` and, if the GUI is open, in the console log view. By default, the tool does not use page mode, so the output might scroll. To enable page mode, set the `sh_enable_page_mode` variable to `true`.

Interrupting or Terminating Command Processing

If you enter the wrong options for a command or enter the wrong command, you can interrupt command processing and remain in `icc2_shell`. To interrupt or terminate a command, press **Ctrl+C**.

Some commands and processes cannot be interrupted. To stop these commands or processes, you must terminate `icc2_shell` at the system level. When you terminate a process or the shell, no data is saved.

When you use **Ctrl+C**, keep the following points in mind:

- If a script file is being processed and you interrupt one of its commands, the script processing is interrupted and no further script commands are processed.
- If you press **Ctrl+C** three times before a command responds to your interrupt, `icc2_shell` is interrupted and exits with this message:

Information: Process terminated by interrupt.

Getting Information About Commands

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

- Command help, which is information about an `icc2_shell` command
- Man pages

See Also

- [Viewing Man Pages](#)

Displaying Command Help

Command help consists of either a brief description of `icc2_shell` commands or a list of the options and arguments supported by the `icc2_shell` command.

- To display a brief description of an `icc2_shell` command, enter the `help` command followed by the command name. For example, to display a brief description of the `report_timing` command, use the following command:

```
icc2_shell> help report_timing
```

- To display the options supported by an `icc2_shell` command, enter the command name with the `-help` option on the `icc2_shell` command line. For example, to see the options supported by the `report_timing` command, use the following command:

```
icc2_shell> report_timing -help
```

Using Variables

In general, the IC Compiler II tool modifies default behavior by using application options rather than application variables. The tool does support user-defined Tcl variables, as well as a minimal number of application variables, such as the `search_path` variable.

Use the `printvar` command to list the variables.

Examples:

```
printvar *
printvar search_path
```

Using Application Options

The IC Compiler II tool uses application options to control the tool behavior. Application options use the following naming convention:

`category[.subcategory].option_name`

where `category` is the name of the engine affected by the application option. Some application option categories have subcategories to further refine the area affected by the application option.

Application options have either a global scope or a block scope.

- Block-scoped application options apply only to the block on which they are set. They are saved in the design library and are persistent across tool sessions.
- Global-scoped application options apply to all blocks, but only within the current session. They are not saved in the design library; you must specify them in each `icc2_shell` session. You might want to consider adding the global settings to your `.synopsys_icc2.setup` file.

To get a list of available application options, use the `get_app_options` command. By default, this command lists all application options. To restrict the reported application options, provide a pattern string as an argument to the command.

For example, to list all available application options, use the following command:

```
icc2_shell> get_app_options
```

To list all available timer application options, use the following command:

```
icc2_shell> get_app_options timer.*
```

To generate a report of application options, use the `report_app_options` command.

For detailed information about application options, see “Application Options” in the *IC Compiler II Data Model User Guide*.

See Also

- [Using Setup Files](#)

Viewing Man Pages

To display the man page for an `icc2_shell` command or application variable, enter the `man` command followed by the command or variable name. For example, to see the man page for the `report_timing` command, use the following command:

```
icc2_shell> man report_timing
```

To display the man page for an `icc2_shell` application option, enter the `man` command followed by the option name. You can also view the man page that summarizes all of the application options for the following types of summaries:

- Category summaries

Enter the `man` command followed by `category_options`. For example, to see the man page that summarizes all floorplan application options, use the following command:

```
icc2_shell> man plan_options
```

- Subcategory summaries

Enter the `man` command followed by `category.subcategory_options`. For example, to see the man page that summarizes all floorplan placement application options, use the following command:

```
icc2_shell> man plan.place_options
```

- Command summaries

Enter the `man` command followed by `command_options`. For example, to see the man page that summarizes all application options that affect the `report_timing` command, use the following command:

```
icc2_shell> man report_timing
```

If you enter the `man` command on the `icc2_shell` command line, the man page is displayed in the IC Compiler II shell and in the console log view if the GUI is open. If you enter this command on the console command line in the GUI, the man page is displayed in the GUI man page viewer.

Using Tcl Scripts

You can use Tcl scripts to accomplish routine, repetitive, or complex tasks. You create a command script file by placing a sequence of `icc2_shell` commands in a text file. Any `icc2_shell` command can be executed within a script file.

In Tcl, a pound sign (#) at the beginning of a line denotes a comment. For example,

```
# This is a comment
```

For more information about writing scripts and script files, see the *Using Tcl With Synopsys Tools* manual.

Use one of the following methods to run a Tcl script:

- Use the `-f` option with the `icc2_shell` command when you start the IC Compiler II tool.
- Use the `source` command from the `icc2_shell` command line.
- Choose File > Execute Script in the GUI.

If an error occurs when running a command, the IC Compiler II tool raises the `TCL_ERROR` condition, which immediately stops the script execution. To tolerate errors and allow the script to continue executing, either

- Check for `TCL_ERROR` error conditions with the Tcl `catch` command on the commands that might generate errors.
- Set the `sh_continue_on_error` variable to `true` in the script file.

See Also

- [Starting the Command-Line Interface](#)

Generating a Tcl Script From Spreadsheet Data

You can quickly generate a PG script from data in a spreadsheet by using the `generate_pg_script -input` command.

However, the data in the spreadsheet should be in a specific format and the spreadsheet should be a CSV file. For the spreadsheet format, see the examples at the end of this topic. When you use these examples to build your CSV file, ensure that you do not alter any of the information that is highlighted in yellow. Altering this text in any way, such as using some other text or not adhering to case-sensitivity might result in the non-generation of the PG script.

The commands that are written out to the script file based on the spreadsheet data are as follows. No other PG commands are supported.

- `create_pg_composite_pattern`
- `create_pg_special_pattern`
- `create_pg_wire_pattern`
- `set_pg_strategy`
- `set_pg_strategy_via_rule`

In the following example, the `generate_pg_script` command takes the data from the `special_pattern_spreadsheet.csv` file and outputs the `test.tcl` file with the PG commands.

```
icc2_shell> generate_pg_script -input special_pattern_spreadsheet.csv \
    -output test.tcl
```

Table 1 Example: Input Spreadsheet, Output Tcl File

CSV data - Example										
special_pattern_command(insert_channel_straps)										
pattern_name	layer	direction	width	spacing	offset	reference	channel_threshold	track_alignment	channel_between_objects	check_one_layer
channelPattern1	M7	vertical	1						macro_placement_blockage_voltage_area	
channelPattern2	M1Z	vertical	2	2			15		macro_placement_blockage	TRUE
endcommand										

Tcl script generated for the CSV data										
<pre># Generate PG create_pg_special_pattern script from spreadsheet create_pg_special_pattern channelPattern1 -insert_channel_straps { {layer: M7} {direction: vertical} {width: 1} {channel_between_objects: macro_placement_blockage_voltage_area} } create_pg_special_pattern channelPattern2 -insert_channel_straps { {layer: M1Z} {direction: vertical} {width: 2} {spacing: 2} {channel_threshold: 15} {channel_between_objects: macro_placement_blockage} {check_one_layer: TRUE} }</pre>										

Chapter 2: Working in Design Planning Mode Using Tcl Scripts

Example 1 Spreadsheet Template for pg_strategy_via_rule_command

pg_strategy_via_rule_command													
set_1	set1_filter_layer	set1_filter_net	set1_filter_width	set1_filter_height	set1_filter_width_height_ratio	set_2	set2_filter_layer	set2_filter_net	set2_filter_width	set2_filter_height	set2_filter_width_height_ratio	via_master	
strategy_core_1	M1	vdd				strategy_core_2	M2		set2_filter_width				via1a
strategy_core_1	M1	vss				strategy_core_2	M2		set2_filter_width				via1B
strategy_core_2	M2					strategy_core_3	M3		set2_filter_width				via2a_rule
strategy_core_3	M3					strategy_core_4	M4		set2_filter_width				via3a
strategy_sram_1	adjacent					strategy_sram_2	adjacent		set2_filter_width				via1B
strategy_sram_2	adjacent					strategy_sram_3	adjacent		set2_filter_width				via2B
endcommand													

Example 2 Spreadsheet Template for composite_pattern_command

composite_pattern_command														
pattern_name	layer	direction	nets	width	pitch_x	pitch_y	track_alignment	low_end	high_end	spacing	offset_x	offset_y	trim	via_rule
P_CORE_PG13	M13	vertical	VDD_A72	0.36			none	0	0	0.36	5	0	FALSE	
PS_pillar_PG	M2	horizontal	VDD_A72	0.02			track	-1.707	1.707	0.02	1.9	0.105	FALSE	{intersection:adjacent}{via_master:default}
PS_pillar_PG	M4	horizontal	VDD_A72	0.022			track	-1.707	1.707	0.022	1.9	0.105	FALSE	
PS_pillar_PG	M6	horizontal	VDD_A72	0.038			track	0	0	0.038	0	0.2	FALSE	
PS_pillar_PG	M8	horizontal	VDD_A72	0.038			track	0	0	0.038	0	0.2	FALSE	
PS_pillar_PG	M10	horizontal	VDD_A72	0.114			track	0	0	0.114	0	0.173	FALSE	
P_CORE_PG0	M0	horizontal	VVDD_CPU	0.056	0.105		track	0	0	0.056	0	0	FALSE	
P_CORE_PG0	M0	horizontal	VSS	0.056	0.105		track	0	0	0.056	0	0	FALSE	
endcommand														

Chapter 2: Working in Design Planning Mode
Using Tcl Scripts

Example 3 Spreadsheet Template for special_pattern_command (insert_channel_straps)

special_pattern_command(insert_channel_straps)											
pattern_name	layer	direction	width	spacing	offset	reference	channel_threshold	track_alignment	channel_between_objects	check_one_layer	
channelPattern1	M7	vertical		1					macro_placement_blockage_voltage_area		
channelPattern2	M1Z	vertical	2	2			15		macro_placement_blockage		TRUE
endcommand											

Example 4 Spreadsheet Template for special_pattern_command (insert_power_switch_alignment_straps)

special_pattern_command(insert_power_switch_alignment_straps)											
pattern_name	lib_cells	layer	width	direction	offset	pitch	track_alignment	number	pin_layers	via_master	
pwrSwitchAlignmentPattern	HEAD16DM	M4									
SW_pat	HEADBUFTI E42Q_D3	D2	0.51	vertical	1.5	0.61			2 M2		
endcommand											

Example 5 Spreadsheet Template for special_pattern_command (insert_physical_cell_alignment_straps)

special_pattern_command(insert_physical_cell_alignment_straps)											
pattern_name	lib_cells	layer	width	direction	offset	track_alignment	number	pin_names	pin_layers	via_master	
phyCellAlignmentPattern	TAP_REF1 TAP_REF2	M4									
endcommand											

*Example 6 Spreadsheet Template for special_pattern_command
(insert_terminal_alignment_straps)*

special_pattern_command(insert_terminal_alignment_straps)			
pattern_name	trim	layer	terminal_alignment_via_rule
terminalPattern			{{{layers: M8}{layers: M7}{via_master: VIA78}} {layers: M7}{layers: M6}{via_master: VIA67}}}}
endcommand			

*Example 7 Spreadsheet Template for special_pattern_command
(honor_max_stdcell_strap_distance)*

special_pattern_command(honor_max_stdcell_strap_distance)					
pattern_name	layer	width	max_distance	offset	check_layers
extraStrapPattern	M8	2	20	20 10	
endcommand					

Example 8 Spreadsheet Template for strategy_command

		strategy_command							
strategy_name	region_type	region_description	pattern_name	nets	offset	offset_start	blockage	extension	
strategy_1	pg_regions	PG_TOP	P_CORE_PG13	VDD_A 72	0 0				
strategy_2	pg_regions	pgr_PSW_cells	PS_pillar_PG	VDD_A 72	0 0				
strategy_3	pg_regions	pgr_PSW_V_cells	PS_pillar_PG_2	VDD_A 72	0 0		{pg_regions : PG_TOP_block SRAM_E* }{placement_blockages:all}		
strategy_3	pg_regions	pgr_PSW_V_cells	PS_pillar_PG_1	VDD_A 72	0 0				
strategy_3	pg_regions	pgr_PSW_V_cells	PS_pillar_PG_3	VDD_A 72	0 0				
	pg_regions	PG_TOP	P_CORE_PGO	VVDD_CPU	0 0		{pg_regions: PG_TOP_block pgr_SRAM_E_macro_cells* pgr_PSW_cells* BOUNDARY_*} {placement_blockages:all}		
	pg_regions	PG_TOP	P_CORE_PG1	VVDD_CPU	0 0		{pg_regions : PG_TOP_block pgr_SRAM_E_macro_cells* }{placement_blockages:all}		
	pg_regions	PG_TOP	P_CORE_PG2	VVDD_CPU	0 0		{pg_regions : PG_TOP_block pgr_SRAM_E_macro_cells* pgr_PSW_V_cells* } {placement_blockages:all}		
endcommand									

Using Setup Files

When you start the IC Compiler II tool, it automatically executes the commands in the `.synopsys_icc2.setup` file.

The tool looks for this file both in your home directory and in the project directory (the current working directory in which you start the IC Compiler II tool). The file is read in the following order:

1. The `.synopsys_icc2.setup` file in your home directory
2. The `.synopsys_icc2.setup` file in the project directory

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

- To set application options that define your IC Compiler II working environment, create setup files in your home directory.
- To set project- or block-specific application options that affect the processing of a block, create a setup file in the design directory.

See Also

- [User Interfaces](#)
- [Using Application Options](#)
- [Using Tcl Scripts](#)

Using the Command Log File

The command log file records the `icc2_shell` commands processed by the IC Compiler II tool, including setup file commands and application option settings. By default, the IC Compiler II tool writes the command log to a file named `icc2_command.log` in the directory from which you invoked `icc2_shell`.

You can change the name of the command log file by setting the `sh_command_log_file` variable in your `.synopsys_icc2.setup` file. You should make any changes to this variable before you start the IC Compiler II tool. If your user-defined or project-specific setup file does not contain this variable, the tool automatically creates the `icc2_command.log` file.

Each IC Compiler II session overwrites the command log file. To save a command log file, move it or rename it. You can use the command log file to

- Produce a script for a particular implementation strategy
- Record the physical implementation process
- Document any problems you are having

Running Tasks in Parallel

To efficiently run the same task on several blocks in your design, you can use the `run_block_script` command to enable distributed processing and perform the tasks in parallel. The `run_block_script` command accepts a Tcl script and applies the commands in the script to the blocks you specify. For more information, see the “Enabling Multicore Processing” topic in the *IC Compiler II Implementation User Guide*.

Monitoring Distributed Tasks

The IC Compiler II tool supports a Distributed Processing Manager to help you monitor the progress of distributed tasks as they run. The Distributed Processing Manager communicates with distributed commands such as `create_abstract` to provide up-to-date status. When a task completes, the GUI updates to show that the task is finished.

To use the Distributed Processing Manager to monitor running jobs,

1. Start the Distributed Processing Manager with the `run_monitor_gui` command.

```
icc2_shell> run_monitor_gui  
10948
```

The command returns the process id for the dpmanager task created by the command.

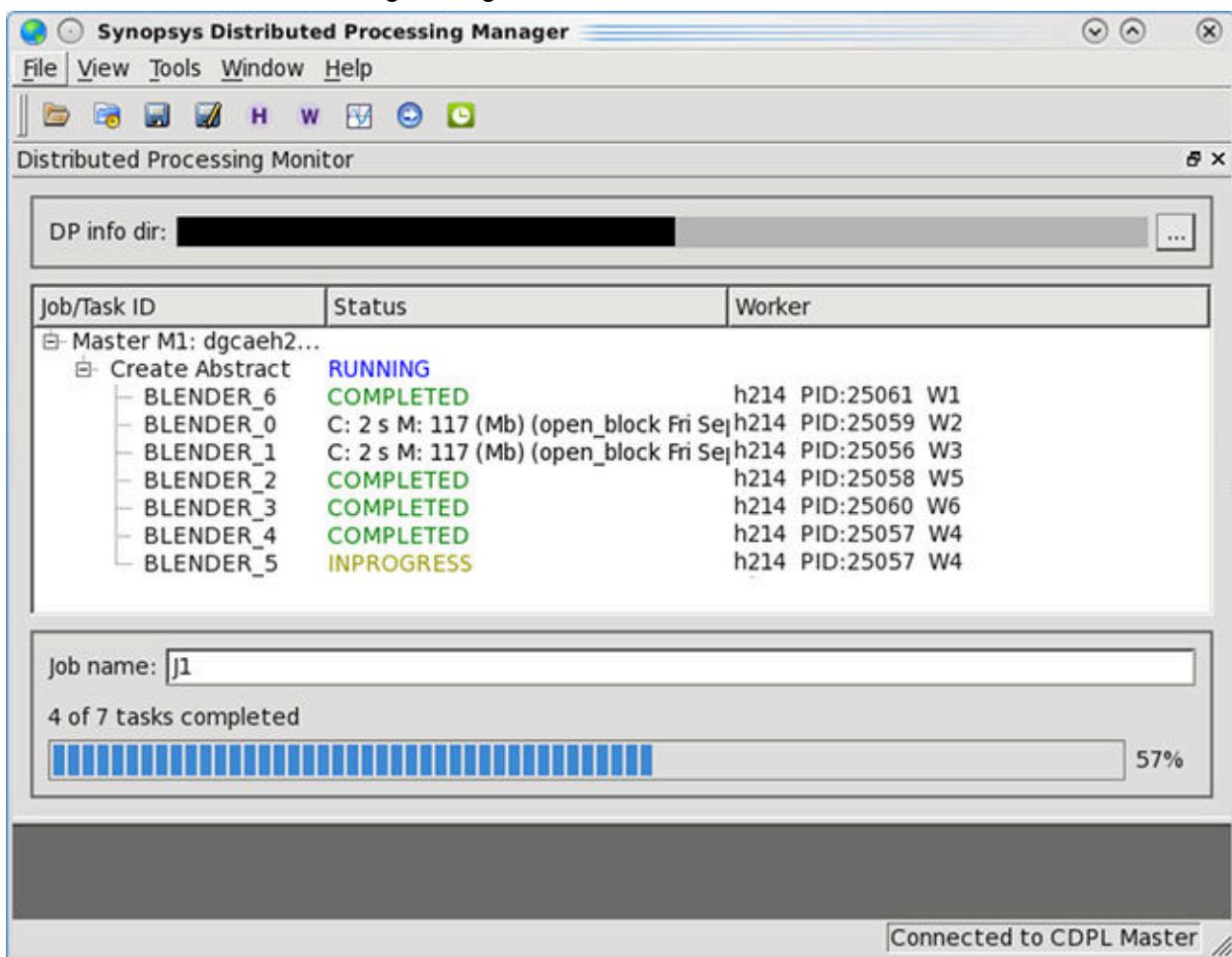
2. Start the distributed task.

```
icc2_shell> create_abstract -placement \  
-host_options block_script -all_blocks  
Submitting job for block BLENDER_0 ...  
Submitting job for block BLENDER_1 ...
```

```
Submitting job for block BLENDER_2 ...
...
```

3. Monitor the running task in the Distributed Processing Manager window as shown in [Figure 2](#).

Figure 2 Distributed Processing Manager



When all tasks are complete, the Distributed Processing Manager updates the Status column to COMPLETED for all job/Task IDs, and updates the status in the Job History window from not done to done.

You can open a maximum of two Distributed Processing Manager windows. To open a second Distributed Processing Manager window, use the `run_monitor_gui` command a second time. To kill any open Distributed Processing Manager windows and restart a single window, use the `run_monitor_gui -kill` command.

The Distributed Processing Manager monitors the progress of the following commands, if they are running in distributed mode:

- `create_abstract -all_blocks | -blocks`
- `create_placement -floorplan`
- `estimate_timing`
- `merge_abstract`
- `route_global -floorplan -virtual_flat`
- `shape_blocks`

3

Splitting Constraints

To generate timing budgets, chip-level Synopsys Design Constraints (SDC) and UPF files must be partitioned into top-level and block-level files. The IC Compiler II tool uses the top-level constraints to constrain the top-level logic when abstract views of blocks are used.

For more information, see the following topics on splitting constraints:

- [Split Constraints Flow](#)
 - [Split Constraints Output Files](#)
 - [Split Constraints Example 1](#)
 - [Split Constraints Example 2](#)
-

Split Constraints Flow

If you have only chip-level constraints for your design, you can use the `split_constraints` command to partition the full-chip SDC and UPF constraints and create separate top-level and block-level constraint files. After running `split_constraints`, the top-level constraint file contain the top-level UPF and all top-level and top-to-block boundary timing, without block-level internal constraints. The block-level constraint files are used during block-level optimization.

The block-level constraint files generated by the `split_constraints` command contain internal constraints for the block and should be applied only one time. You must apply either manually generated constraints or constraints generated by the `split_constraints` command before applying timing budgets.

Note that if you already have block-level SDC and UPF constraints for your design, you can ignore this section and continue without running the `split_constraints` command.

To split the chip-level SDC and UPF constraints,

1. Read in the full Verilog netlist.

```
icc2_shell> read_verilog -top ORCA.v
```

Note that the split constraints flow requires that you read the full Verilog netlist with the `read_verilog` command. You cannot use the `read_verilog_outline` command for this step.

- Load and commit the UPF constraint file for the entire design.

```
icc2_shell> load_upf ORCA.upf
icc2_shell> commit_upf
```

The `load_upf` command reads in the UPF constraints and the `commit_upf` command applies the UPF constraints to the design. The `commit_upf` command also tries to resolve any conflicts between the power intent specified in the UPF file and the actual power and ground nets in the design.

- Set up the timing environment for the design.

For a design with multiple modes and corners, you must use one or more of the following commands:

```
create_corner
create_mode
read_sdc
set_corner_status
set_parasitic_parameters
set_process_number
set_temperature
set_voltage
```

If your design uses UPF to describe the power network, you must use the `set_voltage` command for all supply net groups defined in the UPF file.

- Reset any previous budget constraints and specify the blocks for which to write out constraints.

```
icc2_shell> set_budget_options -reset
icc2_shell> set_budget_options -add_blocks {I_ORCA_TOP/I_BLENDER_1 \
    I_ORCA_TOP/I_BLENDER_2 I_ORCA_TOP/I_BLENDER_3 \
    I_ORCA_TOP/I_BLENDER_4 I_ORCA_TOP/I_BLENDER_5 \
    I_ORCA_TOP/I_BLENDER_6 I_ORCA_TOP/I_BLENDER_7}
```

If your design uses UPF to describe the power network, you must use the `set_voltage` command for all supply net groups defined in the UPF file.

- Split the timing constraints and UPF file into separate files for each block.

```
icc2_shell> split_constraints
```

If your design contains multiple levels of physical hierarchy, specify the intermediate levels of hierarchy with the `-design_subblocks` option. The nested modules in the

design hierarchy are not abstracted when they are used in the context of their parent block.

```
icc2_shell> split_constraints -design_subblocks {block_1 block_2}
```

Split Constraints Options

Use options with the `split_constraints` command to control how the constraints are split.

- Split the SDC file or the UPF file with the `-sdc_only` option or `-upf_only` option.

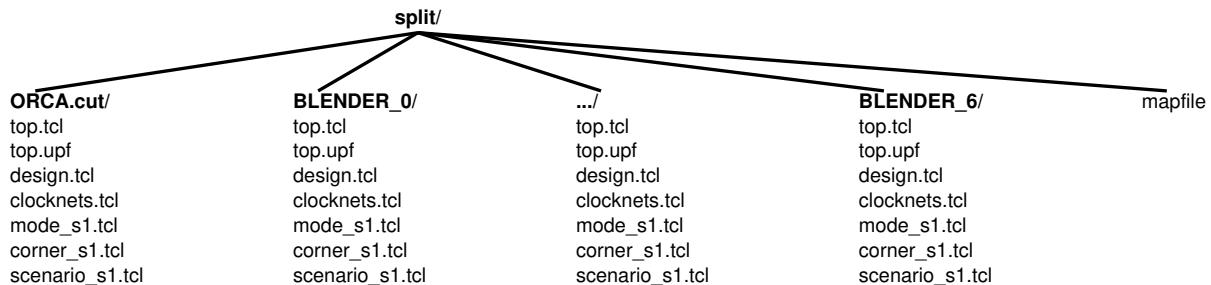
```
icc2_shell> split_constraints -sdc_only
icc2_shell> split_constraints -upf_only
```

- Retain internal constraints for subblocks when using hierarchical abstracts with the `-hier_abstract_subblocks block_names` option. Use this option to indicate that the specific blocks are represented as hierarchical abstracts.
 - Write out constraint files only for certain modes with the `-modes` option.
- ```
icc2_shell> split_constraints -modes {s1.mode}
```
- Write out constraint files only for certain corners with the `-corners` option.
- ```
icc2_shell> split_constraints -corners {s1.corner}
```
- Compress the output constraint files with the `-compress gzip` option.
- ```
icc2_shell> split_constraints -compress gzip
```
- Specify the directory to use for the output with the `-output` option.
- ```
icc2_shell> split_constraints -output split2
```

Split Constraints Output Files

[Figure 3](#) shows the directory structure created by the `split_constraints` command for the ORCA design.

Figure 3 Constraint Files After split_constraints



In this example, ORCA is the top-level design and BLENDER_0 through BLENDER_6 are the modules. The `split_constraints` command creates the following files for the top-level design and for each module that is specified with the `set_budget_options -add_blocks` command:

- `top.tcl`: Provides corner and mode definitions, and sources all other files in the correct order. Source only this file to apply the split constraints for the specified block.
- `top.upf`: Provides power constraints in UPF format..
- `design.tcl`: Provides design-wide constraints with mode and corner associations. Contains `set_corner_status` commands to configure analysis settings for mode and corner combinations.
- `clocknets.tcl`: Provides constraints for clock nets.
- `mode_modename.tcl`: Provides mode-specific constraints for clocks, exceptions, latencies, and so on. The tool creates one Tcl file for each defined mode.
- `corner_cornernname.tcl`: Provides corner-specific constraints for derating factors, parasitic definitions, and PVT values. The tool creates one Tcl file for each defined corner.
- `scenario_scenarioname.tcl`: Provides scenario-specific constraints for clock latency, clock uncertainty, and other factors. The tool creates one Tcl file for each defined scenario.

The `split_constraints` command also creates the constraint mapping file, `mapfile`, which lists the SDC and UPF constraint files for each block. The content of the `mapfile` for this example is as follows:

```

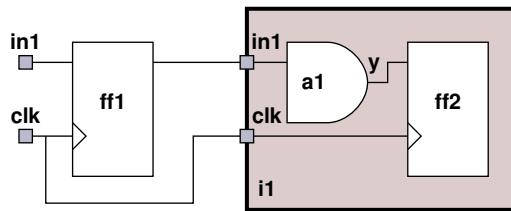
BLENDER_0 CLKNET BLENDER_0/clocknets.tcl
BLENDER_0 SDC BLENDER_0/top.tcl
BLENDER_0 UPF BLENDER_0/top.upf
...
BLENDER_6 CLKNET BLENDER_6/clocknets.tcl
BLENDER_6 SDC BLENDER_6/top.tcl
  
```

```
BLENDER_6 UPF BLENDER_0/top.upf
ORCA CLKNET ORCA/clocknets.tcl
ORCA SDC ORCA/top.tcl
ORCA UPF TOP/top.upf
```

Split Constraints Example 1

The design for Example 1 is shown in [Figure 4](#).

Figure 4 Example 1 Design



This design example uses the following chip-level SDC constraints file.

Example 9 Chip-Level SDC Constraints File

```
create_clock -period 1 [get_ports clk]
set_input_delay 0.5 [get_ports in1]
set_multicycle_path 2 -from ff1/clk -through i1/a1/y
```

The `split_constraints` command partitions the chip-level SDC constraints file into the following top-level and block-level mode_*modename*.tcl constraint files.

Example 10 Top-Level Constraint File after `split_constraints`

```
# Top-level constraints
create_clock -period 1 [get_ports clk]
set_input_delay 0.5 [get_ports in1]
set_multicycle_path 2 -from ff1/clk -through i1/a1/y
```

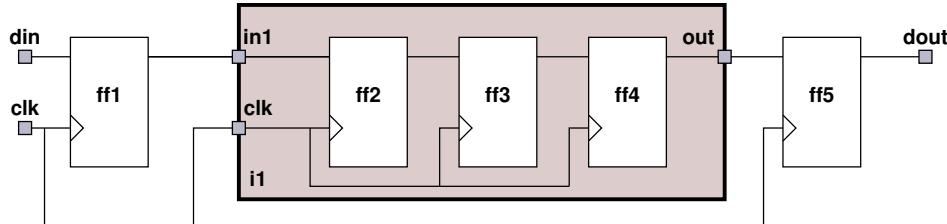
Example 11 i1 Constraint File after `split_constraints`

```
# Block-level constraints
create_clock -add -name "clk" -period 1 [get_ports clk]
create_clock -name virtual_clk -period 1
set_multicycle_path 2 -from virtual_clk -through [get_ports in] \
    -through a1/y
```

Split Constraints Example 2

The design for Example 2 is shown in [Figure 5](#).

Figure 5 Example 2 Design



This design example uses the following top-level SDC constraints file.

Example 12 Chip-Level SDC Constraints File

```

create_clock -period 1 [get_ports clk]
set_input_delay -clock [get_clocks clk] 0.5 [get_ports din]
set_output_delay -clock [get_clocks clk] 0.5 [get_ports dout]
set_false_path -from [get_pins i1/ff2/CK] -to [get_pins i1/ff3/D]

```

The `split_constraints` command partitions the chip-level SDC constraints file into the following top-level and block-level mode_*modename*.tcl constraint files.

Example 13 Top-Level Constraint File After split_constraints

```

# Top-level constraints
create_clock -name clk -period 1 [get_ports {clk}]
set_input_delay -clock [get_clocks {clk}] 0.5 [get_ports {din}]
set_output_delay -clock [get_clocks {clk}] 0.5 [get_ports {dout}]

```

Example 14 Block i1 Constraint File after split_constraints

```

# Block-level constraints
create_clock -name "clk" -period 1 [get_ports {clk}]
set_false_path -from [get_pins {ff2/CK}] -to [get_pins {ff3/D}]

```

4

Creating a Floorplan

This section 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.

For more information, see the following topics:

- [Supported Types of Floorplans](#)
- [Reading the Verilog Netlist](#)
- [Creating Dense and Sparse Outline Views for Large Designs](#)
- [Creating an Initial Floorplan](#)
- [Creating an L-, T-, or U-Shaped Floorplan](#)
- [Creating a Complex Rectilinear Floorplan](#)
- [Creating a Floorplan Based on a Physical Rules File](#)
- [Adjusting the Floorplan Aspect Ratio](#)
- [Defining Routing Tracks](#)
- [Flipping the First Row in the Floorplan](#)
- [Updating the Floorplanning Without Disturbing Specified Object Types](#)
- [Validating the FinFET Grid](#)
- [Scaling the Die Area](#)
- [Reporting Floorplan Information](#)
- [Reading, Writing, and Copying Floorplan Information](#)
- [Creating and Validating Floorplan Rules for Advanced Technology Nodes](#)

Supported Types of Floorplans

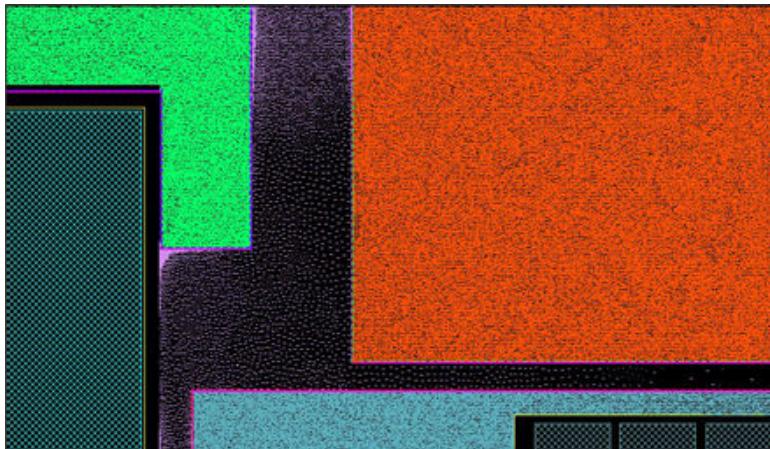
The IC Compiler II tool supports different floorplan styles to meet the requirements of your design. The channeled, abutted, and narrow-channel floorplan styles are described in the following sections.

- [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 6](#). The spacing enables the tool to place standard cells between blocks.

Figure 6 Channeled Floorplan

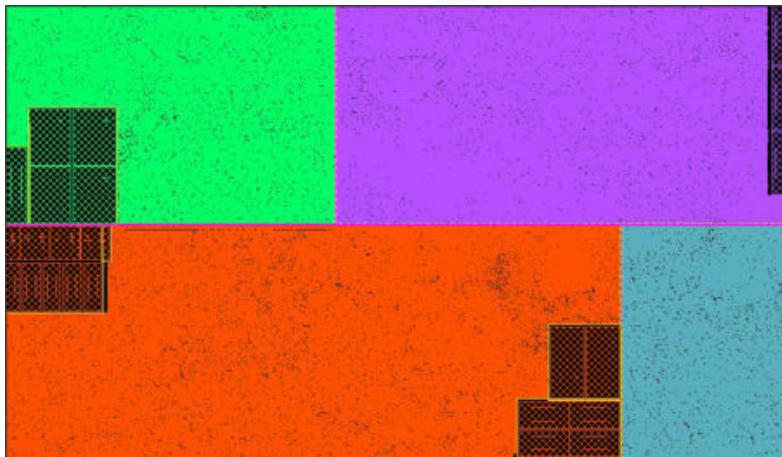


Abutted Floorplans

In the abutted floorplan style, 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 clock planning and feedthrough management. Routing congestion might also become an issue for abutted floorplan designs. Due to the difficulties in implementing an abutted floorplan, the narrow-channel floorplan style is often used to minimize the spacing between blocks.

An example of an abutted floorplan is shown in [Figure 7](#).

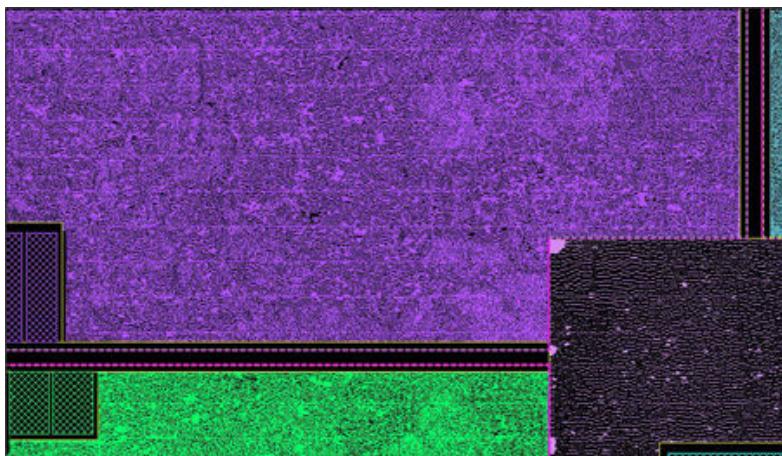
Figure 7 Abutted Floorplan



Narrow-Channel Floorplans

The narrow-channel floorplan style provides a balance between the channeled floorplan and abutted floorplan styles. 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 8](#).

Figure 8 Narrow-Channel Floorplan



Reading the Verilog Netlist

To read the Verilog netlist and begin floorplanning,

1. Create the design library to contain the design by using the `create_lib` command. If the library already exists, use the `open_lib` command to open the library.

```
icc2_shell> set libs [glob -directory ${lib_dir} *.ndm]
icc2_shell> create_lib -technology technology_file.tf \
             -ref_libs $libs lib/design
```

2. Read the Verilog netlist with the `read_verilog_outline` command and specify the appropriate options.

```
icc2_shell> read_verilog_outline -top ORCA ORCA.v
```

The Verilog netlist outline reader in the IC Compiler II tool efficiently processes very large designs. To conserve resources and minimize turnaround time, the netlist reader creates an outline view of the blocks in the different levels of design hierarchy. The outline design contains hierarchy, but no leaf cells or nets.

The `read_verilog_outline` command does not partition the design. To partition the design, you commit the design blocks with the `commit_block` command and expand the blocks to their full netlist representations with the `expand_outline` command. For more information on committing blocks, see [Committing Design Blocks](#).

You can also process a design in the full netlist flow by using the flat netlist reader. For information about reading a design with the flat netlist reader, see the `read_verilog` command.

Creating Dense and Sparse Outline Views for Large Designs

The `read_verilog_outline` command reads the Verilog netlists for large designs and prepares the design database for efficient handling and processing of the design data. During early floorplanning, you can reduce disk and memory requirements for the tool by saving only hard macros, interface logic, standard cell area, and standard cell count information. This reduced information set is stored in an outline view; this view is sufficient for refining block shapes before floorplan-level placement and pin assignment.

To maximize the performance of the tool during design planning, you can further tune the contents of the outline views for the blocks in your design. When performing design planning on a very large design, blocks should contain the smallest amount of information necessary to perform a meaningful analysis. To support the analysis, the tool enables separating Verilog modules into two content types: sparse modules and dense modules. Dense modules retain most of their contents, except for buffers and inverters, and support comprehensive data flow analysis. Sparse modules have greatly reduced content,

contain only macro cells, and support only the size and interface information needed for connectivity analysis.

To help define the contents of the outline views, the tool supports the allocation and partition methods to determine which modules are marked as dense or sparse. Choose the method to use by including the `-partition` or `-allocation` option with the `read_verilog_outline` command. Alternatively, set the `plan.outline.partition` application option to specify the method. By default, the partition method is used.

For data flow analysis, dense connectivity is needed in top-level modules. Use the `plan.outline.dense_module_depth` application option to specify the module hierarchy depth that should be marked as dense. By default, the depth is one and only the top-level module is marked as a dense module. Alternatively, use the `-depth` option to override the default setting. A value of -1 marks all non-sparse modules as dense. The hierarchy depth overrides the target cell count specified by the `-target_cell_count` option.

Partition Method

In the partition method, the tool examines the modules in the hierarchy and determines the module density based on the target block size and number of leaf cells in the block. Top-level modules in the module hierarchy are marked as dense until the hierarchical cell count beneath the module is less than a target block size. Buffers and inverters are included in the hierarchical cell count. This technique maintains connectivity in the top levels of hierarchy for data flow analysis. The target block size is specified by the `plan.outline.target_block_size` application option; by default, the target block size is 1,000,000 leaf cells. The `-target_block_size` option can be used to override the default.

After the partition blocks are designated, small module hierarchies within each partition are designated as dense modules to maintain connectivity for data flow analysis. The target number of leaf cells to allocate to these module hierarchies is specified by the `plan.outline.glue_cell_count` application option; by default the count is 100,000 leaf cells. Specify a value of -1 to avoid marking any of the small modules as dense modules. Buffers and inverters are not represented in the outline design and are not included in the target glue cell count. The `-glue_cell_count` option can be used to override the default setting.

Allocation Method

The allocation method is the default method for designating dense and sparse modules and is used to explore as much of the design hierarchy as possible within a memory budget of a maximum number of leaf cell instances. Specify the allocation method by setting the `plan.outline.partition` application option to `false` or by specifying the `-allocation` option to the `read_verilog_outline` command.

In the allocation method, modules that exceed a leaf cell count threshold are considered to be large modules and are treated as sparse modules in the outline view. Buffers and inverters are not represented in the outline design and are not included in the leaf cell count. The leaf cell count threshold is specified by the `plan.outline.large_threshold`

application option; by default the leaf cell count is 10,000. Alternatively, use the `-large_threshold` option to override the default setting.

Creating an Initial Floorplan

After reading the Verilog netlist, use the `initialize_floorplan` command (or choose Floorplan Preparation > Floorplan Initialization from the Task Assistant in the GUI) to define a floorplan boundary based on your design criteria. You can create a floorplan by specifying the length ratio between the floorplan edges, the length of the core or die edges, the aspect ratio, or the exact width and height. This step uses the outline view for the top-level design.

To create the floorplan,

1. Use the `initialize_floorplan` command with the appropriate options to create the initial floorplan.

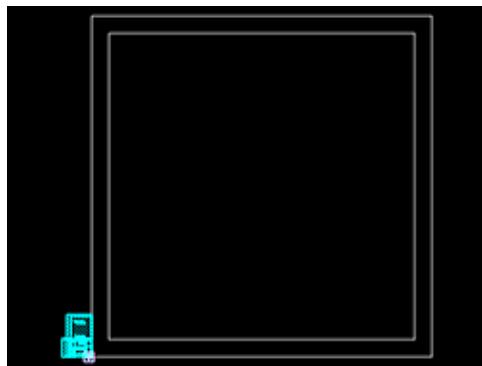
```
icc2_shell> initialize_floorplan -core_offset 100
```

Alternatively, select the Floorplan Initialization panel in the Floorplan Task Assistant from the GUI.

2. If needed, refine the floorplan by running the `initialize_floorplan` command again with different settings for the utilization ratio, side lengths, core-to-boundary spacing, and other settings.

The `initialize_floorplan` command creates the floorplan according to the options you specify. The floorplan is shown in [Figure 9](#).

Figure 9 Floorplan With Core Offset of 100



Creating an L-, T-, or U-Shaped Floorplan

To create an L-, T-, or U-shaped floorplan,

1. Use the `initialize_floorplan` command with the `-shape shape` option to create the floorplan. Supported shapes are R (rectangular), L, T, and U. The following example creates an L-shaped floorplan as shown in [Figure 10](#).

```
icc2_shell> initialize_floorplan -core_utilization 0.7 -shape L \
    -orientation N -side_ratio {1 1 1 1} -core_offset {100.0} \
    -flip_first_row true -coincident_boundary true
```

2. Adjust the length of the sides if needed by changing the arguments to the `-side_ratio` or `-side_length` options and reissue the `initialize_floorplan` command.

To further change the shape of the floorplan, you must set the dimensions of the floorplan boundary by specifying the `-side_ratio {side_a side_b ...}` option or the `-side_length {side_a side_b ...}` option. See [Figure 11](#) for a cross reference between the position of the number in the argument and the edge on the floorplan shape. This information is also available in the Task Assistant.

Figure 10 L-Shaped Floorplan

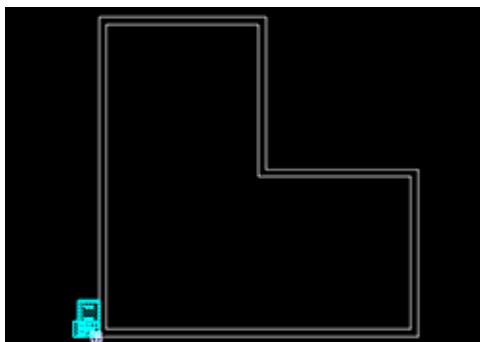
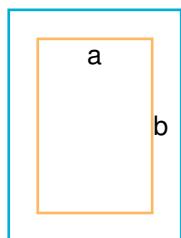
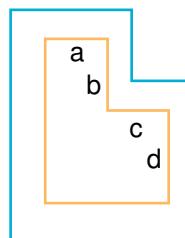


Figure 11 Edge Label Reference for Floorplan Shapes

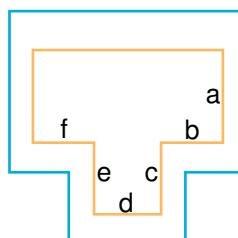
Rectangular



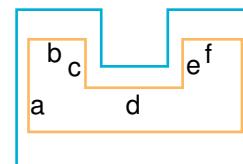
L-shape



T-shape



U-shape



Note that the side ratios apply to the core boundary by default. To apply the ratios to the die boundary, specify the `-control_type die` option.

The following example creates a U-shaped floorplan. The right arm is twice the width of the left arm by specifying the `-side_ratio` option with 1 in the second (b) position and 2 in the sixth (f) position.

```
icc2_shell> initialize_floorplan -core_utilization 0.7 -shape U \
    -orientation N -side_ratio {2 1 1 1 1 2} -core_offset 100 \
    -coincident_boundary true
```

Creating a Complex Rectilinear Floorplan

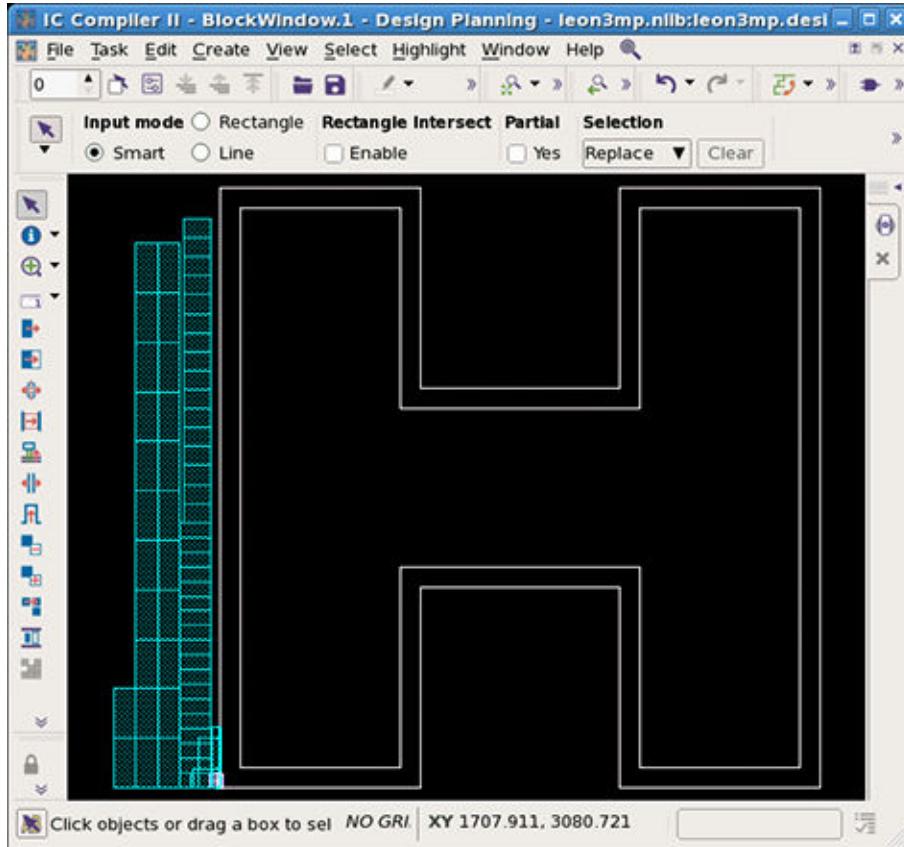
Create a complex floorplan shape by using the `initialize_floorplan` command with the `-boundary` option. Specify the `-control_type die` or `-control_type core` option of the `-boundary` option to specify the coordinates of the die or the core.

The following example creates an H-shaped floorplan by setting the coordinates of the floorplan die.

```
icc2_shell> initialize_floorplan -control_type die \
    -core_offset 100 \
    -boundary { {0 0} {0 3000} {1000 3000} {1000 2000} \
    {2000 2000} {2000 3000} {3000 3000} {3000 0} \
    {2000 0} {2000 1000} {1000 1000} {1000 0} }
```

Figure 12 shows a floorplan created by using the previous command.

Figure 12 Complex Rectilinear Floorplan



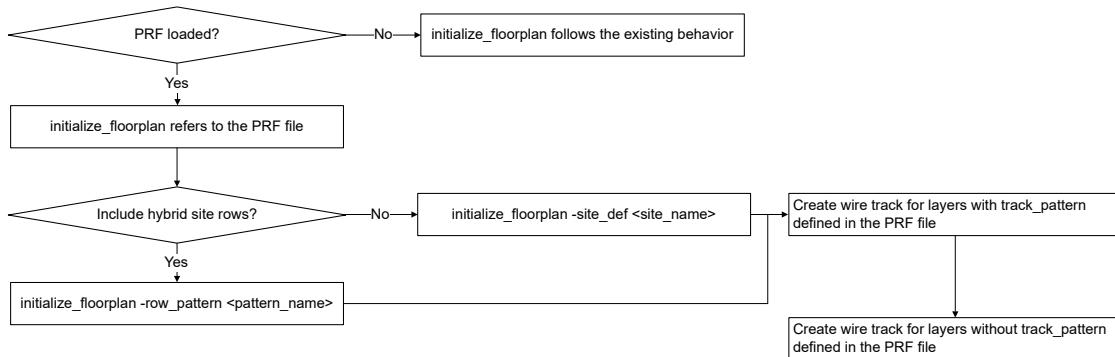
Creating a Floorplan Based on a Physical Rules File

Use the following example flow to create a floorplan based on the specifications listed in a physical rules file.

Note:

In the physical rules file, you can include placement rule constraints, physical cell and pin properties, and stream-in instructions in a unified file format.

Figure 13 Example - Physical Rule File Based Floorplan Creation Flow



1. Set up the physical rules file to meet your requirements.

For more information about the physical rules file, see [SolvnetPlus](#).

2. Load the physical rules file.

You can include the physical rules file content either in a technology file or a separate file that can be read by using the `read_physical_rules` command.

```
icc2_shell> read_physical_rules in.prf
```

3. Run the `initialize_floorplan` command with the necessary options.

For example, if you have defined hybrid rows in the physical row file, you can run the `initialize_floorplan -row_pattern` command. The tool then uses the information in this file to create the floorplan.

Adjusting the Floorplan Aspect Ratio

To create a floorplan with different side lengths, specify an aspect ratio with the `-side_ratio` option. To set the aspect ratio of the floorplan,

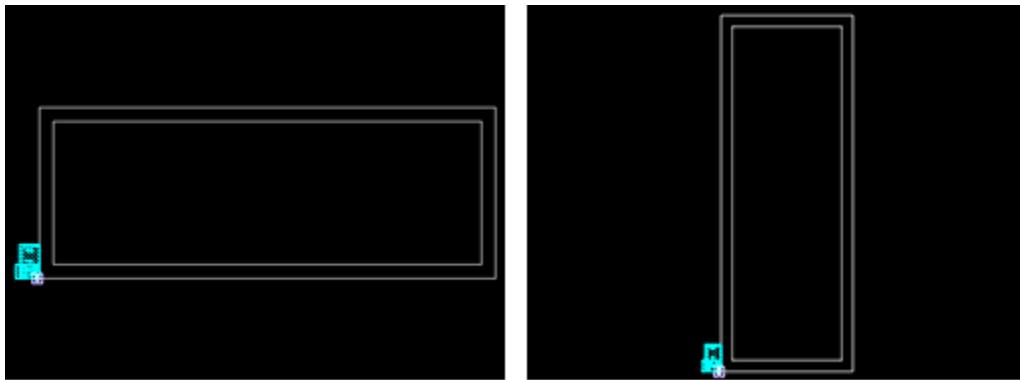
1. Determine the ratio between the width and height of the floorplan.
2. Use the `initialize_floorplan` command with the `-side_ratio` option to create a floorplan with the specified width-to-height ratio.

```
icc2_shell> initialize_floorplan -core_utilization 0.7 -shape R \
    -orientation N -side_ratio {3.0 1.0} -core_offset {100.0} \
    -flip_first_row true -coincident_boundary true
```

Note that the ratios apply to the core boundary by default. To apply the ratios to the die boundary, specify the `-control_type die` option.

[Figure 14](#) shows two floorplans created by using different `-side_ratio` option settings to produce different aspect ratios. The floorplan on the left is created by specifying `-side_ratio {3.0 1.0}`, the floorplan on the right is created by specifying `-side_ratio {1.0 3.0}`.

Figure 14 Floorplans With Aspect Ratios of 3.0 (left) and 0.33 (right)



To create these floorplans by using the Floorplan Task Assistant in the GUI,

1. Open the Floorplan Task Assistant by selecting Task > Task Assistant in the GUI and selecting the Floorplan Initialization panel under Floorplan Preparation.
2. Select Core side ratio and enter a width/height value in the Aspect ratio box.
3. Click Preview to display a wireframe preview of the floorplan.
4. Click Apply to create the floorplan.

To create the floorplans shown in [Figure 14](#), enter 3.0 in the Aspect ratio box to create the floorplan on the left, or enter 0.33 to create the floorplan on the right.

Defining Routing Tracks

To create routing tracks in the floorplan, use the `create_track` command. You can specify the layer, the number of tracks, the pitch distance between tracks, and so on.

You can specify the track location in two ways:

- To specify the location of the first track, use the `-coord` option. The tool uses the x- and y-coordinates set by the `-dir` option, which specifies the stepping direction in which the tracks are placed. By default, the first track is located at half the space distance inside the die area.
- To specify that one of the track lines be aligned to the lower boundary of either the core area bounding box or the block bounding box, use the `-relative_to` option. The

tracks are created completely within the block boundary even if the block boundary is larger than the core boundary.

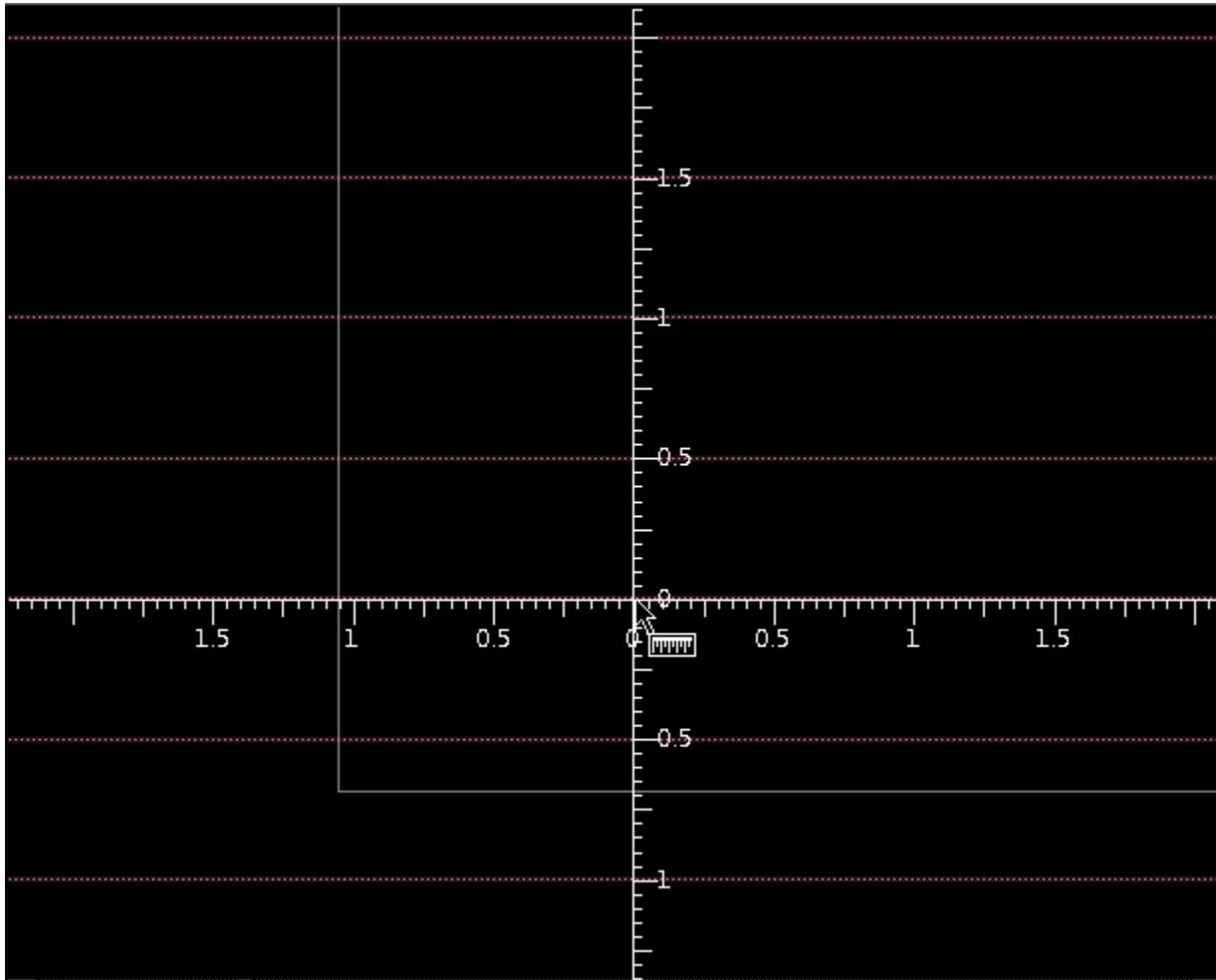
You can also specify that subsequent grid points be created with respect to the core area bounding box or the lower-left of the block bounding box by using the `-end_grid_relative_to` option.

Alternatively, you can create routing tracks in the GUI by choosing Create > Track from the menu.

[Figure 15](#) shows a floorplan created with the `create_track` command. In this example, the `remove_tracks` 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_tracks` command reports the routing track direction, startpoint, number of tracks, and pitch for each set of routing tracks in the design.

```
icc2_shell> remove_tracks -all
1
icc2_shell> create_track -layer METAL3 -space 0.5
{TRACK_0}
icc2_shell> report_tracks
*****
Report report_tracks
Design : TEST
...
*****
Layer      Direction    Start      Tracks      Pitch      Attr
-----      -----
METAL3          Y        0.250     4023      0.500    default
1
```

Figure 15 *Floorplan After Inserting Routing Tracks*



You can also create tracks based on nondefault rules with the `create_track` command. For information about creating tracks with nondefault rules, see [SolvNet article 3001923, "How Can I Create Routing Tracks Based on Nondefault Rules \(NDRs\)?"](#)

Flipping the First Row in the Floorplan

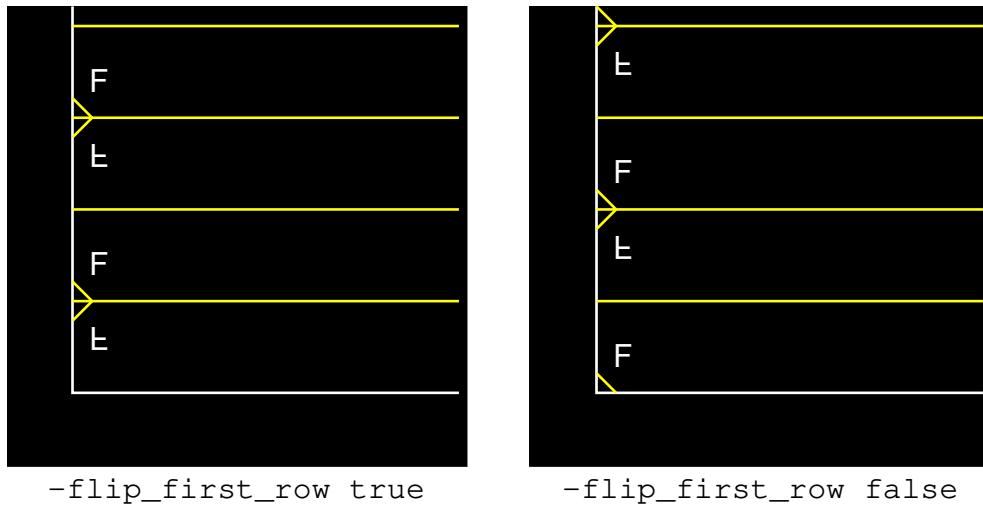
To avoid flipping the first row in the floorplan, specify the `-flip_first_row false` option as shown in the following example.

```
icc2_shell> initialize_floorplan -flip_first_row false
```

In [Figure 16](#), the layout image on the left shows the layout created by using the `-flip_first_row true` option, and the layout image on the right shows the

floorplan created by using the `-flip_first_row false` option. By default, the `initialize_floorplan` command flips the first row.

Figure 16 Floorplan With First Row Flipped and Not Flipped

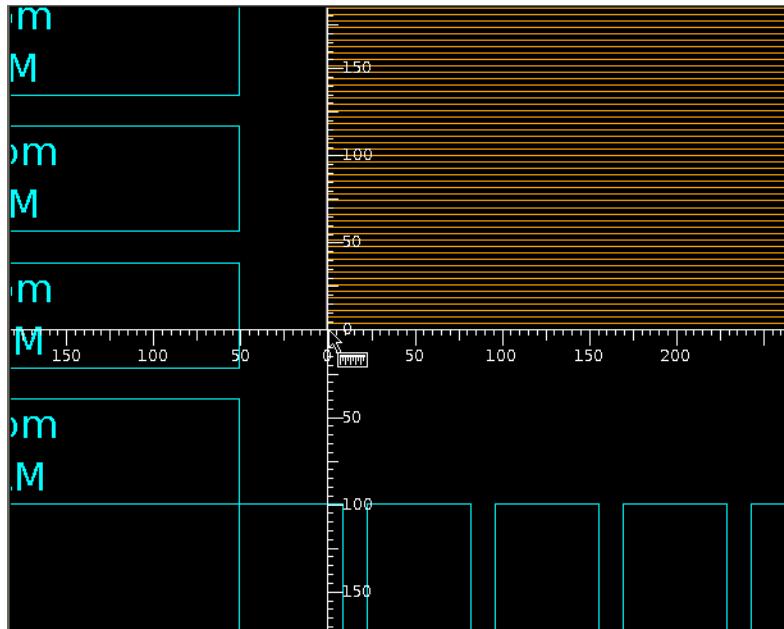


To create an offset between the boundary and the core, specify the `-core_offset` option with the `initialize_floorplan` command as shown in the following example.

```
icc2_shell> initialize_floorplan -core_utilization 0.8 \
    -core_offset {50 100}
```

[Figure 17](#) shows a zoomed-in view of a floorplan created by using the `initialize_floorplan` command with the `-core_offset` option. This option sets the distance between the floorplan core and the floorplan boundary. The `-core_offset` option accepts a list of offsets, where each offset corresponds to an edge of the floorplan. See the `initialize_floorplan` man page for information about mapping the argument position to an edge of the floorplan.

Figure 17 Floorplan With Boundary to Core Spacing



Updating the Floorplanning Without Disturbing Specified Object Types

After creating the initial floorplan with the `initialize_floorplan` command, repeat the command with different options to experiment with different floorplan shapes and characteristics. To retain the placement or shape for specified object types when changing the floorplan, use one or more of the `-keep_*` options.

- `-keep_boundary`: Retain the floorplan boundary when updating the floorplan.
- `-keep_pg_route`: Retain power and ground routes when updating the floorplan.
- `-keep_detail_route`: Retain all routes except power and ground routes when updating the floorplan.
- `-keep_placement {block}`: Retain block placement when updating the floorplan.
- `-keep_placement {io}`: Retain terminal and pad placement when updating the floorplan.
- `-keep_placement {macro}`: Retain macro placement when updating the floorplan.

- `-keep_placement {std_cell}`: Keep standard cell placement when updating the floorplan.
- `-keep_placement {physical_only}`: Keep the placement for physical only cells when updating the floorplan; the cells can be identified with the following commands:

```
icc2_shell> get_cells -filter "design_type==physical_only"
icc2_shell> get_cells -filter "design_type==fill"
```
- `-keep_placement {all}`: Retain terminal, pad, macro, block, standard cell, and physical-only objects when updating the floorplan options.

Validating the FinFET Grid

The IC Compiler II tool supports a FinFET grid to guide the placement of library cells that contain FinFET devices. The x- and y-spacing and offset values for the FinFET grid are specified in the technology file.

To report information about the FinFET grid, use the `report_grids -type finfet` command as follows:

```
icc2_shell> report_grids -type finfet
FinFET Grid          Description
-----
Top Design           orca_hier_lib:ORCA.design
FinFET Grid defined true
FinFET Grid X Pitch 0.001
FinFET Grid Y Pitch 0.048
FinFET Grid X Offset 0
FinFET Grid Y Offset 0
1
```

To check for placement or boundary violations with respect to the FinFET grid, use the `check_finfet_grid` command as follows:

```
icc2_shell> check_finfet_grid
...
Error: boundary point (0 101.97) of hard macro
'I_ORCA_TOP/I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_7'
    is not on FinFET grid. (DPCHK-002)
Error: boundary point (0 101.97) of hard macro
'I_ORCA_TOP/I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_6'
    is not on FinFET grid. (DPCHK-002)
...
```

To check specific object types, specify only those objects to the `check_finfet_grid` command as follows:

```
icc2_shell> check_finfet_grid -objects [get_cells -hierarchical \
-filter "design_type==pad"]
```

```
...
Reporting violations in block ORCA ...
Error: boundary point (0 345.565) of io pad 'test_mode_iopad'
    is not on FinFET grid. (DPCHK-002)
Error: boundary point (0 345.565) of io pad 'sdr_clk_iopad'
    is not on FinFET grid. (DPCHK-002)
```

The following commands are aware of the FinFET grid in the tool:

- `create_blackbox`
- `create_grid`
- `create_io_guide`
- `create_io_ring`
- `create_macro_array`
- `create_placement`
- `create_power_switch_array`
- `create_power_switch_ring`
- `create_site_array`
- `create_site_row`
- `initialize_floorplan`
- `place_io`
- `report_grids`
- `set_block_grid_references`
- `set_grid`
- `set_signal_io_constraints`
- `shape_blocks`

Scaling the Die Area

You can shrink or expand the die area or the size of your design by using the `modify_die_area` command.

- Reducing the size of the design can offer some potential benefits such as, smaller the die, more the number of dies per wafer. However, ensure that you aim for minor changes. Major changes can cause issues including excessive utilization or macros

not being able to fit in a design. You can consider this action successful when your shrunken design is still routable, meets operating speed requirements, and also meets allowable power consumption requirements.

- Enlarging the design can help with congestion.

Table 2 Options to the modify_die_area Command Control How Much the Die Area is Resized

To do this...	Use this...
To control the resizing of the core area, its height, and its width Valid values range from 0.1 to 10	-area_scaling_factor -height_scaling_factor -width_scaling_factor
To specify the required utilization of the core area	-core_utilization
To list the edges whose length you want to remain unchanged	-fixed_edges
To define constraints on aspect ratios	-sizing_type
To specify the row pattern to be used in floorplan creation	-row_pattern
To calculate and print the summary of the estimated floorplan layout and create a new floorplan based on the estimates	-compile
To run a custom script that re-creates the tracks after the resizing of the die By default, the command derives the track information from the input floorplan and reuses it	-run_track_creation_script

The following example shrinks the core area to 85% of its original size.

```
icc2_shell> modify_die_area -area_scaling_factor 0.85
```

The following example shrinks core height to 80% of the original height of the die, which is, 20% reduction.

```
icc2_shell> modify_die_area -height_scaling_factor 0.8
```

The following example shrinks core so that its utilization is 75%.

```
icc2_shell> modify_die_area -core_utilization 0.75
```

The following example increases the core height to 110% of the original height of the core, which is a 10% increase.

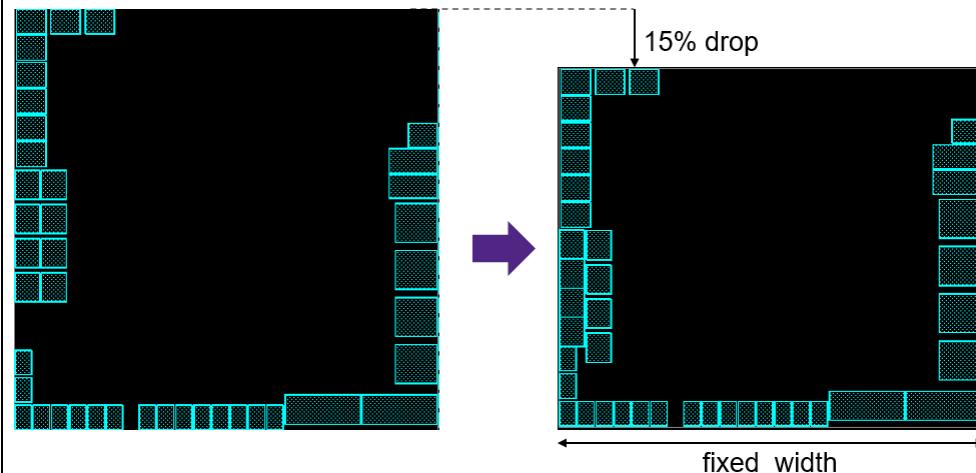
```
icc2_shell> modify_die_area -height_scaling_factor 1.1
```

In the following example, the core area is shrunk to 85% of the original size but the horizontal dimension is fixed.

```
icc2_shell modify_die_area -area_scaling_factor 0.85 -sizing_type fixed_width
-compile
```

```
-----
Die Size Exploration Summary
-----
Original Floorplan Layout
    Utilization Ratio: 0.3934
    Core Area: 3150962.2170
    Core Width: 1775.3000
    Core Height: 1774.8900
    Boundary Co-ordinates: {350.0000 350.0000} {350.0000 2124.8900}
    {2125.3000 2124.8900} {2125.3000 350.0000}

Estimated Floorplan Layout
    Utilization Ratio: 0.4628
    Core Area: 2678317.8845
    Core Width: 1775.3000
    Core Height: 1508.6565
    Boundary Co-ordinates: {350.0000 350.0000} {350.0000 1858.6565}
    {2125.3000 1858.6565} {2125.3000 350.0000}
```



Reporting Floorplan Information

Use the `report_design -floorplan` or `report_design -all` command to report the following information for the floorplan:

- Area: Core and chip area.
- Site row: Name, width, height, number of rows, number of tiles, and area for each site row.
- Blockages: Count and area for each blockage type, including hard placement blockages, soft placement blockages, hard macro blockages, partial placement blockages, routing blockages, category blockages, reserved placement group blockages, shaping blockages, routing for design rule blockages, placement blockages which only allow buffers, and no-register blockages for each blockage type.
- Power domains: Power domain name, voltage area name, and primary power and ground nets for each power domain.
- Voltage areas: Name, number of shapes, area, target utilization, and bounding box coordinates for each voltage area.
- Group bounds: Number of dimensionless group bounds.
- Move bounds: Name, area, utilization, and bounding box for each hard, soft, and exclusive move bound.
- Route guide: Count and area for each extra detour region, route guide over in-chip overlay cell layers, river routing guide, all double vias in the route guides, access preference route guide, default route guide, switched direction only route guide.
- Multibit registers: Number of multibit registers.
- Reserved placement groups: Number of top-level reserved placement groups, number of reserved placement cells, reserved placement cell area, and reserved placement cell area percentage for each reserved placement group.
- Layers: layer name, direction, pitch, default width, minimum width, minimum spacing, and minimum spacing between the same nets for each layer.

Reading, Writing, and Copying 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.

- [Reading in Floorplan Information](#)
 - [Reading DEF Files](#)
 - [Writing Out Floorplan Files](#)
 - [Copying Floorplan Objects From Block to Block](#)
-

Reading in Floorplan Information

To load floorplan information written by the `write_floorplan` command for a flat block, use the `source` command to read in the `fp.tcl` file for the design as follows.

```
icc2_shell> source floorplan/fp.tcl
```

For a hierarchical design which might contain a mix of design and abstract block representations, use the `set_constraint_mapping_file` and `load_block_constraints` commands as follows:

```
icc2_shell> set_constraint_mapping_file floorplan/mapfile
icc2_shell> load_block_constraints -type FLOORPLAN -all_blocks
```

The `load_block_constraints -type FLOORPLAN` command and option `source` each `fp.tcl` file contained in the mapping file to re-create the floorplan for each block. The constraint mapping file written by the `write_floorplan` command contains a list of block names, FLOORPLAN keywords, and `fp.tcl` file locations for each block as shown in the following constraint mapping file:

BLENDER_0	FLOORPLAN	floorplan/BLENDER_0/fp.tcl
BLENDER_1	FLOORPLAN	floorplan/BLENDER_1/fp.tcl
BLENDER_2	FLOORPLAN	floorplan/BLENDER_2/fp.tcl
CONTEXT_MEM	FLOORPLAN	floorplan/CONTEXT_MEM/fp.tcl
ORCA	FLOORPLAN	floorplan/fp.tcl
PCI_TOP	FLOORPLAN	floorplan/PCI_TOP/fp.tcl
RISC_CORE	FLOORPLAN	floorplan/RISC_CORE/fp.tcl
SDRAM_TOP	FLOORPLAN	floorplan/SDRAM_TOP/fp.tcl

Reading DEF Files

To read a DEF file, use the `read_def` command. The following command reads the `orca.def` DEF file.

```
icc2_shell> read_def orca.def
```

To analyze the syntax of the input DEF files before annotating the objects on the design, specify the `-syntax_only` option.

```
icc2_shell> read_def -syntax_only orca.def
```

Writing Out Floorplan Files

To write the current floorplan, use the `write_floorplan` command. The `write_floorplan` command creates a directory named “floorplan” and writes out the `floorplan.def`, `floorplan.tcl`, and `fp.tcl` files to the directory. The `floorplan.def` file contains the physical layout, connectivity, and design constraint information for the design. The `floorplan.tcl` file contains additional design intent that cannot be described in the DEF file, such as hard macro keepout margins. The `fp.tcl` file contains commands to remove design constraints such as voltage areas, bounds, routing guides, and so on. You can control the information written to the files by using command options. The `write_floorplan` command supports both design views and abstract views for blocks.

The `write_floorplan` command also writes out the constraint mapping file which maps each block to its corresponding `fp.tcl` file. When reading in the floorplan data, the mapping file is assigned with the `set_constraint_mapping_file` command and used by the `load_block_constraints` command.

The following examples write out floorplan files for different design requirements:

- Write out macros and set origin, orientation, status, and other attributes.

```
icc2_shell> write_floorplan -output "macros" -objects [get_cells \
    -physical_context -filter is_hard_macro==true]
```

- Write out PG regions, overwrite files in the pg directory if they exist.

```
icc2_shell> write_floorplan -force -objects [get_pg_regions] \
    -output pg
```

- Write out pad cells and macro cells with a physical status of `fixed` or `unplaced`.

```
icc2_shell> write_floorplan -force -include cells \
    -include_physical_status {fixed unplaced} -cell_types {pad macro}
```

- Write out clock, power, and ground nets to the DEF file; signal nets are excluded.

```
icc2_shell> write_floorplan -force -output nets \
    -net_types {clock power ground}
```

- Add the `-add_def_only_objects {cells}` `-no_incremental` options to the `read_def` command in the output `fp.tcl` floorplan file.

```
icc2_shell> write_floorplan -output fp -nosplit -def_version 5.8 \
    -read_def_options {-add_def_only_objects {cells} -no_incremental}
```

Copying Floorplan Objects From Block to Block

To copy floorplan information from one block to another, use the `copy_floorplan` command. The command copies the floorplan data from the design view of source block to

the design view of an existing destination block. The designs can be in the same library or in different libraries. A design label can be specified along with the block name.

The `copy_floorplan` command can include or exclude one or more object types when copying. By default, all object types are copied. Use the `-include object_type_list` or `-exclude object_type_list` option to specify the object types that are included or excluded. Valid object types are as follows:

- blockages
- bounds
- cells
- corridors
- die_area
- edit_groups
- io_guides
- macros
- module_boundaries
- pin_guides
- pins
- route_guides
- rows
- tracks
- vias
- scan_chains
- routing_directions
- voltage_areas
- fills
- pg_metal_fills
- routing_rules
- pg_regions
- port

- supernet
- bundle
- topology_plan
- net_estimation_rule

The `copy_floorplan` command can also copy the following object types from the source block to the destination block.

- block
- design
- routing_blockage
- shaping_blockage
- placement_blockage
- bound
- cell
- routing_corridor
- core_area
- edit_group
- fill_cell
- io_guide
- via
- shape
- net
- pin_guide
- pg_region
- routing_guide
- site_row
- site_array
- track
- voltage_area

- routing_rule
- port
- supernet
- bundle
- topology_plan

Use the `-objects` option to specify the collection of objects. Use other options to the command to specify the source floorplan, destination floorplan, and which floorplan objects are copied.

- Copy all floorplan information from the mem design with the “placed” label in the test library to the currently opened block and library.

```
icc2_shell> copy_floorplan -from_block test:mem/placed
```

If you use the `-force` option, objects are copied to the existing floor plan with new names even though they are present in the floor plan.

- Copy all floorplan information along with power switch logical connectivity information from the mem design with the “placed” label in the test library to the mem_copy block with the “placed” label in the test library by using the `-to_block` option.

```
icc2_shell> copy_floorplan -from_block test:mem/placed \
           -to_block test:mem_copy/placed
```

Note:

The `-force` option has no effect on the power switch cells and connections. This option does not rename the cells unlike other objects. Therefore, if the cells already exist but not the logical connectivity, running the `copy_floorplan` command again with the `-force` option creates only the logical connections.

- Copy only floorplan information for blockages and bounds by using the `-include` option.

```
icc2_shell> copy_floorplan -verbose -from_block test:mem/placed \
           -to_block test:mem_copy/placed -include {blockages bounds}
```

- Copy all floorplan information except for blockages and bounds by using the `-exclude` option.

```
icc2_shell> copy_floorplan -verbose -from_block test:mem/placed \
           -to_block test:mem_copy/placed -exclude {blockages bounds}
```

- Copy only vias VIA_S_0 and VIA_C_0 from the source design to the destination design by using the `-objects` option.

```
icc2_shell> copy_floorplan -verbose -from_block test:mem/placed \
    -to_block test:mem_copy/placed \
    -objects [get_vias {VIA_S_0 VIA_C_0}]
```

- Copy floorplan objects hierarchically with the `-hierarchical` option.

```
icc2_shell> copy_floorplan -verbose -from_block test:mem/placed \
    -to_block test:mem_copy/placed \
    -hierarchical
```

Table 3 shows examples of how the `copy_floorplan` works for some of the objects.

Table 3 How the `copy_floorplan` Command Works

Object		copy_floorplan default	copy_floorplan -objects	copy_floorplan -include	copy_floorplan -exclude	copy_floorplan -force
When the copied objects do not exist in the destination design	supernet	Copy to the destination design	Copy to the destination design	Copy to the destination design	Does not copy to the destination design	Copy to the destination design
	bundle	Copy to the destination design	Copy to the destination design	Copy to the destination design	Does not copy to the destination design	Copy to the destination design
	net_estimation_rule	Copy to the destination design	Cannot be specified with the -objects option	Copy to the destination design	Does not copy to the destination design	Copy to the destination design
	topology_plan	Copy to the destination design	Copy to the destination design	Copy to the destination design	Does not copy to the destination design	Copy to the destination design
	topology_edge	Copy to destination design if the associated topology_plan is also copied	Cannot be specified with the -objects option	Cannot be specified with the -include option	Cannot be specified with the -exclude option	Cannot be specified with the -force option

Table 3 How the *copy_floorplan* Command Works (Continued)

Object	<i>copy_floorplan default</i>	<i>copy_floorplan -objects</i>	<i>copy_floorplan -include</i>	<i>copy_floorplan -exclude</i>	<i>copy_floorplan -force</i>
topology_node	Copy to destination design if the associated topology_plan is also copied	Cannot be specified with the -objects option	Cannot be specified with the -include option	Cannot be specified with the -exclude option	Cannot be specified with the -force option
topology_repeater	Copy to destination design if the associated topology_plan is also copied	Cannot be specified with the -objects option	Cannot be specified with the -include option	Cannot be specified with the -exclude option	Cannot be specified with the -force option
When the copied objects exist in the destination design	supernet	Does not copy to the destination design	Does not copy to the destination design	Does not copy to the destination design	Cannot be specified with the -exclude option
	bundle	Does not copy to the destination design	Does not copy to the destination design	Does not copy to the destination design	Copy to the destination design
	net_estimation_rule	Does not copy to the destination design	Cannot be specified with the -objects option	Does not copy to the destination design	Copy to the destination design
	topology_plan	Does not copy to the destination design	Does not copy to the destination design	Does not copy to the destination design	Copy to the destination design
	topology_edge	Does not copy to the destination design	Cannot be specified with the -objects option	Cannot be specified with the -include option	Cannot be specified with the -force option

Table 3 How the copy_floorplan Command Works (Continued)

Object	copy_floorplan default	copy_floorplan -objects	copy_floorplan -include	copy_floorplan -exclude	copy_floorplan -force
topology_node	Does not copy to the destination design	Cannot be specified with the -objects option	Cannot be specified with the -include option	Cannot be specified with the -exclude option	Cannot be specified with the -force option
topology_repeater	Does not copy to the destination design	Cannot be specified with the -objects option	Cannot be specified with the -include option	Cannot be specified with the -exclude option	Cannot be specified with the -force option

Creating and Validating Floorplan Rules for Advanced Technology Nodes

The IC Compiler II tool supports additional rule creation and checking utilities for floorplans developed for advanced technology nodes. These floorplans have additional requirements for the placement and spacing of standard cells, macros, and other objects in the design.

Creating Additional Floorplan Rules

To create floorplan rules for floorplans that have additional requirements, use the commands described in the following table. The commands are described in detail following the table.

Command	Description
set_floorplan_area_rules	Defines a minimum area, maximum area, list of valid or invalid areas, or a range of valid or invalid areas.
set_floorplan_enclosure_rules	Defines an enclosure rule to check for legal spacing between an enclosed object and its enclosing object.
set_floorplan_halo_rules	Defines a halo rule to check for legal spacing between an enclosed object and its enclosing object.
set_floorplan_spacing_rules	Defines a spacing rule to check for legal spacing between objects.
set_floorplan_width_rules	Defines a width rule to check for legal spacing between objects.

Command	Description
set_floorplan_density_rules	Defines a rule to check for floorplan density.

Floorplan Area Rules

To define a minimum area, maximum area, a list of valid or invalid areas, or a range of valid or invalid areas for a set of library cells or design objects, use the `set_floorplan_area_rules` command. Valid objects are block boundary, core area, hard macro, placement blockage, routing blockage, soft macro, `va_boundary`, and standard cell area.

The following example creates a floorplan area rule named `area_rule_1`. The rule specifies that the minimum area for a standard cell is 100.

```
icc2_shell> set_floorplan_area_rules -name area_rule_1 \
    -object_types std_cell_area -min 100
```

Floorplan Enclosure Rules

To define an enclosure rule to check for legal spacing between an enclosed object and its enclosing object, use the `set_floorplan_enclosure_rules` command. Use options to specify object types, object sides, object corners, allowed object rotation, and valid or invalid spacing ranges and values.

The following example creates a floorplan enclosure rule named `enclosure_rule_1`. The rule specifies that when a routing blockage encloses one of the specified library cells, the routing blockage must fully enclose the cell on all sides and there must be a spacing of 5.0 between the routing blockage and the library cell.

```
icc2_shell> set_floorplan_enclosure_rules -name enclosure_rule_1 \
    -from_object_types routing_blockage -layers METAL \
    -to_lib_cells $lib_cells -follow_rotations -sides all \
    -must_enclose -min 5.0
```

Floorplan Halo Rules

To define a halo rule to check for legal spacing between an enclosed object and its enclosing objects, use the `set_floorplan_halo_rules` command. Use options to specify object types, object sides, object corners, allowed object rotation, and valid or invalid spacing ranges and values.

The following example creates a floorplan halo rule named `halo_rule_1`. The rule specifies that when a routing blockage encloses a macro, the routing blockage must fully enclose the macro on all sides and there must be a spacing of 3.0 between the routing blockage and the library cell.

```
icc2_shell> set_floorplan_halo_rules -name halo_rule_1 \
    -from_object_types routing_blockage -layers METAL2 \
    -to_object_types hard_macro -sides all -type outer \
    -must_enclose -min 3.0
```

Floorplan Spacing Rules

To define a spacing rule to check for legal spacing between objects, use the `set_floorplan_spacing_rules` command. Use options to specify object types, object sides, object layers, allowed object rotation, minimum parallel run length, overlaps, and valid or invalid spacing ranges and values.

The following example creates a floorplan spacing rule named `spacing_rule_1`. The rule specifies the spacing between the specified library cells and the standard cell area.

```
icc2_shell> set_floorplan_spacing_rules -name spacing_rule_1 \
    -from_lib_cells $lib_cells -follow_rotations \
    -to_object_types std_cell_area -directions vertical \
    -min_parallel_run_length -1.0 -step 2.0 -offset 13.0
```

Floorplan Width Rules

To define a width rule to check for legal spacing between objects, use the `set_floorplan_width_rules` command. Use options to specify object types, width type of the check, side or direction, object layers, allowed object rotation, maximum and minimum width, and valid or invalid width ranges and values.

The following example creates a floorplan width rule named `width_rule_1`. The rule specifies that the horizontal width of the standard cell area must be an even multiple of 8.0 plus the offset of 40.0.

```
icc2_shell> set_floorplan_width_rules -name width_rule_1 \
    -object_types std_cell_area -type simple -direction horizontal \
    -step 8.0 -offset 40.0
```

Floorplan Density Rules

To define a floorplan density rule, use the `set_floorplan_density_rules` command. An example of a floorplan density rule is the maximum bump density of the die. Use options to specify object types, library cells, routing layers, density calculation method, cut method, region of the design where the density should be calculated, window size, window step, and distance to be maintained between objects.

The following example creates a floorplan density rule named `ds1`.

```
icc2_shell> set_floorplan_density_rules -name ds1 \
    -from_object_types shape -to_object_types io_pad \
    -numerator amount -cut_method drop -density_method local_window \
    -window_size {1 2} -window_step {3 4} -max 10
```

Validating Floorplan Rules and Repairing Errors

After creating advanced node floorplan rules with the commands listed in [Creating Additional Floorplan Rules](#), use the `check_floorplan_rules` command to check your floorplan against the rules.

To display violations in the error browser, select the “Display errors in layout window” option in the `check_floorplan_rules` dialog box in the GUI.

Use options and arguments of the `check_floorplan_rules` command as follows to control how the floorplan rule checks are performed.

- Specify the floorplan rules to check by listing them after the command name. Use the `check_floorplan_rules` command to list the currently specified floorplan rules.

```
icc2_shell> check_floorplan_rules {floorplan_rule_1 floorplan_rule_2}
...
Error: found region {{30 115} {47 262}} from {hard_macro: MACRO1} to
{hard_macro: MACRO2} violates rule floorplan_rule_1 on vertical
direction:
not on grid. (DPCHK-010)
```

- Specify the object types (`hard_macro`, `std_cell_area`, `soft_macro`, `block_boundary`, or `core_area`) to check by using the `-object_types` option.

```
icc2_shell> check_floorplan_rules -object_types {std_cell hard_macro}
...
Error: found region {{340 1} {431 58}} from
{block_boundary} to {hard_macro: MACRO5} violates rule rule_1
on horizontal direction: not on grid. (DPCHK-010)
```

- Specify the library cells to check by using the `-lib_cells` option.

```
icc2_shell> check_floorplan_rules -lib_cells MACRO4
Error: found region {{262 58} {299 58}} from {cell: U3/MACRO4} to
{cell: U0/MACRO4} violate rule rule_2 on vertical direction: less than
offset, not on grid. (DPCHK-010)
```

After creating and checking the floorplan rules with the `set_floorplan_area_rules`, `set_floorplan_enclosure_rules`, `set_floorplan_halo_rules`, `set_floorplan_spacing_rules`, `set_floorplan_width_rules`, and `check_floorplan_rules` commands, use the `fix_floorplan_rules` command to modify the design based on the rules. The command performs the following changes to the design:

- Moves macros to satisfy all grid-type spacing rules and enclosure rules from the block boundary
- Creates a hard keepout margin around hard macros to satisfy spacing and halo rules

- Creates routing blockages after macro placement to satisfy all routing blockages to macro enclosure and halo rules
- Creates placement blockages after macro placement to satisfy all width rules, spacing rules to macros, enclosure rules from block boundary, and area rules for the standard cell area
- Adjusts core area and block boundary to mitigate block boundary to core area enclosure rule violations and core area width rule violations
- (Optional) Writes out a Tcl file that contains commands to implement the design changes made by the `fix_floorplan_rules` command

The following example checks the floorplan rules, uses the `fix_floorplan_rules` command to adjust the position of hard macros, and then performs a second check of the floorplan rules. The second check shows that all floorplan rules are repaired by the `fix_floorplan_rules` command.

```
icc2_shell> check_floorplan_rules
INFO: found 4 error(s) violate rule: rule_1
INFO: found 4 error(s) violate rule: rule_2
INFO: found 50 error(s) violate rule: rule_3
INFO: found 8 error(s) violate rule: rule_4
INFO: checking floorplan rules finished.
```

```
icc2_shell> fix_floorplan_rules -include hard_macro
Fixing hard macro floorplan spacing rules.
```

```
icc2_shell> check_floorplan_rules
INFO: found 0 error(s) violate rule: rule_1
INFO: found 0 error(s) violate rule: rule_2
INFO: found 0 error(s) violate rule: rule_3
INFO: found 0 error(s) violate rule: rule_4
```

5

Handling Black Boxes

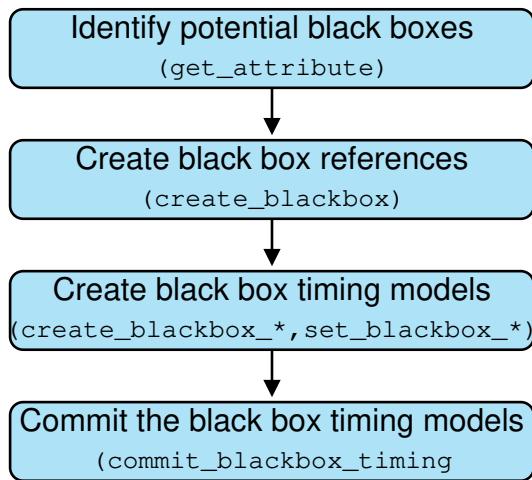
The IC Compiler II tool supports black boxes for modules in the design planning flow. A black box is a module that has no netlist, has only a partial netlist, or is not bound to a higher level module. Use black boxes to continue design planning without requiring a complete netlist for all modules. The tool supports block boxes for both physical and timing purposes, and for all stages of design planning. Note that only physical hierarchies can be black boxes in the tool.

The following design planning operations are available for black boxes:

- Multiply-Instantiated black box support
- Shaping
- Macro and standard placement (for partial netlist Black Boxes)
- PG Routing
- Pin assignment and feedthrough creation
- Push-down and pop-up
- Feedthrough buffering
- Abstract creation and merging
- Timing model creation
- Budgeting

The flow to create black box physical models and timing models is shown in [Figure 18](#).

Figure 18 Black Box Flow



For more details, see the following topics:

- [Identifying Potential Black Boxes](#)
- [Creating Black Box References](#)
- [Creating a Black Box Timing Model](#)
- [Black Box Timing Example](#)
- [UPF for Black Box Designs](#)

Identifying Potential Black Boxes

Black boxes can be identified by retrieving cells with a `black_box` attribute set to `true`. Black boxes can be further classified into missing/unbound, empty, partial, feedthrough, and tie-off. In addition, feedthrough and tie-off modules can be identified by examining the following attributes on the block:

- `has_timing_model`
- `hierarchy_hard_macro_count`
- `hierarchy_has_shadow_netlist`
- `hierarchy_has_shadow_netlist_only`
- `hierarchy_pad_cell_count`
- `hierarchy_physical_only_cell_count`

- hierarchy_std_cell_count
- hierarchy_switch_cell_count
- is_unbound
- target_boundary_area

Use the following guidelines to identify different types of black boxes:

- Missing modules

Missing modules have no module definition (no `module` statement) in the Verilog netlist and have the `is_unbound` attribute set to `true`.

```
icc2_shell> get_attribute -objects $cells -name is_unbound
true
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_std_cell_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_has_shadow_netlist
false
```

- Empty modules

Empty modules have a module definition but no content. Use the hierarchy count attributes for standard cells and hard macros to identify these types of modules.

```
icc2_shell> get_attribute -objects $cells -name is_unbound
false
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_std_cell_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_hard_macro_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_has_shadow_netlist
false
```

- Partial modules

Modules with only a partial netlist contain an incomplete set of cells. Use the hierarchy count attributes to identify these types of modules.

```
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_hard_macro_count
3
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_std_cell_count
119
```

```
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_has_shadow_netlist
true
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_has_shadow_netlist_only
false
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_pad_cell_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_physical_only_cell_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_switch_cell_count
0
```

- Feedthrough modules

Feedthrough modules contain net connections between input and output ports, but contain no cells.

```
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_std_cell_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_hard_macro_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_has_shadow_netlist
true
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_has_shadow_netlist_only
true
```

- Tie-off modules

Tie-off modules have only output ports with are tied to a logical value.

```
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_std_cell_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_hard_macro_count
0
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_has_shadow_netlist
false
icc2_shell> get_attribute -objects $cells \
    -name hierarchy_has_shadow_netlist_only
false
```

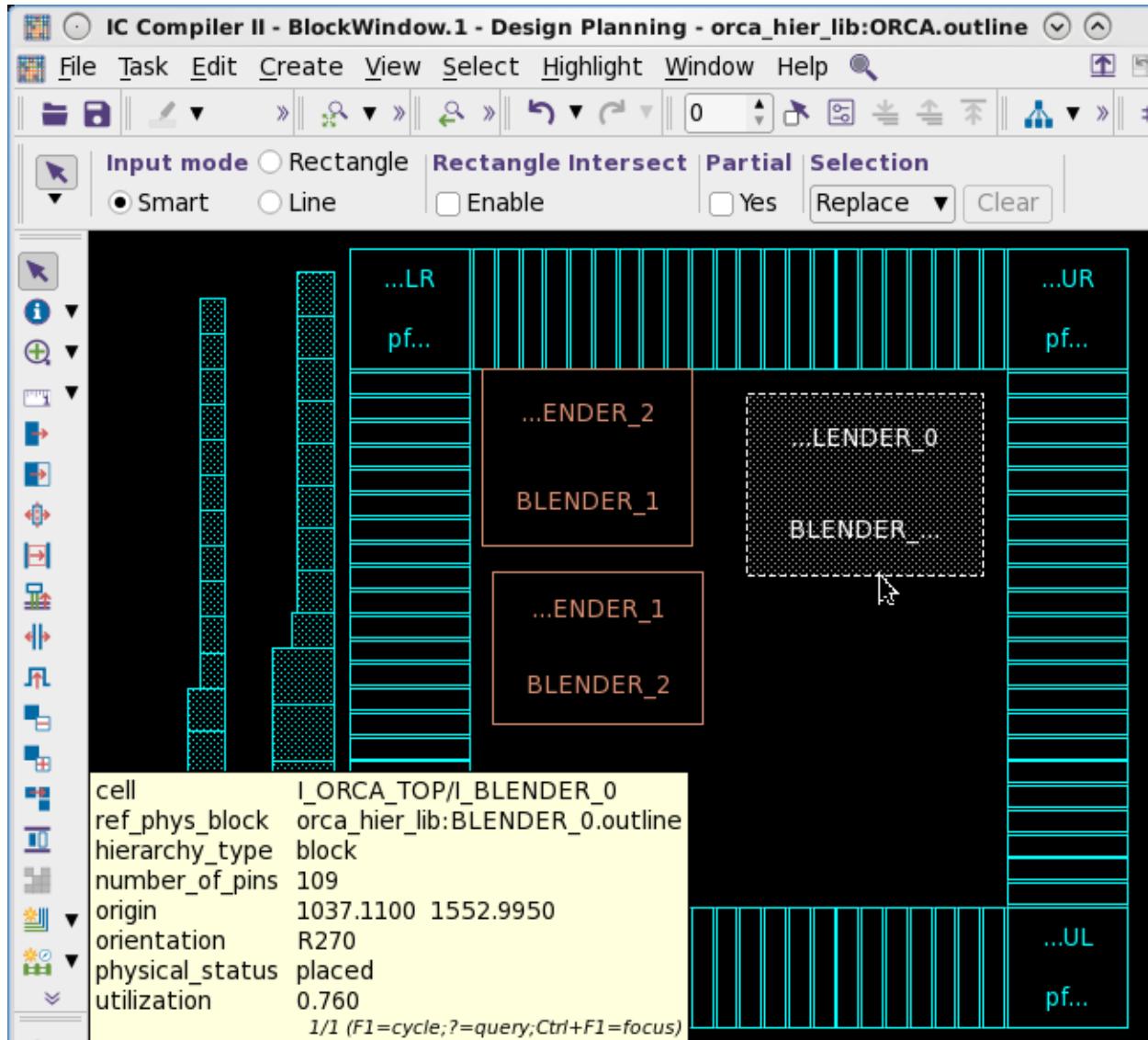
Creating Black Box References

After identifying the modules which to treat as black boxes, use the `create_blackbox` command to create a black box physical hierarchy for the cell. The command also updates the reference for the specified cell. Note that you can convert empty, partial, or missing modules to black boxes after reading in the netlist, or after committing the blocks.

The following example creates a black box in the current library for instance u0_3 and sets a target boundary area of 2000000. The layout window after creating the black box is shown in [Figure 19](#):

```
icc2_shell> create_blackbox -target_boundary_area 2000000 u0_3  
0
```

Figure 19 Black Box in the GUI



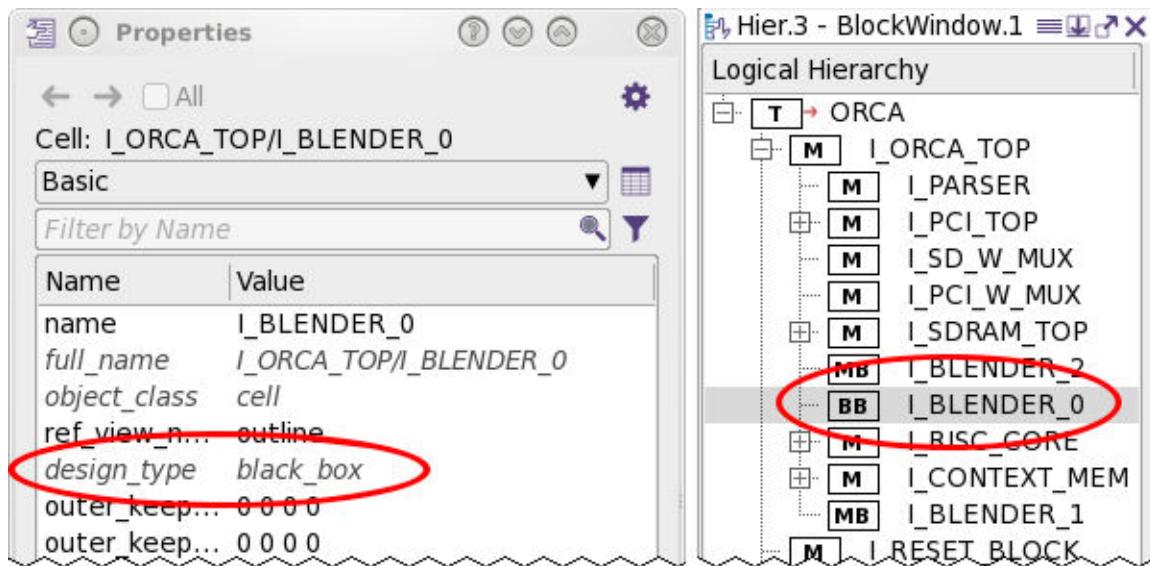
To find black boxes in the design created with the `create_blackbox` command, or to check whether a specific module is a black box, check the `design_type` attribute of the module as follows:

```
icc2_shell> get_cells -hierarchical \
    -filter ref_block.design_type==black_box
{I_ORCA_TOP/I_BLENDER_0}

icc2_shell> get_attribute [get_cells I_ORCA_TOP/I_BLENDER_0] design_type
black_box
```

The GUI provides annotations in the Hierarchy Browser and Properties Editor to help identify black boxes as shown in [Figure 20](#).

Figure 20 Black Box Display in Hierarchy Browser and Properties Editor



To modify the area set by the `create_blackbox -target_boundary_area area` command, set the `target_boundary_area` attribute directly on the black box cell with the following command:

```
icc2_shell> set_attribute -objects [get_cells I_ORCA_TOP/I_BLENDER_0] \
    -name target_boundary_area -value 1500000
{u0_3}
```

For black boxes created from a partial Verilog netlist, you can specify a target utilization with the following command:

```
icc2_shell> set_attribute -objects [get_cells I_ORCA_TOP/I_BLENDER_0] \
    -name target_utilization -value 0.5
{u0_3}
```

Both the `target_boundary_area` and `target_utilization` attributes are honored by the `shape_blocks` command during block shaping.

Creating a Black Box Timing Model

A black box contains no timing information by default. Any path that begins or ends at a black box is unconstrained. The IC Compiler II tool provides commands to create a timing model for a black box, which allows the module to be used for timing budgeting and optimization.

To create a black box timing model,

1. Define the clock ports, drive and load.

```
icc2_shell> set_blackbox_clock_port ...
icc2_shell> set_blackbox_port_drive ...
icc2_shell> set_blackbox_port_load ...
```

2. Set the clock network delay, pin-to-pin delays, setup and hold constraints.

```
icc2_shell> create_blackbox_clock_network_delay ...
icc2_shell> create_blackbox_constraint ...
icc2_shell> create_blackbox_delay ...
icc2_shell> create_blackbox_drive_type ...
icc2_shell> create_blackbox_load_type ...
```

3. Commit the timing model to the black box.

```
icc2_shell> commit_blackbox_timing
1
```

You create 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.

Black box timing information is stored in the design view. When an abstract is created from a black box, specially marked cells, nets, and connections are created on-the-fly within the abstract view. This approach enables timing on an empty or partial netlist and allows newly created feedthroughs to be timed immediately without recreating the models.

For a partial black box, only the unconnected ports can be constrained with black box timing constraints. Other ports on the partial black box should be constrained by using Synopsys Design Constraints (SDC) files.

When generating timing budgets, the tool maintains the timing you specified with the black box timing commands. To allow the budgeter to adjust the black box timing budgets, set the `plan.budget.bbt_fixed_delay` application option as follows:

```
icc2_shell> set_app_options -name plan.budget.bbt_fixed_delay \
    -value false
plan.budget.bbt_fixed_delay false
```

Black Box Timing Example

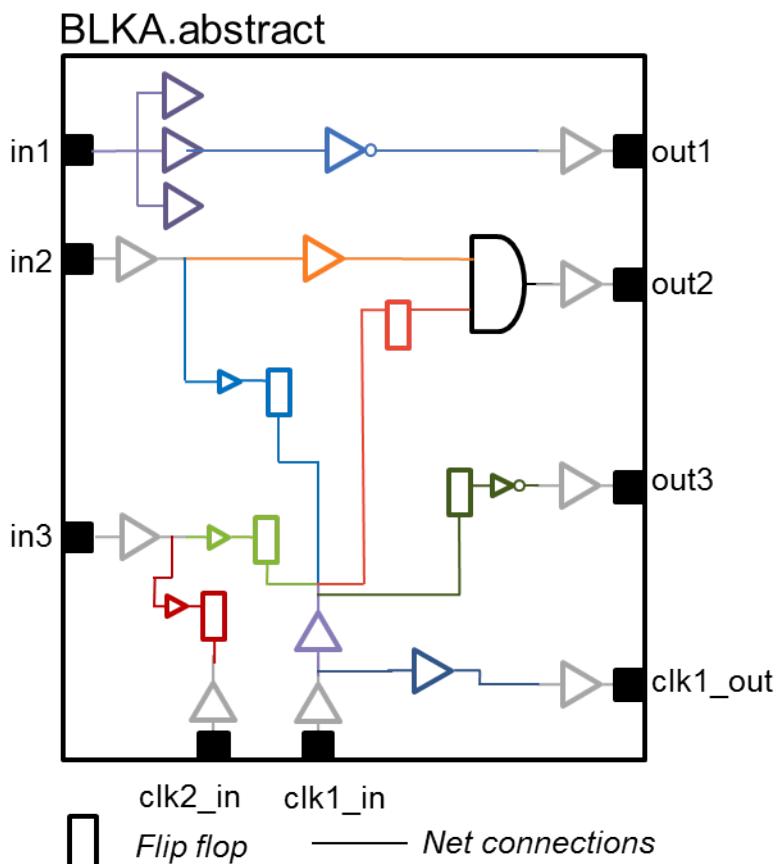
The following example shows the commands used to create a black box timing model. The resulting model is shown in [Figure 21](#).

```
set_blackbox_clock_port [get_ports {clk1_in clk1_out clk2_in}]
create_blackbox_load_type -lib_cell buf1_lt1
set_blackbox_port_load -type lt1 -factor 3 [get_ports in1]
```

```

create_blackbox_clock_network_delay -value 3.5 clk_1
create_blackbox_delay -rise_from in1 -fall_to out1 -value 5.0
create_blackbox_delay -fall_from in1 -rise_to out1 -value 5.2
create_blackbox_delay -from in2 -to out2 -clock main_clock \
    -value 0.4
create_blackbox_constraint -setup -edge rise -from clk2_in \
    -to in3 -value 1.2
create_blackbox_constraint -hold -edge rise -from clk2_in \
    -to in3 -value 0.8
create_blackbox_delay -from clk1_in -to clk1_out -value 1.1
create_blackbox_constraint -edge rise -from clk1_in -to in3 \
    -clock main_clock -value 0.5
create_blackbox_constraint -edge rise -from clk1_in -to in2 \
    -value 1.3
create_blackbox_delay -rise_from clk1_in -to out2 -value 2.2
create_blackbox_delay -fall_from clk1_in -to out3 -value 1.7
commit_blackbox_timing
    
```

Figure 21 Example Black Box Timing Model



UPF for Black Box Designs

The UPF constraints for black boxes are specified differently from the UPF constraints for normal blocks, because black boxes act as leaf cells and the black box UPF constraints describe the context in terms of the top level. The black box UPF also defines the receiver supply information to the block input ports.

The UPF for black boxes can be written specifically for the black box or extracted from an existing design with the `save_upf -for_empty_blackbox` command and option as shown in the following example.

```
create_blackbox I_ORCA_TOP/I_BLENDER
current_design BLENDER
load_upf inputs/BLENDER.upf
commit_upf
save_upf -for_empty_blackbox ./split/BLENDER/top.upf
```

You would not typically run the `split_constraints` command to generate UPF constraints for black boxes or empty modules. If a net does not have a real driver, the primary supply for the domain is used. Black boxes are the only hierarchical cells where the `set_port_attributes` command or the `set_related_supply_net` (SRSN) command can be used to override the supply information.

You should create abstracts for black boxes in your design, as this creates black box timing cells. In addition to providing drive and load information for outputs and inputs, these cells also provide drivers and receivers on the net and enable multivoltage-related commands such as `check_mv_design` to perform as expected.

6

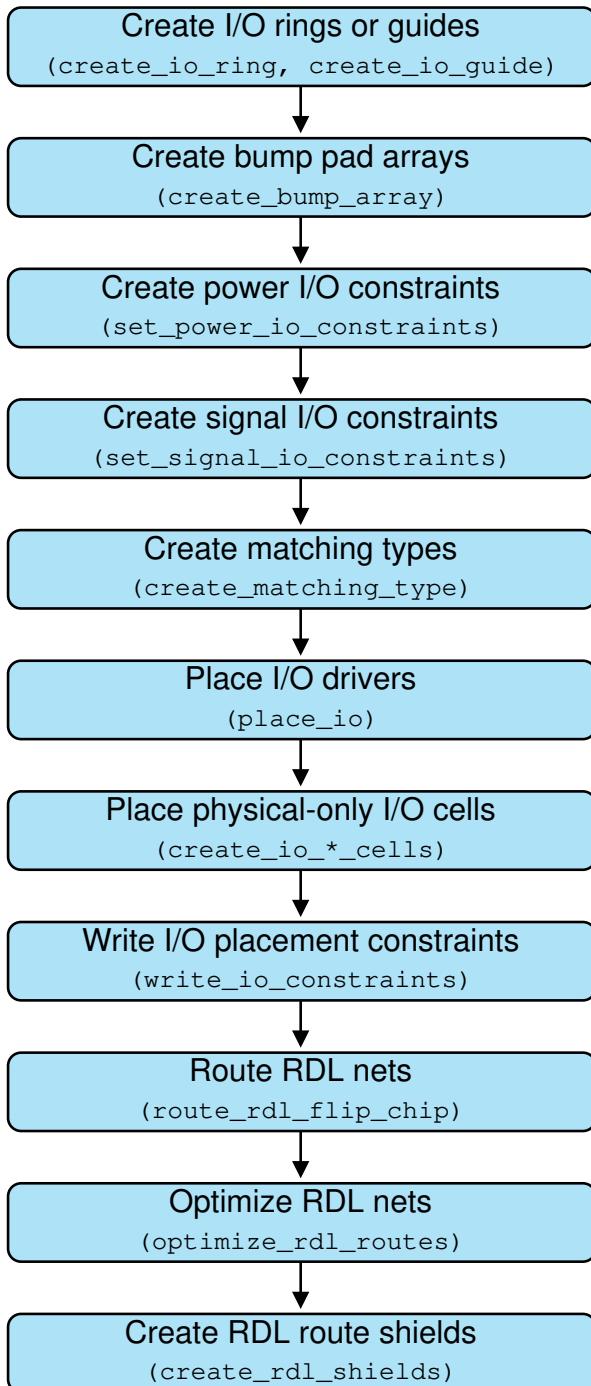
Planning I/Os and Flip-Chip Bumps

After creating the floorplan, you can instantiate the I/O drivers and bump cells for your design. The IC Compiler II tool supports advanced features for unconstrained and constraint-based placement of I/O drivers and flip-chip bump cells. You can create I/O placement constraints and specify the ordering, xy coordinate, and placement side for each I/O. The tool supports both package-driven and die-driven I/O placement flows, and you can adapt the tool for different chip packaging scenarios. Different I/O driver styles, such as pads with integrated drivers, are supported by the tool.

Chip I/Os can be placed along the chip periphery or in islands within the chip core. Constraints include signal constraints and power constraints to specify I/O adjacency requirements and power requirements. You can perform I/O placement by using Tcl commands and GUI interfaces.

The flow for instantiating I/O drivers and pads is shown in [Figure 22](#).

Figure 22 I/O Planning Flow



For more details, see the following topics:

- [Validating Library Requirements](#)
- [Creating I/O Rings or Guides](#)
- [Creating Arrays of Bump Cells](#)
- [Creating Rows or Columns of Bump Cells](#)
- [Placing Bump Cells in Predetermined Locations](#)
- [Creating Power I/O Placement Constraints](#)
- [Creating Signal I/O Placement Constraints](#)
- [Assigning I/O Pads to Bumps](#)
- [Placing I/Os and Writing Constraints](#)
- [Adding and Placing Physical-Only I/O Cells](#)
- [Checking I/O Placement](#)
- [Routing RDL Nets](#)
- [Optimizing RDL Routes](#)
- [Creating RDL Net Shields](#)
- [Creating a Flip-Chip Design With Multiple Levels of Hierarchy](#)

Validating Library Requirements

Before beginning a flip-chip design, you must ensure that the reference library that contains flip-chip component cells is properly configured for the tool. Specifically, the `design_type` attribute should be checked for the following flip-chip reference cells by using the IC Compiler II Library Manager tool or the IC Compiler II tool:

- Bump cells: the `design_type` attribute must be set to `flip_chip_pad`.
- Driver cells: the `design_type` attribute must be set to `flip_chip_driver` or `pad`.

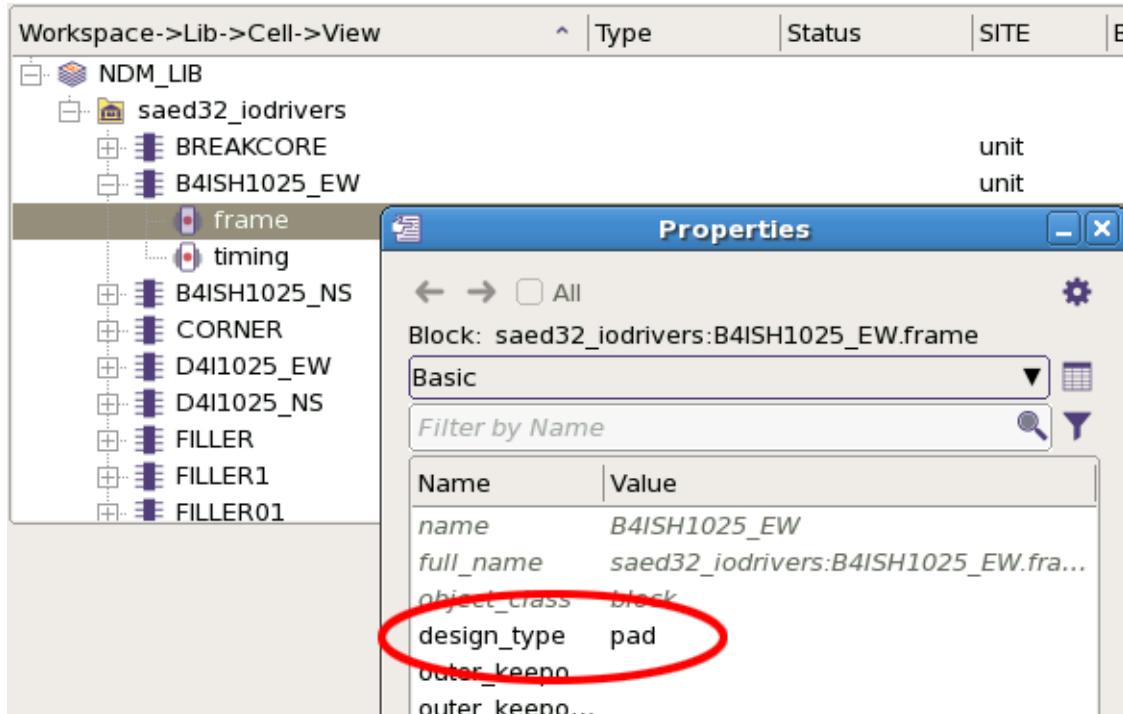
You can also set the `design_type` attribute to `macro` and use the driver cell for I/O placement if the flip-chip matching types are defined.

The terminal or pin shape of the driver cell to be connected to an RDL route should have the `class` attribute set to `bump`. If there is no terminal with this setting, the RDL router chooses the topmost layer terminal of the port.

- Filler and break cells: the `design_type` attribute must be set to `pad_spacer`.
- Corner cells: the `design_type` attribute must be set to `corner`.

Property types and attribute settings on the frame view for flip-chip library cells can be examined in the IC Compiler II Library Manager as shown in the following figure.

Figure 23 *design_type* Property for Driver Cell



Note that the `design_type` attribute setting can be derived from the LEF file for the library cell. A value of `pad` is created based on the "PAD" macro class in the LEF file. A value of `flip_chip_pad` is created based on the "COVER BUMP" macro class in the LEF file.

Some attributes of a library reference cell can be overridden by using the `set_attribute` command in the IC Compiler II tool. Attribute settings performed in the IC Compiler II tool are not propagated to the reference library.

For macros, the default `design_type` attribute setting is `macro` and bump assignment is not performed. To perform bump assignment for macros, you must create matching types that include macros, pins of macros, or terminals of macro pins. Matching any part of the

macro is sufficient for the `place_io` command to identify a cell instance. To identify the pins that are assigned to bumps, do one of the following:

- Explicitly include the pin in the matching type. This is the recommended methodology.
- Set the `class` attribute to `bump` for any terminal pin during library preparation, or use the `set_attribute` command in the IC Compiler II tool.

Creating I/O Rings or Guides

The IC Compiler II tool supports I/O guides, which form a placement area for I/O drivers. You can create an I/O ring by placing four I/O guides along the periphery of the floorplan with four `create_io_guide` commands, or by using one `create_io_ring` command. The IC Compiler II tool also supports multiple I/O rings and supports multiple I/O guides on the same side of the die.

I/O guides can be placed anywhere in the hierarchy, if the following are honored:

- All flip-chip bump cells must be placed at the top level.
- I/O guides are not allowed to cross block boundaries for a placed block.
- I/O pad cells and the associated I/O guide must be placed within the same physical block instance.
- Only pads in the same logical hierarchy as the I/O guide can be constrained with the `set_signal_io_constraints` command.
- I/O guides must not overlap.

You must create I/O guides or I/O rings before placing I/O cells. The following commands create a ring containing four I/O guides around the periphery of the floorplan and retrieve the names of the I/O guides created by the `create_io_ring` command.

```
icc2_shell> create_io_ring -name outer_ring -corner_height 300
{outer_ring}
icc2_shell> get_io_guides
{outer_ring.left outer_ring.bottom outer_ring.right outer_ring.top}
```

You can also use four `create_io_guide` commands to create I/O guides on the left, top, right, and bottom sides of the die.

```
icc2_shell> create_io_guide -name "ring_left" -line {{0 300} 3000} \
    -side left
{ring_left}
icc2_shell> create_io_guide -name "ring_top" -line {{300 3640} 3000} \
    -side top
{ring_top}
icc2_shell> create_io_guide -name "ring_right" -line {{3680 3380} 3000} \
```

```
-side right
{ring_right}
icc2_shell> create_io_guide -name "ring_bottom" -line {{3380 0} 3000} \
    -side bottom
{ring_bottom}
icc2_shell> get_io_guides
{ring_left ring_top ring_right ring_bottom}
```

Alternatively, you can use the Floorplan Task Assistant to create the I/O rings and I/O guides in the GUI. Select Task > Task Assistant, then select Floorplan Preparation > I/O Planning > I/O Ring and Guide. Enter the information for the I/O ring or I/O guide and click Apply to create the ring or guide.

I/O Ring and I/O Guide Tasks

Use options with the `create_io_ring` and `create_io_guide` commands to control how the ring is created:

- Assign a name to the ring with the `-name` option. The name is used as a prefix for the I/O guide names.

```
icc2_shell> create_io_ring -name outer_ring
{outer_ring}
icc2_shell> get_io_rings
{outer_ring}
icc2_shell> get_io_guides
{outer_ring.left outer_ring.bottom outer_ring.right outer_ring.top}
```

- Create a ring inside an existing ring with the `-inside` option. The existing ring is used as the outer boundary for the new ring.

```
icc2_shell> create_io_ring -name inner -inside outer_ring
```

- Specify the distance between the inner and outer ring boundaries with the `-offset` option.

```
icc2_shell> create_io_ring -name inner -inside outer -offset 500
```

- Create a ring within a specific bounding box with the `-bbox` option.

```
icc2_shell> create_io_ring -bbox {{1000 1000} {2000 2000}}
```

- Specify a list of pad cell instances to assign to the ring with the `-pad_cell_list` option.

```
icc2_shell> create_io_ring -pad_cell_list {pad_iopad_0 pad_iopad_1}
```

- Include specific I/O guides in the ring with the `-guides` option.

```
icc2_shell> get_io_guides
{ring_left ring_top ring_right ring_bottom}
```

```
icc2_shell> create_io_ring -guides [get_io_guides]
{_default_io_ring1}
```

- Specify the minimum distance between the startpoint of the I/O guide and the first I/O driver, and the minimum distance between the endpoint of the I/O guide and the last I/O driver with the `-offset` option of the `create_io_guide` command.

Editing, Reporting, and Removing I/O Guides and I/O Rings

- Add I/O pad cells to an I/O guide with the `add_to_io_guide` command.

```
icc2_shell> add_to_io_guide guide_left {pad_iopad_0 pad_iopad_1}
```

- Create a collection of I/O guides with the `get_io_guides` command and create a collection of I/O rings with the `get_io_rings` command.

```
icc2_shell> get_io_guides
{guide_left guide_top guide_right guide_bottom}
icc2_shell> get_io_rings
{ring_outer}
```

- Remove I/O pad cells from an I/O guide with the `remove_from_io_guide` command.

```
icc2_shell> remove_from_io_guide guide_left {pad_iopad_0 pad_iopad_1}
total 2 cells removed from io guide
```

- Remove an I/O guide from an I/O ring with the `remove_from_io_ring` command.

```
icc2_shell> remove_from_io_ring ring_outer {guide_left guide_top}
```

- Report I/O rings or I/O guides in the block with the `report_io_rings` and `report_io_guides` commands.

- Remove I/O rings or I/O guides from the block with the `remove_io_rings` and `remove_io_guides` commands.

Creating Arrays of Bump Cells

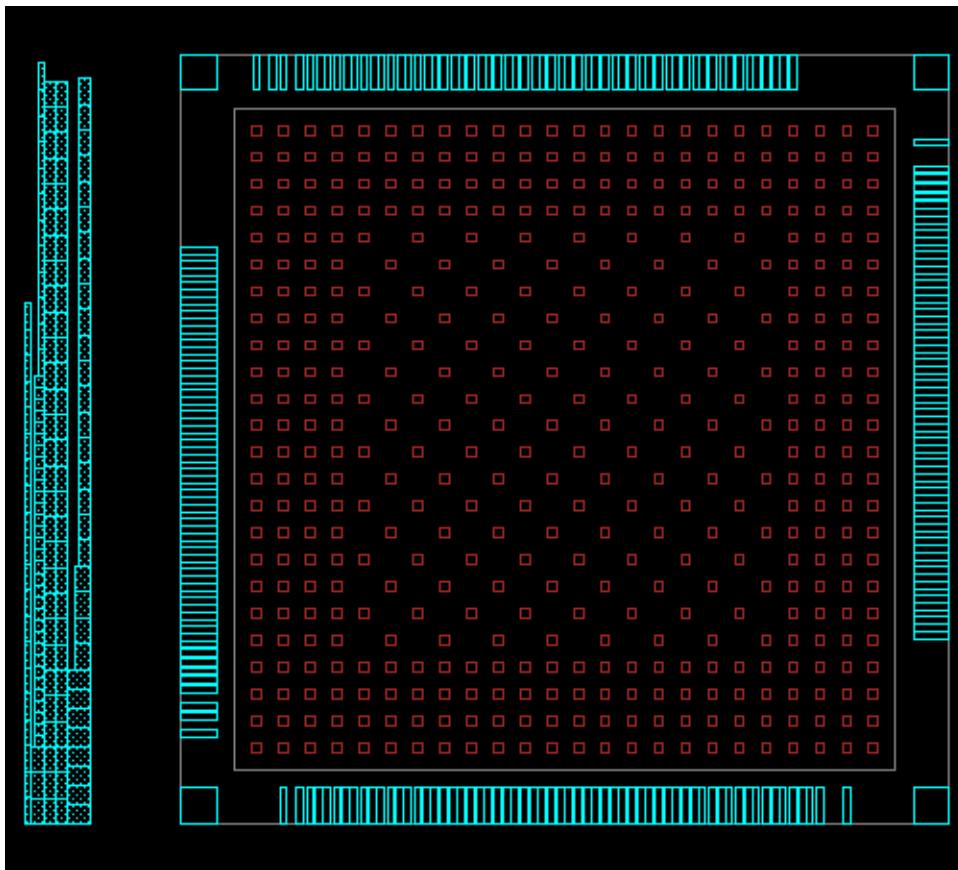
The IC Compiler II tool supports bump cell placement for flip-chip designs. You can place bump cells based on placement coordinates in a DEF or AIF file, or by creating a bump array with the `create_bump_array` command. You should place bump cells before placing I/O driver cells. In the die-driven flow, you create a region for bump cell placement and let the tool determine the locations for the bump cells. The following example creates five bump array regions, one each for the left, top, right, bottom, and center die areas, and fills each region with BUMP library cells.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {140 140} \
    -bbox {{410 450} {1175 3278}} -name left
icc2_shell> create_bump_array -lib_cell BUMP -delta {140 140} \
    -bbox {{2620 450} {3325 3278}} -name right
```

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {140 140} \
    -bbox {{1100 450} {2650 1125}} -name bottom
icc2_shell> create_bump_array -lib_cell BUMP -delta {140 140} \
    -bbox {{1100 2555} {2650 3250}} -name top
icc2_shell> create_bump_array -bbox {{1100 1150} {2650 2550}} \
    -pattern staggered_1 -lib_cell {BUMP} -delta {140 140} -name pg_bump
```

[Figure 24](#) shows the layout after running the previous commands.

Figure 24 Layout After create_bump_array



Bump Array Options

Use options with the `create_bump_array` command to control how the bump array is created:

- Specify the horizontal and vertical spacing between adjacent rows with the `-delta` option; `-delta` and `-lib_cell` are required options.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {150 150}
```

- Create a name for the bump array with the `-name` option.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {150 150} \
    -name pg_bump
```

- Specify a rectangular bounding box to contain the bump cells with the `-bbox` option.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {150 150} \
    -bbox {{1000 1000} {3000 3000}}
```

- Create the bump array within a complex rectilinear shape with the `-boundary` option.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {150 150} \
    -boundary {{1000 1000} {3000 1000} {3000 3000} {1000 3000}}
```

- Limit the bump array size by number of columns or number of rows with the `-repeat` option. Bumps are centered in the die or bounding box.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {150 150} \
    -repeat {5 5}
```

- Set the spacing between the lower-left corner of the bump array bounding box and the lower-left corner of the bump cell instance at column 0 and row 0 with the `-origin` option.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {150 150} \
    -origin {1000 1000}
```

- Specify the orientation of each bump cell in the bump array with the `-orientation` option.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {150 150} \
    -orientation N
```

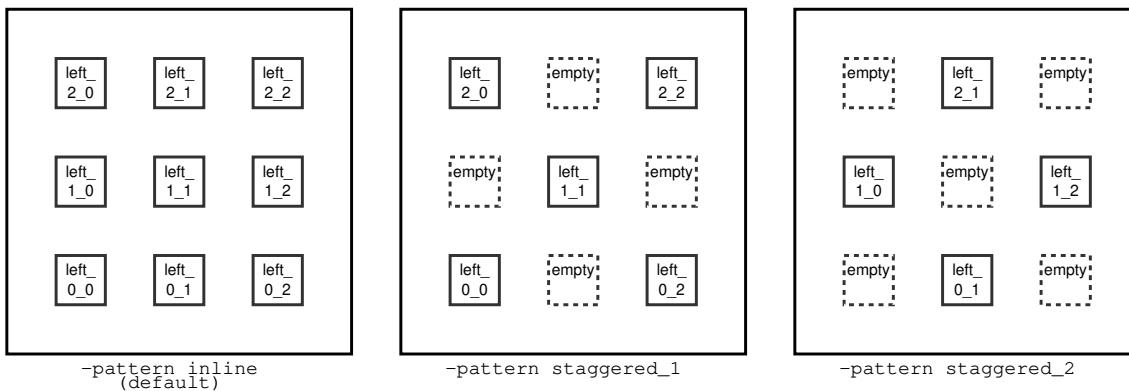
Legal orientation values are N, W, S, E, FN, FS, FE, and FW.

- Specify whether the bump cells occupy each possible location or alternate locations with the `-pattern inline | staggered_1 | staggered_2` option.

```
icc2_shell> create_bump_array -lib_cell BUMP -delta {150 150} \
    -pattern staggered_1
```

The `inline` keyword places bump cells at each point in the array specified by the `-delta` option. The `staggered_1` keyword places bump cells at points where the sum of the column index and the row index is even. The `staggered_2` keyword places bump cells at points where the sum of the column index and the row index is odd. [Figure 25](#) shows a small bump placement example that uses these three placement patterns.

Figure 25 Bump Array Placement Pattern Options



Creating Rows or Columns of Bump Cells

In designs where I/O pad cells are not placed by using I/O guides, you can use the `create_bump_pattern` command to instantiate bump cells and specify bump spacing, matching types, and other information. This command is useful for designs that contain clusters of I/O cells and require that the bump cells be placed in rows or columns perpendicular to the I/O pad cells.

To create a bump pattern for a set of pad cells, create the bump pattern file as described in this section and load the file with the `create_bump_pattern` command as follows:

```
icc2_shell> create_bump_pattern -file_name bump_patterns.txt
12 bumps and 5 matching types are created successfully
```

The following bump pattern file is used in this example; the layout result is shown in Figure 26.

```
# Bump pattern file for area I/O pad cluster
bump_lib_cell BUMP_PAD // Bump library cell
bump_pitch 150          // (A) Bump pitch

io_group {               // I/O group description
    {{bus_3} {bus_2} {bus_1} {bus_0 s}
     {vss_g0_} {vcc_main p_} {vss_io g1_} {vcc_io p1_}
     {bus_7} {bus_6} {bus_5} {bus_4}}
    {orientation row} // Valid orientations are row, column,
                       // or perpendicular
    {-50} // (B) Offset between center line of bumps
           // and pad bbox center
    {100} // (C) Spacing between the left or bottom
           // of pad bbox and nearest bump cell center
    {s s p_ g0_} // Left or bottom bumps
    {75} // (D) Spacing between the right/top
          // of pad bbox and nearest bump cell center
```

```
{ s p1_ g1_ // Right or top bumps
  s s s s s}
}
}
```

The pattern file specified by the `-file_name` option defines the bump library cell, bump placement pitch, spacing, matching types, and other information used to create the bump pattern. Commented lines begin with a pound sign (#), and inline comments begin with two forward slashes (//). The pattern file uses the following format:

- `bump_lib_cell`: The bump library cell; specify the bump library cell as *library_cell* or *library_name/library_cell*
- `bump_pitch`: The bump pitch in microns
- `io_group`: The I/O group description, surrounded by curly braces ({}), contains the following information:
 - I/O pad instance names: The list of instance names specified within curly braces. Each instance name is specified within its own curly braces together with an optional matching type prefix. If no matching type prefix is specified, the “s” matching type prefix is used. Bump instance names are based on the I/O pad name of the first instance.

In this example, the `{vss g0_}` item creates a bump cell with an instance name of `bus_3_n` and a matching type named `g0_bus_3`. “`g0_`” is the matching type prefix. “`bus_3`” is the first I/O pad specified and this string is used for all bump cell instances created with this pattern file.

- `orientation`: (Optional) the orientation specification, either `row`, `column`, or `perpendicular`.

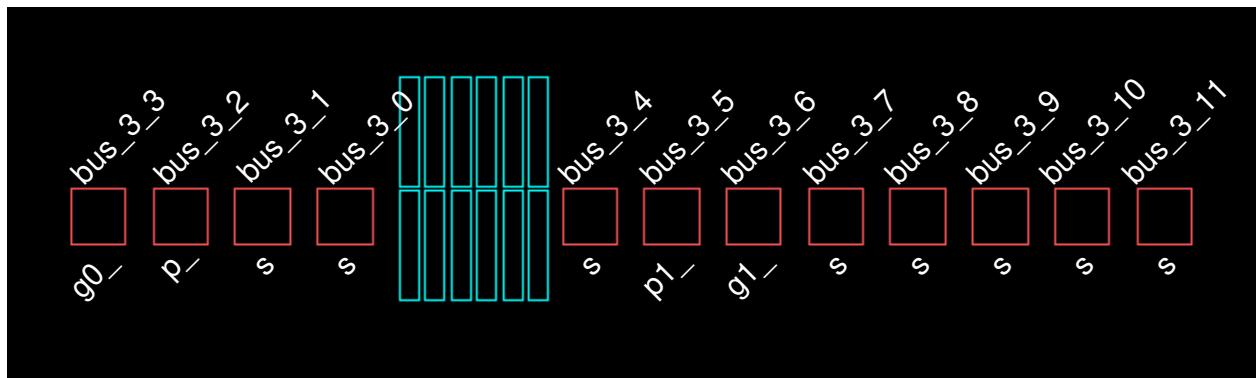
If the `orientation` keyword is omitted, the command places the bump cells based on the height and width of the I/O pad bounding box. If the width is greater than the height, the tool places the bump cells in columns. If the height is greater than the width, the command places the bump cells in rows.

- Pad ordering and spacing: The ordering and spacing list is specified within curly braces; a pattern file can contain multiple ordering and spacing sections. The specification is as follows:
 - Center line offset: The distance in microns between the center line of new bump cells the center line of the bounding box defined by the I/O pads
 - Left or bottom first bump spacing: The distance between the left or bottom edge of the I/O pad bounding box and the nearest bump cell center
 - Left or bottom bump cells: The list of matching type prefixes corresponding to the matching type prefixes defined in the list of I/O pad instances

- Right or top first bump spacing: The distance between the right or top edge of the I/O pad bounding box and the nearest bump cell center
- Right or top bump cells: The list of matching type prefixes corresponding to the matching type prefixes defined in the list of I/O pad instances
- orientation: (Optional) the orientation specification for this group, either `row`, `column`, or `perpendicular`.

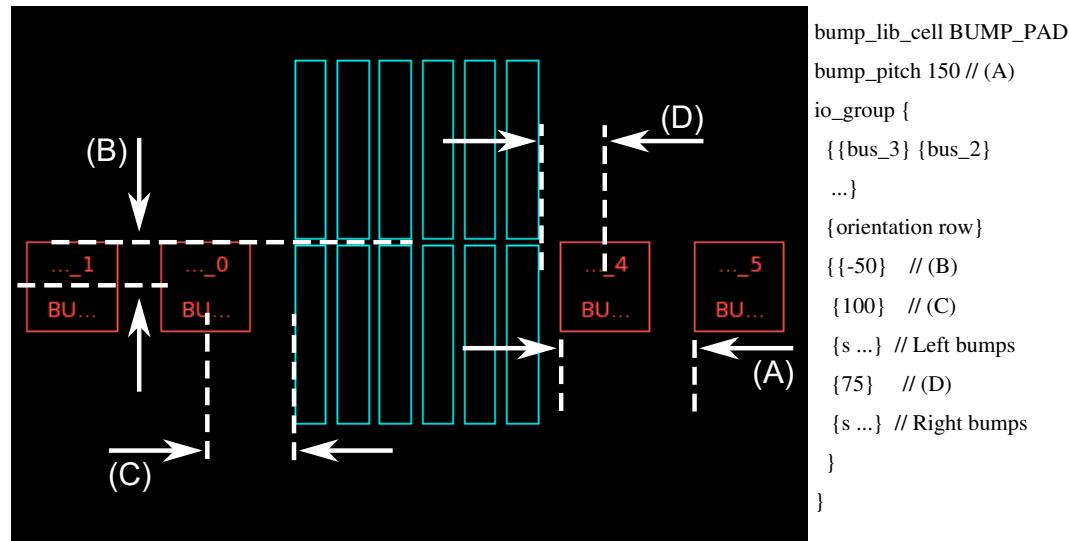
The following figure shows the layout result created by the example bump pattern file.

Figure 26 Bump Cell Instances



The following figure shows how the spacing values in the pattern file correspond to the bump cell and I/O pad cell spacings in the layout.

Figure 27 Bump Cell Offsets and Spacings



The output of the `report_matching_types` command shows the matching types generated by the preceding pattern file. Note that the Uniquify and Object Type columns are removed from this example.

Matching Type	Object Name	Matching Type	Object Name
sbus_3	bus_0	g0_bus_3	bus_3_3
	bus_1		vss
	bus_2	g1_bus_3	bus_3_6
	bus_3		vss_io
	bus_3_0	p1_bus_3	bus_3_5
	bus_3_1		vcc_io
	bus_3_10	p_bus_3	bus_3_2
	bus_3_11		vcc_main
	bus_3_4		
	bus_3_7		
	bus_3_8		
	bus_3_9		
	bus_4		
	bus_5		
	bus_6		
	bus_7		

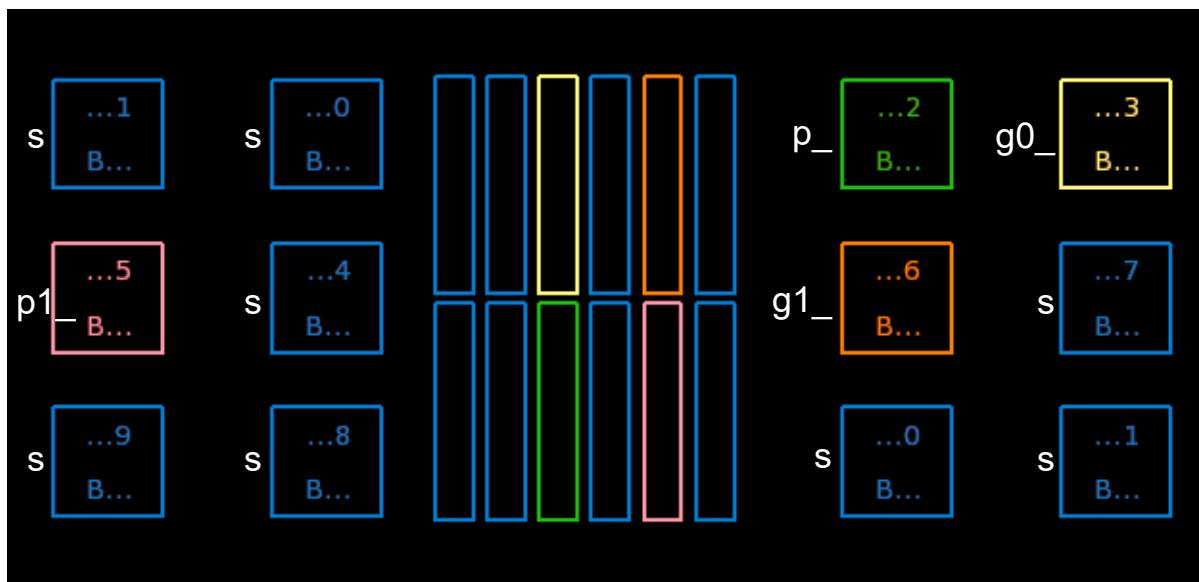
You can create more complex bump patterns by including multiple groups of row or column specifications. The following bump pattern file creates three rows of bump cells by including three groups of offsets and matching types. Each row contains two bumps to the left of the I/O drivers and two bumps to the right of the I/O drivers.

```
# Pattern file for three bump rows
bump_lib_cell BUMP_PAD // bump lib cell
bump_pitch 200 // pitch

io_group { // I/O group description
    {{bus_3} {bus_2} {bus_1} {bus_0 s}
     {vss g0_} {vcc_main p_} {vss_io g1_} {vcc_io p1_}
     {bus_7} {bus_6} {bus_5} {bus_4}}
    {orientation row}
    {{150} {100} {s s} {150} {p_ g0_}} // top row
    {{0} {100} {s p1_} {150} {g1_ s}} // middle row
    {{-150} {100} {s s} {150} {s s}} // bottom row
}
```

The layout result is shown in the following figure with cells highlighted by matching types.

Figure 28 Bump Pattern With Three Rows



Placing Bump Cells in Predetermined Locations

If the locations of the bump cells are determined by the constraints of the package, use the `read_aif` command to read in a list of bump cells and locations. The location information is typically read from a DEF or AIF file, which contains x- and y-coordinates and bump cell

names for the bump cells in the block. Use the `-hierarchy` option to read in bump cell and I/O pad locations and place the I/O pads in any level of physical hierarchy.

```
icc2_shell> read_aif bumps.aif
```

A short example from an AIF file is as follows:

```
[DATABASE]
TYPE=AIF
VERSION=2.0
UNITS=UM
[DIE]
NAME=mydesign_die
WIDTH=3840.575700
HEIGHT=3690.006000
[PADS]
BUMP_N = POLY -17.000 46.500 -17.000 46.475 ...
[NETLIST]
;Netname Pad#      Type     Pad_X        Pad_Y        Ball#
- left_0_0          BUMP_N   -1463.7878   -1348.5030
- left_0_1          BUMP_N   -1323.7878   -1348.5030
...
- right_0_0         BUMP_N   746.2122    -1348.5030
- right_0_1         BUMP_N   886.2122    -1348.5030
...
- bottom_0_0        BUMP_N   -773.7878   -1348.5030
- bottom_0_1        BUMP_N   -633.7878   -1348.5030
...
- top_0_0           BUMP_N   -773.7878   756.4970
- top_0_1           BUMP_N   -633.7878   756.4970
...
- pg_bump_0_0       BUMP_N   -773.7878   -648.5030
- pg_bump_0_2       BUMP_N   -493.7878   -648.5030
...
```

Creating Power I/O Placement Constraints

Power I/O placement constraints specify spacing and other requirements for power I/O drivers. Use the `set_power_io_constraints` command to set constraints for the placement of I/O driver pads and use the `place_io` command to place the drivers into the I/O guides. The following example sets placement constraints on power I/O drivers.

```
icc2_shell> set_power_io_constraints \
    -io_guide_object [get_io_guides "*left *right"] \
    {{reference: VDD_EW}
     {ratio: 3}
     {prefix: VDD}
     {connect: {VDDIO VDDIO} {VDD VDD1} {VSS VSS1}}}
icc2_shell> set_power_io_constraints \
    -io_guide_object [get_io_guides "*left *right"] \
```

```
{reference: VSS_EW}
{ratio: 7}
{prefix: VSS}
{connect: {VSSIO VSSIO} {VDD VDD1} {VSS VSS1}}}
Power IO constraints set successfully
1
```

In this example, the first `set_power_io_constraints` command specifies the following constraint:

- The constraint applies only to the left and right I/O guides.
- The power pad cell VDD_EW is inserted based on the constraint.
- The number of signal pads between successive VDD_EW pads is three.
- The string “VDD” is inserted into the instance name.
- The VDDIO pad cell pin is connected to the VDDIO net, the VDD pad cell pin is connected to the VDD1 net, and the VSS pad cell pin is connected to the VSS1 net.

The second `set_power_io_constraints` command creates a similar constraint for the VSS_EW cell references, but specifies that seven signal I/O pads might be inserted between successive VSS_EW cells.

[Figure 29](#) shows a small section of the lower-left corner of the layout after running the `set_power_io_constraints` and `place_io` commands.

Figure 29 Layout With Power I/O Constraints



If you do not specify constraints with the `set_power_io_constraints` command, the `place_io` command places the power pads evenly throughout the I/O guide.

Power I/O Placement Constraint Options

Use options with the `set_power_io_constraints` command to further control power signal I/O placement.

- Assign more than one power pad to a given bump cell with the `-share` option.

```
icc2_shell> set_power_io_constraints \
             -reference_cell {VDD_NS VSS_NS} -share {{2 VDD_NS} {3 VSS_NS}}
```

In this example, `-share {{2 VDD_NS} {3 VSS_NS}}` allows up to two VDD power pads to drive a single VDD bump cell, and three VSS power pads to drive a single VSS bump cell.

- Define the number of signal pads that can be placed between successive power pads with the `-ratio` option.

```
icc2_shell> set_power_io_constraints \
             -reference_cell {VDD_NS VSS_NS} -ratio {{7 VDD_NS} {6 VSS_NS}}
```

In this example, the tool places no more than seven I/O pads between VDD_NS pad cells, and no more than six I/O pads between VSS_NS pad cells.

- Specify the maximum spacing between successive power pads of the same type with the `-spacing` option.

```
icc2_shell> set_power_io_constraints -reference_cell {VDD_NS VSS_NS} \
    -spacing {{100 VDD_NS} {100 VSS_NS}}
```

- Specify the maximum distance between the starting point of the I/O guide and the closest edge of the first power pad of the specified type with the `-offset` option.

```
icc2_shell> set_power_io_constraints -reference_cell {VDD_NS VSS_NS} \
    -offset {{100 VDD_NS} {100 VSS_NS}}
```

Creating Signal I/O Placement Constraints

To create signal I/O placement constraints to specify the ordering and spacing for signal I/O drivers, use the `set_signal_io_constraints` command. This command can be used to specify constraints for I/O guides at the top level or specify constraints for I/O guides within lower levels of the design hierarchy.

In the following example, the `set_signal_io_constraints` command specifies an order-only constraint for the placement of three signal I/O drivers in the `_default_io_ring1.left` I/O guide: `u_p/clk_pad_P1`, `u_p/sdclk_pad_P1`, and `u_p/test_so_4_pad_P1`.

```
icc2_shell> set_signal_io_constraints -constraint {{order_only} \
    u_p/clk_pad_P1 u_p/sdclk_pad_P1 u_p/test_so_4_pad_P1} \
    -io_guide_object {_default_io_ring1.left}
Signal IO constraints set successfully
1
```

The `-constraint {constraint_spec}` option supports different formats for specifying the placement of signal I/O drivers. Use the `-constraint {{order-only} pad_1 pad_2 ...}` option to place the pads in the specified order. Note that additional spacing and other pad cells might be inserted between adjacent pads. To create a fixed space between I/O drivers, insert a spacing value in microns between the signal names. For example, the `-constraint {pad_1 10 pad_2 20 pad_3}` option inserts a 10 micron space between pads `pad_1` and `pad_2`, and a 20 micron space between pads `pad_2` and `pad_3`. To place each signal I/O driver at a specified pitch, use the `-constraint {{pitch} pad_1 pad_2 ...}` option format.

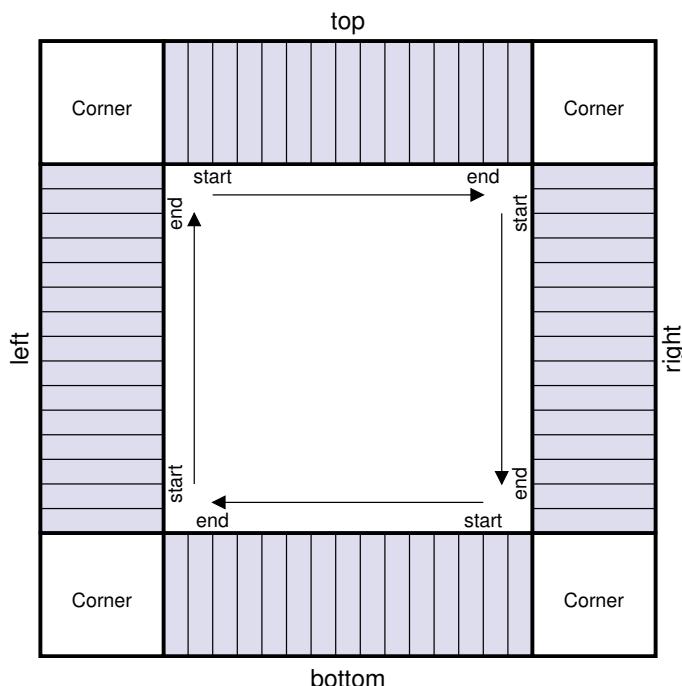
Alternatively, you can write out the signal I/O placement constraints with the `write_io_constraints -filename constraint_file_name` command and use the `set_signal_io_constraints -file constraint_file_name` command to load the constraints. The constraint file is useful when there is a large number of signal constraints.

to specify. The following example loads signal I/O placement constraints from the `signal_io_constraints.txt` constraints file.

```
icc2_shell> set_signal_io_constraints -file signal_io_constraints.txt
Signal IO constraints set successfully
1
icc2_shell> sh cat signal_io_constraints.txt
_default_io_ring1.left
{{order_only}
u_p/clk_pad_P1
u_p/sdclk_pad_P1
u_p/test_so_4_pad_P1};
```

Signal ordering begins at the lower-left pad for the left edge, the upper-left pad for the top edge, the upper-right pad for the right edge, and the lower-right pad for the bottom edge as shown in [Figure 30](#).

Figure 30 Signal I/O Constraints Side Ordering



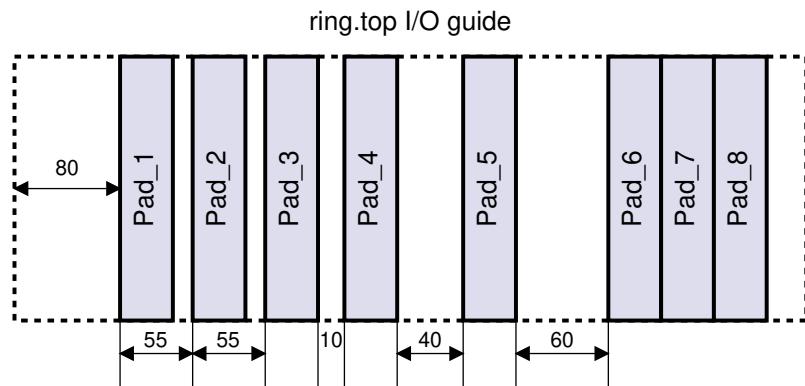
In the following example, a signal I/O constraints file creates an ordering and spacing constraint for the top I/O guide in the ring. The constraint creates an 80 micron gap between the start of the I/O guide and the first signal I/O pad. A 55 micron pitch is used to place the Pad_1, Pad_2, and Pad_3 I/O drivers. The constraint creates a 10 micron spacing between the Pad_3 and Pad_4 I/O drivers, a 40 micron spacing between Pad_4 and Pad_5, a 60 micron spacing between Pad_5 and Pad_6, and no spacing between

Pad_6, Pad_7, and Pad_8. [Example 15](#) shows the signal I/O constraint file, and [Figure 31](#) shows the layout results after running the `place_io` command.

Example 15 I/O Constraints File Example

```
ring.top
{80}
{{55} Pad_1 Pad_2 Pad_3}
{10}
{Pad_4 40 Pad_5 60 Pad_6}
{Pad_7 Pad_8}
;
```

Figure 31 ring.top I/O Placement



Specifying I/O Placement Constraints With a CSV File

Instead of a constraint file, you can use a comma-separated values (CSV) file to define the ordering and location of I/O cells for a specific I/O guide. Note the following when placing I/Os by using a CSV file:

- The CSV file must contain a header row that defines the ordering of the fields
- Only the “Pad Name” field is required in the CSV file, the “Offset Value” field is optional
- Any other fields in the CSV file are read and ignored
- Fields can appear in any order
- Fields must not be repeated
- The number of fields must be the same for each row

To specify an order-only constraint, create a CSV file and list the I/O pad cell names starting from the beginning of the I/O guide. Use the `-csv` option with the `set_signal_io_constraints` command to specify the CSV file name. In the following example, the “ref_cell” field is for information only and is not used by the `set_signal_io_constraints` command.

```
icc2_shell> set_signal_io_constraints -csv iopads.csv \
    -io_guide_object io_guide_1
Signal IO constraints set successfully

icc2_shell> sh cat iopads.csv
ref_cell,Pad Name
B4ISH1025_EW,pad0
B4ISH1025_EW,pad1
B4ISH1025_EW,pad2
B4ISH1025_EW,pad3
```

To set an offset value for each I/O pad, specify both the “Pad Name” and “Offset Value” fields as shown in the following example. Any other fields, such as the “Comment” field in this example, are for information only and are not used by the tool:

```
icc2_shell> sh cat iopads_offsets.csv
Pad Name,Offset Value,Comment
pad0,200,pad0 offset is 200
pad1,400,pad1 offset is 400
pad2,600,pad2 offset is 600
pad3,800,pad3 offset is 800
pad4,1000,pad4 offset is 1000
pad5,1200,pad5 offset is 1200
pad6,1400,pad6 offset is 1400
pad7,1600,pad7 offset is 1600
```

If your CSV file contains custom column names, create a map file with comma-separated pairs of custom column names and standard column names and use the `-map_file` option to specify the map file name with the `set_signal_io_constraints` command. The following example uses the custom column names `REFERENCE_CELL`, `PAD_NAME`, and `PAD_OFFSET` in the CSV file. The map file lists the custom column name and standard column name, separated by a comma. Note that the `REFERENCE_CELL` column is not used by the tool and is not included in the map file.

```
icc2_shell> sh head ioconstraints2.csv
REFERENCE_CELL,PAD_NAME,PAD_OFFSET
B4ISH1025_EW,pad_iopad_0,0
B4ISH1025_EW,pad_iopad_1,100
...
icc2_shell> sh cat map.txt
PAD_NAME,Pad Name
PAD_OFFSET,Offset Value

icc2_shell> set_signal_io_constraints -csv ioconstraints2.csv \
    -io_guide_default_io_ring1.left -map_file map.txt
Setting PAD_NAME as custom name for column Pad Name.
Setting PAD_OFFSET as custom name for column Offset Value.
Signal IO constraints set successfully
1
```

Placing I/Os Pads Together With Fixed Objects

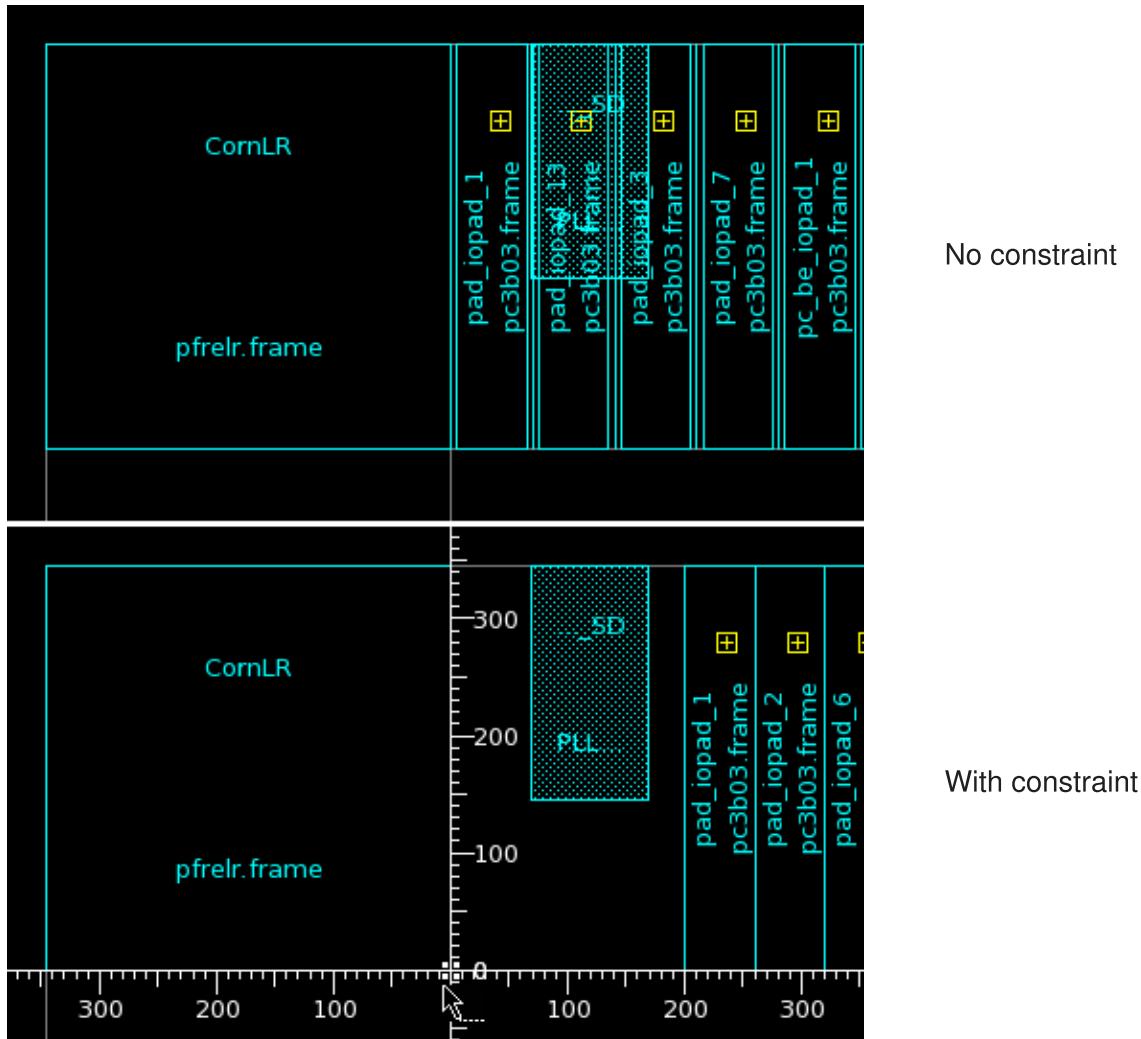
By default, the `place_io` command places I/O pads into the I/O guides and overlaps any existing fixed objects. To honor a fixed object that is placed at the start of the I/O guide and specify that I/O pads are inserted away from the object, create a constraint file that contains an absolute constraint and order-based constraint. The absolute constraint contains the first pad to be placed and the spacing value. The order-based constraint also contains the first pad and other order-dependent pads.

The following constraint file creates a constrained I/O placement for the top I/O ring named `ring1.top`. The constraint places the `pad_iopad_1` pad 200 microns from the start of the I/O guide. Note that `pad_iopad_1` is specified both in the absolute constraint and in the order-based constraint.

```
ring1.top
{{absolute} pad_iopad_1 200.0}
{
    pad_iopad_1
    pad_iopad_2
    pad_iopad_6
    pad_iopad_12
    pc_be_iopad_0
}
;
```

The following figure shows the design after running the `place_io` command without the previous constraint and with the constraint.

Figure 32 place_io Fixed Pad Constraint Comparison



Assigning I/O Pads to Bumps

The IC Compiler II tool supports both automatic and user-specified pad-to-bump assignment as explained in the following topics.

- [Automatic Bump Assignment](#)
- [User-Specified Bump Assignment With Matching Types](#)

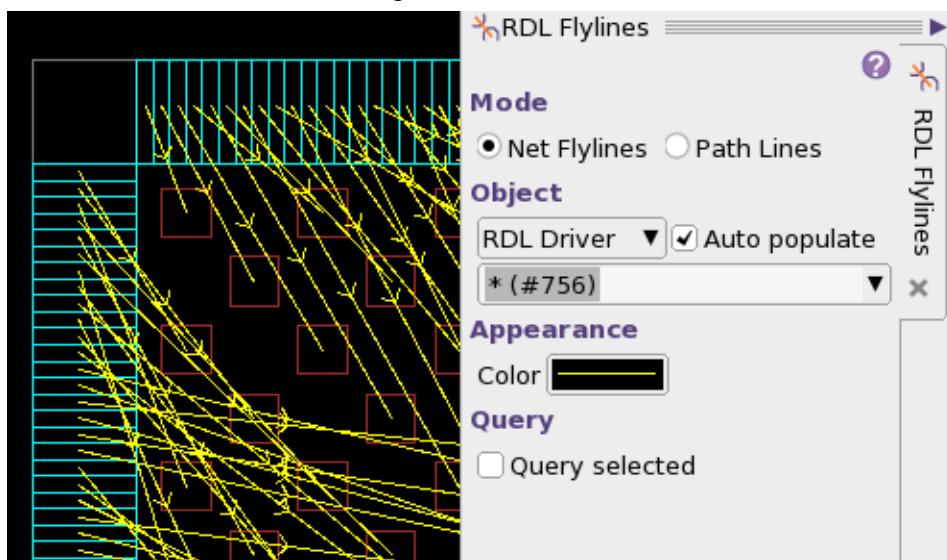
Automatic Bump Assignment

Before routing flip-chip drivers to flip-chip bump cells, you must create assignments between the I/O drivers and bump cells. The tool supports both automatic assignment with the `place_io` command and user-generated assignment with the `create_matching_type` command. To perform automatic assignment, specify the `-bump_assignment_only` option with the `place_io` command as follows:

```
icc2_shell> place_io -bump_assignment_only
```

To verify the assignments, use the RDL Flylines panel in the GUI to view the assignments. Choose View > Flylines > RDL Flylines from the menu and select the RDL net names to view the RDL flylines.

Figure 33 Default RDL Net Assignments



Alternatively, use the `write_matching_types` command with the `-from_existing_assignment` and `-file_name` options to write out the assignments created by the `place_io` command. Examine the file written by this command to view the assignments created by the tool.

```
icc2_shell> write_matching_types -from_existing_assignment \
           -file_name existing.tcl
1

icc2_shell> sh head existing.tcl
create_matching_type -name existing_assignment_0
  -uniqualify 1 [list [get_cells { bottom_0_0 }] \
    [get_pins { __added_power_driver_1/VDDPAD }]]
create_matching_type -name existing_assignment_1
  -uniqualify 1 [list [get_cells { bottom_0_1 }]] \
```

```
[get_pins { sdram_DQ_iopad_59/PADIO }]
create_matching_type -name existing_assignment_2
    -uniqualify 1 [list [get_cells { bottom_0_10 }]
    [get_pins { sdram_DQ_iopad_38/PADIO }]]
...

```

User-Specified Bump Assignment With Matching Types

Instead of automatic bump assignment, you can associate flip-chip drivers and bump cells by creating one or more user-specified matching types. The matching type for a cell has two primary functions: provide support for driver-to-bump pairing and provide support for driver placement. The tool assigns flip-chip drivers to flip-chip bump cells that have the same matching type.

The following example uses two `create_matching_type` commands. The first command associates pad cells `pad_iopad_46` through `pad_iopad_49` with bump cells in row 25. The second command associates pad cells `pad_iopad_42` through `pad_iopad_45` with bump cells in row 24. The `-uniqualify` option specifies the number of pad pins that can be assigned to a single bump cell; a value of 0 specifies that each bump cell can be assigned to only one pad cell.

```
icc2_shell> create_matching_type -name match1 \
    -uniqualify 0 {left_25_0 left_25_1 left_25_2 left_25_3 \
    pad_iopad_46 pad_iopad_47 pad_iopad_48 pad_iopad_49}
icc2_shell> create_matching_type -name match2 \
    -uniqualify 0 {left_24_0 left_24_1 left_24_2 left_24_3 \
    pad_iopad_42 pad_iopad_43 pad_iopad_44 pad_iopad_45}
```

After creating the matching types, use the `check_pre_place_io` command with the `-matching_types` option to check the following:

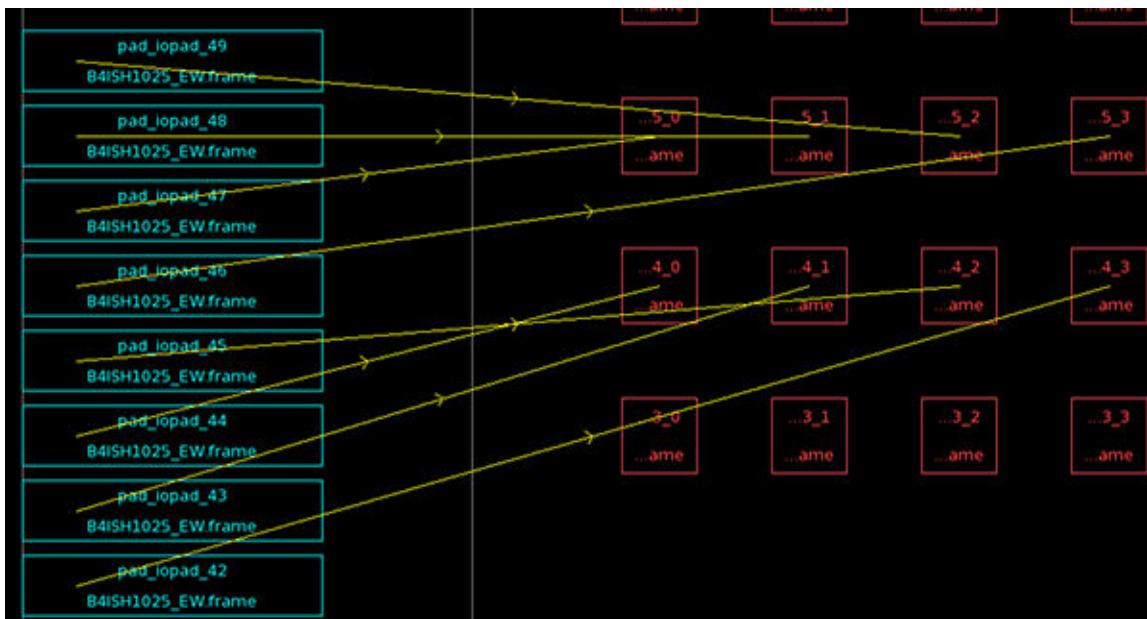
- Top-level matching type consistency
- Matching types do not contain bump cells, bump pins, or bump terminals at lower levels of hierarchy
- Each object has only one matching type
- Each matching type has enough bumps for bump assignment
- Netlist violations; components within the same matching type must be connected to the same net or be unconnected
- Matching types agreement across all multiply instantiated block instances

```
icc2_shell> check_pre_place_io -matching_types
Warning: Matching type m1 and matching type m2 conflict
    over bump io_pg_0. (DPI-575)
Warning: bump io_pg_0 in matching type m1 cannot be skipped since
    it is connected to an IO pad pin. (DPI-547)
```

```
Warning: bump io_pg_0 in matching type m2 cannot be skipped since
it is connected to an IO pad pin. (DPI-547)
0
```

The `place_io` command applies the matching type assignments to the pad and bump cells; this command is required to view the updated flylines with the RDL flylines panel. [Figure 34](#) shows the RDL flyline connections created by the `place_io` command. View RDL flylines in the layout by choosing View > Flylines > RDL Flylines in the GUI.

Figure 34 RDL Flylines Between Matching Cells



Other Matching Type Command Examples

Use the following commands to modify, report, and delete matching types.

- Return a collection of matching types with the `get_matching_types` command.

```
icc2_shell> get_matching_types
{SIGNAL POWER}
```

- Create a collection of cells that are assigned a specified matching type name with the `get_cells` command and `matching_type.name` attribute.

```
icc2_shell> get_cells -filter "matching_type.name == SIGNAL"
{pad_iopad_49 pad_iopad_48 ...}
icc2_shell> get_cells -filter "matching_type.name =~ SIG*"
{pad_iopad_49 pad_iopad_48 ...}
```

- Add I/O pad or bump cells to the matching type with the `add_to_matching_type` command.

```
icc2_shell> add_to_matching_type match2 {pad_iopad_40 pad_iopad_41}
```

- Write out your user-defined matching type assignments with the `write_matching_types` command.

```
icc2_shell> write_matching_types -file_name user_matching.tcl
```

If you did not create any matching type assignments, the file is empty.

- Write out the default matching types assigned by the tool with the `write_matching_types -from_existing_assignment` command.

```
icc2_shell> write_matching_types -from_existing_assignment \
           -file_name existing.tcl
```

1

```
icc2_shell> sh head existing.tcl
create_matching_type -name existing_assignment_0
  -uniquify 1 [list [get_cells { bottom_0_0 }] \
    [get_pins { __added_power_driver_1/VDDPAD }]]
create_matching_type -name existing_assignment_1
  -uniquify 1 [list [get_cells { bottom_0_1 }] \
    [get_pins { sdram_DQ_iopad_59/PADIO }]]
create_matching_type -name existing_assignment_2
  -uniquify 1 [list [get_cells { bottom_0_10 }] \
    [get_pins { sdram_DQ_iopad_38/PADIO }]]
...

```

You can re-create the matching types at a later time by sourcing the file written by the `write_matching_types` command.

```
icc2_shell> source existing.tcl
```

- Remove one or more matching types with the `remove_matching_types` command.

```
icc2_shell> remove_matching_types {match2}
```

1

```
icc2_shell> remove_matching_types -all
```

1

- Remove specific I/O pad or bump cells with the `remove_from_matching_type` command.

```
icc2_shell> remove_from_matching_type match2 {pad_iopad_42}
```

- Create a list of matching types and associated I/O pad or bump cells with the `report_matching_types` command.

```
icc2_shell> report_matching_types match2
Matching Type  Uniquify  Object Type  Object Name
-----
match2          0          cell        left_24_0
                cell        left_24_1
                cell        pad_iopad_42
                cell        pad_iopad_43
...
...
```

Placing I/Os and Writing Constraints

After setting power and signal I/O placement constraints, you can place the I/O drivers and write out the placement constraints.

To place I/Os and write out the placement constraints,

1. Place the I/Os with the `place_io` command.

```
icc2_shell> place_io
```

2. Write out the I/O placement constraints file based on the current I/O placement with the `write_io_constraints` command.

```
icc2_shell> write_io_constraints -filename io_constraints.txt
1
icc2_shell> sh cat io_constraints.txt
_default_io_ring1.left
{{order_only}} u_p/test_si_13_pad_P1
u_p/data_pad_1_x0_3_P1
u_p/data_pad_2_x0_3_P1
u_p/data_pad_2_x0_2_P1
...
...
```

The `place_io` command supports options to control how I/Os are placed. Use the `-io_guide io_guide_list` option to place I/Os only in the specified I/O guides. If the I/O guide is not long enough to contain all pads, the pads are placed after the endpoint of I/O guides. Use the `-rule rule_name` option to specify a single routing spacing rule for redistribution layer (RDL) routes. Use the `-incremental` option to place additional I/O drivers for nonflip-chip designs.

Note that the `place_io` command removes existing filler cells if the `physical_status` property is not set to `fixed`. This condition occurs when all pads are considered during flip-chip I/O placement. If only a fraction of pads are incrementally replaced, then filler cells are not removed.

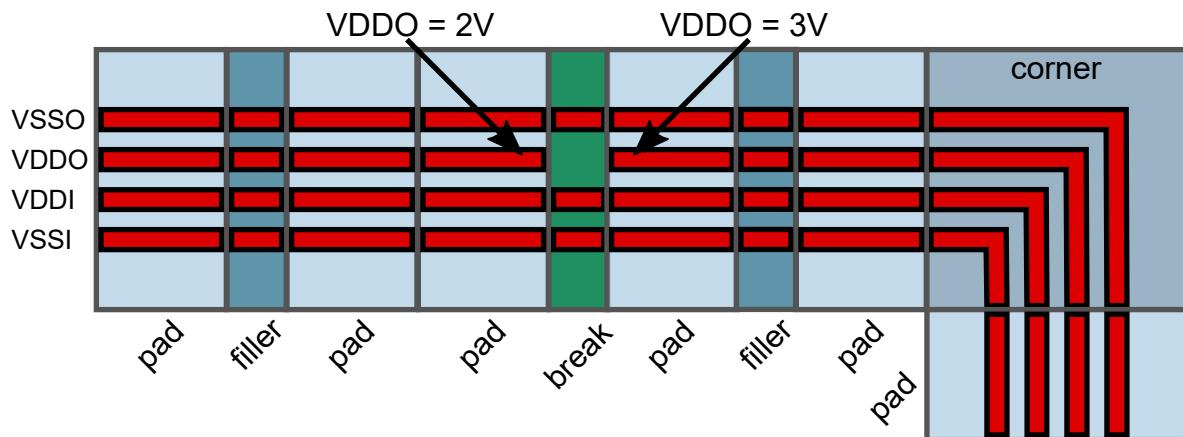
If you dedicate a matching type to a collection of only bump cells, the tool does not connect these bump cells to I/O drivers. Use this feature to reserve certain bump cells for power or ground and avoid connecting the bump cells to I/O driver cells. If you apply a matching type to I/O drivers, I/O driver pins, and I/O driver terminals, but do not assign a matching type for bumps, the `place_io` command skips bump assignment and continues to place other I/Os. If an I/O driver pin is already connected to a bump through a two-pin net, the pad pin remains connected to the bump. For isolated pad cells which are not assigned to an I/O guide, the tool locates the closest unassigned bump cell and calculates a flyline route between the pad cell and bump cell. For groups of pad cells not within an I/O guide, the tool creates a virtual I/O guide.

The `write_io_constraints` command supports options to control the name and contents of the output file. Use `-filename file_name` option to specify the output constraints file name. Use the `-format order_only | spacing | pitch | fixed` option to specify the type of constraints to write to the output file. The `order_only` argument writes only pad order information to the file. The `spacing` argument writes the spacing values between pads and the pad cell names to the file. The `pitch` argument writes pad cell locations by using cell pitch instead of spacing values and requires that all pad cells be placed evenly in the I/O guide. The `fixed` argument specifies that all pad cells are fixed at their current locations.

Adding and Placing Physical-Only I/O Cells

After I/O cells are placed, additional cells are inserted into the I/O pad ring to fill the gaps between the I/O driver cells to create more uniform density in the die. The IC Compiler II tool provides the `create_io_break_cells`, `create_io_corner_cell`, and `create_io_filler_cells` commands for this purpose. The following figure shows a typical I/O ring with pad, corner, filler, and break cells.

Figure 35 I/O Pad, Corner, Filler, and Break Cells



For information about inserting these cells, see the following topics:

- [Inserting Corner Cells](#)
- [Inserting Break Cells](#)
- [Inserting Filler Cells](#)

Inserting Corner Cells

Corner cells are placed at the intersection or projected intersections of two perpendicular I/O guides. Use options to the `create_io_corner_cell` command as shown in the following examples to control how the cells are inserted.

- Insert a reference of the corner cell named “CORNER” at the intersection of the I/O guides named “left” and “bottom”.

```
icc2_shell> create_io_corner_cell \
    -reference_cell CORNER {left bottom}
```

- Place an existing corner cell instance at the intersection of the I/O guides named “bottom” and “right.”

```
icc2_shell> create_io_corner_cell -cell corner_cell_1 {bottom right}
```

- In a design with multiple levels of hierarchy and overlapping blocks, insert a reference of the corner cell named “CORNER” at the intersection of the I/O guides named “right” and “top” and instantiate the corner cell within the `I_ORCA_TOP` block.

```
icc2_shell> create_io_corner_cell \
    -reference_cell CORNER {right top} -force I_ORCA_TOP
```

Inserting Break Cells

I/O break cells are used to create a space between the abutted connections of power nets in the I/O ring to allow for different voltages. A library might contain multiple I/O break cells, depending on the supply structure of the I/O drivers and the requirements of the design.

I/O break cells are inserted by using the `create_io_break_cells` command. Use options to the command as shown in the following examples to control how the cells are inserted.

- Insert a reference of the break cell named “BREAK” just before the start of the I/O guide named “left”.

```
icc2_shell> create_io_break_cells -reference_cells BREAK \
    -location start left
```

Use the `-location end` option to place the break cell just after the end of the I/O guide. Use the `-location both` option to place the break cell at both ends of the I/O guide.

- Move the existing `break_cell_1` instance 600 microns from the start of the I/O guide named “left”.

```
icc2_shell> create_io_break_cells -cells break_cell_1 \
    -location 600 left
```

- Insert a reference of the break cell named “BREAK” at the intersection of the I/O guides named “top” and “right”.

```
icc2_shell> create_io_break_cells -reference_cells BREAK {right top}
```

Inserting Filler Cells

Filler cells are used to populate empty spaces between driver cells and complete an abutted power connection for I/O drivers. Many cell libraries contain multiple filler cell references of varying widths; wide filler cells are used to fill wide gaps and narrow filler cells are used to fill narrow gaps.

Use options to the `create_io_filler_cells` command as shown in the following examples to control how the cells are inserted. Note that when multiple filler cell references are specified with the `-reference_cells` option, the tool chooses the largest possible cell from the list that fills the space.

- Insert filler cell references `FILLER20A`, `FILLER10A`, `FILLER5A`, `FILLER1A`, `FILLER05A`, and `FILLER0005A` into gaps between I/O drivers in all I/O guides and use “`filler_`” as a prefix when creating the instance name. The largest filler cell, `FILLER20A`, is specified at the beginning of the list with decreasing filler cell sizes specified in order.

```
icc2_shell> create_io_filler_cells \
    -prefix filler_ \
    -reference_cells {{ FILLER20A FILLER10A FILLER5A \
        FILLER1A FILLER05A FILLER0005A }}
```

- Insert filler cell references `FILLER10A` on I/O guides named “top” and “right”. An error message is issued if gaps remain after inserting the filler cells.

```
icc2_shell> create_io_filler_cells -reference_cells FILLER10A \
    -io_guides {right top}
Error: In IO guide right, filler cells cannot cover the gap
at (6500,6090) with length 2. (DPI-089)
```

- Insert filler cell references FILLER10A on the I/O guides named “guide_1” and force the filler cell to be placed within the block named “top”. In this example, the guide_1 I/O guide overlaps multiple blocks and the design contains multiple levels of hierarchy.

```
icc2_shell> create_io_filler_cells -reference_cells FILLER10A \
    -io_guides guide_1 -force top
```

Checking I/O Placement

After placing I/O cells with the `place_io` command, use the `check_io_placement` command to validate the I/O placement. The command writes a report containing a list of violating cells and returns a collection of the cells. Options to the `check_io_placement` command control which checks are performed.

- Check specific cells with the `-cells` option.
- Check whether unplaced I/O cells exist in the design with the `-unplaced` option.

```
icc2_shell> check_io_placement -unplaced
----- Start of Unplaced Pads Check -----
Cell u13 is unplaced.
Total 1 cell is unplaced in current design.
----- End of Unplaced Pads Check -----
```

- Check for overlapping cells with the `-overlap overlap_checks` option, where `overlap_checks` is one or more of: `pad2pad`, `pad2filler`, `filler2filler`, `bump`, `corner2pad`, OR all.

```
icc2_shell> check_io_placement -overlap {pad2pad bump}
----- Start of Overlap Check -----
Pad u488 and pad u547 overlap.
Pad u488 and pad u553 overlap.
Total 3 cells have overlap violation.
----- End of Overlap Check -----
```

- Check specific I/O guides with the `-io_guides` option.

```
icc2_shell> check_io_placement \
    -io_guides [get_io_guides {_default_io_ring1.bottom}]
----- Start of Pad to Guide Assignment Check -----
Pad u547 violates pad-to-guide constraints: assigned to
    IO guide _default_io_ring1.bottom, but placed outside
    of _default_io_ring1.bottom with 4089.49 gap.
Total 1 violation is found.
----- End of Pad to Guide Assignment Check -----
```

- Check the placement of I/O cells against the minimum pitch specified for the I/O guide with the `-min_pitch` option.

```
icc2_shell> check_io_placement -min_pitch
----- Start of Min Pitch Check -----
bus_0 to bus_4 violates min_pitch 5. Current spacing is 28.
bus_2 to bus_7 violates min_pitch 5. Current spacing is 42.
Total 2 violations are found.
----- End of Min Pitch Check -----
```

- Check whether I/O pad placement is consistent with the `is_flipped` attribute for the pad with the `-flipping` option.

```
icc2_shell> check_io_placement -flipping
----- Start of Pad Flip Check -----
Pad bus_0 is not flipped, but has
    is_flipped attribute set to true.
Total 3 violations are found.
----- End of Pad Flip Check -----
```

- Check for spaces between I/O cells, including pads, corners, fillers and break cells, with the `-gap` option.

```
icc2_shell> check_io_placement -gap
----- Start of Gap Check -----
Cell bus_0 and cell bus_4 have a gap between them.
Total 2 cells have gap violation.
----- End of Gap Check -----
```

- Check the I/O pad placement against constraints set with the `set_signal_io_constraints` command by using the `-signal_constraints` option.

```
icc2_shell> check_io_placement -signal_constraints
----- Start of Signal I/O constraints Check -----
The order of pad bus_1 and pad vss violates I/O guide
    default_guide_0 order constraint.
The order of pad bus_6 and pad bus_1 violates I/O guide
    default_guide_0 order constraint.
Total 2 violations are found.
----- End of Signal I/O constraints Check -----
```

Signal I/O constraints include order, pitch, spacing, offset of pads and boundary spacing.

- Check whether I/O pads are placed on the correct I/O guides with the `-pad_to_guide_assignment` option.

```
icc2_shell> sh cat padAssign
bus_5 default_guide_0 default_guide_2;
{bus_1 bus_6} default_guide_0;

icc2_shell> check_io_placement -pad_to_guide_assignment \
```

```
-pad_assignment_file padAssign
----- Start of Pad to Guide Assignment Check -----
Pad bus_5 violates pad-to-guide constraints: assigned to
    IO guide default_guide_0, but placed outside of
    default_guide_0 with 500 gap.
Total 1 violation is found.
----- End of Pad to Guide Assignment Check -----
```

Use the `-pad_assignment_file` option to specify a file that contains I/O pads and corresponding I/O guides. The format of the file is

```
pad1 guide1;
pad2 guide2 guide3;
{pad3 pad4} guide4;
```

where one or more I/O guides can be specified for a single pad or collection of pads, and collections of I/O pads are surrounded with curly brackets ({}).

- Write the I/O placement report to a specific file with the `-filename` option.

```
icc2_shell> check_io_placement -filename io_placement_report.txt
```

- Write out I/O placement violation reports to a specific directory with the `-output_directory` option.

```
icc2_shell> check_io_placement -output_directory io_checks
Writing IO placement checking reports to directory io_checks.
```

```
icc2_shell> ls io_checks/*
io_checks/bumpassignment_violations.rpt
io_checks/flip_violations.rpt
io_checks/gap_violations.rpt
io_checks/minPitch_violations.rpt
io_checks/overlap_violations.rpt
io_checks/pad2guide_violations.rpt
io_checks/powerIOConstraints_violations.rpt
io_checks/signalIOConstraints_violations.rpt
io_checks/unplacedpads_violations.rpt
```

The command creates the specified directory and writes out individual files for each violation type.

Routing RDL Nets

After placing the bump and I/O driver cells, you can route the redistribution layer (RDL) nets.

To route the RDL nets,

1. Define the routing rules for the RDL net routes. If you plan to add net shields for the RDL routes, specify the `-shield_spacings` and `-shield_widths` options as described in [Creating RDL Net Shields](#).

```
icc2_shell> create_routing_rule rdlrule -widths {MRDL 2} \
           -spacings {MRDL 1}
```

2. Apply the routing rule to the flip-chip bump cells.

```
icc2_shell> set rdl_nets [get_nets -of_objects [get_pins \
           -of_objects [get_cells -filter "design_type==flip_chip_pad"]]] \
           {...}
icc2_shell> set_routing_rule [get_nets $rdl_nets] -rule rdlrule
```

3. Set the `flip_chip.route.layer_routing_angles` application option to `90_degree` or `45_degree` to specify the routing type for the RDL layer.

```
icc2_shell> set_app_options \
           -name flip_chip.route.layer_routing_angles \
           -value {{MRDL 90_degree}}
1
```

4. Route the RDL nets.

```
icc2_shell> route_rdl_flip_chip -layers {MRDL}
-- START GLOBAL ROUTING --
-- END GLOBAL ROUTING --
-- START DETAIL ROUTING --
-- END DETAIL ROUTING --
1
```

5. Check the RDL routes for open nets.

```
icc2_shell> report_rdl_routes -open_nets true
*****
Report : RDL routes
*****
Open Net:
DATA[7]
DATA[6]
...

```

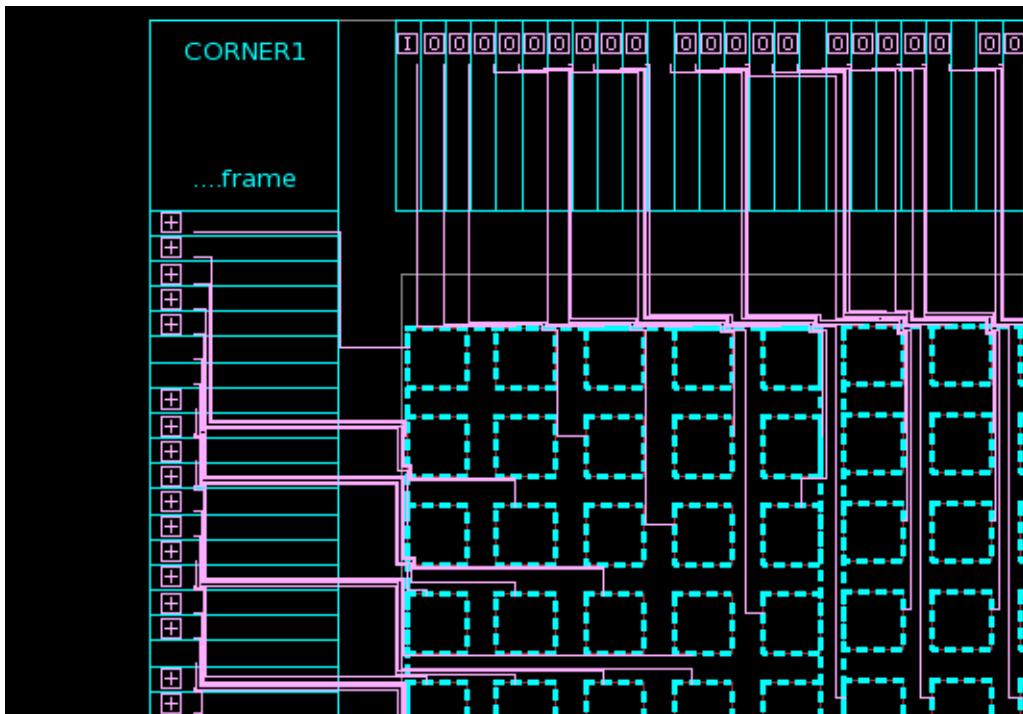
6. (Optional) Write out an error database for the Error Browser in the GUI.

```
icc2_shell> report_rdl_routes -create_error_data verification
```

Use the `verification` keyword with the `-create_error_data` option to write out error data for open and short violations. Use the `flyline_only` keyword to write out flyline data.

[Figure 36](#) shows the layout result after RDL routing. The RDL nets are routed at 90-degree angles and connect the I/O drivers and bump cells.

Figure 36 Layout After Flip-Chip Routing



Other RDL Routing Tasks

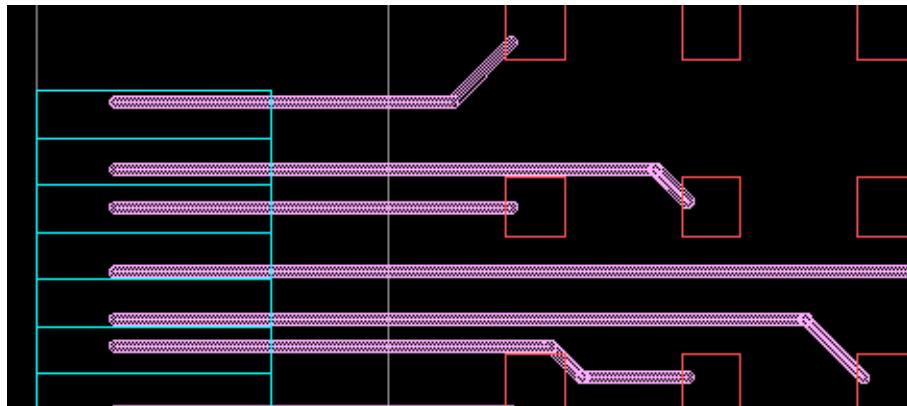
Use the following commands and application options to perform different RDL routing tasks.

- Create 45-degree routes by setting the `flip_chip.route.layer_routing_angles` application option to `45_degree` before running the `route_rdl_flip_chip` command.

```
icc2_shell> set_app_options \
    -name flip_chip.route.layer_routing_angles \
    -value {{MRDL 45_degree} {M9 45_degree}}
icc2_shell> route_rdl_flip_chip -layers {MRDL M9} -nets $rdl_nets
```

The tool creates RDL routes as shown in [Figure 37](#).

Figure 37 45-Degree RDL Routes

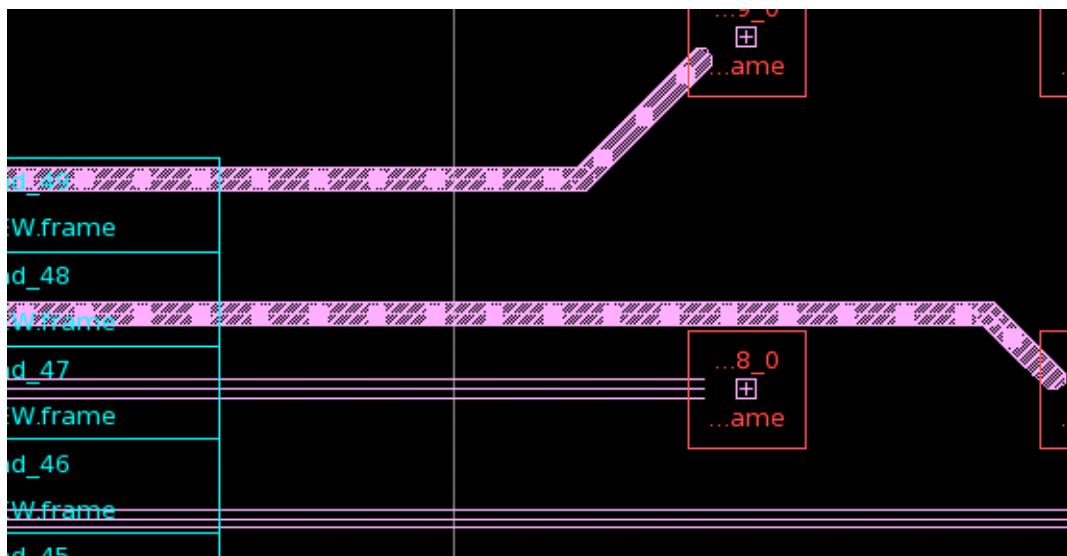


- Create an RDL route on an adjacent layer that overlaps the current route by using the `split_rdl_routes` command.

```
icc2_shell> split_rdl_routes -nets {pad[48] pad[49]} \
    -mode adjacent_layer -via_interval 25 -from_layers MRDL \
    -to_layers M9
```

The tool adds a parallel route on the M9 layer and inserts vias as shown in the upper two routes in [Figure 38](#).

Figure 38 Split Routes



- Split an existing route into multiple thin routes on the same layer with the `split_rdl_routes` command and the `-widths`, `-spacings`, and `-number_of_routes` options.

```
icc2_shell> split_rdl_routes -nets {pad[46] pad[47]} \
    -widths {MRDL {2 2 2}} -spacings {MRDL {2 2}} \
    -number_of_routes {MRDL 3}
```

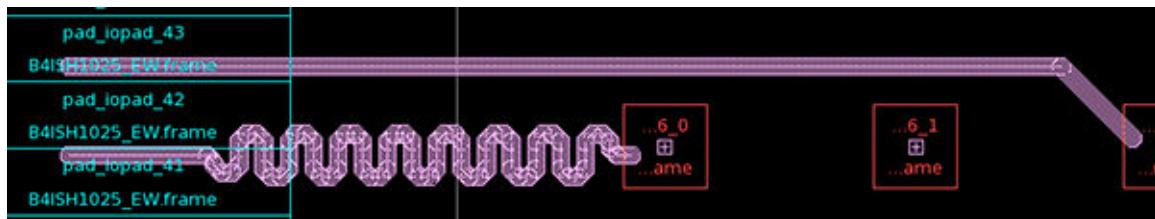
The tool splits the existing route into multiple thin routes as shown in the lower two routes in [Figure 38](#).

- Create length-matched routes for the specified routes by using the `route_rdl_differential` command.

```
icc2_shell> route_rdl_differential -layers {MRDL} \
    -nets {pad[41] pad[43]}
```

The tool creates routes for the specified nets as shown in [Figure 39](#).

Figure 39 Differential Routes

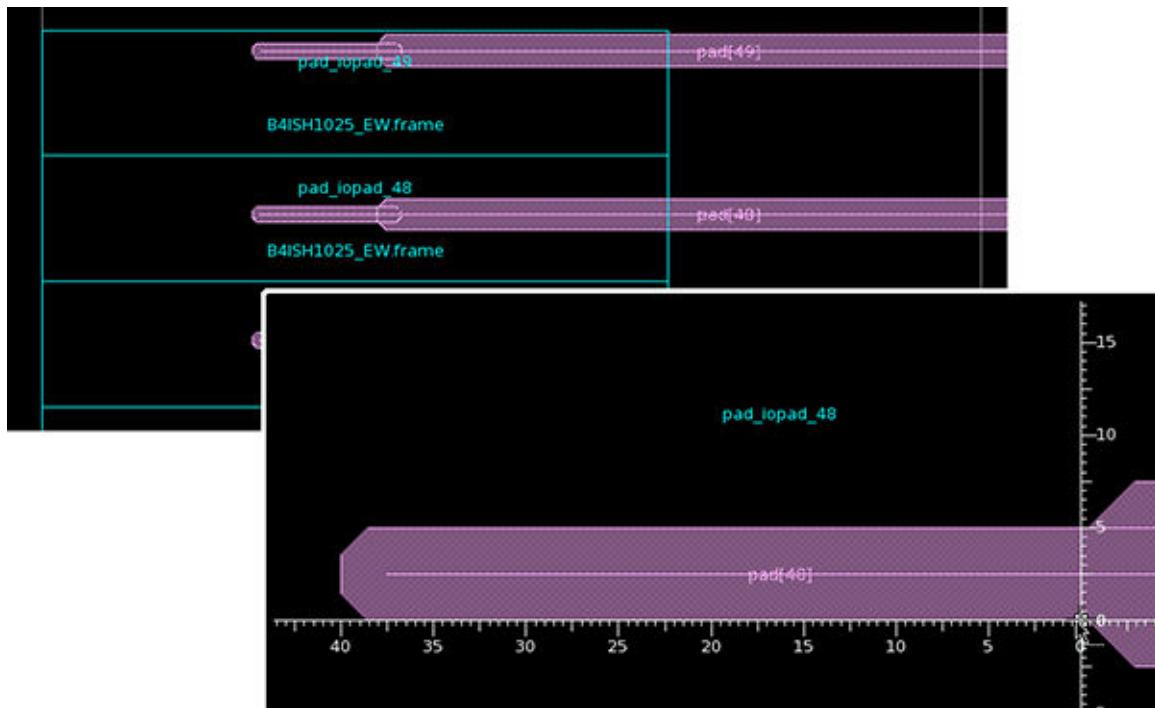


- Create a tapered route by setting taper distances and widths with the `create_routing_rule` command. Create the tapered route with the `route_rdl_flip_chip` command.

```
icc2_shell> create_routing_rule taperrule \
    -widths {MRDL 10 M9 10} -spacings {MRDL 10 M9 10} \
    -rdl_taper_distances {MRDL 40 M9 40} \
    -rdl_taper_widths {MRDL 5 M9 5}
icc2_shell> set_routing_rule [get_nets $rdl_nets] -rule taperrule
icc2_shell> route_rdl_flip_chip -layers {MRDL M9} -nets $rdl_nets
```

The tool creates the tapered routes as shown in [Figure 40](#).

Figure 40 Tapered RDL Routes

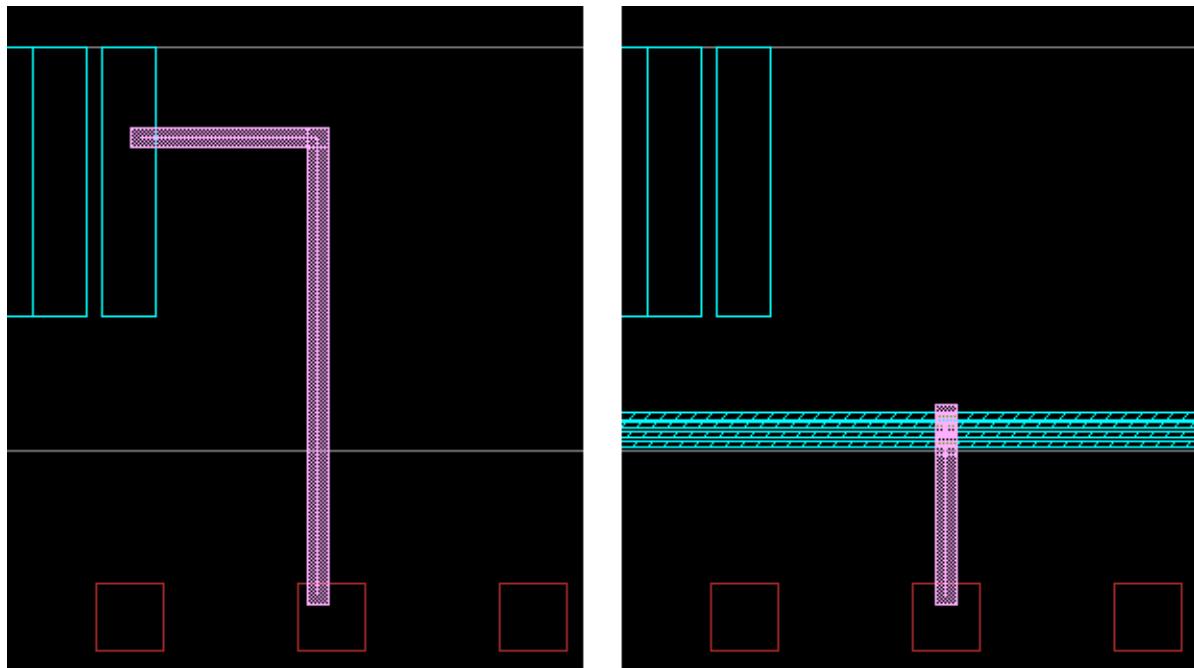


- Create an RDL route between a bump cell and a power ring net by setting the `flip_chip.route.route_to_ring_nets` application option to the net name. Use the `flip_chip.route.route_to_stripe_nets` application option to create an RDL route between a bump cell and PG net shape in the PG mesh. The following example specifies these application options.

```
icc2_shell> set_app_options \
    -name flip_chip.route.route_to_ring_nets -value VDD1
icc2_shell> set_app_options \
    -name flip_chip.route.route_to_stripe_nets -value VDD1
icc2_shell> route_rdl_flip_chip -layers $rdl_layers -nets VDD1
```

The layout for a typical flip-chip route is shown on the left; the layout for the flip-chip route created by the preceding commands is shown on the right.

Figure 41 RDL Route to Pad and PG Stripe

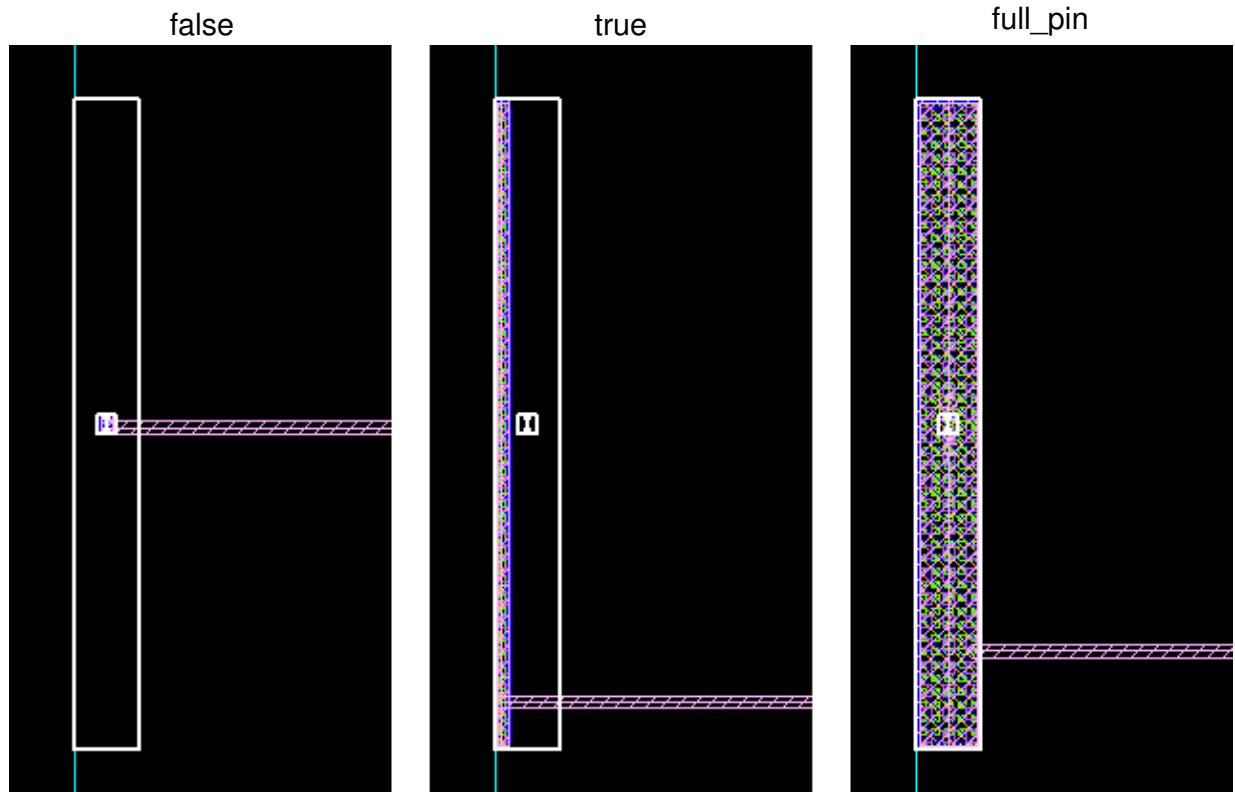


- Create additional stacked vias between RDL routes and I/O driver pins by setting the `flip_chip.route.extend_vias_on_pin` application option as shown in the following figure. By default, the value is `false` and the tool creates a simple connection to the pin. Set the application option to `true` to extend the wire and insert additional stacked vias. The width of the wire extension is the same as the wire width defined by the routing rule. Set the application option to `full_pin` to extend and widen the wire to cover the pin and insert additional stacked vias. The tool ignores the route width set by the routing rule and attempts to cover the entire I/O driver pin.

```
icc2_shell> set_app_options \
    -name flip_chip.route.extend_vias_on_pin -value full_pin
```

Figure 42 Additional Stacked Vias for RDL Routes to I/O Driver Pins

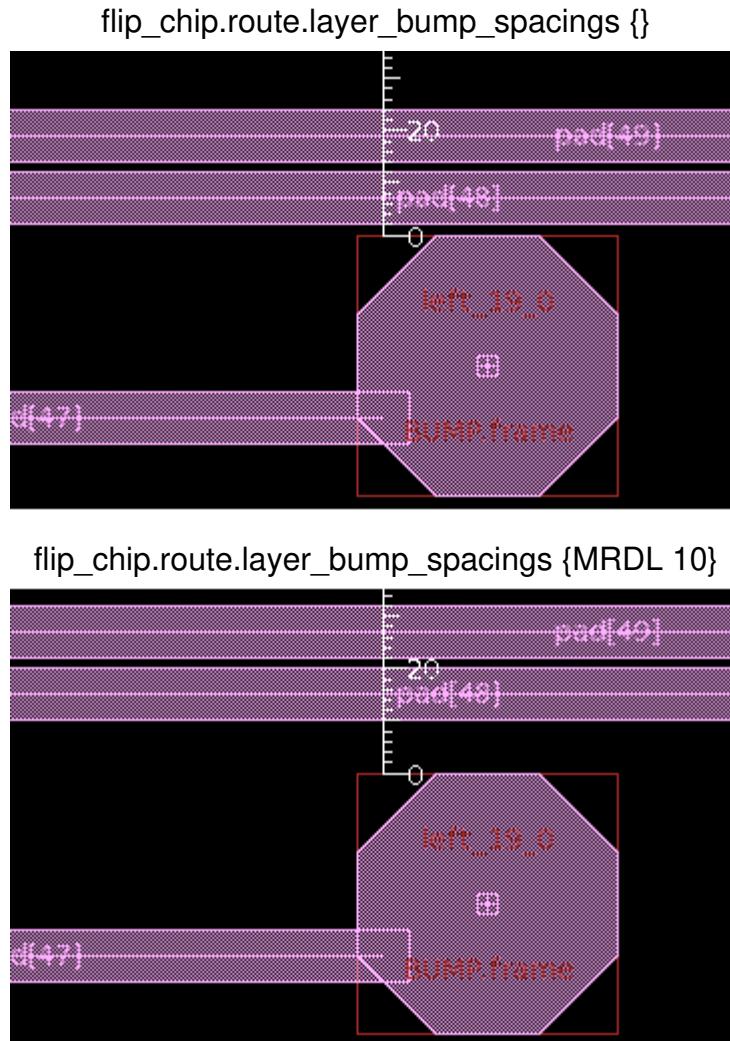
`flip_chip.route.extend_vias_on_pin`



- Specify the bump-to-route spacing values in microns for each flip-chip routing layer by using the `flip_chip.route.layer_bump_spacings` application option. You can also use the `sameNetRDLMetalToPadMinSpacing` and `diffNetRDLMetalToPadMinSpacing` technology file rules to specify the bump-to-route spacing.

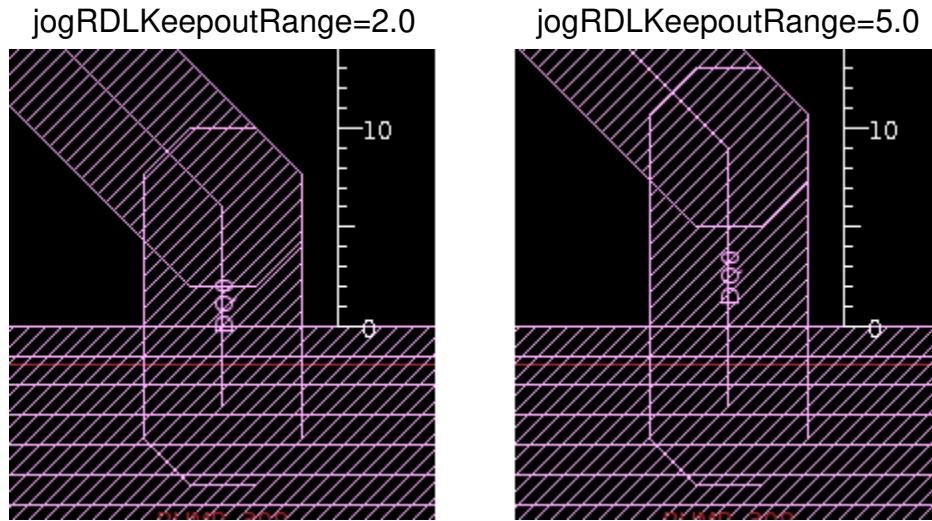
```
icc2_shell> set_app_options \
    -name flip_chip.route.layer_bump_spacings \
    -value {{M9 10.0} {MRDL 10.0}}
```

Figure 43 Bump-to-Route Spacing



The jogRDLKeepoutRange design rule specifies the keepout range between the bump cell and the route. The following figure shows the RDL routing result for two different jogRDLKeepoutRange settings.

Figure 44 Jog Keepout Range



- Remove all RDL routes with the `remove_routes -rdl` command.

```
icc2_shell> remove_routes -rdl
Successfully removed 321 route shapes.
```

Optimizing RDL Routes

After creating RDL routes with the `route_rdl_flip_chip` command or by using the GUI, use the `optimize_rdl_routes` command to improve the RDL routing patterns by

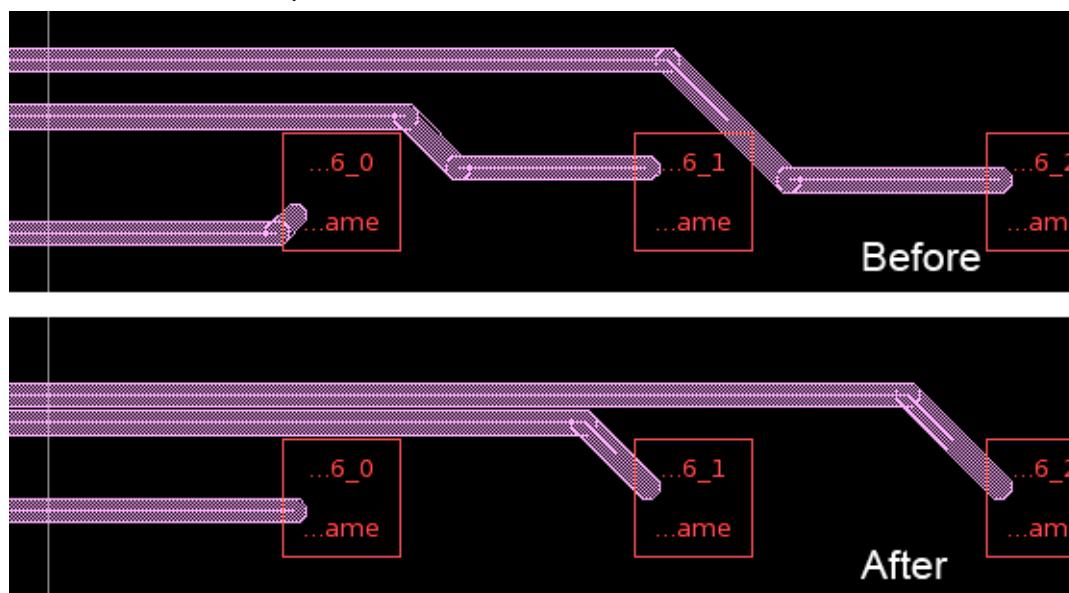
reducing the number of U- and Z-shaped routes in the block and create additional space for RDL power routes. Perform the following tasks to optimize RDL routes:

- Improve RDL routing.

```
icc2_shell> optimize_rdl_routes -nets $rdl_nets
```

[Figure 45](#) shows the layout result before and after running the previous command.

Figure 45 RDL Route Optimization

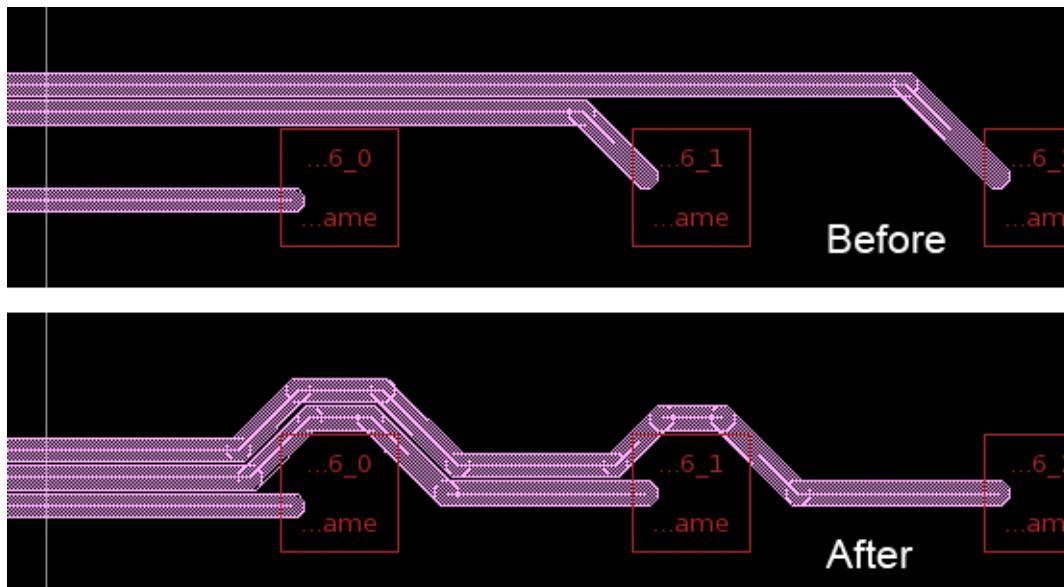


- Move signal routes to create additional routing area for power and ground routes with the `-reserve_power_resources` option.

```
icc2_shell> optimize_rdl_routes -layer {MRDL} -nets $rdl_nets \
    -reserve_power_resources true
```

After running the command with this option, the amount of U- and Z-shaped routes might increase. [Figure 46](#) shows the layout result before and after running the previous command.

Figure 46 RDL Route Optimization to Reserve Power Resources



Creating RDL Net Shields

RDL net shields provide isolation for RDL nets. You can shield nets on the same layer (known as side-wall shielding) or shield nets above and below the layer (known as coaxial shielding). Coaxial shielding provides better signal isolation than side-wall shielding, but it uses more routing resources.

The following procedure creates RDL routes and shields the routes using side-wall shielding:

1. Set the `flip_chip.route.shielding_net` application option to define the shielding net connection.

```
icc2_shell> set_app_options -name flip_chip.route.shielding_net \
    -value VSS
```

2. Create the routing rule and specify the shield spacings and shield widths.

```
icc2_shell> create_routing_rule shieldrule \
    -shield_spacings {MRDL 2 M9 2} -shield_widths {MRDL 5 M9 5} \
    -widths {MRDL 6 M9 5}
```

3. Assign the routing rule to the RDL nets.

```
icc2_shell> set_routing_rule $rdl_nets -rule shieldrule
```

4. Route the RDL nets.

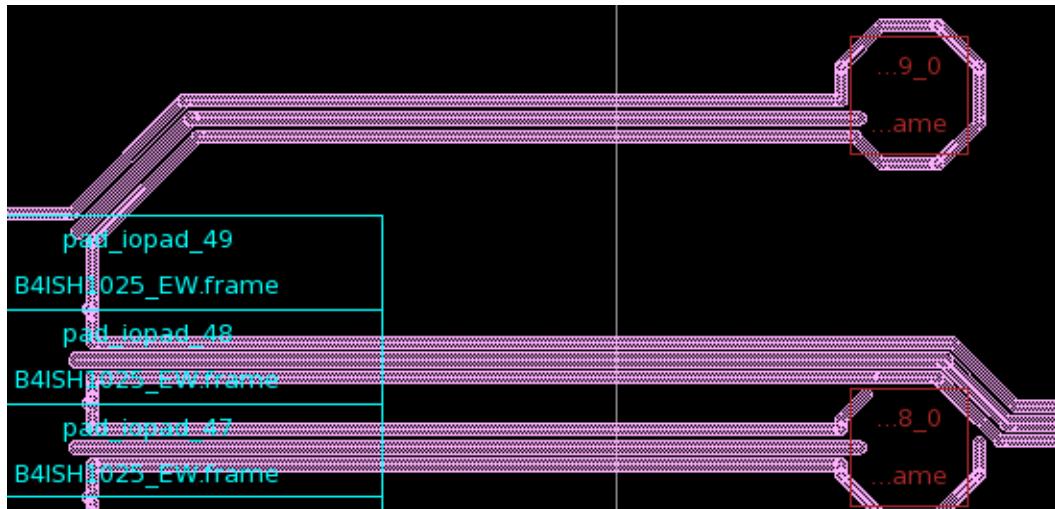
```
icc2_shell> route_rdl_flip_chip -layers {MRDL M9} -nets $rdl_nets
```

5. Create the RDL net shields.

```
icc2_shell> create_rdl_shields -layers {MRDL M9} -nets $rdl_nets
```

[Figure 47](#) shows the layout result produced by the previous commands.

Figure 47 RDL Net Shields



RDL Net Shielding Options

Use the following options with the `create_rdl_shields` command to control how the shields are created.

- List the RDL nets in a file and specify the file name with the `-nets_in_file` option.

```
icc2_shell> create_rdl_shields -layers {MRDL M9} \
    -nets_in_file rdlnets.txt
icc2_shell> sh cat rdlnets.txt
pad[40]
pad[41]
...
```

- Enable or disable shielding on via ties with the `-shield_via_tie` option.
- Remove floating shields after creating them with the `-trim_floating true` option.

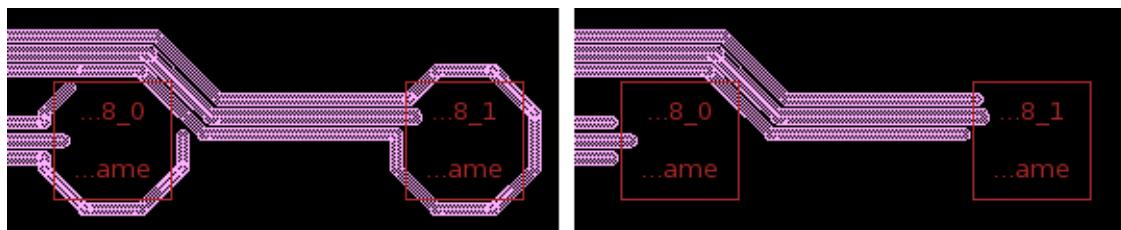
For side-wall shielding only,

- Enable or disable shielding on bumps with the `-shield_on_bump` option.

```
icc2_shell> create_rdl_shields -layers {MRDL M9} -nets $rdl_nets \
    -shield_on_bump true
```

[Figure 48](#) shows the layout result with the `-shield_on_bump` option set to `true` (left) and `false` (right).

Figure 48 Bump Shields -shield_on_bump Option



- Create a side-wall shield on one side of a net by using the `-half_shield` option and specifying one of the following values: `up`, `down`, `left`, or `right`. To create a shield on the upper or lower side of a horizontal flyline, specify `up` or `down`. To create a shield on the left or right side of a vertical flyline, specify `left` or `right`. By default, the `create_rdl_shields` command creates side-wall shields on both sides of a net.

For coaxial shielding only,

- Specify the reference layer for routing by using the `-reference_layer` option. The target layer and reference layer must be different. To specify the target routing layer, use the `-layers` option. To specify the center line of the net shield, use the `-offset` option.
- Report the coaxial shield ratio by using the `-report_coaxial_shield_ratio_offsets` option and specifying the following layers and offset value: `{shield_layer route_layer shield_route_offset}`

The following example creates METAL7 coaxial shielding wires to shield METAL8 routes. The shield-to-route center line offset is 5.

```
icc2_shell> create_rdl_shields -reference_layer METAL8 -layers METAL7 \
    -offset 5 -report_coaxial_shield_ratio_offsets {{METAL7 METAL8 5}}
```

The following figures show nets before and after coaxial shielding:

Figure 49 Before Coaxial Shielding

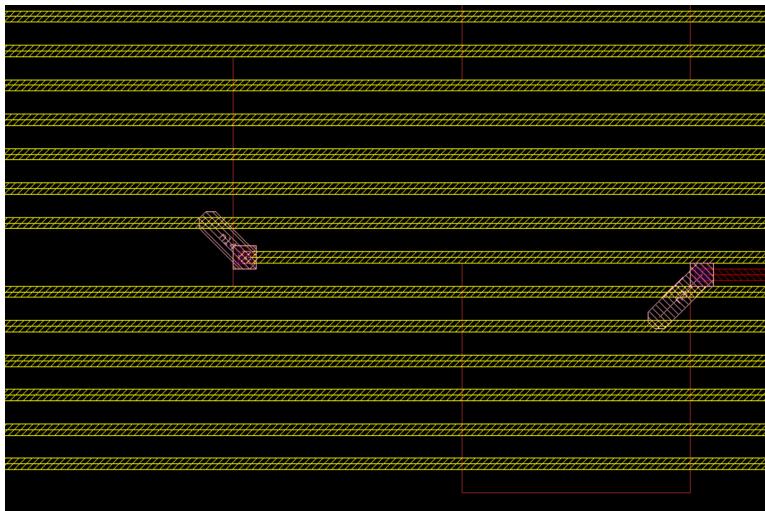
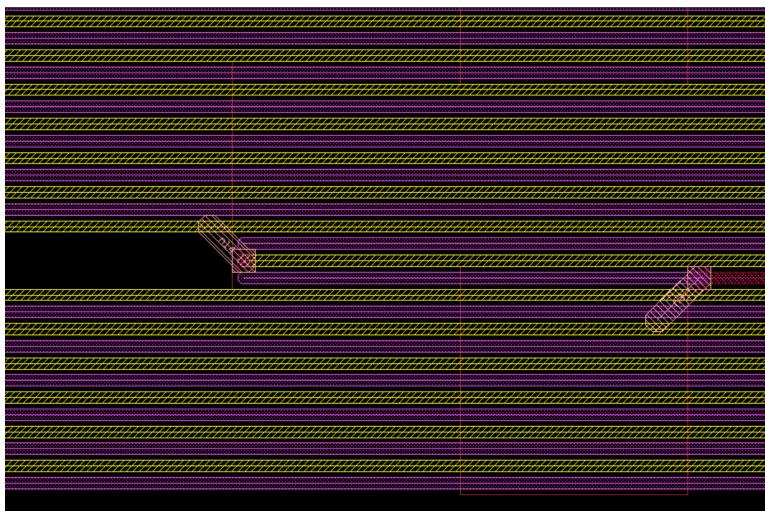


Figure 50 After Coaxial Shielding



Creating a Flip-Chip Design With Multiple Levels of Hierarchy

The tool supports top-level and block-level flip chip planning for designs that contain multiple levels of physical hierarchy. For this design style, performing top-level flip-chip routing connects bump cells at the top level and ignores bump cells within the blocks.

Note the following additional considerations for top-level flip-chip routing in a design with multiple levels of hierarchy:

- Blocks which contain I/O bump cells or pad cells must be preplaced.
- I/O pad cells can be placed at the top level or at the block level.
- I/O pad cells at the block level are considered as fixed.
- I/O guides are created at the top level.
- Matching types that assign top-level bumps to block-level I/O cells must be created at the top level.
- I/O pads are placed at the block level for blocks that contain the I/O pads.
- Block-level I/O pads should be specified with the `create_matching_type` command.
- For block-level I/O pads not specified with the `create_matching_type` command, the pin shapes should have their `class` attribute set to `bump` in the IC Compiler II Library Manager.

For IC Compiler II versions before N-2017.09, you can use a frame-based flow to assign bump cells to I/O drivers in the blocks. To prepare the frame view for I/O assignment,

1. Push into the block or edit the block that contains the I/O pad.

```
icc2_shell> set_working_design -push [get_selection]
```

2. (Optional) For power and ground pins, use the `create_shape -port` command to create terminal shapes based on the pin shapes of the pad cell.

Pin shapes for I/O signal pins are created automatically in the next step by the `create_frame` command.

3. Create the frame view for the blocks.

```
icc2_shell> create_frame
```

To reduce runtime when creating frame views for multiple blocks, you can create a script that contains the `create_frame` command and run the script with the `run_block_script` command.

4. Return to the top-level design.

```
icc2_shell> set_working_design -pop
```

5. Select the block and change the view type to the frame view of the block.

```
icc2_shell> change_view -view frame [get_selection]
```

6. Change the `design_type` attribute of the block to `flip_chip_driver`.

```
icc2_shell> set_attribute [get_selection] design_type flip_chip_driver
```

7. Select the terminals created in step 2 in the block and change the `class` attribute of terminals to `bump`.

```
icc2_shell> set_attribute [get_selection] class bump
```

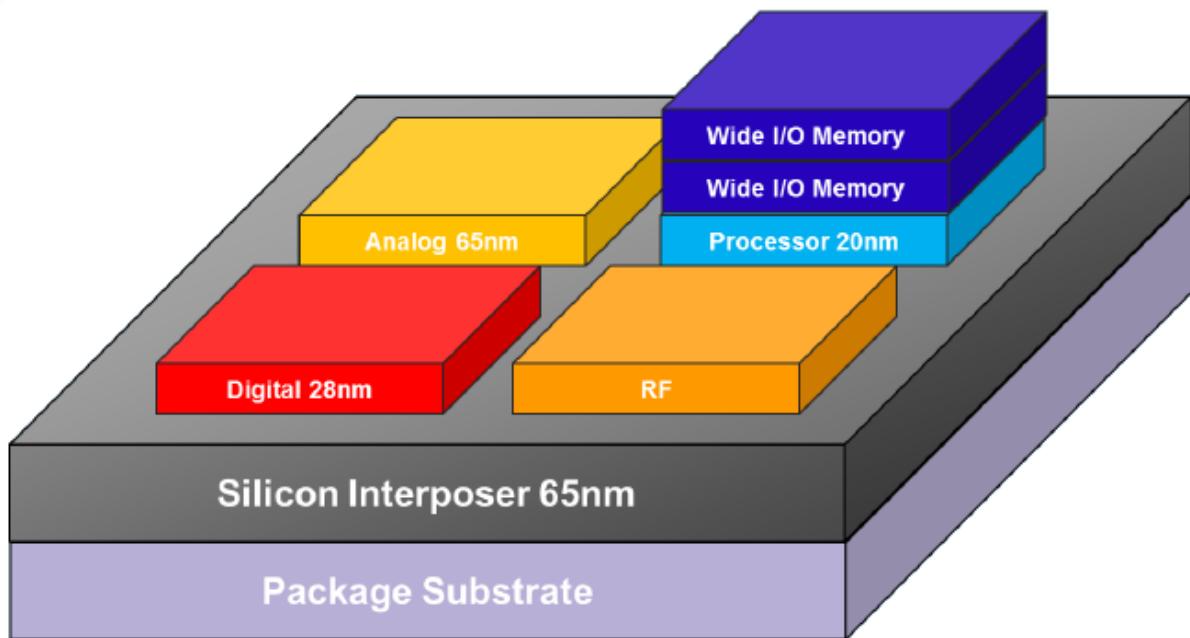
8. Continue with the normal top-level flip-chip flow.

7

Creating a 3DIC Design

In a 3DIC design, two or more die are directly stacked vertically to create a complete design or system that contains multiple dies. In a 2.5D IC design, one or more die are mounted on an interposer; the interposer also connects the die to each other and to the package. The interposer can be manufactured from various materials including silicon or glass.

Figure 51 Complex 3DIC Design

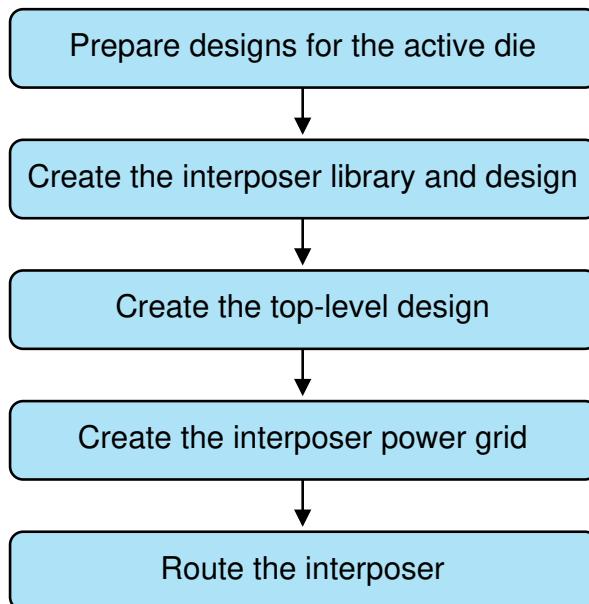


The IC Compiler II tool supports a 3DIC flow and commands to create designs that contain multiple dies. The die can be placed separately on a silicon interposer or stacked upon each other and connected with through-silicon vias (TSV). The tool supports commands to place the die, create the connections between the different die, and validate the 3DIC design.

The topics in this section describe a flow used to create a 2.5D design. The term “active die” is used to describe the logic, memory, RF, or other die that contain active devices. The interposer is the silicon die which mounts and connects the active die and is not required to contain active components or logic.

The flow for creating a 2.5D design with active die placed separately on a passive silicon interposer is shown in the following figure.

Figure 52 3DIC Flow for 2.5D Design



For more details, see the following topics:

- [Preparing Libraries for a 3D IC Design](#)
- [Preparing the Designs for the Active Die](#)
- [Creating the Silicon Interposer Design](#)
- [Reading and Writing Bump Cell and TSV Locations Using CSV Files](#)
- [Creating the Top-Level Design](#)
- [Creating Virtual Blocks](#)
- [Creating the Power and Ground Meshes for the Interposer](#)
- [Routing the Interposer](#)
- [3D IC Glossary of Terms](#)

Preparing Libraries for a 3D IC Design

A 3DIC design contains multiple subdesigns, and each subdesign can reference unique libraries and use a different technology. The libraries might have different scale factors; however, these libraries need to be used together in the complete 3DIC design. The IC Compiler II tool provides application options to support libraries with different scale factors.

To set the scaling factor for a library used in a 3DIC design, specify the `design.session_scale_factor` application option before the first design library is opened or created. The scale factors for the various libraries in a 3DIC design must be a multiple of scale factors of all the libraries. For example, if the lib1 library has a scale factor of 4000 and the lib2 library has a scale factor of 10000, the application option must be set to the least common multiple (20000) or an even multiple of 20000. By default, this option is not set. If you open a library with a different scale factor, the command issues an error message and exits. The following example sets a scale factor of 20000.

```
icc2_shell> set_app_options -name design.session_scale_factor \
    -value 20000
```

The tool also supports half-node scaling of libraries in a 3DIC design. The half-node scale factor of a library is specified by the `half_node_scale_factor` attribute on the library. If the `design.is_3dic_mode` application option is set to `true` on the top design, the coordinates in the library are scaled by the `half_node_scale_factor` setting when the library is read.

Preparing the Designs for the Active Die

Begin the 2.5D flow by creating the individual active die to be placed on the silicon interposer or on one another. Each active die has an initial floorplan and contains bump cells that are used to connect the active die to the silicon interposer or to another active die.

The following simplified flow can be used as a guideline to prepare each active die. These steps are similar to those used to create a flip-chip design, see [Creating Arrays of Bump Cells](#) for more information. It is not necessary to create a fully placed design, since the critical pieces of information for placing the active die on interposer are the bump cell locations and the active die size.

1. Read the netlist with the `read_verilog` command.

```
icc2_shell> read_verilog logic.v
```

2. Initialize the floorplan with the `initialize_floorplan` command or by reading the DEF file with the `read_def` command.

```
icc2_shell> read_def logic.def
```

3. Instantiate and place the bump cells with the `create_bump_array` command.

```
icc2_shell> create_bump_array -lib_cell BUMP -name BUMP_RING \
-delta {225 225} -bbox {{100 100} {4000 4000}}
```

Alternatively, place existing bump cells with the `read_aif` command.

Similar to a flip-chip design, the bump cell locations are derived from one of the following sources:

- A DEF file
- An AIF file
- The `create_bump_array` command

The RDL routes from pad cell to bump cell can be created during a later design stage.

Creating the Silicon Interposer Design

To create the silicon interposer design, you must create the design library for the silicon interposer, create the interposer netlist, initialize the floorplan, and create the power and ground connections.

To create the interposer design,

1. Create the design library for the interposer, use the `create_lib` command as follows:

```
icc2_shell> create_lib interposer.ndm -technology interposer.tf \
-ref_libs {c4_bump.nlib front_ubump.nlib physicalonly.nlib ...}
```

2. Create an empty netlist for the interposer and read it with the `read_verilog` command.

```
icc2_shell> sh cat interposer.v
module interposer (dummypin);
  input dummypin;
endmodule
icc2_shell> read_verilog interposer.v
```

The following example uses an empty interposer netlist. The top-level netlist contains the connections. The IC Compiler II tool automatically updates the interposer netlist with ports and all connections and creates feedthroughs as needed.

3. Initialize the floorplan for the interposer.

```
icc2_shell> initialize_floorplan -control_type die \
-boundary {{0 0} {15000 10000}}
```

4. Create the bump arrays to connect the signal and power pads of the package to the interposer with the `create_bump_array` command.

```
icc2_shell> create_bump_array -lib_cell C4_BUMP \
    -name C4_logic_SIGNAL_B -origin {500 500} -delta {190 190} \
    -repeat {74 3}
...

```

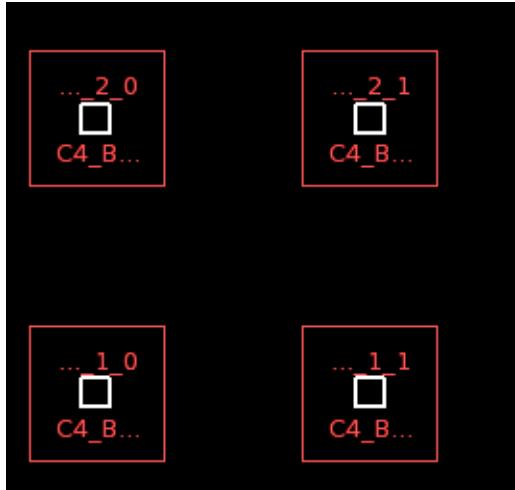
5. Create through-silicon vias (TSVs) for each of the bump cells added in the previous step with the `derive_3d_interface` command.

```
icc2_shell> derive_3d_interface -from [get_cells C4_logic_SIGNAL_B*] \
    -to_object_ref [get_via_defs -tech [get_techs *] VIAB1] \
    -name_prefix VIAB1_C4_logic_SIGNAL_B
...

```

The `derive_3d_interface` command creates new 3DIC structures based on the properties of existing structures in the design. The following figure shows the TSV cells created by the previous command.

Figure 53 TSV Cells Created by `derive_3d_interface`



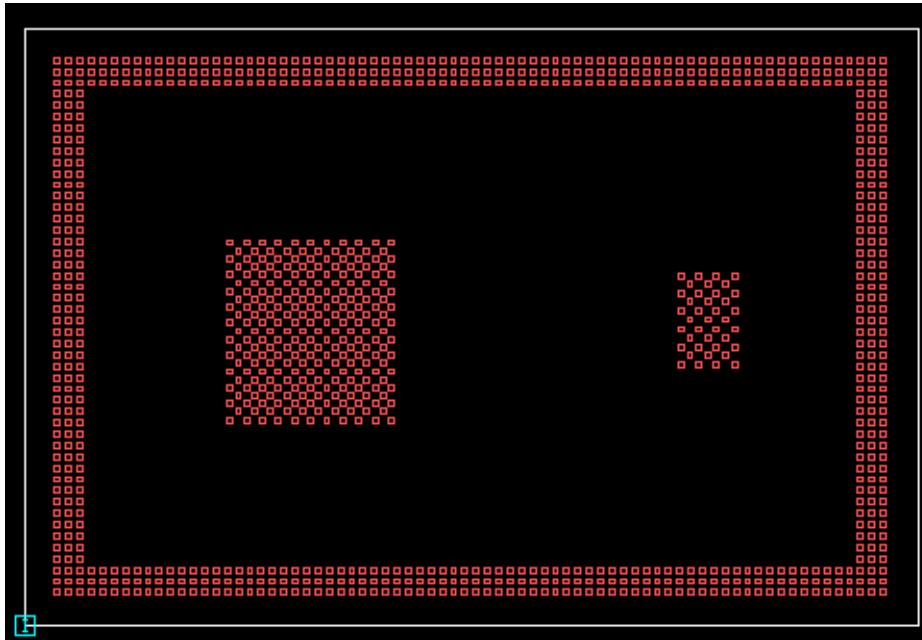
Name	Value
name	VIAB1_C4_logic_SIGNAL_B_2_3
full_name	VIAB1_C4_logic_SIGNAL_B_2_
object_class	cell
ref_view_name	design
design_type	tsv
outer_keepout...	
outer_keepout...	
outer_keepout...	
outer_keepout...	
display_view	
number_of_pins	1
origin	7006.5000 926.5000

6. Create matching types for signals, power, and ground nets for RDL routing.

```
icc2_shell> create_matching_type \
    -name Signal [all_connected [get_nets *SIGNAL*]]
```

After performing these steps, the tool creates the interposer floorplan as shown in the following figure. The bump cells around the periphery connect to signal bumps on the active dies. The two bump arrays in the center connect to power and ground for the two active dies.

Figure 54 Interposer Floorplan



Reading and Writing Bump Cell and TSV Locations Using CSV Files

Instead of setting placement constraints and matching types to place and connect bump cells and through-silicon vias (TSVs), you can specify location and connection information by using one or more comma-separated values (CSV) files. Use the `read_design_io` command to read the placement and connection information from the CSV file and place the cells on the interposer. Use the `write_design_io` command to write out the location information to a CSV file.

The following example writes out the locations of the bump cells and TSVs in the current design.

```
icc2_shell> write_design_io -file_name bump_tsv_locations.csv
icc2_shell> sh cat bump_tsv_locations.csv
Reference_name,C4_inst,Ubump_inst,TSV_inst,X_origin,Y_origin,
Orientation,Port_name,Net_name,Comments
C4_BUMP,C4_logic_SIGNAL_B_0_0,,,500,500,R0,,C4_logic_SIGNAL_B_0_0_net,
C4_BUMP,C4_logic_SIGNAL_B_0_1,,,690,500,R0,,C4_logic_SIGNAL_B_0_1_net,
C4_BUMP,C4_logic_SIGNAL_B_0_2,,,880,500,R0,,C4_logic_SIGNAL_B_0_2_net,
...
VIAB1,,,VIAB1_C4_mem_VSS_4_0,11165,5725,R0,,,
VIAB1,,,VIAB1_C4_mem_VSS_4_1,11465,5725,R0,,,
VIAB1,,,VIAB1_C4_mem_VSS_4_2,11765,5725,R0,,,
...
```

The CSV file contains a header row and a row of data for each cell instance. The following list shows a description for each field and the default name for the field in the column header:

- Cell reference name (`Reference_name`)
- C4-bump instance name (`C4_inst`)
- U-bump instance name (`Ubump_inst`)
- TSV instance name (`TSV_inst`)
- X-origin (`x_origin`)
- Y-origin (`y_origin`)
- Cell orientation (`Orientation`)
- Port name (`Port_name`)
- Net name (`Net_name`)
- Comments (`Comments`)

You can specify multiple input files and provide different fields in each file. Not all fields are required for each cell instance or connection. The header row is required and describes the keyword for each column. After the header row, each row describes a cell instance or a connection. Cell rows specify the cell instance name, reference name, location, and orientation of the cell. Connection rows specify connections between terminals, and port rows are used to create a port.

If a cell instance does not exist, the tool creates the specified cell instance. By default, the tool performs the following checks when creating new cell instances:

- Cells of the same type cannot overlap.
- A C4-bump cell can overlap an existing u-bump cell, TSV, or standard cell.
- A u-bump cell can overlap an existing C4-bump cell, TSV, or standard cell.
- A TSV can overlap an existing C4-bump cell or u-bump cell.

To disable checking, use the `-overlap_check none` option. To prevent any overlaps, use the `-overlap_check strict` option.

If the cell instance already exists, it is moved to the specified location. If the cell instance overlaps an existing instance, it is skipped. The tool issues warning messages when reading an invalid row and skips the row.

By default, if a port does not exist, the tool ignores it. To create the port instead, use the `-create_ports` option.

By default, if a port or net already exists, it retains the current setting of its `port_type` or `net_type` attribute. To set this attribute to `power` for specific ports or nets, use the `-power` option. To set this attribute to `ground` for specific ports or nets, use the `-ground` option. If the ports or nets specified in these options do not exist, the tool creates them.

If your CSV file contains custom column names, create a map file with a list of custom column names and standard column names, then specify the map file with the `-map_file` option. Each line in the map file contains the comma-separated custom column name and corresponding standard column name as shown in the following example.

```
custom_reference_name,Reference_name
custom_C4_instance_name,C4_inst
...

```

In the following example, the `locations_mapped.csv` file uses custom column names and the `mapfile.txt` file contains the list of custom column names and the standard column names.

```
icc2_shell> sh head mapfile.txt
re,Reference_name
c4,C4_inst
ub,Ubump_inst
ts,TSV_inst
...
icc2_shell> sh head locations_mapped.csv
re,c4,ub,ts,x_,y_,or,po,ne,co
C4_BUMP,C4_logic_SIGNAL_B_0_0,,,500,500,R0,,
C4_BUMP,C4_logic_SIGNAL_B_0_1,,,690,500,R0,,
...
icc2_shell> read_design_io -file_name locations_mapped.csv \
    -map_file mapfile.txt
...
1
```

Use other options with the `read_design_io` command to control how the CSV file is read.

- Specify a delimiter character with the `-delimiter_char` option.

```
icc2_shell> sh head location_splat.csv
Reference_name!C4_inst!Ubump_inst!TSV_inst!X_origin!...
C4_BUMP!C4_logic_SIGNAL_B_0_0!!!500!500!R0!!!
C4_BUMP!C4_logic_SIGNAL_B_0_1!!!690!500!R0!!!
C4_BUMP!C4_logic_SIGNAL_B_0_2!!!880!500!R0!!!
...
icc2_shell> read_design_io -file_name location_splat.csv \
    -delimiter_char !
1
```

- Specify a C4-bump reference name with the `-c4_ref_name` option.
- Specify a TSV reference name with the `-tsv_ref_name` option.
- Specify a u-bump reference name with the `-ubump_ref_name` option.
- Specify the die origin and orientation.

By default, the tool uses the lower-left corner of the die as its origin and assumes no rotation (`R0` orientation). To specify the origin coordinates, use the `-die_origin` option. To specify the die orientation, use the `-die_orientation` option. Valid values are `R0`, `R90`, `R180`, `R270`, `MX`, `MXR90`, `MY`, and `MYR90`.

- Specify the cell origin location.

By default, the coordinates specified for the cell origin are interpreted as the lower-left corner of the cell. To interpret the coordinates as the cell center instead, use the `-cell_origin_type center` option.

- Specify the length unit used in the CSV file.

By default, the command assumes the x-coordinate and y-coordinate in the CSV file use the design library length units. If the CSV file uses a different length unit, specify a multiplier with the `-units` option. The command divides the values in the CSV file by the specified value to determine the length in microns. The multiplier must be an integer value greater than or equal to one.

- Specify a comment character with the `-comment_char` option.

Use options with the `write_design_io` command to control how the CSV file is written:

- Specify the objects to write to the output file.

By default, the command writes all C4-bump cells, u-bump cells, and TSVs to the output file. To modify the objects written to the output file, use one of the following options:

- `-contents`: Output only a single object type. Supported types are `c4`, `ubump`, or `tsv`.
- `-objects`: Output only the specified objects. The specified objects must be C4-bump cells, u-bump cells, or TSVs.
- `-within`: Output only the C4-bump cells, u-bump cells, and TSVs that are within the specified bounding box.
- Specify the cell origin location.

By default, the command uses the lower-left corner of a cell as its origin. To use the cell center instead, use the `-cell_origin_type center` option. When you use this option, the tool calculates the cell center based on its bounding box.

- Specify the length unit used in the CSV file.

By default, the command writes the x-coordinate and y-coordinate to the CSV file using the design library length units. To use a different length unit, specify a multiplier with the `-units` option. The command multiplies the values in microns by the specified value and writes this value to the CSV file. The multiplier must be an integer value greater than or equal to one.

- Specify the data fields to write to the CSV file by placing a list of field names in a file and referencing the file with the `-columns_file` option.

```
icc2_shell> sh cat c4_columns.csv
Reference_name,C4_inst,X_origin,Y_origin

icc2_shell> write_design_io -file_name c4bumps.csv \
    -columns_file c4_columns.csv -contents c4
Parsing columns file c4_columns.csv

icc2_shell> sh head c4bumps.csv
Reference_name,C4_inst,X_origin,Y_origin
C4_BUMP,C4_logic_SIGNAL_B_0_0,500,500
...
```

Creating the Top-Level Design

A 3DIC design contains two or more active dies and a silicon interposer; the active dies and silicon interposer are instantiated into a top-level design. The top-level design also contains connections between the package, silicon interposer, and active dies as well as ports that represent connections to the package.

Note that the top-level design specifies only the physical and logical connectivity between the actual dies and the interposer; the top-level design is never actually manufactured in silicon.

Using the top-level design, you can

- Specify the coordinates and orientation of the dies on the silicon interposer
- Copy or mirror bumps between the interposer and the dies
- Perform 3D-physical and logical design checks of the entire system
- Assign bumps to signals based on the shortest paths

The following top-level Verilog netlist, which contains two die instances and an interposer instance, is used in the following example.

```
module top (resetn , clk , pllref , ...);
input resetn ;
input clk ;
```

```
input pllref ;
...
leon3mp_chip logic_die_inst (.resetn(resetn), ...);
mem mem_die_inst (.sa(sa), .sd(sd), ...);
interposer interposer_inst ();
endmodule
```

To create a top-level design that contains the active dies and interposer,

1. Create the design library to contain the top-level design.

```
icc2_shell> create_lib top.ndm -technology top.tf \
    -ref_libs {mem.ndm leon3mp.ndm interposer.ndm}
```

Because the top-level design has references to the active dies and the silicon interposer, the reference libraries must include the silicon interposer library and the library for each active die.

2. Read the Verilog netlist for the top-level design.

```
icc2_shell> read_verilog top.v
```

3. Initialize the floorplan.

```
icc2_shell> initialize_floorplan -boundary {{0 0} {15000 10000}}
```

4. Change the `design_type` attribute on the current block to `3dic`.

```
icc2_shell> set_attribute [current_block] design_type 3dic
```

5. Place the interposer and active dies with the `set_cell_location` command. Note the `-z_offset` option that specifies the relative z-location for the interposer and active dies.

```
icc2_shell> set_cell_location interposer_inst \
    -coordinates {0 0} -z_offset 0 -orientation N
icc2_shell> set_cell_location logic_die_inst \
    -coordinates {2500 2600} -z_offset 1 -orientation FN
icc2_shell> set_cell_location mem_die_inst \
    -coordinates {10300 3000} -z_offset 1 -orientation FN
```

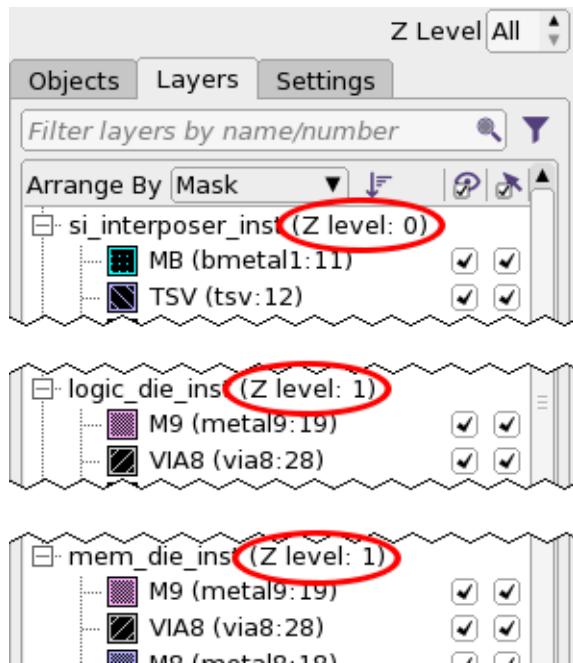
6. Confirm the placement of the interposer and active dies with the `report_3d_chip_placement` command.

```
icc2_shell> report_3d_chip_placement -all
```

```
-----
chip_name      stack_z   location      orientation scaling_factor
-----
logic_die_inst 1          (2500 2600)  FN           1
mem_die_inst   1          (10300 3000) FN           1
interposer_inst 0          (0 0)        N            1
-----
```

To review the z-ordering of the active dies and interposer, you can also query the `z_order` attribute on the die instances. Alternatively, click the Layers tab in the View Settings panel to view the z-order (Z level) setting associated with each die instance as shown in the following figure.

Figure 55 Z-Order in View Settings Panel



7. (Optional) Determine if there are any overlaps between the active dies by using the `check_3d_design` command with the `-chip_placement` option.

```
icc2_shell> check_3d_design -chip_placement
```

8. Save the design, save the library, and close the library.

```
icc2_shell> save_block
icc2_shell> save_lib
icc2_shell> close_lib
```

9. Reopen the library.

```
icc2_shell> open_lib top.ndm -ref_libs_for_edit
```

10. Ensure the blocks are editable by using the `set_editability` command.

```
icc2_shell> set_editability \
    -blocks leon3mp_chip.ndm:leon3mp_chip.design
icc2_shell> set_editability -blocks mem.ndm:mem.design
icc2_shell> set_editability -blocks siip.ndm:interposer.design
```

11. Create bump cells on the interposer by copying the bump cell locations from the two active dies.

```
icc2_shell> create_3d_mirror_bumps -from mem_die_inst \
    -to si_interposer_inst -ref_cell BUMP -prefix mem
icc2_shell> create_3d_mirror_bumps -from logic_die_inst \
    -to si_interposer_inst -ref_cell UBUMP -prefix logic
```

When the source and target dies are stacked vertically, this command copies and places bump cells, bump clusters, and bond pads from the source die to the target die. The tool also supports 3DIC flows that copy the bump locations from the interposer to the dies.

If you are creating mirror bump cells for the pins of hard macro cells, use the `-pins` option with the `create_3d_mirror_bumps` command to specify the hard macro pins.

12. (Optional) Create logical connections between the bump cells or pseudo bumps on the interposer added in the previous step and the bump cells on the active dies with the `propagate_3d_connections` command.

```
icc2_shell> propagate_3d_connections
Created 215 new nets after inter-chip nets splitting.
Propagate 67 nets.
```

13. Create matching types on the interposer.

```
icc2_shell> current_design si_interposer
{siip.ndm:si_interposer.design}

icc2_shell> add_to_matching_type Signal [get_cells *ubump*]
1
...
icc2_shell> create_matching_type -name Power [get_pins *VDD_CORE*/PAD]
{Power}
...
```

14. Create matching types at the top level.

```
icc2_shell> current_design top
icc2_shell> create_matching_type \
    -name Signal { logic_die_inst/test_so ... }
icc2_shell> add_to_matching_type Signal { si_interposer_inst/a1 ... }
icc2_shell> add_to_matching_type Signal { mem_die_inst/sa[14] ... }
```

15. Copy the matching types of cells, pins, and terminals to other pins with the `propagate_3d_matching_types` command.

```
icc2_shell> propagate_3d_matching_types
=====
```

```
Matching types to be propagated from si_interposer_inst to
mem_die_inst:
{ 'Signal' 'Power' 'Ground' }
=====
...
```

After copying the matching types, you can use the `report_matching_types` command to display the matching types assigned by the `propagate_3d_matching_types` command.

16. Assign connections between the bump cells or pseudo bumps and nearest driver cells with the `assign_3d_interchip_nets` command.

```
icc2_shell> assign_3d_interchip_nets
...
Doing assignment between chip logic_die_inst and
si_interposer_inst.
Successfully assigned 149 nets between chip logic_die_inst and
si_interposer_inst.
...
Doing assignment between chip si_interposer_inst and Package.
Successfully assigned 149 nets between chip si_interposer_inst
and Package.
```

The `assign_3d_interchip_nets` command logically connects TSV, bump cells, and pseudo bumps to nets, creates ports and nets in the interposer based on the top-level connectivity, creates feedthroughs for signals that connect the package to the dies, and creates die-to-die net connections in the interposer.

If the design contains multiple bump cells for the same interchip net, use the `-set_port_terminals incremental | all` option to resolve ambiguity, allow the tool to choose the preferred connection, and avoid creating an overlapping port. The tool sets the `preferred_pin` attribute on the port when assigning the connection; use the `get_attribute` command to review the assignment. The `incremental` argument sets the `preferred_pin` attribute only on ports without the attribute assignment. The `all` argument resets and reassigns the preferred pin attribute settings for all ports. Similarly, the command supports logical connection between pseudo bumps and drivers. Use the `-bump_regions` and `-pseudo_bumps` options to specify pseudo bumps that are to be considered for logical connection propagation.

Creating Virtual Blocks

A 3DIC design contains capacitance due to cross-coupling between two dies; this capacitance affects nets at the top-level. To capture this capacitance, tools such as StarRC extract the parasitic capacitance and store it in a virtual interface block that is added to the design database. Before extracting the capacitance with StarRC, the virtual interface block must be created in the IC Compiler II tool with the `create_3d_virtual_blocks` command.

The virtual interface block contains the ports, nets, shapes, and vias at the interfaces between two stacked dies and between a die and the interposer. The `create_3d_virtual_blocks` command supports options to modify the netlist and connect the interface information, or only add the interface information without connecting it to the existing blocks as shown in the following examples.

- Create a virtual interface block between the `mem_inst` block and the `interposer_inst` block and copy four layers from `mem_inst` and two layers from `interposer_inst`.

```
icc2_shell> create_3d_virtual_blocks \
    -blocks {{mem_inst:4} {interposer_inst:2}}
```

- Create a virtual interface block between the `mem_inst` block and the `logic_inst` block and copy four layers from both blocks.

```
icc2_shell> create_3d_virtual_blocks \
    -blocks { logic_inst mem_inst} -layers 4
```

- Create virtual interface blocks between all dies and the interposer based on the `z-offset` attribute setting for the block. The virtual interface blocks are connected to the top-level netlist.

```
icc2_shell> create_3d_virtual_blocks -in_netlist
```

Creating the Power and Ground Meshes for the Interposer

The silicon interposer mounts the active dies and provides connections between them. In addition to signals, power and ground are routed through the interposer to provide power to the active dies. To provide adequate power and minimize IR drop, you should create a power mesh inside the interposer to efficiently distribute power from the package to the dies.

Silicon interposers connect to the package through C4 bumps on the back side, then distribute power to the microbumps on the front side which connect to flip-chip bumps on the die. A power mesh on the intermediate routing layers of the interposer helps minimize potential IR drop due to additional routing on the interposer. Standard IC Compiler II power planning commands are used to create the power meshes on the interposer.

To create a power mesh on interposer for each active die,

1. Create the mesh pattern by using the `create_pg_mesh_pattern` command.

```
icc2_shell> create_pg_mesh_pattern mesh1 \
    -layers {{vertical_layer: M4} {width: 12} \
    {spacing:88} {pitch: 200} {offset: 65}} \
    {{horizontal_layer: M3} {width: 12} \
    {spacing:88} {pitch: 200} {offset: 40}}}
```

2. Create the PG strategy to associate the pattern with the region of the interposer beneath the active die.

```
icc2_shell> set_pg_strategy strat1 \
    -pattern {{name: mesh1} {nets: VDD_mem VSS_mem}} \
    -polygon {{11060 3800} {12095 6755}}
```

3. Instantiate the power plan with the `compile_pg` command.

```
icc2_shell> compile_pg -strategies strat1
```

4. Repeat the preceding steps for each active die.

Routing the Interposer

After placing bump cells on the silicon interposer, creating through-silicon vias (TSVs), and creating a power grid beneath the active die instances, the interposer can be routed. The IC Compiler II tool supports the following methods for creating routes on the interposer: routing with Zroute, routing with the RDL router, and combining both routing methods.

Before routing the interposer, you can specify a route plan to place vias and nets. The route plan uses a pattern file that contains constraints and other data used to create and place the vias and route guides. You can also modify the pattern file and add additional routing rules, such as routing bumps to pre-placed vias and specifying point-to-point routing connections, as described in [Creating a Route Plan](#).

Creating a Route Plan

When routing the interposer for high-bandwidth memory designs, you typically want finer control over placing the vias and nets. To do this, use the `create_interposer_routeplan` command to create a route plan. The command analyzes an interposer design containing pre-placed and preassigned frontside and (optionally) backside bumps. Based on the configuration of the bumps, the command creates and places vias that connect to the bumps to a second layer for routing and creates routing guidance in the form of wire stub pre-routes (route guides). The command also routes the center section between the bump arrays.

The route guides are used by the RDL router to complete complex routing paths. The placed bump vias and route guides are collectively known as a route plan. The goals of the route plan are to optimize routing and meet specified routing requirements during the routing step.

The `create_interposer_routeplan` command supports the following interposer style modes: `default`, `single_hbm` (for high-bandwidth memory designs), and `fpga` (for FPGA designs). By default, the command inserts vias only. To insert vias and route guides, specify `single_hbm` or `fpga` with the `-interposer_style` option.

The tool processes different types of pattern files based on the interposer style you choose. The pattern file contains the constraints and other data used to create and place the vias and route guides.

You can specify additional routing guidance by modifying the pattern file. For example, you can

- Create vias between bump cells and metal layers on the interposer
- Specify via definitions, via locations, a single via or a stacked via, and via spacing rules
- Route wire stubs to other wire stubs and vias (point-to-point routing)
- Route pre-placed landing vias to bumps

The pattern file is an ASCII file containing constraints and directives to drive the `create_interposer_routeplan` command. It contains one or more sections, where each section begins with a keyword followed by a pair of curly brackets. Constraints are specified inside the curly brackets. The tool uses a default pattern file unless you modify the file with new constraints and specify the updated file with the `-pattern_file` option.

To see a `single_hbm` interposer style pattern file, see [Pattern File Example](#).

The following example creates single vias between the RDL routing layer and the top metal layer for all bump cells.

```
icc2_shell> create_interposer_routeplan -pattern_file interposer.pat
```

Routing High-Bandwidth Memory Designs

3DIC interposer designs support high-bandwidth memory, a high-performance RAM interface that uses stacked RAMs. High-bandwidth memory designs are often paired with CPUs or GPUs on a silicon interposer die. These designs move memory closer to the processor using a wider data pipeline, which speeds up data throughput, reduces the amount of power necessary to drive a signal, and reduces RC delay.

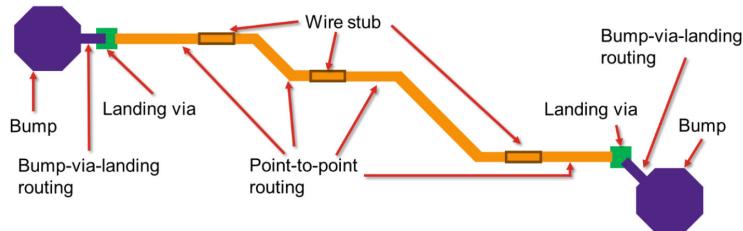
Interposer designs that incorporate high-bandwidth memory have specific, detailed physical design requirements for routing that are more stringent than other interposer configurations. These requirements include:

- High-density routing, often with track utilization rate nearing 100%
- Limited number of routing layers (often two)
- Highly parallel bus routing
- Length matching for critical data signals
- Coaxial shielding

The following routing strategy is beneficial for routing high-bandwidth memory designs.
Use this strategy for high-density, high-bandwidth memory designs only.

- Routing bumps to landing vias

You can direct the router to search for the closest via on the same net as a bump and connect the bump to the via, as shown in the following figure:



Vias that touch a bump are considered connected and do not need to be routed unless there are DRC violations between the bump and via.

- Point-to-point routing

After routing bumps to landing vias, you can specify point-to-point connections to route between bumps, landing vias, and wire stubs by providing the following information in a route plan file before routing:

```
#netName #layer #point1_x #point1_y #point2_x #point2_y
```

The route plan file specifies the net to be routed, the connection points, and the target routing layer for the point-to-point connections.

The connection points must meet the following requirements:

- The points must fall inside the same net shapes, such as wire, via, or pin shapes.
- For wire shapes, the point must be exactly at the endpoint of the wire. However, the router does not necessarily connect to pins and vias exactly at the point specified in the file.
- The points must be on the minimum grid.
- The points must be on coordinates that do not have DRC violations, meaning that when routing a wire to the specified point, there are no DRC violations with neighboring shapes. Routing pairs with DRC violations are left unrouted.

In the following example, the RDL router routes the Net1 net from point (10.5, 15.0) to point (20.5, 20.0) on the AP layer:

```
Net1 AP 10.5 15.0 20.5 20.0
```

The following example routes the interposer for a high-bandwidth memory design:

```
### Open the floorplanned interposer block

open_lib interposer_floorplan
open_block interposer

### Set the design style to a 3DIC interposer

set_app_options -name flip_chip.route.design_style \
    -value 3dic_interposer

### Create the interposer RDL vias and route guides (routeplanning)
### Set the style to HBM and specify the pattern constraint file

create_interposer_routeplan -interposer_style single_hbm \
    -pattern_file HBM_pattern.txt

### Define the HBM channel signals to be routed

set net_all "DQa24 DMa2 DMa3 DQa16 DQa27 DQa26 DQa25 DQa20 \
    DQa21 RDQSa0_t DQa28 DQa29 DERRa0 RDQSa0_c DQa18 DQa19 DQa17 \
    RDa1 DQa22 DQa23 DBIa3 DQa31 DQa30 DBIa2_DMe2 DQe25 DMe3 \
    RDQSe0_t DQe18 DQe19 DQe16 DERRe0 DQe26 DQe27 DQe24 DQe17 \
    DQe30 DQe23 DQe21 DQe20 RDQSe0_c DQe31 DQe29 DQe28 RDe1 DBIe2 \
    DQe22 DBIe3"

### Define the routing rules

set fc_route_layer_m4 {M4 AP}
set fc_route_space_m4 {M4 2.96 AP 3}
set fc_route_width_m4 {M4 1.6 AP 5}

set fc_route_layer_m2 {M2 AP}
set fc_route_space_m2 {M2 2.96 AP 3}
set fc_route_width_m2 {M2 1.6 AP 5}

create_routing_rule rdlrule_m4 -widths $fc_route_width_m4 \
    -spacings $fc_route_space_m4
create_routing_rule rdlrule_m2 -widths $fc_route_width_m2 \
    -spacings $fc_route_space_m2

set_routing_rule [get_nets $net_m2] -rule rdlrule_m2
set_routing_rule [get_nets $net_m4] -rule rdlrule_m4

### Set the shape type for pre-placed vias to user_route
### to preserve vias until the routing stage

set_attribute [get_vias *] shape_use user_route

set_app_options -name flip_chip.route.connect_via_center \
    -value true
```

```

### Specify bump-to-via routing and route the
### RDL layer

set_app_options -name flip_chip.route.bump_via_landing_mode \
    -value true

### Route the nets
### Use the -coordinates option to speed up routing by
### routing inside the specified region only

route_3d_rdl -nets [get_nets $net_all] -layer AP \
    -coordinates {{12300 11980} {12515 18814} }

### Set the options for via-to-stub routing on the lower layers

set_app_options -name flip_chip.route.bump_via_landing_mode \
    -value false

### Specify the point-to-point connection file created by the
### route plan command for the first layer (M2)

set_app_options \
    -name flip_chip.route.point_to_point_connection_file_name \
    -value hbm_p2p.txtM2

### Route the via-to-stub nets on the specified layer (M2)

route_3d_rdl -nets [get_nets $net_m2] -layer M2 \
    -coordinates {{12300 11980} {12515 18814} }

### Set the point-to-point connection file created by the route
### plan command for the second layer (M4)

set_app_options \
    -name flip_chip.route.point_to_point_connection_file_name \
    -value hbm_p2p.txtM4

### Route the via-to-stub nets on the specified layer (M4)

route_3d_rdl -nets [get_nets $net_m4] -layer M4 \
    -coordinates {{12300 11980} {12515 18814} }

```

Routing With a Combined Zroute and RDL Router Flow

If routing on the M1-Mn layers does not require the special configurations needed for high-bandwidth memory designs and if connections to through-silicon via (TSV) pins are needed, route the interposer with a combined Zroute and RDL router flow.

To route the interposer with Zroute and the RDL router,

1. Create the interposer RDL vias (route planning in the default mode).

```
icc2_shell> create_interposer_routeplan -pattern_file via_pattern.txt
icc2_shell> set_attribute [get_vias *] shape_use user_route
```

2. Create the routing rules. In this example, AP is the RDL metal layer and MB is the backside metal layer.

```
icc2_shell> create_routing_rule Wide \
    -widths {AP 5 MB 5 M1 2 M2 2 M3 2 M4 2}
{Wide}
icc2_shell> set_routing_rule $nets -rule Wide
```

3. Route the bumps to landing vias on the RDL layer.

```
icc2_shell> set_app_options -name
    flip_chip.route.bump_via_landing_mode \
    -value true
icc2_shell> route_3d_rdl -nets $nets -layer AP
icc2_shell> set_app_options -name
    flip_chip.route.bump_via_landing_mode \
    -value false
```

4. Route the M1-Mn layer nets on the interposer with the `route_group` command.

```
icc2_shell> set_ignored_layers -min_routing_layer M1
    -max_routing_layer M4
icc2_shell> route_group -nets $nets
```

5. (If needed) Route the nets on backside layer.

```
icc2_shell> route_3d_rdl -nets $nets -layer MB
```

Pattern File Example

The following example shows a `single_hbm` pattern file. The pattern file contains one or more sections, where each section begins with a keyword followed by one or more constraints specified inside a pair of curly brackets. You can modify the pattern file to specify additional routing rules by adding constraints (keywords and values).

```
config {
  {via_def 0 VIA3AP_1cut}
}
exception {
```

```
{hbm_bump_array_box {5194.670 4225.350} {6337.725 4362.85}
{9119.675 4387.700} {10357.175 4525.2}}
{hbm_right_array_routing_layer M3}
{hbm_left_array_routing_layer M2}
{hbm_wire_width 1.8}
{hbm_wire_spacing 3.5}
{hbm_channel_direction horizontal}
{hbm_mid_shielding HBM_VSS}
{hbm_middle_turn_offset 72.0}
{hbm_boundary_stub_corners auto_align}
{hbm_via_bump_abutment false}
{hbm_rdl_via_offset 0}
{hbm_stub_alignment_type staggered_before}
```

Creating RDL Net Shields in High-Bandwidth Memory Designs

You can shield nets on the same layer (known as side-wall shielding) or shield nets above and below the layer (known as coaxial shielding). Coaxial shielding provides better signal isolation than side-wall shielding, but it uses more routing resources. It is best to use a combination of side-wall shields and coaxial shields in high-bandwidth memory designs. In this flow, the `create_interposer_routeplan` command creates wire stubs (route guides) and leaves as much space for shielding wires as possible.

To create RDL routes and shield the routes for a high-bandwidth memory design,

1. Set the `flip_chip.route.shielding_net` application option to define the shielding net connection.

```
icc2_shell> set_app_options -name flip_chip.route.shielding_net \
-value VSS
```

2. Create the routing rule.

```
icc2_shell> create_routing_rule rdrlrule -widths {MRDL 6 M9 5}
```

3. Assign the routing rule to the RDL nets.

```
icc2_shell> set_routing_rule $rdl_nets -rule rdrlrule
```

4. Route the RDL nets.

```
icc2_shell> route_3d_rdl -layer {MRDL} -nets $rdl_nets
```

5. Create the routing rule and specify the shield spacing and shield widths.

```
icc2_shell> create_routing_rule shieldrule -shield_spacings {M9 2} \
-shield_widths {M9 5} -widths {MRDL 6 M9 5}
```

6. Assign the routing rule to the RDL nets.

```
icc2_shell> set_routing_rule $rdl_nets -rule shieldrule
```

7. Create the RDL net shields.

```
icc2_shell> create_rdl_shields -layers {M9} -reference_layer {MRDL} \
    -offset {8} -nets $rdlnets
```

To speed up routing by routing only within a specific region, use the `-coordinates` option and specify a list of rectangles.

3D IC Glossary of Terms

The following terminologies and conventions are used in 3DIC design:

- Interposer: The specialized silicon die that provides connections to the package and mounts for the dies. The passive interposer typically contains no logic, but does contain TSVs, C4 bumps, microbumps, and routing.
- Active dies: The dies that contact the interposer, face down, using flip-chip technology.
- Front side: The top side of a silicon die. The front side is opposite the substrate and device layers where most of the standard cell metal routing is performed.
- Back side: The bottom side of a silicon die. The bottom side contains the substrate and device layers where the back metal layers are created.
- Through-silicon via (TSV): A via that connects the back-side metal through the substrate with the device layers to the front-side metal. TSVs are required on the silicon interposer to propagate the signals from the package substrate through the interposer to the dies.
- Microbump: A back-side or front-side cell with a single pin. The cell contains the RDL metal and passivation opening for external contact. The microbump can be used for signal and power and ground connections to another die.
- C4 bump: A back-side or front-side cell with a single pin used for a connection to the package. The cell contains the back-side metal and passivation opening for external contact. C4 bumps are used for signal and power and ground connections to the package.

C4 stands for Controlled Collapse Chip Connect.

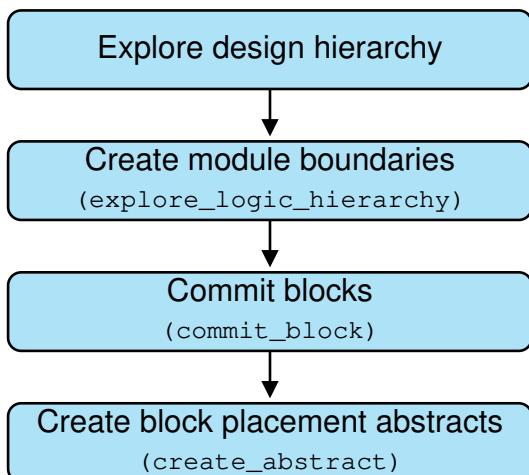
8

Managing Design Blocks

To manage design blocks, the IC Compiler II tool supports operations to easily partition your design and commit a logical hierarchy cell to a physical hierarchy block early in the design flow. After committing to blocks, you can create multiple optimized, abstract views for design blocks that contain only the information needed to perform placement, timing, and other tasks. This approach enables you to minimize the system requirements needed to efficiently distribute and process very large designs.

The flow to commit blocks and create abstracts is shown in [Figure 56](#).

Figure 56 Manage Block Flow



For more details, see the following topics:

- [Exploring the Design Hierarchy](#)
- [Creating Module Boundaries](#)
- [Committing Design Blocks](#)
- [Creating Block Placement Abstracts](#)
- [Changing the Block Boundary and Block Origin](#)

- Moving Objects Between the Top and Block Levels
- Generating ECO Scripts for Netlist Editing

Exploring the Design Hierarchy

The IC Compiler II tool provides tools to explore your design hierarchy and determine which logical hierarchy cells to commit to physical hierarchy blocks. The hierarchy browser in the GUI displays a view of the design hierarchy and provides other information about each hierarchy cell. To explore the design hierarchy with the Hierarchy Browser,

1. Choose View > Hierarchy Browser in the GUI.
2. Expand or collapse the hierarchy by clicking the [+] symbols next to the hierarchy cells.
3. (Optional) Click and drag the column headers to rearrange the table.
4. Click the block name to perform additional operations on the block.

An example hierarchy as displayed in the Hierarchy Browser is shown in [Figure 57](#).

Figure 57 Hierarchy Browser

Logical Hierarchy	Util	Pin	HM	Hier HM	Std	Hier Std
ORCA	0.868	72		41		13494
I_CLOCK_GEN		15	3	3	8	25
I_ORCA_TOP	178	0	38	203	13354	
I_BLENDER_1	0.884	193	0	0	274	274
I_BLENDER_2	0.864	166	0	0	258	258
I_BLENDER_3	0.793	119	0	0	206	206
I_BLENDER_4	0.851	219	0	0	283	283
I_BLENDER_5	0.858	284	0	0	322	322
I_BLENDER_6	0.881	172	0	0	257	257
I_BLENDER_7	0.864	131	0	0	214	214

The Hierarchy Browser shows the cell name, utilization, number of pins, number of hard macros, number of standard cells in the entire hierarchy, and other information.

Creating Module Boundaries

In preparation for committing logical hierarchy cells to physical hierarchy blocks, you can create module boundaries in the GUI. Module boundaries represent uncommitted cells (modules) in the logical hierarchy. Using module boundaries, you can quickly assess the

approximate block size that is required for a given utilization for a top-level block, and arrange module boundaries by hand within the floorplan for rough estimation of block placement. You can also set different parameters to control which top-level blocks are treated as module boundaries in this view.

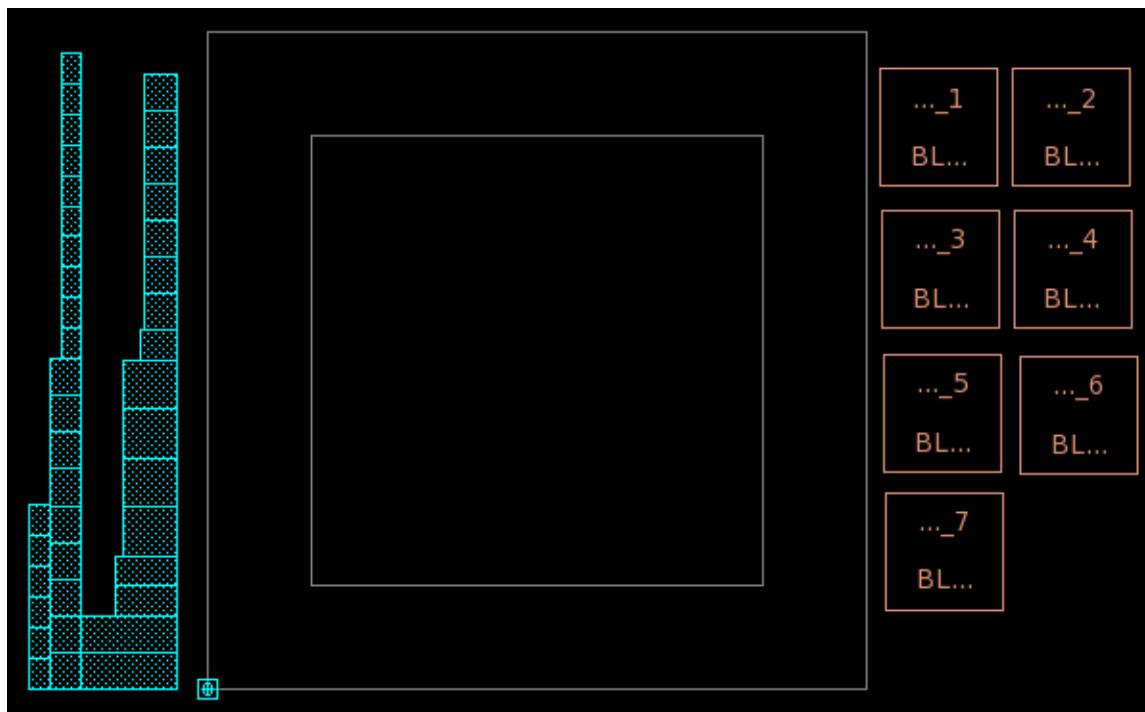
To create module boundaries,

1. Use the `explore_logic_hierarchy` command and specify the cell names for which to create module boundaries.

```
icc2_shell> explore_logic_hierarchy -create_module_boundary \
    -cell { I_ORCA_TOP/I_BLENDER_1 I_ORCA_TOP/I_BLENDER_2 \
    I_ORCA_TOP/I_BLENDER_3 I_ORCA_TOP/I_BLENDER_4 \
    I_ORCA_TOP/I_BLENDER_5 I_ORCA_TOP/I_BLENDER_6 \
    I_ORCA_TOP/I_BLENDER_7 }
```

The tool creates module boundaries based on the options you specify. By default, the tool determines the size of the boundary based on the utilization of the top-level floorplan. After creating the module boundaries, the tool updates the display in the GUI as shown in [Figure 58](#). The original floorplan boundary is shown on the left, the I/Os are located above the floorplan boundary, the module boundaries are placed at the bottom right, and any unplaced macros are placed above the module boundaries.

Figure 58 Module Boundaries

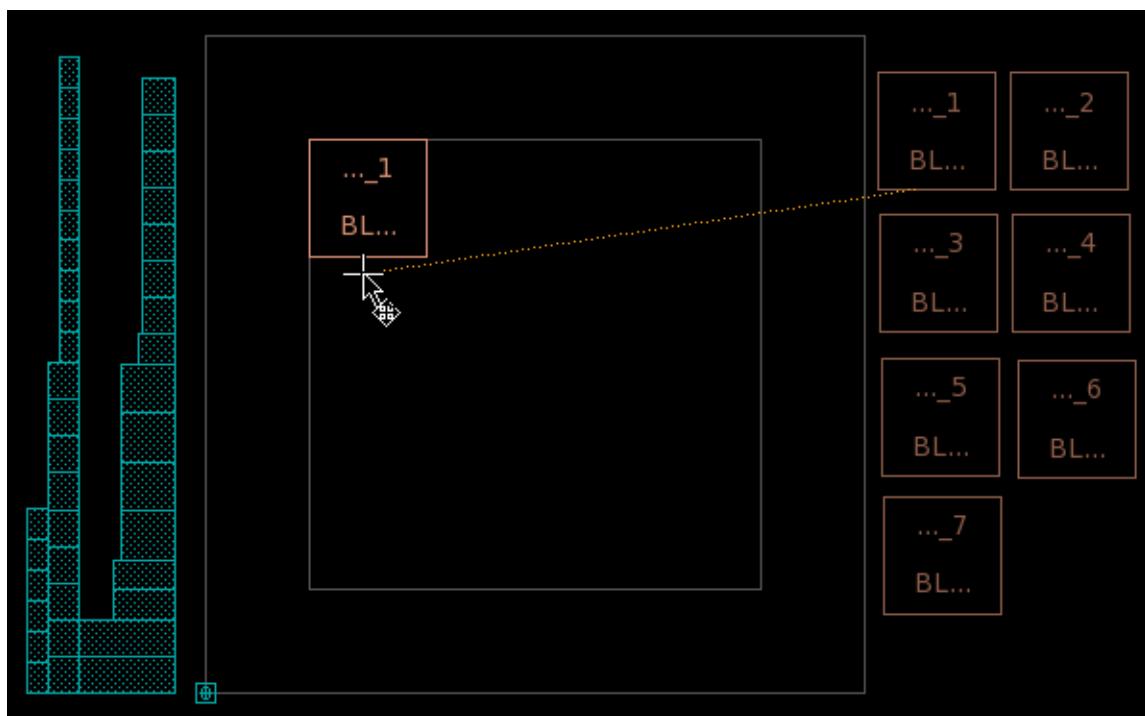


You can retain a parent module boundary and create additional module boundaries from within the parent hierarchy by using the `-nested` option. When you specify this option, the tool keeps the parent-level module boundary and creates a new module boundary for the lower level of hierarchy. The following example creates two module boundaries: a module boundary for `I_ORCA_TOP` and a module boundary for `I_ORCA_TOP/I_BLENDER_0`.

```
icc2_shell> explore_logic_hierarchy -create_module_boundary \
    -cell {I_ORCA_TOP}
icc2_shell> explore_logic_hierarchy -create_module_boundary \
    -cell {I_ORCA_TOP/I_BLENDER_0} -nested
```

2. (Optional) Experiment with different block placements by selecting a module boundary and moving it into the floorplan boundary as shown in [Figure 59](#).

Figure 59 Moving Module Boundaries



3. Reset the module boundary positions or remove the module boundaries to explore other hierarchies.

```
icc2_shell> explore_logic_hierarchy -organize
icc2_shell> explore_logic_hierarchy -remove
```

The `-organize` option moves the module boundaries outside the floorplan to the locations created by the `explore_logic_hierarchy -create_module_boundary`

command as shown in [Figure 58](#). The `-remove` option removes all module boundaries. You can also combine the `-remove` and `-cell` options to remove specific module boundaries.

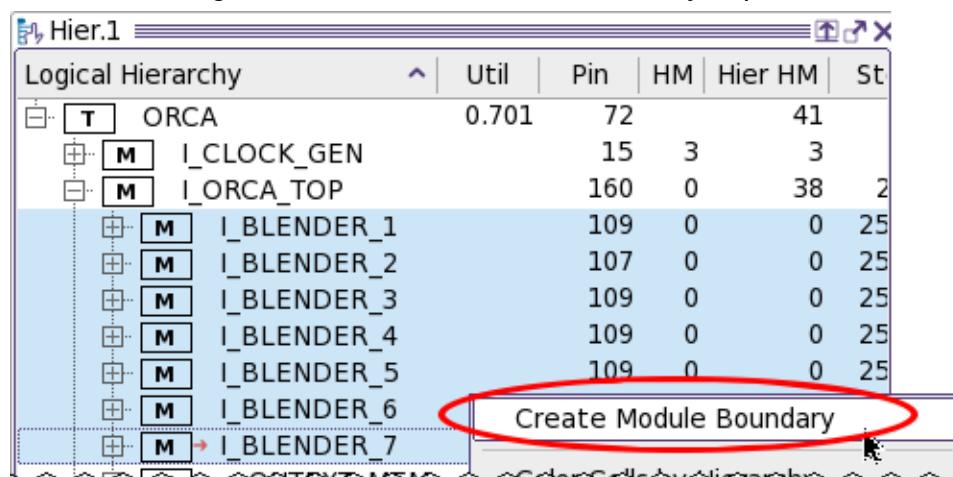
4. Repeat the flow by creating module boundaries for other cells to explore different hierarchies.

To create the initial module boundaries with the GUI,

1. Open the Hierarchy Exploration panel in the GUI by choosing View > Assistants > Hierarchy Exploration, or select Floorplan Preparation > Hierarchy Exploration in the Task Assistant.
2. Select the blocks for which to create module boundaries.
3. Click Create Module Boundary from the context menu as shown in [Figure 60](#).

The tool creates module boundary for the selected logic hierarchies.

Figure 60 Creating Module Boundaries in the Hierarchy Explorer



Creating Module Boundaries for Flat Designs

The `explore_logic_hierarchy` command can also create module boundaries for a flat design. Set the `plan.place.hierarchy_by_name` application option to `true`; this enables the `explore_logic_hierarchy` command to extract virtual hierarchies by name and create move bounds for the virtual hierarchies. Any existing module boundaries are removed before creating the new move bounds. To create individual move bounds for each module, specify the modules explicitly with the `-cell` option as follows:

```
icc2_shell> explore_logic_hierarchy -create_module_boundary \
    -cell {I_ORCA_TOP/I_PCI_TOP I_ORCA_TOP/I_SDRAM_TOP}
```

To create a single move bound to group two or more modules, specify the `-virtual_group` option as follows:

```
icc2_shell> explore_logic_hierarchy -name group1 \
    -virtual_group {I_ORCA_TOP/I_PCI_TOP I_ORCA_TOP/I_SDRAM_TOP}
```

Use the `get_attribute` command as follows to report the individual modules associated with the grouped move bound:

```
icc2_shell> get_attribute [get_bounds group1] -name flat_hier_names
```

Committing Design Blocks

In the IC Compiler II design planning flow, you commit logical hierarchy cells to physical hierarchy blocks early in the design process. You can use the connectivity of the cells, size of the cells, and other factors to determine which logical hierarchy cells to commit. After deciding which logical hierarchy cells to commit to physical hierarchy blocks, you can convert them to an abstract representation for placement, power planning, pin placement and timing budgeting. Interactive tools support your analysis by displaying the net connectivity between different modules.

To commit a block in the design,

1. (Optional) Create a new design library to contain the block and save the library. This enables each block to be processed in parallel by using distributed processing.

```
icc2_shell> create_lib ../lib/orca -technology orca.tf \
    -ref_libs $libs
icc2_shell> save_lib
icc2_shell> set_ref_libs -add orca
```

2. Commit the block. The block is saved to the specified design library.

```
icc2_shell> commit_block -library orca ORCA
```

In this example, the command creates a new physical hierarchy for the block named orca and writes the block to the orca design library. The GUI updates to show outline views for the committed blocks.

Creating Block Placement Abstracts

After committing the logical hierarchy cells to physical hierarchy blocks, you can create block placement abstracts in preparation for block shaping and initial macro placement. Block placement abstracts are lightweight representations of physical hierarchy blocks and contain hard macros, original standard cell count and area information, and enough standard cells and registers to represent the I/O interfaces of a block. To create the block

placement abstracts, you must load the full netlist representation of the top-level design, without the details for the committed blocks.

To create block placement abstracts,

1. Set the constraint mapping file.

```
icc2_shell> sh cat split/mapfile
BLENDER_0 CLKNET BLENDER_0/clocknets.tcl
BLENDER_0 SDC BLENDER_0/top.tcl
BLENDER_0 UPF BLENDER_0/top.upf
...
BLENDER_6 CLKNET BLENDER_6/clocknets.tcl
BLENDER_6 SDC BLENDER_6/top.tcl
BLENDER_6 UPF BLENDER_0/top.upf
ORCA CLKNET ORCA/clocknets.tcl
ORCA SDC ORCA/top.tcl
ORCA UPF TOP/top.upf
```

The constraint mapping file is generated automatically by the `split_constraints` command and specifies a list of blocks and their associated SDC and UPF constraint files. See [Split Constraints Output Files](#) for more information.

2. Verify that the currently loaded top-level block representation is “outline”. If the current representation is already “design”, proceed to step 3.

```
icc2_shell> current_design
{orca:ORCA.outline}
```

The `current_design` command reports the current block name (ORCA), representation (outline), and design library that contains the design (orca).

3. Expand the outline representation for the top-level block with the `expand_outline` command and save the design library.

```
icc2_shell> expand_outline
...
icc2_shell> current_design
{orca:ORCA.design}
icc2_shell> save_lib -all
```

In this example, the `expand_outline` command reads the netlist and expands the top level. Note that the representation reported by the `current_design` command is changed from “outline” to “design”.

Note:

For the command to function correctly, you must maintain the original Verilog file in the same location referenced by the `read_verilog_outline` command, or make the file available in the `search_path`.

4. Set the host options for distributed computing.

```
icc2_shell> set_host_options -name block_script host_settings
```

Note:

You can verify the current host option settings with the `check_host_options` command.

5. Create the block placement abstracts for all blocks by using distributed processing.

```
icc2_shell> create_abstract -placement \
    -host_options block_script -all_blocks
Submitting job for block BLENDER_0 ...
Submitting job for block BLENDER_1 ...
...
Running distributed create_abstract ...
...
1
```

Use the `abstract.allow_all_level_abstract` and `plan.flow.design_view_only` application options to control abstract creation for hierarchical blocks. When the `abstract.allow_all_level_abstract` application option is `true`, you can create an abstract view for any child block at any level of hierarchy if the `plan.flow.design_view_only` application option is `false` for that child block. When the `abstract.allow_all_level_abstract` application option is `false`, the tool only allows you to create abstract views for the lowest level block that has the `plan.flow.design_view_only` application option set to `false`.

You can create block placement abstracts without using distributed processing by omitting the `-host_options` option. To create block placement abstracts for only a subset of blocks, use the `-blocks {blocks}` option instead of the `-all_blocks` option.

Changing the Block Boundary and Block Origin

When you reshape a block by setting the `boundary` attribute or by changing the block shape or location with the GUI, the `origin` attribute does not change. The `origin` attribute specifies the block origin and might require modification if you reshape or relocate the block.

To change the block origin together with the block boundary, use the `set_block_boundary` command and specify the `-origin` option. If you omit the `-origin` option, the command places the origin at the lower-left corner of the bounding box for the block.

```
icc2_shell> set_block_boundary -cell U0 \
    -boundary {{2500 100} {3000 100} {3000 1000} {2500 1000}} \
    -origin {2500 100} -orientation R0
Information: Block instance U0 boundary is set. (DPP-057)
1
```

To change only the origin for the current top-level block, use the `move_block_origin` command. The location you specify is relative to the current origin. For example, to move the origin of the current top-level block to the center of the block, where the current origin is at the lower-left corner and the bounding box for the block is {0 0} {1000 1000}, use the following command:

```
icc2_shell> move_block_origin -to {500 500}
```

Moving Objects Between the Top and Block Levels

You can move objects such as cells, routes, route guides, routing corridors, blockages, and pin guides between the top and block levels as described in the following topics:

- [Pushing Objects Down From the Top Level to the Block Level](#)
- [Popping Objects Up From the Block Level to the Top Level](#)

You move objects to plan hierarchical interfaces using fully routed signals or wire stubs as pin markers and move routing objects that have no associated net.

Pushing Objects Down From the Top Level to the Block Level

To move objects from the top level to the block level, use the `set_push_down_object_options` and `push_down_objects` commands. By default, objects created by the `push_down_objects` command have a suffix of `_PUSHDOWN_n`. To assign a different suffix for port and net names, use the `plan.pins.new_port_name_tag` application option. To assign a different suffix for cell names, use the `plan.pins.new_cell_name_tag` application option.

To push objects down from the top level to the block level,

1. Specify the types of objects to move, and how to copy them, with the `set_push_down_object_options` command.

```
icc2_shell> set_push_down_object_options -object_type signal_routing \
           -block_action {copy create_pin_shape}
```

For information about the push down object options, see [Push Down Object Options](#).

2. (Optional) Verify the option settings with the `report_push_down_object_options` command.

```
icc2_shell> report_push_down_object_options \
           -object_type signal_routing
*****
Report : report_push_down_object_options
*****
--- push_down_objects options for signal_routing ---
```

```
Top action: remove
Block action(s): { copy create_pin_shape }
Routing overlap check: false
Ignore misalignment: false
```

3. (Optional) Check the nets for potential routing issues before pushing down to the block level with the `check_objects_for_push_down` command.

```
icc2_shell> check_objects_for_push_down \
    [get_nets -of_objects [get_cells CLK_BUFFER*]]
0 nets determined to have problems
Overall runtime: 0.008u 0.000s 0:00.01e 100.9%
```

The `check_objects_for_push_down` command checks the routes or charging stations to be pushed down for potential issues. The command checks for collinear routes and vias, problems with physical routing that are incompatible with the logical definition of the net, and incomplete routing at the top level. Charging stations are checked for potential UPF issues. The `check_objects_for_push_down` command uses the option settings specified by the `set_push_down_object_options` command when checking the design.

4. Push the specified objects from the top level to the block level with the `push_down_objects` command.

```
icc2_shell> push_down_objects \
    [get_nets -of_objects [get_cells CLK_BUFFER*]]
Examined 8 nets for push-down
Pushed down routing for 8 nets
Overall runtime: 0.008u 0.000s 0:00.01e 106.6%
```

When you push down routes and other objects from the top level or create feedthroughs on a block, a shadow netlist is created that contains the objects that were pushed down or created. Within the block that receives the new objects, the tool inserts the objects and

- Sets the `is_shadow` attribute on the objects to `true`

Feedthroughs that are pushed down from the top level have the `is_shadow` attribute set to `true` on the pins, nets, and buffers of the feedthrough net.

- Sets the `shadow_status` property to `pushed_down`

Other possible shadow status property values are `copied_down`, `copied_up`, `normal`, `pulled_up`, `pushed_down`, and `virtual_flat`.

See Also

- [Push Down Object Options](#)
- [Pushing Down Feedthrough Nets](#)

- [Using Via Locations as New Pin Locations During Push Down](#)
- [Writing Pushed Down Objects to the Netlist](#)

Push Down Object Options

Use the following options with the `set_push_down_object_options` command to control the behavior of the `push_down_objects` commands.

- Specify the object types with the `-object_type` option. The `set_push_down_object_options` command supports the following object types: `blockage`, `cells`, `charging_station`, `marker_layer`, `pg_routing`, `pin_blockage`, `pin_guide`, `routing_corridor`, `routing_guide`, and `signal_routing`.
- ```
icc2_shell> set_push_down_object_options -object_type {cells blockage}
```
- Specify how the object is treated at the top level with the `-top_action` option. Valid top actions are `keep` or `remove`.

```
icc2_shell> set_push_down_object_options -object_type blockage \
 -top_action keep
```

- Specify how the object is treated at the block level with the `-block_action` option. Valid block actions are `center_pin_from_wire`, `copy`, `create_blockage`, `create_pin_shape`, `create_route_blockage_from_cell`, and `retain_obsolete_ports`, depending on the object type specified with the `-object_type` option.

```
icc2_shell> set_push_down_object_options -object_type cells \
 -top_action remove -block_action {copy retain_obsolete_ports}
```

- Check for overlaps with existing routing objects with the `-routing_overlap_check` option. This option can be used when specifying the `pg_routing` and `signal_routing` object types with the `-object_type` option.

```
icc2_shell> set_push_down_object_options -object_type pg_routing \
 -block_action {create_pin_shape copy} \
 -routing_overlap_check true
```

- Issue a warning message when pushing down misaligned PG routing objects into multiply instantiated blocks with the `-ignore_misalignment` option.

```
icc2_shell> set_push_down_object_options -object_type pg_routing \
 -block_action {copy create_pin_shape} -ignore_misalignment true
```

- Push down multivoltage cells with multiple rails by specifying the `-allow_multi_rail_cells` option.

```
icc2_shell> set_push_down_object_options -object_type cells \
 -allow_multi_rail_cells true
```

- Treat parallel PG straps outside the block boundary but within the specified distance setting as collinear routes with the `-collinear_margin` option. Note that these PG straps are created in the child block as copies of the original top-level strap and the original strap is left untouched at the top level.

```
icc2_shell> set_push_down_object_options -object_type pg_routing \
 -block_action {copy create_pin_shape} -ignore_misalignment true \
 -collinear_margin 2.1 -pin_meet_fatwire_rule true
```

- Invoke fat-wire spacing rules with the `-pin_meet_fatwire_rule` option as shown in the previous example.
- Specify whether the command creates a unique pin name based on the layer and location of the pin with the `-location_based_terminal_naming` option.

## Pushing Down Feedthrough Nets

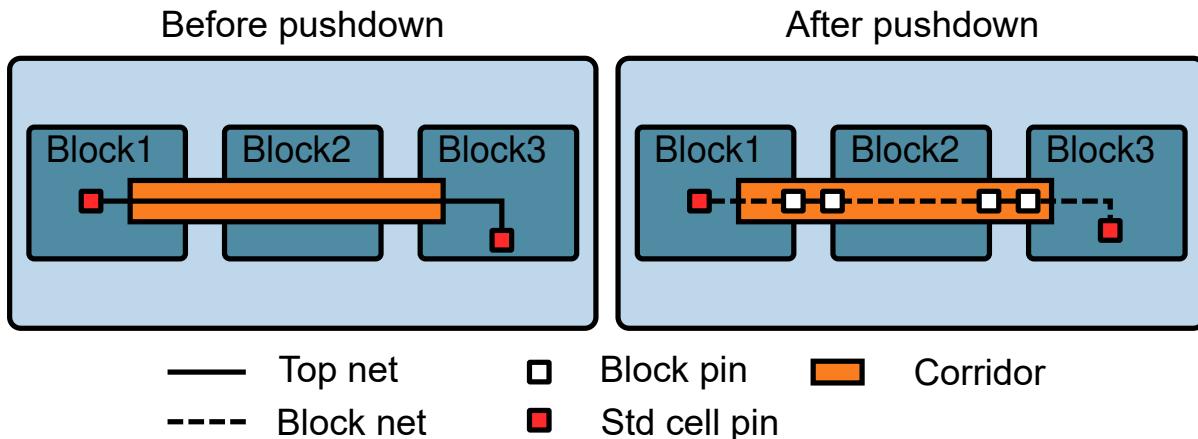
If a block contains feedthrough nets, the `push_down_objects` command creates an electrically equivalent port on the lower-level block that the feedthrough net crosses over. At the top level, the command connects the top-level net to the new electrically equivalent pin on the block. To create a feedthrough port in the lower-level block instead of an electrically equivalent port, specify the `-allow_feedthroughs true` option with the `create_pin_constraint` command. To allow feedthroughs on only certain types of nets during push down, apply a filter as shown in the following example:

```
icc2_shell> create_pin_constraint -type individual \
 -nets [get_nets * -filter "net_type == signal"] -allow_feedthroughs
 true
```

In the following example, the `push_down_objects` command pushes down all nets associated with routing corridor `corridor_a` into the blocks that are crossed by the net. Note that feedthrough creation must be enabled. [Figure 61](#) shows the design block diagram before and after pushing down the route. The tool creates a pure feedthrough in `Block2` and creates physical pins for detailed routed nets at the intersection of the net and the block boundaries.

```
icc2_shell> push_down_objects [get_nets \
 -of_objects [get_routing_corridors corridor_a]]
```

Figure 61 Pushdown Global Route



### See Also

- [Creating Individual Pin Constraints](#)

## Using Via Locations as New Pin Locations During Push Down

By default, the `push_down_objects` command creates pins on block edges as needed when pushing down net-related objects. To avoid creating the pin at the block edge and instead treat a via as the new pin location on the net being pushed down, perform the following steps. Note that using the via location as the pin location helps to limit the number of new pin locations created for designs that contain multiply instantiated blocks.

- Specify the `-block_action {copy create_pin_shape}` option to the `set_push_down_object_options` command.

```
icc2_shell> set_push_down_object_options -object_type signal_routing \
 -top_action remove -block_action {copy create_pin_shape}
```

- Push down the via into the block specified by the `-cells` option.

```
icc2_shell> push_down_objects [get_vias VIA_S_5] -cells u1/u2
...
Examined 1 nets for push-down
Pushed down routing for 1 nets
...
```

## Writing Pushed Down Objects to the Netlist

The `push_down_objects` command sets the `is_shadow` and `shadow_status` attributes to identify pushed-down objects.

- The `is_shadow` attribute is set to `true`.
- The `shadow_status` attribute is set to `pushed_down` or `copied_down`.

By default, the `write_verilog` command writes out the shadow netlist objects to the netlist for the lower-level block. You can specifically include or exclude these objects by using the `-include shadow_netlist` or `-exclude shadow_netlist` option with the `write_verilog` command.

To write out shadow information in the form of an ECO script instead of a shadow netlist, use the `write_shadow_eco` command. The ECO script can be used by a physical design tool like the IC Compiler II tool; the script can also provide useful information to a front-end tool such as the Design Compiler tool. Note that the Design Compiler tool consumes the ECO script differently than the IC Compiler II tool. Use the `-command_style icc2` option when targeting the IC Compiler II tool and use the `-command_style dc` option when targeting the Design Compiler tool.

Use the following steps to query the pushed-down objects in the lower-level block,

1. Open the block that contains the pushed-down objects.

```
icc2_shell> open_block lib/myblock:myblock.design
```

2. Return the list of pushed-down cells and nets.

```
icc2_shell> get_cells -filter is_shadow
{x0_mem0_ram0_0_x0_PUSHDOWN x0_mem0_ram0_0_x1_PUSHDOWN ...
icc2_shell> get_nets -filter is_shadow
{from_top_1_PUSHDOWN_0 ...
```

3. Examine the `shadow_status` attribute as needed on the objects returned by the previous command.

```
icc2_shell> get_attribute \
 [get_cells x0_mem0_ram0_0_x0_PUSHDOWN] shadow_status
pushed_down

icc2_shell> get_attribute [get_nets from_top_1_PUSHDOWN_0]
 shadow_status
pushed_down
```

### See Also

- [Pushing Objects Down From the Top Level to the Block Level](#)

## Popping Objects Up From the Block Level to the Top Level

To move objects from the block level to the top level, use the `set_pop_up_object_options` and `pop_up_objects` commands.

To pop objects up from the block level to the top level,

- Specify the types of objects to move, and how to copy them, with the `set_pop_up_object_options` command.

```
icc2_shell> set_pop_up_object_options -object_type blockage \
 -block_action keep
```

For information about the pop-up object options, see [Pop Up Object Options](#).

- (Optional) Verify the option settings with the `report_pop_up_object_options` command.

```
icc2_shell> report_pop_up_object_options -object_type blockage

Report : report_pop_up_object_options

--- pop_up_objects options for blockage ---
Block action: keep
```

- Pop up the placement blockage in a specific cell to the top level with the `pop_up_objects` command.

In the following example, the placement blockage from cell34 is popped up to the top.

```
icc2_shell> pop_up_objects [get_placement_blockages cell34/*]

--- Process cell34: Popping up blockages from block block1 ---
Block action: keep
Number of Placement blockages processed: 6
Overall runtime: 0.017u 0.000s 0:00.02e 103.7%
```

After objects are pulled up, the tool sets the `shadow_status` property to `pulled_up`. Other possible shadow status property values are `copied_down`, `copied_up`, `normal`, `pulled_up`, `pushed_down`, and `virtual_flat`.

## Pop Up Object Options

Use the following options with the `set_pop_up_object_options` command to control the behavior of the `pop_up_objects` command.

- Specify the object types with the `-object_type` option. Valid object types are `blockage`, `cells`, `marker_layer`, `pg_routing`, `pin_blockage`, `pin_guide`, `routing_guide`, and `signal_routing`.
 

```
icc2_shell> set_pop_up_object_options -object_type {cells blockage}
```
- Specify how the object is treated at the block level with the `-block_action` option. Valid block actions are `keep` or `remove`.
 

```
icc2_shell> set_pop_up_object_options -object_type cells \
 -top_action remove -block_action {keep}
```
- Check for overlaps with existing routing objects with the `-routing_overlap_check` option. This option can be used when specifying the `pg_routing` and `signal_routing` object types with the `-object_type` option.
 

```
icc2_shell> set_pop_up_object_options -object_type pg_routing \
 -block_action {keep} \
 -routing_overlap_check true
```
- Specify whether the `pop_up_objects` command creates top-level terminals at the parent level with the `-pins_as_terminals` option.

## Generating ECO Scripts for Netlist Editing

You can create an ECO script to make the following changes in the netlist. Using an ECO script provides convenience and enhanced capabilities.

- [Push Up the Branching of a Physical Multiple-Fanout Net to the Top Level](#)
- [Push Down a Tree of Standard Cells One Level](#)
- [Create Spare Ports and Nets on a Child Block](#)

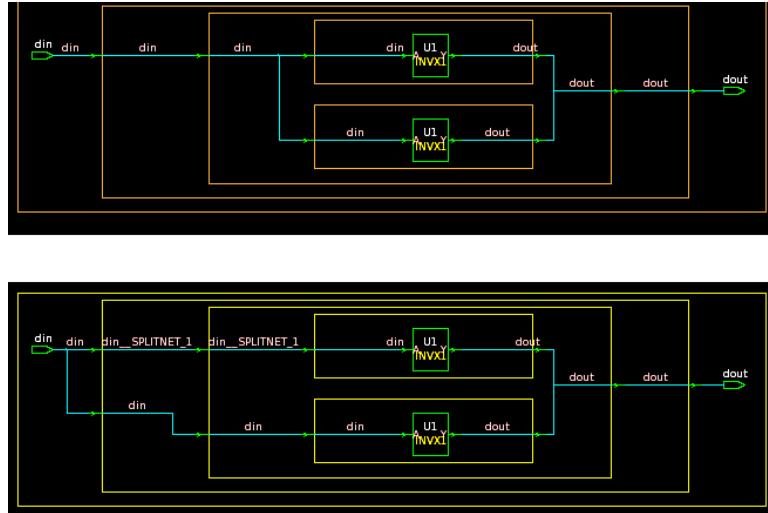
### Push Up the Branching of a Physical Multiple-Fanout Net to the Top Level

A physical multiple-fanout net is a net that connects to more than one physical block. To generate an ECO script to move the branch of a physical multiple-fanout net to the top level, use the `write_split_net_eco` command. By default, the command splits all physical multiple-fanout nets in the block. To split specific nets, specify the nets with the `-nets` option. The nets must connect to multiple physical block pins to be split. If a net qualifies to be split, the branching net is moved to the highest level of hierarchy possible. This command moves the branch for both logic wrappers and for physical nets.

By default, the command writes the ECO script to the console. To write the script to a Tcl file, specify the file name with the `-filename` option. Source the script by using the `source` command.

The following figures show the branching of a physical multiple-fanout net before and after being pushed up to the top level.

*Figure 62 Before and After Pushing to the Top Level*



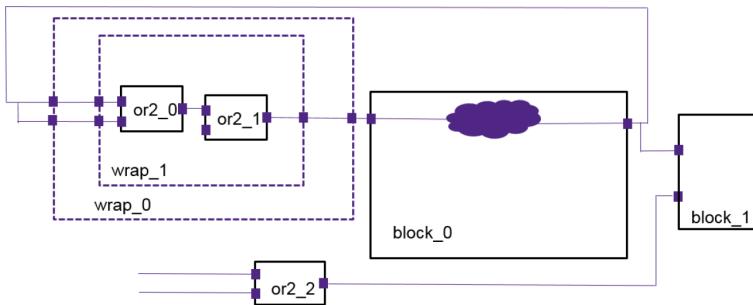
### Push Down a Tree of Standard Cells One Level

To generate an ECO script to move a tree of standard cells down one level from the top physical level to the overlapping blocks on the next physical level, use the `write_push_down_eco` command. The script provides a way to push down OR-gate trees; the `push_down_objects` command does not have this capability.

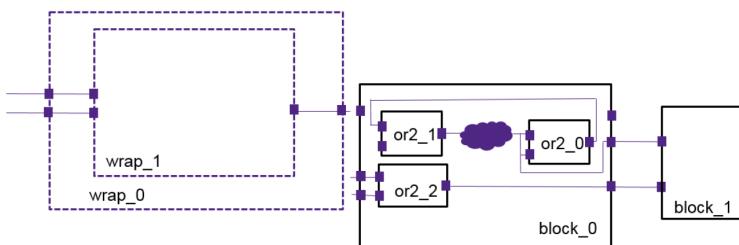
When you use the `write_push_down_eco` command, specify the cells to be pushed down. A cell can be pushed down into a block only when it completely fits within the boundary of that block.

As the tool pushes down the standard cells into a block, it creates new ports on the block. The original ports are not reused. Any unused input ports are tied to ground (VSS).

In the following figure, all three or2 gates overlap with block\_0 in the physical floorplan:



After sourcing the ECO script, the cells are pushed down into the block\_0 block:



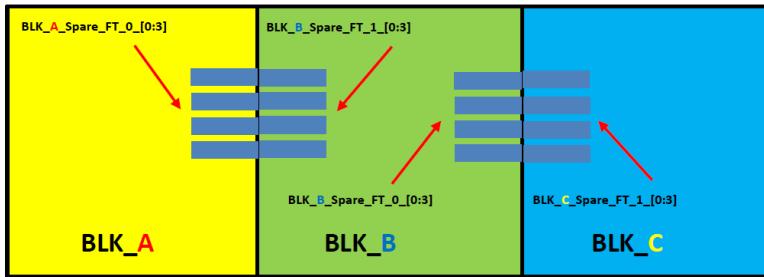
The `write_push_down_eco` command generates an ECO file for each block. By default, the ECO files are written to a directory named `push_down_eco_scripts`. You can specify the directory name by using the `-dir` option. To run the ECO scripts for all the blocks, source the top-level block Tcl script with the `source` command.

### Create Spare Ports and Nets on a Child Block

To generate an ECO script to create spare (dummy) ports and nets on a child block, use the `write_spare_ports_eco` command and specify

- A list of cells by using the `cell_list` argument.
- The name of the ports by using the `-name` option. By default, the command uses `dummyFT`.
- The direction of the ports by using the `-direction` option and specifying `in`, `out`, or `inout`. The default is `in`.
- The bus width by using the `-bits` option.

After creating the ports and nets, the command creates the top-level nets and connects them to the child block's pins.



The `write_spare_ports_eco` command generates an ECO file for each block. By default, the ECO files are written to a directory named `create_spare_ports`. You can specify the directory name by using the `-dir` option. To run the ECO scripts for all the blocks, source the top-level block Tcl script with the `source` command.

The following example writes out a script for the `U*` cell instances and specifies new ports named `spareFT`:

```
icc2_shell> write_spare_ports_eco [get_cells U*] -direction in \
 -dir test_path -bits {0 4} -name spareFT
```

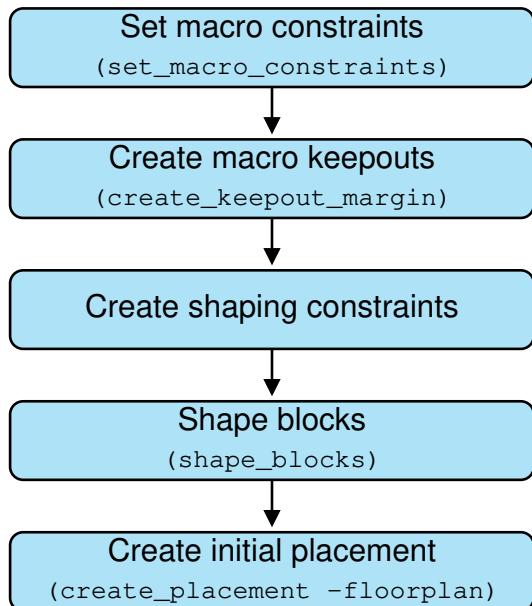
## 9

## Performing Block Shaping and Macro Placement

The block shaping flow refines the boundary for the block based on the rough rectangular or rectilinear shape defined during hierarchy exploration. When creating the block shape, the tool considers design constraints such as target utilization for the block, channel width and keepout settings, while minimizing feedthroughs and interface wire lengths. You can create an optional block grid for your design; the tool aligns block shapes to the grid.

The flow to shape blocks is shown in [Figure 63](#).

*Figure 63 Block Shaping Flow*



For more details, see the following topics:

- [Setting Macro Constraints](#)
- [Creating Relative Placement Constraints for Macros and Macro Arrays](#)
- [Setting Macro Keepouts](#)

- [Reserving Space for Macros Between Connected Blocks](#)
- [Creating Macro Arrays](#)
- [Shaping Blocks](#)
- [Performing Block Shaping With Tcl Constraints](#)
- [Creating Channel Constraints for Block Shaping](#)
- [Creating the Block Grid for Multiply Instantiated Blocks](#)
- [Creating the Initial Macro Placement](#)
- [Placing Macros Island Style](#)
- [Placing Macros Away From Block Edges](#)
- [Placing Macros Among Standard Cells](#)
- [Using the Macro Placement Assistant](#)
- [Performing Timing-Driven and Congestion-Driven Placement](#)

## Setting Macro Constraints

Before block shaping, you can specify macro placement constraints and keepout margins for some or all blocks and macros. Placement constraints limit the orientations in which macros can be placed. To set macro placement constraints,

1. Create a collection of macro cells. The following example assigns a variable to a collection containing all macro cells in the design.

```
icc2_shell> set macro_cells [get_cells -physical_context \
 -filter "is_hard_macro && !is_physical_only" -quiet]
```

2. Specify the `set_macro_constraints` command with the appropriate arguments. The following example limits the placement orientation for all hard macros to R0 or R180.

```
icc2_shell> set_macro_constraints \
 -allowed_orientations {R0 R180} $macro_cells
```

3. (Optional) Use the `report_macro_constraints` command to verify the current constraint settings. Use one or more of the `-allowed_orientations`, `-preferred_location`, `-alignment_grid`, and `-align_pins_to_tracks` options to `report_macro_constraints` to limit the report to only those constraints.

```
icc2_shell> report_macro_constraints

Report : report_macro_constraints
Design : ORCA
```

```

macro allowed legal
name orientations orientations

I_ORCA_TOP/I_RISC_CORE/I_REG_FILE/REG_FILE_D_RAM
 R0,R180 all
I_ORCA_TOP/I_RISC_CORE/I_REG_FILE/REG_FILE_C_RAM
 R0,R180 all
...

```

Use the following options with the `set_macro_constraints` command to constrain the placement of hard macros and I/O cells in the design:

*Table 4 Specifying Macro Constraint Options*

| To do this                                                                                                                                                                                                                                               | Use this option                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------|
| Specify a list of allowed orientation values that further restrict legal placement orientations specified in the reference library                                                                                                                       | <code>-allowed_orientations {list}</code>              |
| Place macros by aligning the signal pins of the macro to the tracks for the layer the pin shape belongs to                                                                                                                                               | <code>-align_pins_to_tracks</code>                     |
| Based on your requirement, use either the <code>-align_pins_to_tracks</code> or <code>-alignment_grid grid_name</code> option; not both                                                                                                                  |                                                        |
| Align macros to a specified placement grid                                                                                                                                                                                                               | <code>-alignment_grid grid_name</code>                 |
| Based on your requirement, use either the <code>-alignment_grid grid_name</code> or <code>-align_pins_to_tracks</code> option; not both                                                                                                                  |                                                        |
| Only apply the <code>-alignment_grid</code> or <code>-align_pins_to_tracks</code> option specified in the same command to the specified macros placed in the specified orientation set, either R0: {R0, R180, MX, MY}, or R90: {R90, R270, MXR90, MYR90} | <code>-alignment_orientation_set R0   R90   all</code> |
| Assign a location on the macro that must be on-grid. If you do not specify the coordinates, they default to {0 0}.                                                                                                                                       | <code>-alignment_point {x y}</code>                    |
| Use this option with the <code>-alignment_grid grid_name</code> option. Otherwise, the tool ignores the <code>-alignment_point {x y}</code> setting.                                                                                                     |                                                        |
| Specify a preferred placement location for a specified macro, where <code>x_range</code> and <code>y_range</code> are values from zero to one                                                                                                            | <code>-preferred_location {x_range y_range}</code>     |

**Table 4 Specifying Macro Constraint Options (Continued)**

| To do this                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Use this option                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| <p>Specify the style used to place each macro when the <code>plan.macro.style</code> application option is set to <code>hybrid</code>.</p> <p>The valid values are <code>auto</code>, <code>on_edge</code>, and <code>freeform</code>. The default is <code>auto</code>, which means that the tool decides whether to place the macros along the edges or in freeform style. The automatic selection is based on the size of the macros and the number of routing layers that are blocked. The tool limits the on-edge selection to only those macros that should not be in the standard cell area.</p> <p>For more information about the <code>plan.macro.style</code> application option, see the man page.</p> | <code>-style on_edge   freeform   auto</code> |

## Creating Relative Placement Constraints for Macros and Macro Arrays

During design planning, macros and standard cells are placed by using the `create_placement -floorplan` command. To constrain a macro or macro array to be placed with respect to another object in the design, use the `set_macro_relative_location` command. The `create_placement -floorplan` command honors the constraints set by the `set_macro_relative_location` command and places the target macro or macro array according to the specified offset distance from the anchor object.

To create the constraint for macro placement,

1. Set the constraint with the `set_macro_relative_location` command.

```
icc2_shell> set target .../I_RISC_CORE/I_REG_FILE/REG_FILE_D_RAM
icc2_shell> set anchor .../I_RISC_CORE/I_REG_FILE/REG_FILE_B_RAM
icc2_shell> set_macro_relative_location \
 -target_object [get_cells $target] \
 -target_orientation R0 -target_corner b1 \
 -anchor_object [get_cells $anchor] -anchor_corner tl \
 -offset {0.0000 35.2550} -offset_type fixed
```

2. (Optional) Verify that the constraint was set correctly with the `report_macro_relative_location` command.

```
icc2_shell> report_macro_relative_location

Report : report_macro_relative_location
Design : ORCA
```

```

Macro relative location constraints in block ORCA/dp:

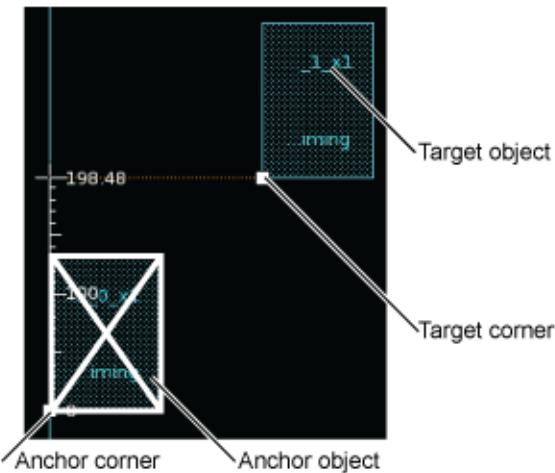
target target target anchor anchor anchor offset...
name orientation corner name type corner X ...

.../I_RISC_CORE/I_REG_FILE/REG_FILE_D_RAM
 R0 bl .../I_RISC_CORE/I_REG_FILE/REG_FILE_B_RAM
 macro tl 0.0000 ...

```

**Figure 64** shows the layout after running the `set_macro_relative_location` and `create_placement -floorplan` commands.

**Figure 64    Macro Relative Location**



The `set_macro_relative_location` command constrains the placement of an object with respect to the anchor object. If the anchor object is not specified, the constraint is with respect to the parent-level design that contains the target object. The following anchor objects are supported:

- Hard Macro
- Macro Array
- Block
- I/O Pad
- Block Pin
- Core Area
- Move Bound
- Voltage Area

You can change the relative location constraints with the following

`set_macro_relative_location` command options:

- Specify which corner of the anchor object to use when creating the offset between the anchor object and target object; valid corner values are bl (bottom left), br (bottom right), tl (top left), and tr (top right).

```
icc2_shell> set_macro_relative_location -anchor_corner bl ...
```

- Specify the instance name of the object to use as the reference location for the target object.

```
icc2_shell> set_macro_relative_location -anchor_object {u1/u2} ...
```

- Specify the x- and y-offset for the target object with respect to the anchor object.

```
icc2_shell> set_macro_relative_location -offset {200 200} ...
```

- Specify the corner of the target macro to apply the constraint; valid corner values are the same as for the `-anchor_corner` option.

```
icc2_shell> set_macro_relative_location -target_corner tr ...
```

- Specify the instance name of the target macro to constrain.

```
icc2_shell> set_macro_relative_location -target_object {u3/u4} ...
```

- Specify the orientation of the target macro; valid orientations are R0, R90, R180, R270, MX, MXR90, MY, and MYR90. The orientation that you specify must be an allowed orientation for the macro.

```
icc2_shell> set_macro_relative_location -target_orientation R0 ...
```

- Specify a scalable offset ratio, positive or negative, from the anchor object.

```
icc2_shell> set_macro_relative_location -offset_type scalable \
-offset {0.5 0.5} ...
```

The scalable offset is calculated using the following formula:

$$\text{target position} = \text{anchor position} + \text{offset} * (\text{scale\_edge\_length} - \text{used\_length})$$

- Specify different offset types for the x- and y-directions.

```
icc2_shell> set_macro_relative_location \
-offset_type {fixed scalable} ...
```

## Setting Macro Keepouts

Keepout margins maintain spacing around blocks and macros and help prevent congestion and DRC errors. The following example creates a collection of all hard macros,

then uses the `create_keepout_margin` command to create four types of keepout margins:

```
icc2_shell> set all_hm [get_cells -hierarchical \
 -filter "is_hard_macro==true"]
icc2_shell> create_keepout_margin -type hard \
 -tracks_per_macro_pin 0.05 $all_hm
icc2_shell> create_keepout_margin -type hard_macro \
 -tracks_per_macro_pin 0.5 $all_hm
icc2_shell> create_keepout_margin -type routing_blockage \
 -layers "M2 M3 M4" -tracks_per_macro_pin 0.05 $all_hm
icc2_shell> create_keepout_margin -type soft \
 -tracks_per_macro_pin 1.0 $all_hm
```

The `create_keepout_margin` command supports several options to control the keepout margins. To create a keepout margin around the outside of the block or macro, use the `-outer {left bottom right top}` option. To create a keepout margin around the inside of the block or macro, use the `-inner {left bottom right top}` option. To create a keepout margin based on a specified number of wire tracks, use the `-tracks_per_macro_pin tracks` option. When you use this option, the command multiplies the number of tracks you specify by the number of pins on that side of the macro to calculate the keepout margin size. You can further restrict the size of the keepout by specifying the `-min_padding_per_macro_size` and `-max_padding_per_macro_size` options together with the `-tracks_per_macro_pin` option. To specify the layers on which to apply a route\_blockage keepout, use the `-layers {layer_list}` option.

The keepout is not a placement blockage, but behaves like one and moves with the hard macro.

The tool supports four types of keepout margins: hard, soft, hard\_macro, and routing\_blockage. The following table summarizes these four types.

**Table 5** Summary of Keepout Types

| Keepout Type     | Description                                                                                                                                                                                                                                                                                                                                            |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hard (default)   | Prevents standard cells from being placed within the keepout margin.                                                                                                                                                                                                                                                                                   |
| soft             | Prevents standard cells from being placed within the keepout margin. However, the placer can move cells into the keepout during optimization.                                                                                                                                                                                                          |
| hard_macro       | Prevents the placement of hard macros or other hard macro keepout margins within the keepout margin. This type affects only hard macro placement and ensures a minimum channel between hard macros. The entire macro and keepout are kept within the site array. You can use this keepout type to allocate space for routing to the pins of the macro. |
| routing_blockage | Prevents the power and ground (PG) router from inserting via arrays that block access to macro pins. This type requires a list of layers to include in the blockage.                                                                                                                                                                                   |

## Reserving Space for Macros Between Connected Blocks

The number of available routes between blocks can be reduced if macros are placed near the block edges. To control how macros are placed near block edges with the `create_placement -floorplan` command, set the `plan.macro.cross_block_connectivity_planning` application option on the block to one of the following values:

- `hard`: The tool analyzes the connectivity between subblocks and avoids placing macros in a section of the boundary between blocks. The free edge can be used for pin placement later in the flow.
- `soft`: This setting is similar to the `hard` constraint setting, but creates a soft constraint which can be violated as needed.
- `unset`: The application option setting is inherited from the parent block.
- `false`: This constraint setting disables cross-block connectivity planning.

The following example creates a hard constraint on block BLK1 to enable cross block connectivity planning.

```
icc2_shell> set_app_options -block BLK1 \
 -name plan.macro.cross_block_connectivity_planning -value hard
```

You can perform similar connectivity planning on move bounds and voltage areas by setting the `interface_connectivity_planning` attribute on the object. The attribute accepts the same values as the `plan.macro.cross_block_connectivity_planning` application option. The following example sets a hard interface connectivity planning constraint on voltage area VA1.

```
icc2_shell> set_attribute -name interface_connectivity_planning \
 -value hard [get_voltage_areas VA1]
```

## Creating Macro Arrays

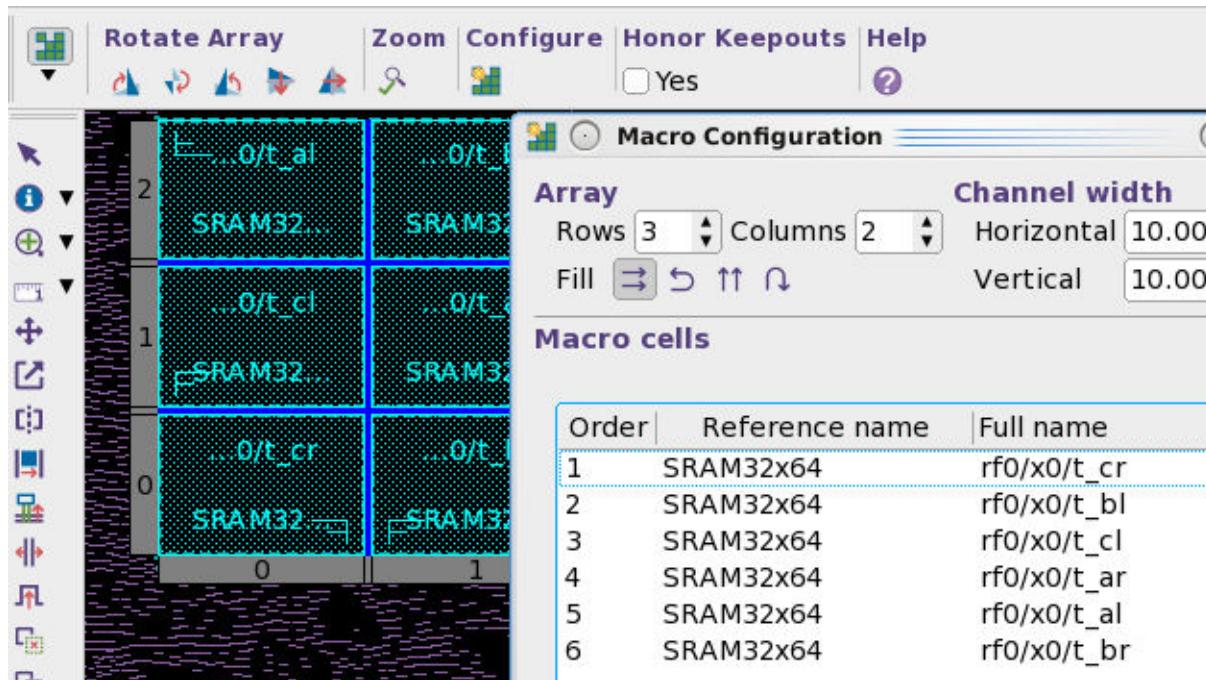
Macro arrays can be used to manage two or more macros as a single unit. Use macro arrays to enforce orientation, consistent channel width, macro flipping, and fill pattern for all macros in the group. To create a macro array, specify the `create_macro_array` command with the number of rows and columns in the array.

```
icc2_shell> create_macro_array -num_rows 2 -num_cols 6 [get_selection]
{MACRO_ARRAY_0}
```

After creating a macro array, you can use the macro array tool in the GUI to rotate and swap macros. You can also rotate the entire macro array with the macro array tool. To

start the macro array tool, select two or more macros in the macro array and choose Edit > Macro Array from the menu. The macro array editor is shown in the following figure.

*Figure 65 Macro Array Editor*



To remove a macro array, use the `remove_edit_groups` command to remove the edit group associated with the macro array.

```
icc2_shell> remove_edit_groups {MACRO_ARRAY_0}
```

#### See Also

- [Creating the Initial Macro Placement](#)

## Shaping Blocks

After committing the logical hierarchy cells to physical hierarchy blocks, you can create a rough placement of the physical hierarchy blocks. The tool shapes and places the physical blocks, including power domains and voltage areas, based on the specified utilization, channel size and constraints.

To place and shape the committed physical hierarchy blocks,

1. Set the block placement options with the `set_shaping_options` command.

```
icc2_shell> set_shaping_options -keep_top_level_together true
```

Any options set with the `set_shaping_options` command remain active during the session. You can use the `report_shaping_options` command to validate the current shaping settings.

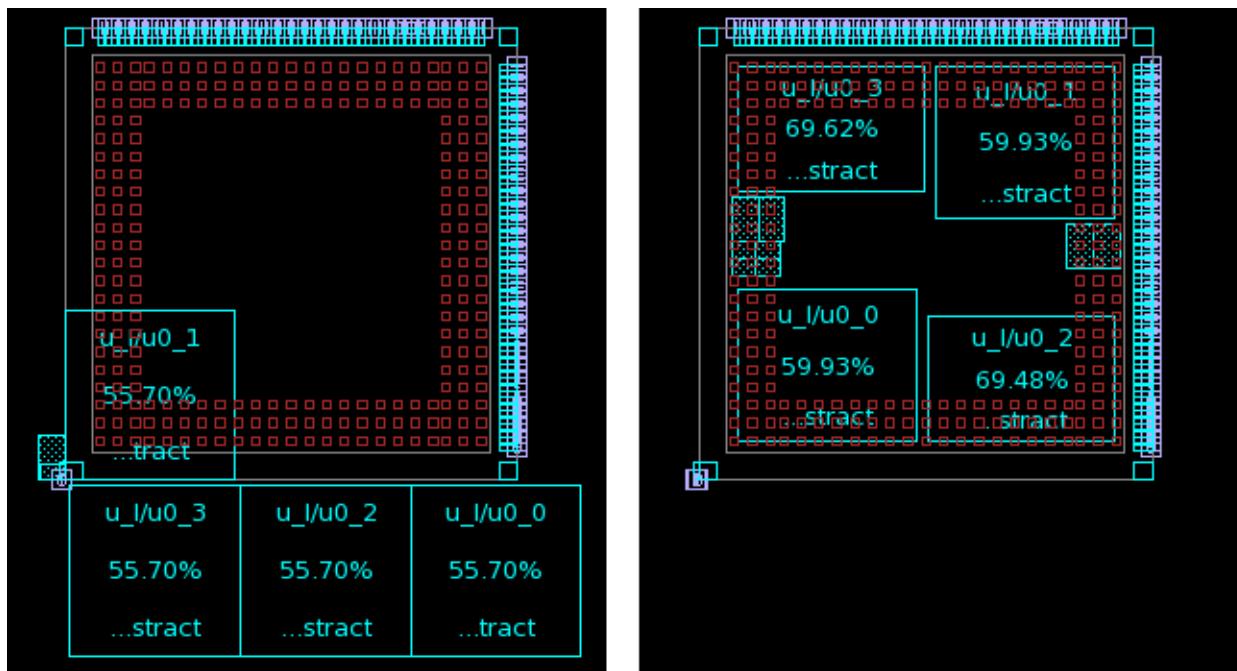
2. Shape and place the blocks with the `shape_blocks` command.

```
icc2_shell> shape_blocks -channels true
```

In this example, the `-channels true` option directs the command to insert channels between blocks. Note that `shape_blocks` command options are not persistent.

[Figure 66](#) shows the floorplan before and after block shaping.

*Figure 66 Before (left) and After (right) Block Shaping*



During block shaping, the tool minimizes the number of feedthroughs and minimizes interface wire lengths. The command uses macro packing to place the top-level macros. The tool also aligns the block shape to the block grid, if specified. If you created block placement abstracts as described in [Creating Block Placement Abstracts](#), you can use the abstracts to reduce the runtime required for placement.

3. Report any violations that occurred during block placement with the `report_block_shaping` command.

```
icc2_shell> report_block_shaping -core_area_violations -overlaps \
-flyline_crossing

Report : report_block_shaping
Design : ORCA

Block core area violation report
=====
Core area violations in design ORCA: 0

Block overlap report
=====
Block overlaps in design ORCA: 0

Block crossing flyline report
=====
Maximum flyline crossings is between block pairs
(I_ORCA_TOP/I_BLENDER_2 I_ORCA_TOP/I_BLENDER_1) and
(I_ORCA_TOP/I_BLENDER_7 I_ORCA_TOP/I_BLENDER_4): 4x19 = 76
Flyline crossings in design ORCA: 400
1
```

The `report_block_shaping` command produces a quality of results (QoR) report for the current block shaping result. Use the command to check for block overlaps, blocks outside the core boundary, excessive flyline crossings. The command also produces estimates for the number of unaligned pins and net detours.

4. (Optional) Repeat the flow from step 1 to adjust the shaping options and produce a higher quality result.

The `shape_blocks` commands supports different options to modify and refine the shaping result. Use the `-channels true | false` option to specify whether the command inserts channels between different blocks or between the blocks and the core boundary. Use the `-constraint_file file_name` option to apply utilization, preferred locations, grouping and fixed shape constraints for each block. Use the `-incremental congestion_driven | target_utilization_driven` option to apply incremental shaping with a priority on congestion or target utilization.

## Creating Block Shaping Constraints

To specify preferred block and voltage area placement locations for the `shape_blocks` command, create a block shaping constraint file and specify the file with the `-constraint_file` option. The tool sizes and places the specified regions based on the target utilization, channel size, and other constraints you specify in the file.

The shaping constraints file supports the following syntax:

```

Global shaping constraints
utilization { target: 0.2; }
guard_band { width: 0; height: 0; }
channel_size { size=5; }

TOP constraints
block constraints cannot be nested
block TOP {
 # Define channel constraints for TOP
 # and blocks within TOP
 # Channel constraints can be defined for
 # block_inst, voltage_area, group, or current_block
 channel_size {
 # Define constraint for TOP/U1
 block_inst U1;
 # Default channel is 20 micron
 # You can also specify "min = 20, max = 20"
 size = 20;
 # Left and right channels are between 30 and 50 microns
 left, right: min = 30, max = 50;
 }
 # Create a guard band constraint for voltage area VA1
 # of 50 microns, and set the voltage area boundary
 # to {{0 0} {300 600}}
 voltage_area VA1 {
 contents: voltage_area VA2, block_inst U3;
 # Define spacing around voltage area VA2
 guard_band {
 width: 50; height: 50;
 }
 boundary {
 type: rigid;
 shape: {{0 0} {300 600}};
 }
 }
 # Define spacing around any other voltage area
 # beneath TOP
 guard_band {
 width: 40;
 height: 40;
 }
 # Define a group that contains a set of blocks or
 # voltage areas within TOP
 define_group GROUP1 {
 # Organize blocks and voltage areas in a row or column
 # Legal directions are:
 # east (from left to right)
 # west (from right to left)
 # north (from bottom to top)
 # south (from top to bottom)
 arrange_in_array {
 }
}

```

```

 contents: block_inst U1, block_inst U2, voltage_area VA3;
 direction: east;
 }
 # Specify a relative placement location
 # for instance U4 in the middle of TOP block,
 # alignment_point is optional
 arrange_in_box {
 block_inst U4;
 location: x = 0.5, y = 0.5;
 alignment_point: x = 1, y = 0;
 }

 # aspect_ratio specifies the relative width
 # and height. Use aspect_ratio together with
 # arrange_in_box
 aspect_ratio {
 block_inst U1;
 target: width = 1, height = 2;
 }
}
}

BLOCK2 constraints
block BLOCK2 {
 # Reserve a space between the current block
 # boundary and any regions shaped inside the block
 boundary_channel_size {
 current_block;
 left: min = 3, max = 5;
 right, bottom: min = 5, max = 10;
 }
 # Specify a 50% target utilization
 # and a maximum of 70%
 utilization {
 target: 0.5;
 max: 0.7;
 }
}
}

```

The following example uses the shaping constraints file to organize blocks in the design. The shaping constraints file specifies two groups that contain five blocks each. The five blocks are stacked horizontally, and the two groups are stacked vertically. A guard band of 100 is applied between the blocks.

```

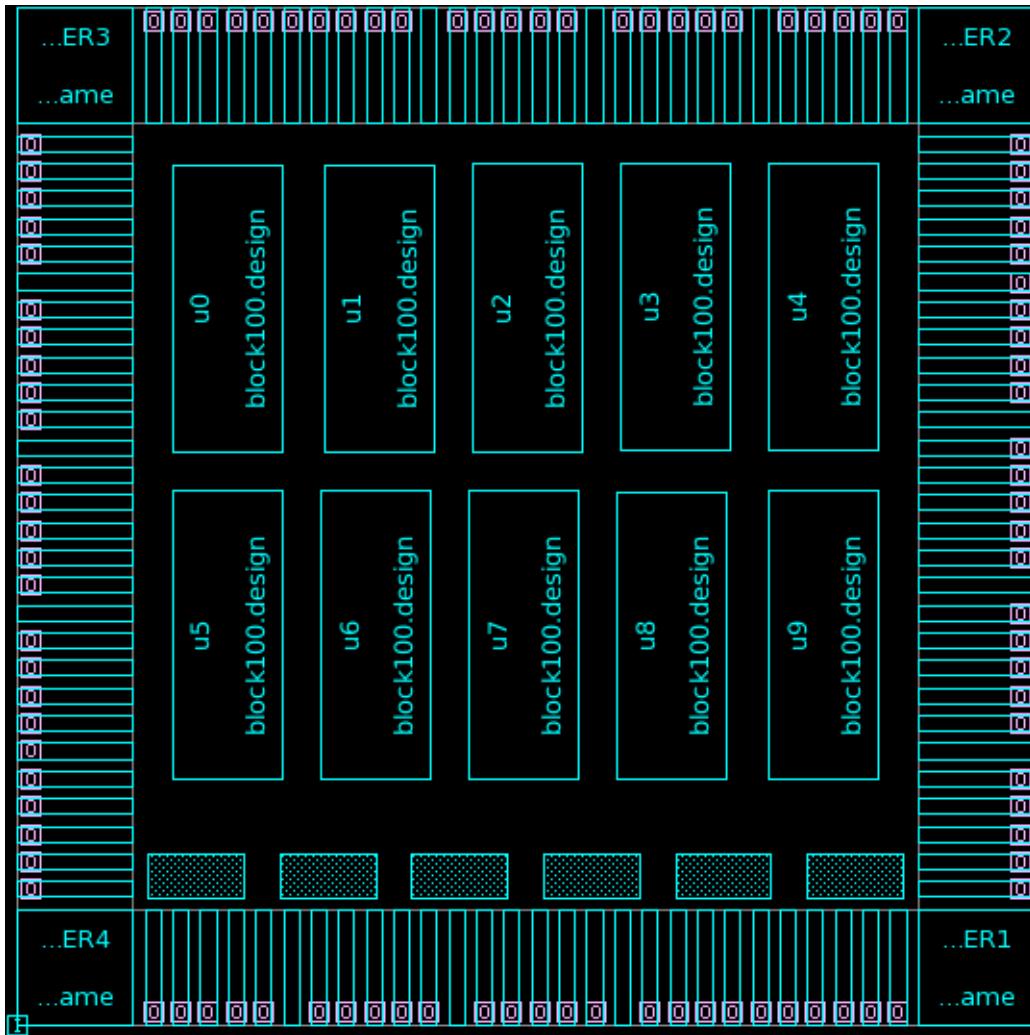
icc2_shell> sh cat shaping.con
Shaping constraint file
channel_size { size=100; }
block top {
 define_group UPPER {
 arrange_in_array {
 contents: block_inst u0, block_inst u1, block_inst u2,
 block_inst u3, block_inst u4;

```

```
 direction: east;
 }
}
define_group LOWER {
 arrange_in_array {
 contents: block_inst u5, block_inst u6, block_inst u7,
 block_inst u8, block_inst u9;
 direction: east;
 }
}
define_group TOP {
 arrange_in_array {
 contents: group UPPER, group LOWER;
 direction: south;
 }
}
icc2_shell> shape_blocks -constraint_file shaping.con
```

Figure 67 shows the layout after running the `shape_blocks` command.

Figure 67 Layout With Shaping Constraints



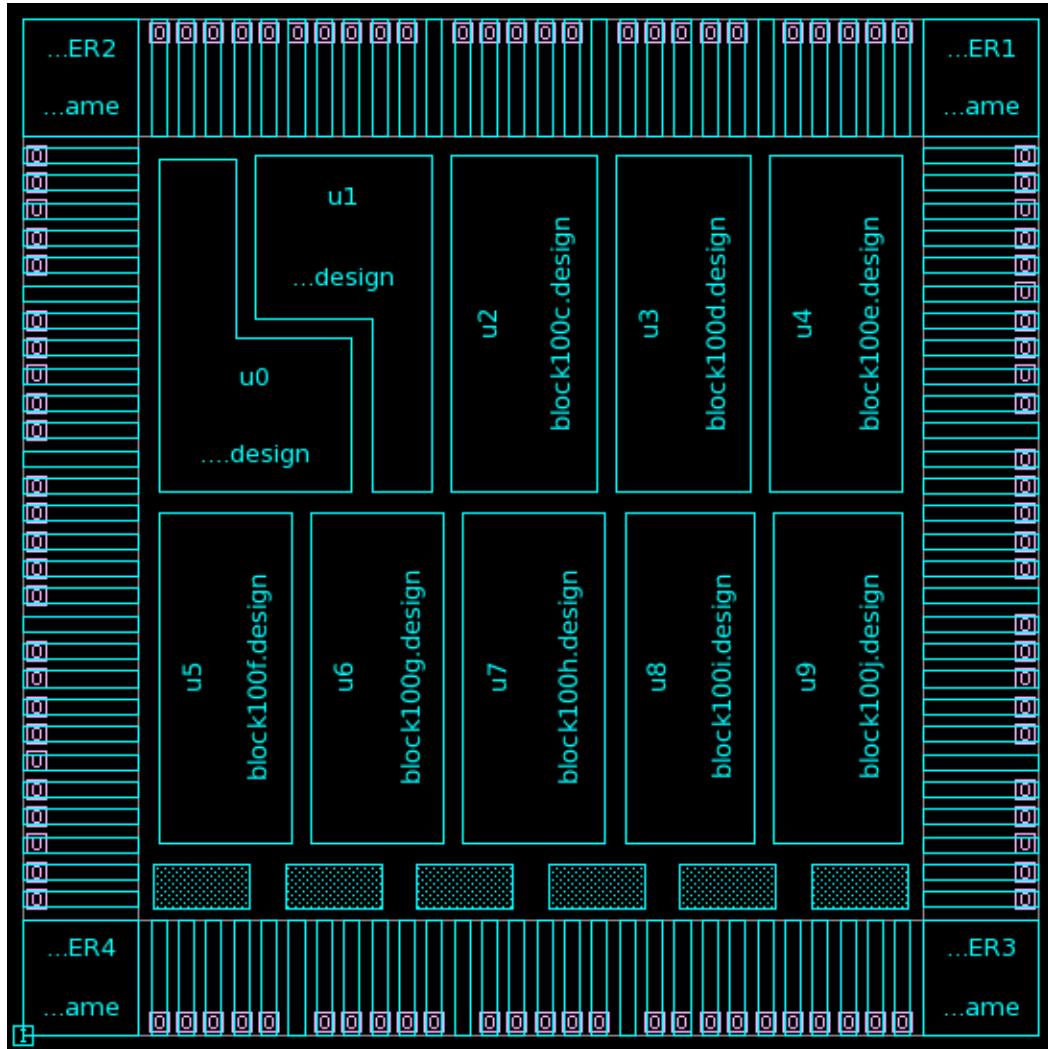
The following example specifies a fixed boundary for a block in the design. The boundary is specified with the `shape:` and `type: rigid` constraints. The coordinates are relative and the tool can move the shape to a new location in the parent block. The location of the block within the design is specified by `arrange_in_array` or `arrange_in_box` constraints.

```
icc2_shell> sh cat fixedblock.con
Shaping constraints with fixed block
channel_size { size=50; }
block block100a {
 boundary {
 type: rigid;
 shape: {{350 500} {350 1360} {550 1360}
 {550 900} {850 900} {850 500}};
 }
}
```

```
 }
 block top {
 define_group UPPER {
 arrange_in_array {
 contents: block_inst u0, block_inst u1, block_inst u2,
 block_inst u3, block_inst u4;
 direction: east;
 }
 }
 define_group LOWER {
 arrange_in_array {
 contents: block_inst u5, block_inst u6, block_inst u7,
 block_inst u8, block_inst u9;
 direction: east;
 }
 }
 define_group TOP {
 arrange_in_array {
 contents: group UPPER, group LOWER;
 direction: south;
 }
 }
 }
```

Figure 68 shows the layout after running the `shape_blocks` command. Note the shape of the block100a block, instantiated in the design as instance u0.

Figure 68 Layout With Fixed Boundary



## Performing Block Shaping With Tcl Constraints

In addition to the block shaping constraint file, the tool supports Tcl-based constraints that work with the `shape_blocks` command to specify constraint-based block, blockage, channel, move bound, and voltage area shaping. You create these constraints by specifying Tcl commands or by using the GUI to create the constraints in the layout.

Shaping constraints can be applied to single objects or groups of heterogeneous objects. Groups can also contain other groups to form a hierarchy. Use the `create_group` command to create a group that contains the objects in the group and the `get_groups` and `report_groups` commands to query the groups.

The following example creates a specific arrangement of ten blocks as shown in [Figure 69](#). Blocks u1, u2, u3, u4 and u5 are grouped and arranged left-to-right in the top half of the layout. Blocks u6, u7, u8, u9, and u10 are arranged left-to-right and placed at the bottom half of the layout.

1. Set the `plan.shaping.import_tcl_shaping_constraints` application option to `true` to specify that the `shape_blocks` command should honor the Tcl-based constraints.

```
icc2_shell> set_app_options \
 -name plan.shaping.import_tcl_shaping_constraints -value true
```

2. (Optional) Set the `plan.shaping.report_import_constraints` application option to `file`, `log`, or `both` to write a detailed summary of the block-shaping constraints to the console, log file, or both. The application option is `off` by default.

3. Create one group named `TOP_ROW` to contain the top set of blocks and another group named `BOTTOM_ROW` to contain the bottom set of blocks.

```
icc2_shell> create_group -name TOP_ROW \
 -shaping [get_cells {u1 u2 u3 u4 u5}]
{TOP_ROW}
icc2_shell> create_group -name BOTTOM_ROW \
 -shaping [get_cells {u6 u7 u8 u9 u10}]
{BOTTOM_ROW}
```

4. Create a parent group named `TOP_AND_BOTTOM` to contain the `TOP_ROW` and `BOTTOM_ROW` groups.

```
icc2_shell> create_group -name TOP_AND_BOTTOM \
 -shaping [get_groups -shaping {TOP_ROW BOTTOM_ROW}]
{TOP_AND_BOTTOM}
```

#### Note:

You can modify the order of the objects within a shaping group by using the `set_shaping_group_order` command. The command allows you to move specific objects or specify a new order for the entire group.

5. Create one shaping constraint for the `TOP_ROW` group and another shaping constraint for the `BOTTOM_ROW` group to specify the type of layout for each group. The “east” array layout orders the objects from left-to-right in the order they were specified by the `-shaping` option of the `create_group` command.

```
icc2_shell> create_shaping_constraint [get_groups \
 -shaping TOP_ROW] -type array_layout -array_layout east
{TOP_ROW/SHAPING_CONSTRAINT_0_array_layout}
icc2_shell> create_shaping_constraint [get_groups \
 -shaping BOTTOM_ROW] -type array_layout -array_layout east
{BOTTOM_ROW/SHAPING_CONSTRAINT_1_array_layout}
```

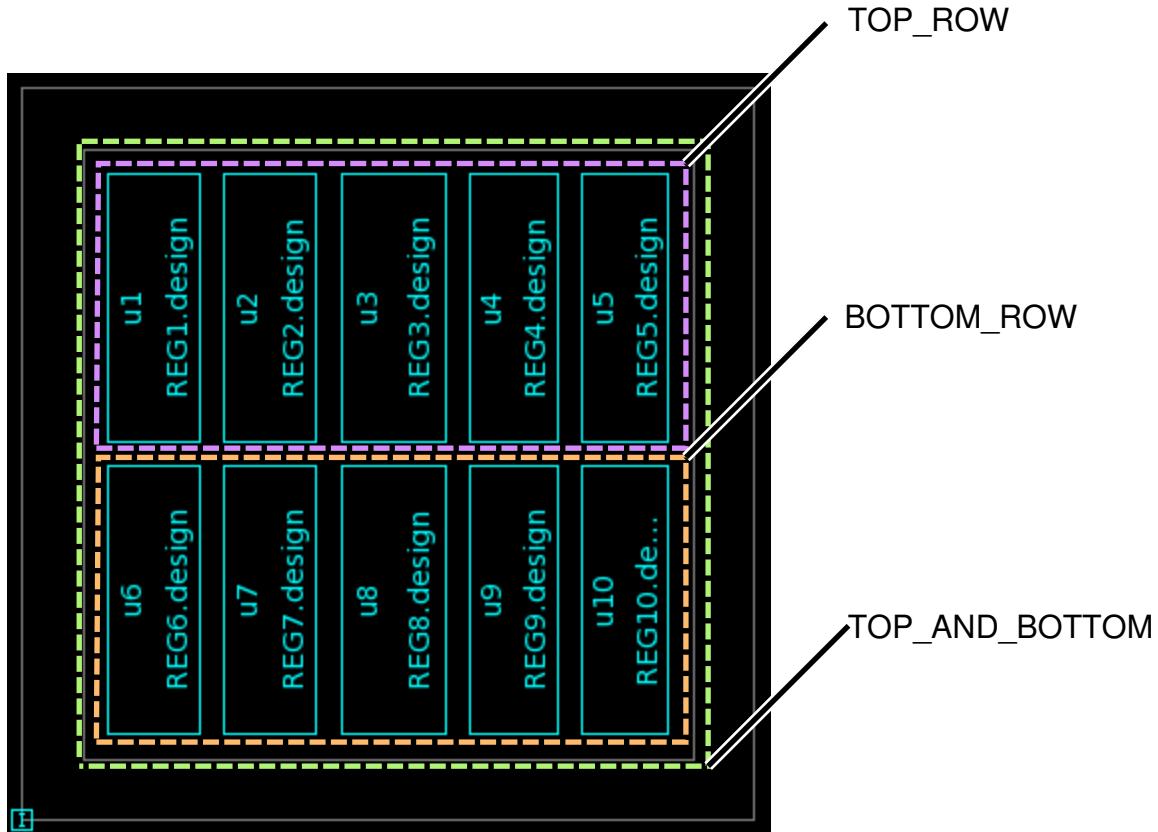
6. Create a shaping constraint that specifies that the BOTTOM\_ROW group is placed below the TOP\_ROW group.

```
icc2_shell> create_shaping_constraint [get_groups \
 -shaping TOP_AND_BOTTOM] -type array_layout -array_layout south
{TOP_AND_BOTTOM/SHAPING_CONSTRAINT_2_array_layout}
```

7. Perform block shaping with the `shape_blocks` command.

```
icc2_shell> shape_blocks
```

*Figure 69 Block Shaping Example*



To remove shaping constraints, use the `remove_shaping_constraints` command. The following command removes all shaping constraints from the u1/u2 block:

```
icc2_shell> remove_shaping_constraints u1/u2/*
```

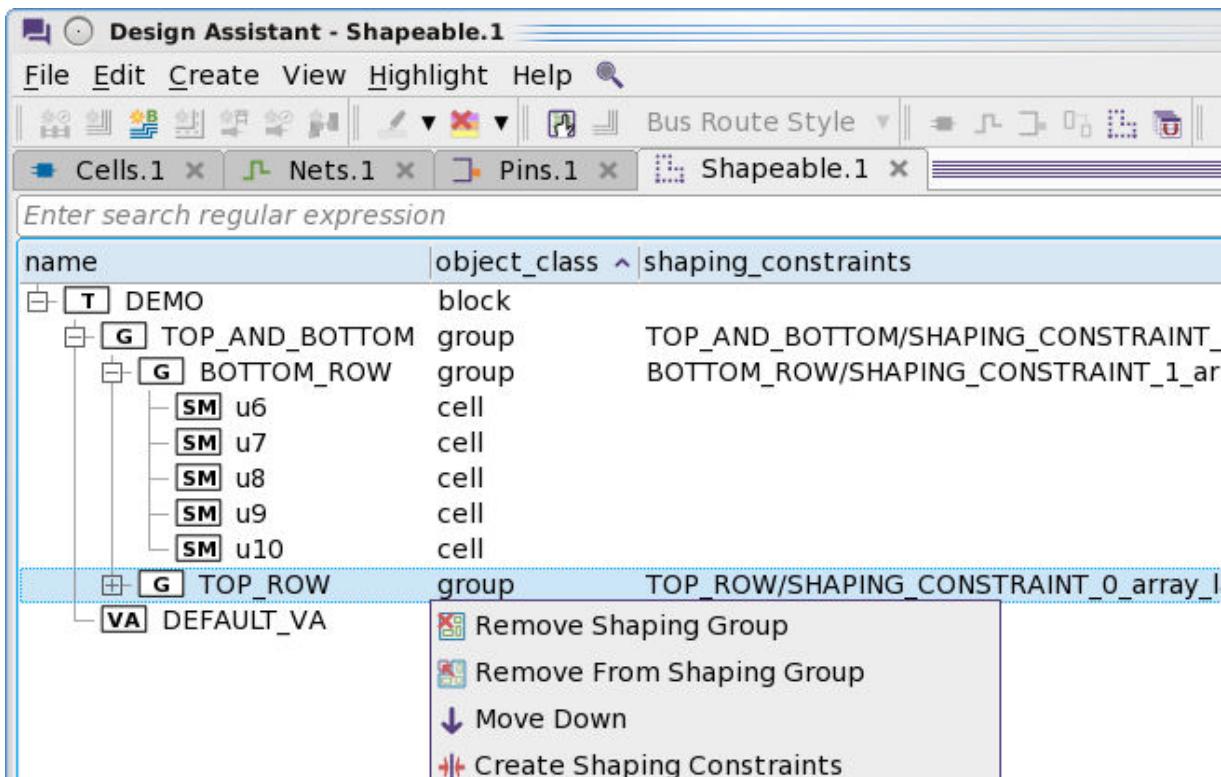
To report the shaping constraints for the current block or a specific block, use the `report_shaping_constraints` command. The following example reports all the shaping

constraints for the current block and all its hierarchically nested blocks that are visible to the `shape_blocks` command:

```
icc2_shell> report_shaping_constraints
```

The Design Assistant in the GUI displays the Tcl-based shaping constraints applied to the design. Open the Design Assistant by choosing View > Assistants > Design Assistant from the menu. In the Design Assistant, choose View > Shapeable Objects to view the objects and constraints. Use this tool to create shaping constraints, rearrange objects in shaping groups, remove objects from shaping groups, and delete shaping groups. The following figure shows the shapeable objects tab in the Design Assistant for the previous example.

*Figure 70 Shapeable Objects in Design Assistant*



The following example applies Tcl-based shaping constraints to two blocks and a voltage area. The `create_shaping_constraint` command is used to create a 50 micron channel around each shaped object. Note that the Tcl `list` command is used to gather heterogeneous objects for the `-shaping` option with the `create_group` command.

```
set_app_options \
 -name plan.shaping.import_tcl_shaping_constraints -value true

create_group -name G1 \
 -shaping [list [get_cells u0_1] \
```

```
[get_voltage_areas u_m/PD_LEON3_misc] \
[get_cells U2]

create_group -name USG_1 \
 -shaping [list [get_groups -shaping G1] \
 [get_cells U1]]

create_shaping_constraint \
 [get_blocks leon3mp.nlib:leon3mp/shaping.design] \
 -type boundary_channels \
 -object_channel [create_shaping_channel -left_min 50 \
 -left_max 50 -bottom_min 50 -bottom_max 50 -right_min 50 \
 -right_max 50 -top_min 50 -top_max 50]

create_shaping_constraint [get_groups -shaping G1] \
 -type array_layout -array_layout east

create_shaping_constraint [get_groups -shaping USG_1] \
 -type array_layout -array_layout south

shape_blocks
```

This example creates the following layout.

Figure 71 Block and Voltage Area Shaping



## Creating Channel Constraints for Block Shaping

Block shaping channels are used to create space between different objects such as blocks, voltage areas, move bounds, and shaping groups in the layout. To create a shaping channel, use the `create_shaping_channel` command with options to define the minimum and maximum channel width along the top, bottom, left, and right edges of the object. Use the `-neighbor` option with the `create_shaping_channel` command to specify a spacing constraint with respect to a neighboring object in the

layout. After creating the shaping channel, assign the channel to an object by using the `create_shaping_constraint` command.

You can assign three types of channels by using the `-type` option with the `create_shaping_constraint` command:

- `boundary_channel`: Creates a boundary channel constraint for the top-level design.
- `external_channels`: Creates a channel constraint along the sides of an objects. Note that you cannot assign an external channel constraint to the current top-level block (returned by the `current_block` command).
- `child_default_channels`: Creates default external channels for the nested child regions of the specified block, voltage area, move bound, or shaping group.

The following example creates a 100 micron boundary channel between the core boundary and top-level objects. A neighbor channel constraint is created for the `u0_1` block and assigned to the `U1` block. The `shape_blocks` command shapes the blocks and channels based on the constraints.

```
Create the TOP_ROW shaping group to contain blocks
u0_1 and U2, and voltage area u_m/PD_LEON3_misc.
Create the TOP_AND_BOTTOM group to contain TOP_ROW
group and block U1
set_app_options \
 -name plan.shaping.import_tcl_shaping_constraints -value true
create_group -name TOP_ROW \
 -shaping [list [get_cells u0_1] \
 [get_voltage_areas u_m/PD_LEON3_misc] \
 [get_cells U2]]
create_group -name TOP_AND_BOTTOM -shaping \
 [list [get_groups -shaping TOP_ROW] [get_cells U1]]

Create the boundary shaping channel and
neighbor shaping channel
set boundary_channel_100 \
 [create_shaping_channel -left_min 100 \
 -right_min 100 -top_min 100 -bottom_min 100]
set neighbor_channel \
 [create_shaping_channel -left_min 200 -left_max 250 \
 -right_min 200 -right_max 250 \
 -top_min 200 -top_max 250 \
 -bottom_min 200 -bottom_max 250 \
 -neighbor [get_cells u0_1]]

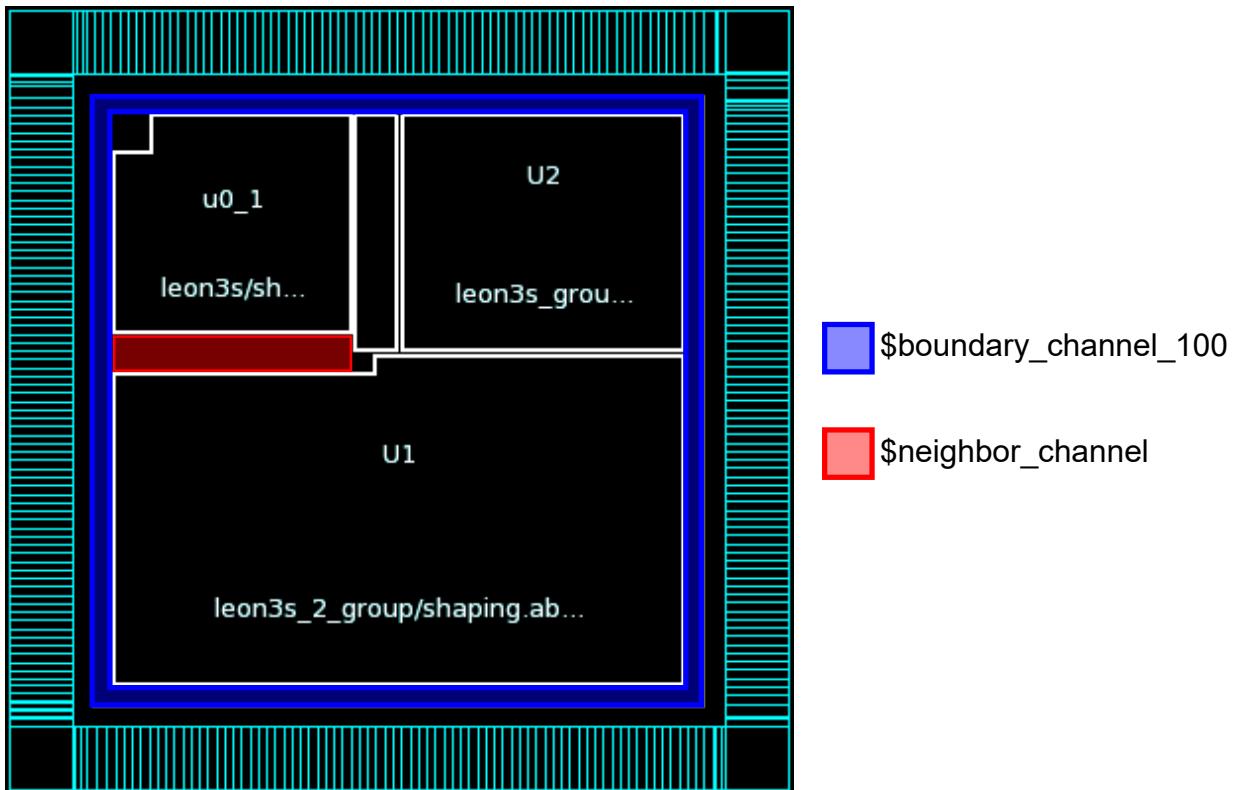
Assign the boundary channel to the current block and
the neighbor channel to block U1
create_shaping_constraint [current_block] -type boundary_channel \
 -object_channel $boundary_channel_100
create_shaping_constraint [get_cells U1] -type external_channels \
 -neighbor_channels $neighbor_channel
```

```
Arrange the objects in the TOP_ROW group in
left-to-right order
create_shaping_constraint [get_groups -shaping TOP_ROW] \
 -type array_layout -array_layout east

Arrange the objects in the TOP_AND_BOTTOM group in
top-to-bottom order
create_shaping_constraint [get_groups -shaping TOP_AND_BOTTOM] \
 -type array_layout -array_layout south
shape_blocks
```

The constraints produce the layout result in [Figure 72](#).

*Figure 72 Block Shaping and Channel Constraints*



To remove shaping channels, use the `remove_shaping_channels` command. The following command removes all channels associated with a specific shaping constraint object:

```
icc2_shell> remove_shaping_channels \
 DEFAULT_VA/SHAPING_CONSTRAINT_0_external_channels/*
```

To report the shaping channels created by the `create_shaping_channel` and `create_shaping_constraint` commands, use the `report_shaping_channels` command. The following example reports the shaping channels related to the MB2 move bound:

```
icc2_shell> report_shaping_channels -of_objects [get_bounds MB2]
```

## Creating the Block Grid for Multiply Instantiated Blocks

For designs that contain multiply instantiated blocks (MIBs), you must create a block grid to enable the tool to properly align MIB instances with other design objects. Proper MIB alignment ensures that objects can be successfully pushed down into blocks while the objects maintain alignment with the original block. During block shaping with the `shape_blocks` command, the tool aligns MIB instances with the block grid.

The block grid can be derived from power plan strategies or derived from existing site rows in the floorplan. You can also define the grid manually by specifying the origin and grid spacing. After creating the grid, you must associate the block references with a block grid to ensure that the tool aligns all instances with the grid in the same way.

To create the block grid from existing site rows and power ground strategy settings,

1. Use the `create_pg_mesh_pattern` and `set_pg_strategy` commands to define the power grid (it is not necessary to instantiate the grid with the `compile_pg` command).

```
icc2_shell> create_pg_mesh_pattern pat_mesh \
-layers { \
 {{vertical_layer: ME8} {width: 3} \
 {spacing: 3} {pitch: 20} {offset: 2}} \
 {{horizontal_layer: ME7} {width: 3} \
 {spacing: 3} {pitch: 20} {offset: 2}}}
icc2_shell> set_pg_strategy s_mesh -core \
-pattern {{name: pat_mesh} {nets: VDD90 VSS}}
```

2. Create the grid by using the `create_grid` command with the `-site_rows` and `-pg_strategy` options.

```
icc2_shell> create_grid grid1 -type block \
 -site_rows [get_site_rows] -pg_strategy s_mesh1
Calculating block grid:
...
{grid1}
```

If your design contains a site array instead of site rows, use the following command to create the grid:

```
icc2_shell> create_grid grid1 -type block \
 -site_arrays [get_site_array] -pg_strategy s_mesh1
Calculating block grid:
```

```
...
{grid1}
```

3. (Optional) Verify that the grid was created correctly by using the `report_grids` command.

```
icc2_shell> report_grids

Report : report block grid
Design : ORCA

Auto Block grid grid1:
 Step in X direction: 20.0000 micron
 Step in Y direction: 20.0000 micron
 X offset: 0.0000 micron
 Y offset: 0.0000 micron
 Allowed orientations: R0
 Derived from PG strategies.
 Strategies Used Layers:
 s_mesh1 \
```

4. Associate the block grid with the MIB references by using the `set_block_grid_references` command.

```
icc2_shell> set_block_grid_references -designs {ORCA} \
 -grid grid1
```

Alternatively, you can manually create the block grid by specifying the x- and y- offsets and step sizes for the grid. Use the `report_grids` and `set_block_grid_references` commands to verify the grid and associate the grid with the MIB references. Create the grid manually as follows:

```
icc2_shell> create_grid -type block grid2 -x_step 20.0 \
 -y_step 20.0 -x_offset 19.0 -y_offset 13.0
{grid2}
```

Note that you do not need to specify the power planning patterns or strategy if you create the grid manually.

### Setting the Snap Point for a Multiply Instantiated Block

To ensure that multiply instantiated blocks (MIBs) can be flipped and rotated correctly while keeping the MIBs on-grid, you can change the snap point for the block. The tool aligns the snap point of the block to the nearest grid point. By default, the snap point for the design is (0,0). To set the snap point and verify the setting, use the

`set_block_grid_references` command followed by the `get_attribute` command as follows:

1. Set the snap point for the multiply instantiated design.

```
icc2_shell> set_block_grid_references -grid grid1 -designs BLENDER_2 \
-snap_point {100.0 100.0}
```

2. (Optional) Verify the settings by examining the attributes on the multiply instantiated block instances.

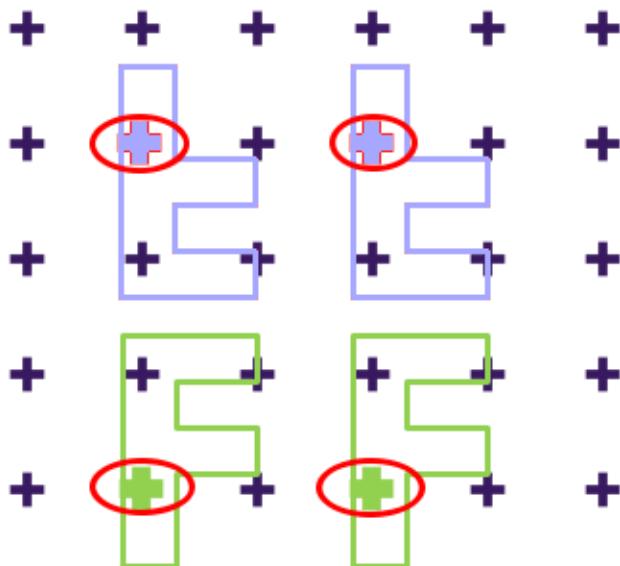
```
icc2_shell> get_attribute [get_cells I_BLENDER_1] snap_point
100.0000 100.0000
icc2_shell> get_attribute [get_cells I_BLENDER_1] block_grid
grid1
```

3. (Optional) Snap the multiply instantiated blocks to the grid using one of the following commands.

```
icc2_shell> snap_cells_to_block_grid -grid grid1
icc2_shell> snap_cells_to_block_grid -designs BLENDER_2
icc2_shell> snap_cells_to_block_grid
-cells [get_cells I_BLENDER_1]
```

The snap point can be set to any location on the block. In the following figure, the small plus symbols represent points on the grid and the large plus symbols (circled in red) represent the snap point defined for the design.

Figure 73    *Multiply Instantiated Block Instances With Snap Point*



## Creating the Initial Macro Placement

After shaping the blocks, you can create an initial global macro placement with the `create_placement` command. The objective of macro placement is to decide which macro to place where. The macro placement is done at the early stages of the design planning flow.

To create the initial macro placement,

1. (Optional) Set the host options for the creating the floorplan-level placement with the `set_host_options` command.

```
icc2_shell> set_host_options -name distributed ...
```

Use the `set_host_options` command to create the setting for distributed processing.

2. (Optional) Enable macro-only placement by setting the `plan.macro.macro_place_only` application option to true.

```
icc2_shell> set_app_options -name plan.macro.macro_place_only \
 -value true
```

3. (Optional) Review the settings for the application options for the placer and make any necessary changes.

You can use the `report_app_options plan.place*` commands to report the current placer-related settings. Use the `set_app_options` command to change a setting.

4. (Optional) Limit placement to specific levels of design hierarchy with the `set_editability` command. The following example prevents placement at lower levels of the design hierarchy and specifies placement at the top level only.

```
icc2_shell> set_editability -from_level 1 -value false
```

5. Perform global macro placement by using the `create_placement -floorplan` command.

```
icc2_shell> create_placement -floorplan -host_options distributed
```

Optionally,

- If you set host options in step 1, enable distributed processing by using the `-host_options` option
- Specify the level of effort to apply to the placement by using the `-effort low | medium | high` option

As you increase the effort level, the tool increases the number of internal loops to create a placement with the best possible quality of results. To provide better

convergence, the tool decreases the average distance the cells are allowed to move in successive placement passes.

- Update the placement after changing the boundaries of macro blocks by using the `-incremental` option

When you use this option, the command tries to create legal macro placement close to the current one. If the current macro placement is legal, it is retained; otherwise, macros are moved to other legal positions. Examples of illegal positions include macro overlapping, sticking out of core, or not being on grid.

- Reduce congestion by using the `-congestion` option

When you use this option, the tool tries to reduce congestion by moving macros and creating wider channels.

## 6. Validate the placement with the `report_placement` command.

```
icc2_shell> report_placement -physical_hierarchy_violations all \
 -wirelength all -hard_macro_overlap

Report : report_placement
Design : ORCA

Wire length report (all)
=====
wire len in design ORCA/dp: 828662.662 microns.
wire len in design ORCA/dp (see through blk pins): 950442.075 microns.

Physical hierarchy violations report
=====
Violations in design ORCA/dp:
Information: cell .../I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_5
 overlaps block .../I_BLENDER_1 (DPP-403)
Information: cell .../I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_5
 overlaps block .../I_BLENDER_5 (DPP-403)
2 cells have placement violation.

Voltage area violations report
=====
Voltage area placement violations in design ORCA/dp:
 0 cells placed outside the voltage area which they belong to.

Hard macro to hard macro overlap report
=====
Information: .../I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_4 and
 .../I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_7 overlap (DPP-405)
Information: .../I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_3 and
 .../I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_6 overlap (DPP-405)
HM to HM overlaps in design ORCA/dp: 2

Total hard macro to hard macro overlaps: 2
```

7. (Optional) If the design contains overlaps, generate a complete list of overlapping instances with the `report_placement` command.

```
icc2_shell> report_placement -verbose high > report.txt
```

You can examine the report that is created by the `report_placement` command and investigate the overlaps.

8. (Optional) To remove the placement created with the `create_placement` command, use the `reset_placement` command.

```
icc2_shell> reset_placement
```

For more information on congestion-driven and timing-driven placement, see [Performing Timing-Driven and Congestion-Driven Placement](#).

## Identifying Channels Between Objects

To identify and report channels that exist between specified objects,

1. Specify the objects between which to identify channels.

```
icc2_shell> set macros {I_CONTEXT_RAM_3_1 I_CONTEXT_RAM_3_2 \
I_CONTEXT_RAM_3_3 I_CONTEXT_RAM_3_4}
```

2. Run the `identify_channels` command to write out a file that contains the coordinates of the channels between the specified objects.

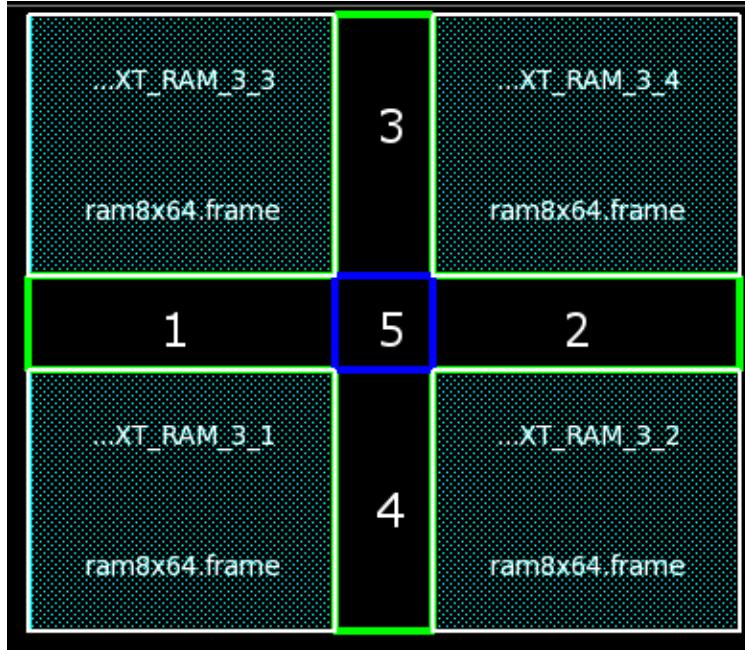
```
icc2_shell> identify_channels -cross_area $macros
Creating channel file channels.txt
```

The tool writes the channels to the `channels.txt` file. The file lists the type of channel (horizontal, vertical, or crossed\_area) and the coordinates of the channel as follows:

```
icc2_shell> sh cat channels.txt
horizontal (848.17 1632.765) (970.3 1669.995)
horizontal (1009.175 1632.765) (1131.305 1669.995)
vertical (970.3 1669.995) (1009.175 1774.075)
vertical (970.3 1528.685) (1009.175 1632.765)
crossed_area (970.3 1632.765) (1009.175 1669.995)
```

In the following layout, the channels are outlined and numbered corresponding to the line number of the channel in the `channels.txt` file.

Figure 74 Identified Channels



Use options with the `identify_channels` command to control how the channels are reported.

- Report channels less than a specified height by using the `-horizontal_threshold` option.
- Report channels less than a specified width by using the `-vertical_threshold` option.
- Specify the output file name with the `-output_filename` option.
- Report cross areas by using the `-cross_area` option. Cross areas occur between the lower-left edge of one object and the upper right edge of another object, and vice versa.
- Report cross channels less than a specified height by using the `-cross_area_height` option.
- Report cross channels less than a specified width by using the `-cross_area_width` option.

## Application Options for Macro Placement

*Table 6 Macro Placement Application Options*

| To do this                                                                                                                                                                                                                                       | Use this application option                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| Specify whether pins are aligned in both the x- and y-directions or are aligned only with the direction for the layer they occupy.                                                                                                               | <code>plan.macro.align_pins_on_parallel_edges</code>       |
| Specify the maximum height of an automatically generated macro array.                                                                                                                                                                            | <code>plan.macro.auto_macro_array_max_height</code>        |
| Specify the maximum number of columns in an automatically generated macro array.                                                                                                                                                                 | <code>plan.macro.auto_macro_array_max_num_cols</code>      |
| Specify the maximum number of rows in an automatically generated macro array.                                                                                                                                                                    | <code>plan.macro.auto_macro_array_max_num_rows</code>      |
| Specify the maximum width in micros of an automatically generated macro array.                                                                                                                                                                   | <code>plan.macro.auto_macro_array_max_width</code>         |
| Specify whether macros with pins on only one side are flipped to abut the pins of neighboring macros ( <code>pins_in</code> ), or flipped to abut the sides without pins ( <code>pins_out</code> ), or not changed ( <code>none</code> ).        | <code>plan.macro.auto_macro_array_minimize_channels</code> |
| Specify the amount of array packing: <code>none</code> , <code>low</code> , <code>medium</code> , <b>or</b> <code>high</code> .                                                                                                                  | <code>plan.macro.auto_macro_array_size</code>              |
| Specify the height of the channel created when the macro stack height meets or exceeds the value specified by the <code>plan.macro.max_buffer_stack_height</code> application option.                                                            | <code>plan.macro.buffer_channel_height</code>              |
| Specify the width of the channel created when the macro stack width meets or exceeds the value specified by the <code>plan.macro.max_buffer_stack_width</code> application option.                                                               | <code>plan.macro.buffer_channel_width</code>               |
| Specify the number of iterations used when estimating congestion using the <code>-congestion</code> option with the <code>create_placement</code> command.                                                                                       | <code>plan.macro.congestion_iters</code>                   |
| Specify a command or script to use to create temporary PG grids to help reduce congestion during macro placement. Optionally, in the script, you can include code to insert temporary power switch cells in narrow channels along with PG grids. | <code>plan.macro.create_temporary_pg_grid</code>           |

**Table 6      Macro Placement Application Options (Continued)**

| To do this                                                                                                                                                                                                                                                                                                                                                                               | Use this application option                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| Specify the user script to run, to remove the temporary PG grids and the temporary power switch cells that were inserted to help improve congestion estimate.                                                                                                                                                                                                                            | plan.macro.remove_temporary_pg_grid          |
| Specify the constraint type ( <code>soft</code> or <code>hard</code> ) or behavior ( <code>false</code> , <code>true</code> , or <code>unset</code> ) to use when avoiding macro placement along a block boundary in a channel between blocks.                                                                                                                                           | plan.macro.cross_block_connectivity_planning |
| Specify that the <code>create_placement -floorplan</code> command create a temporary PG mesh to enhance the precision of the congestion estimation.                                                                                                                                                                                                                                      | plan.macro.estimate_pg                       |
| Specify what happens if an error occurs when aligning macros to the grid. Specify <code>continue</code> to print warning messages and continue floorplan placement. Specify <code>macro_place_only</code> to print warning messages and stop after completing macro placement. Specify <code>quit</code> to print error messages and stop immediately when an alignment error occurs.    | plan.macro.grid_error_behavior               |
| Specify that the <code>create_placement -floorplan</code> command groups macros by logical hierarchy. When <code>true</code> , the macro placer places all macros within a hierarchy into the same group if the ratio of the size of the hierarchy and the size of the block exceeds the threshold specified by the <code>plan.macro.hierarchy_area_threshold</code> application option. | plan.macro.grouping_by_hierarchy             |
| Specify whether the <code>create_placement -floorplan</code> command stops or continues after placing macros.                                                                                                                                                                                                                                                                            | plan.macro.macro_place_only                  |

**Table 6** *Macro Placement Application Options (Continued)*

| To do this                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Use this application option                             |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| Specify the preferred placement of macros. This application option supports <code>on_edge</code> , <code>islands</code> , and <code>freeform</code> , and <code>hybrid</code> styles. The default is <code>on_edge</code> that places the macros along the edges of blocks, voltage areas, and move bounds. If this application is set to <code>islands</code> , some of the macros are placed away from partition edges in islands, that is, floating tightly packed groups of macros. If <code>freeform</code> , the macros are placed individually and loosely among standard cells. If <code>hybrid</code> , the tool places large macros along the edges of the partitions while the smaller ones are placed using freeform placement. However, it is also possible that the tool places all macros on edge or all macros in <code>freeform</code> style (for example, if all the macros are large, block a lot of routing layers, or both, the tool could place them along the edges, whereas for opposite reasons, the tool could place all the macros in <code>freeform</code> style). The difference between the <code>hybrid</code> style and the <code>islands</code> style is that in the <code>hybrid</code> style, the floating macros are placed individually in <code>freeform</code> style versus being grouped and packed in islands. Also, in the <code>hybrid</code> style, any number of macros can be floating, while in the <code>islands</code> style, only small percentage of the design area can be islands. | <code>plan.macro.style</code>                           |
| Specify the height of the macro stack after which the tool creates a channel for buffer placement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | <code>plan.macro.max_buffer_stack_height</code>         |
| Specify the width of the macro stack after which the tool creates a channel for buffer placement.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | <code>plan.macro.max_buffer_stack_width</code>          |
| Specify the minimum size of the macro keepout created to ensure routability to the macro pins.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | <code>plan.macro.min_macro_keepout</code>               |
| Specify the width (for vertical layers) or height (for horizontal layers) within which the tool tries to color match pin shapes and tracks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <code>plan.macro.pin_shape_track_match_threshold</code> |
| Specify a pair of distances, an exact distance and a minimum distance, for placing neighboring macros. Neighboring macros must be placed at the exact vertical distance apart, or placed with a vertical spacing greater than the minimum distance.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | <code>plan.macro.spacing_rule_heights</code>            |
| Specify a pair of distances, an exact distance and a minimum distance, for placing neighboring macros. Neighboring macros must be placed at the exact horizontal distance apart, or placed with a horizontal spacing greater than the minimum distance.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <code>plan.macro.spacing_rule_widths</code>             |

**Table 6      Macro Placement Application Options (Continued)**

| To do this                                                                                                                                                                                                                                                                                                 | Use this application option                           |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------|
| Control whether to automatically create soft placement blockages in channels between hard and soft objects. Hard objects include hard macros, hard keepout margins, block boundaries, VA boundaries, and hard placement blockages. Soft objects include soft placement blockages and soft keepout margins. | plan.place.auto_create_blockages                      |
| Control whether to automatically create vertical channel blockages.                                                                                                                                                                                                                                        | plan.place.auto_create_blockage_channel_widths        |
| Control whether to automatically create horizontal channel blockages.                                                                                                                                                                                                                                      | plan.place.auto_create_blockage_channel_heights       |
| Control whether slightly rectilinear macros are considered to be the shape of their rectangular bounding box when determining blockages.                                                                                                                                                                   | plan.place.auto_generate_blockages_smooth_rectilinear |
| Control whether to allow standard cells to clump together during top-level placement of a design that contains blocks.                                                                                                                                                                                     | plan.place.auto_max_density                           |
| Specify the type of cell to consider for congestion reduction: <code>macro</code> , <code>std_cell</code> , or both.                                                                                                                                                                                       | plan.place.congestion_driven_mode                     |
| Control whether the tool creates a default hard keepout for all macros to force spacing between macros.                                                                                                                                                                                                    | plan.place.default_keepout                            |
| Control whether the tool creates a default hard keepout for all macros to force spacing between macros and standard cells.                                                                                                                                                                                 | plan.place.auto_create_hard_keepout                   |
| Specify the tracing method used when tracing signals for macro placement. Specify <code>normal</code> to trace only buffers and inverters to registers; specify <code>dfa</code> to trace buffers, inverters, and combinational logic to registers.                                                        | plan.place.trace_mode                                 |

---

## Performing Freeform Macro Placement

With the increase in the number of macros, the traditional approach of placing the macros on the edge does not scale well. To address this issue as well improve the QoR and EoU, the tool allows the placement of macros among standard cells, which is called the freeform

macro placement style. Freeform macro placement provides the following benefits as compared to on-edge macro placement:

- Considers congestion driven channel sizing
- Adds partial blockages to macro channels
- Considers PG resources when calculating channel congestion
- Provides automatic pin protection blockages
- Provides support for hard inclusive bounds with standard cells
- Honors any functional safety-related Dual Core Lock Step (DCLS) or nCLS (core lock step with n cores) constraints

To use the freeform style of macro placement, set the following application option:

```
icc2_shell> set_app_options -name plan.macro.style -value freeform
```

Use the following application options for finer control of the freeform macro placement:

- `plan.macro.create_auto_pin_blockages`: Specifies whether to create blockages around pins when running freeform macro placement. These blockages are of `hard_macro` type and they protect the design pins from having macros too close to them.
- `plan.macro.remove_auto_pin_blockages`: Specifies whether to remove the temporary blockages that were placed around the pins during freeform macro placement, to allow easy access to the pins.
- `plan.macro.auto_pin_blockages_width`: Specifies the width (if the pins are on the vertical block edges) or height (if pins are on horizontal block edges) of pin blockages created during freeform macro placement.

To enable timing driven placement using the buffering aware timing model, use the `create_placement -buffering_aware_timing_driven` command. This approach estimates the effects of buffering long nets and high-fanout nets later in the design planning flow. It provides a better starting point for later timing optimizations.

```
icc2_shell> create_placement -floorplan -congestion \

-buffering_aware_timing_driven
```

To perform incremental freeform macro placement after changes due to one or more of the following reasons, use the `create_placement -from` command.

- Netlist changes (during optimization due to timing or area changes)
- Constraint changes (for example, hard macro keepouts)
- Congestion reduction

The `-from` option enables you to choose the level of macro movement from the most aggressive and time consuming (`combined_placement`) to the least aggressive (`channel_sizing`). A value of `combined_placement` results in the tool running the full freeform flow. The default is `macro_legalization`. Ensure that you use the `-from` option with the `-floorplan` and `-incremental` options.

For incremental congestion-driven freeform macro placement, use the following options in the `create_placement` command:

- `-floorplan`
- `-incremental`
- `-congestion`
- (Optional) `-from` option from the following table

| <b>Starting point for incremental freeform macro placement</b> | <b>Use this</b>                                                                                                        |
|----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| Incremental combined placement                                 | <code>-from combined_placement</code>                                                                                  |
| Legalize macros                                                | <code>-from macro_legalization</code><br>which is the default starting point for incremental freeform macro placement. |
| Macro channel sizing<br>and                                    | <code>-from channel_sizing</code>                                                                                      |
| Incremental standard cell placement                            |                                                                                                                        |

For example,

```
icc2_shell> create_placement -floorplan -incremental -congestion
 -from combined_placement
```

For noncongestion driven freeform macro placement,

| <b>Starting point for incremental freeform macro placement</b> | <b>Use this</b>                       |
|----------------------------------------------------------------|---------------------------------------|
| Incremental combined placement                                 | <code>-from combined_placement</code> |
| Legalize macros                                                | <code>-from macro_legalization</code> |
| Incremental standard cell placement                            |                                       |

## Integrated Freeform Macro Placement

To have a better runtime efficiency during macro placement, you can use the integrated freeform macro cells placement (iFFMP) approach.

Typically, freeform macro placement (FFMP) is performed before running the `place_opt` flow. The limitations of this standalone FFMP are inefficient runtime, non-adaptive results, and the possibility of duplication when running the `place_opt` flow.

To overcome these limitations, the tool supports Integrated Free Form Macro Placement (iFFMP), where freeform macro cells placement is integrated with the `place_opt` flow. The advantages of such an integrated flow are better runtime efficiency and the ability to adjust macro cells placement during the `place_opt` flow to improve QoR.

You can run the iFFMP for a single pass, two pass, and manual two pass `place_opt` flows. Apart from this, iFFMP also supports PG grid insertion and native chip-finishing without requiring any changes in the `place_opt` flow.

To run iFFMP without the support for PG grid, switch, or tap insertion, perform the following steps:

1. Open a design.

2. Ensure that all the macro cells to be placed are unfixed. For example,

```
icc2_shell> set_attribute [get_flat_cells -filter is_hard_macro]
unplaced
```

3. Remove any boundary, switch, and tap cells.

```
icc2_shell> remove_cells-force \
[get_flat_cells-quiet -all -filter \
"(is_power_switch|| is_physical_only) && !is_io"]
```

4. Set the following application options to enable the iFFMP flow:

```
icc2_shell> set_app_options -name plan.macro.style -value freeform
```

```
icc2_shell> set_app_options -name plan.macro.integrated -value true
```

**Note:**

When setting the `plan.macro.integrated` to `true`,

- If `place_opt.initial_place.two_pass` is `false`, the tool enables the single pass flow.
- If `place_opt.initial_place.two_pass` is `true`, the tool enables the two pass flow.

When running the manual two pass flow, set the `plan.macro.integrated` application option to `two_pass`.

5. Run the `place_opt` command to start the `place_opt` flow with iFFMP.
6. Ensure all macros are fixed after the `place_opt` flow by using the `set_fixed_objects` command. For example,

```
icc2_shell> set_fixed_objects [get_flat_cells -filter is_hard_macro]
```

### Performing iFFMP With Temporary PG Grid Insertion

Inserting a PG grid before macro cells placement increases runtime as macro cells have a significant impact on the PG grid. To improve performance, the tool supports iFFMP with a temporary PG grid creation to model the congestion effect of a power grid during macro and standard cells placement.

The temporary PG grid flow is quicker as it is inserted without the PG via DRC checking. After the `final_place` step of the `place_opt` flow is completed, the PG grid is removed and reinserted to analyze the capacity impact of the PG grid on the tracks available for routing.

To perform iFFMP with temporary PG grid insertion,

1. Open a design.
2. Ensure that all the macro cells to be placed are unfixed. For example,

```
icc2_shell> set_attribute [get_flat_cells -filter is_hard_macro] unplaced
```

3. Remove any boundary, switch, and tap cells.

```
icc2_shell> remove_cells-force \

[get_flat_cells-quiet -all -filter \

"(is_power_switch|| is_physical_only) && !is_io"]
```

4. Set the following application options to enable iFFMP flow and PG grid insertion:

```
icc2_shell> set_app_options -name plan.macro.style -value freeform
```

```
icc2_shell> set_app_options -name plan.macro.integrated -value true
```

```
set_app_options -name plan.macro.create_temporary_pg_grid \

-value <pg_script.tcl|Tcl code block>
```

```
set_app_options -name plan.macro.remove_temporary_pg_grid \

-value <pg_script.tcl|Tcl code block>
```

5. Run the `place_opt` command to start the `place_opt` flow with iFFMP.
6. Ensure that all macro cells are fixed after the `place_opt` flow by using the `set_fixed_objects` command. For example,

```
icc2_shell> set_fixed_objects [get_flat_cells -filter is_hard_macro]
```

7. Remove and reinsert the PG grid with via DRC checking.

### Performing iFFMP With Floorplan Finishing

You can do floorplan finishing when running freeform or hybrid macro placement to insert boundary cells, tab cells, switch cells, and so on by using the `plan.macro.floorplan_finishing` application option.

It is important that these cells are modeled during the iFFMP flow as the placement of these cells has a significant impact on the placeable area of the design. To improve the accuracy of the iFFMP `place_opt` flow, the tool uses the `plan.macro.floorplan_finishing` application option while enabling the iFFMP flow. You can use this application option to specify a script file or Tcl code that inserts boundary cells, tap cells, and switch cells.

To perform iFFMP with floorplan finishing,

1. Open a design.
2. Make sure all the macro cells to be placed are unfixed. For example,

```
icc2_shell> set_attribute [get_flat_cells -filter is_hard_macro] unplaced
```

3. Remove any boundary, switch, and tap cells.

```
icc2_shell> remove_cells-force \
[get_flat_cells-quiet -all -filter \
"(is_power_switch|| is_physical_only) && !is_io"]
```

4. Set the following application options to enable iFFMP flow, PG grid insertion, and floorplan finishing:

```
icc2_shell> set_app_options -name plan.macro.style -value freeform

icc2_shell> set_app_options -name plan.macro.integrated -value true

set_app_options -name plan.macro.create_temporary_pg_grid -value \
<pg_script.tcl|Tcl code block>

set_app_options -name plan.macro.remove_temporary_pg_grid -value \
<pg_script.tcl|Tcl code block>
```

```
set_app_options -name plan.macro.floorplan_finishing \
 -value <floorplan_finishing.tcl|Tcl code block>
```

5. Run the `place_opt` command to start the `place_opt` flow with iFFMP.
6. Ensure that all macro cells are fixed after the `place_opt` flow by using the `set_fixed_objects` command. For example,

```
icc2_shell> set_fixed_objects [get_flat_cells -filter is_hard_macro]
```

7. Remove and reinsert the PG grid with via DRC checking.

## Performing Machine Learning-Based Macro Placement

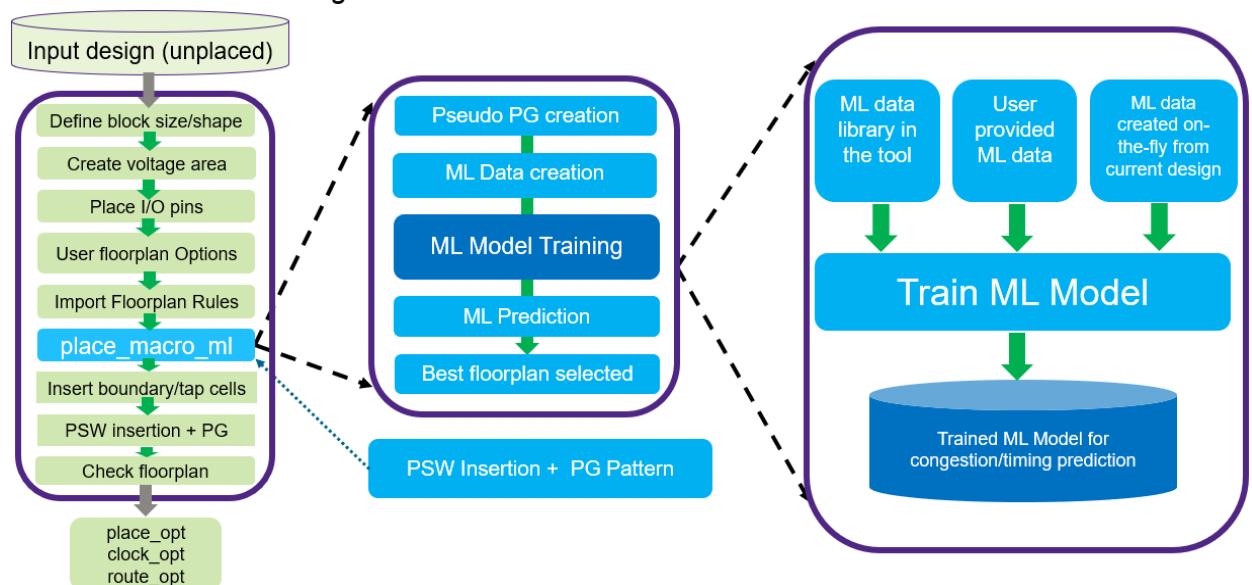
Location of macros can have a big impact on performance, power, and area (PPA) and therefore you should consider macro placement as one of the essential steps in any PPA analysis.

You can use either a manual or automated way to place macro cells. The machine learning-based macro placement is one of the automated ways that helps you to get the best floorplan among all possible macro placement results.

In the machine learning-based macro placement approach, you have the flexibility to tune between accuracy and runtime. You can switch between them by using the `-effort` option.

The flow is as follows:

**Figure 75 Machine Learning-Based Macro Placement Flow**



To place macros using the machine learning techniques, use the `place_macro_ml` command. This command places macros in such a way that would improve congestion, total negative slack, and power.

Machine learning macro placement predicts the best floorplan from possible macro placement results. You can place macros on edge or in freeform placement by using the `-style` option. Use a combination of the following options to achieve the required results.

**Table 7** *Macro Placement Command Options*

| To do this                               | Use this <code>place_macro_ml</code> command option                                                                                                                                                                                                                                                                                                                                             |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-host_options</code><br>(Required) | <p>To determine what type of jobs you want to run.<br/>         In the following example, the ABC host option is used to specify a batch job. The host option name (in this example, ABC) is case-sensitive.</p> <pre>icc2_shell&gt; set_host_options -name ABC \     -submit_command "qsub -P batch -l mem_free=30G" \     -max_cores 8  icc2_shell&gt; place_macro_ml -host_options ABC</pre> |
| <code>-mode</code>                       | <p>To optimize the macro placement for congestion, total negative slack, and power.</p> <ul style="list-style-type: none"> <li><code>both</code> - To optimize for congestion and total negative slack<br/>             This is the default setting.</li> <li><code>all</code> - To optimize for congestion, total negative slack, and power</li> </ul>                                         |
| <code>-effort</code>                     | <p>To specify the effort - whether low, medium (the default), or high<br/>         When the effort is high, the tool strives to get more accurate machine learning data, which makes for better machine learning model and results in better floorplans.</p>                                                                                                                                    |
| <code>-style</code>                      | To place the macros only on edge (the default) or in hybrid placement.                                                                                                                                                                                                                                                                                                                          |
| <code>-work_directory</code>             | <p>To specify where the machine learning internal data should be stored.<br/>         The default is <code>mlmp_dir</code>.<br/>         The generated machine learning data is stored in the <code>mlmp_dir/design_local_data/hm_local.csv</code> file.</p>                                                                                                                                    |

*Table 7 Macro Placement Command Options (Continued)*

| To do this                  | Use this <code>place_macro_ml</code> command option                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-on_edge_ratio</code> | <p>To specify the ratio of macros that should be identified as on edge macros over all the available macros.</p> <p>This option is only for hybrid macro placement.</p> <p>If the option is not set, the tool determines this value automatically. To know the value used by the tool, search for the text "On_edge ratio:" in the tool log file.</p> <p>To visualize what macros are regarded as on edge macros for the specified value of <code>-on_edge_ratio</code> option, use the <code>identify_on_edge_macros -ratio</code> command. This command identifies and highlights specific on edge macros (on the GUI) that are impacted by the macro placement depending on the ratio specified. This command can act as a guidance for you to get the best results for your design based on your requirements.</p> |

Here are a few examples of using the `place_macro_ml` command:

- Perform machine learning-based hybrid macro placement with medium effort to reduce both congestion and total negative slack.

```
icc2_shell> set_host_options -name mlmp \
-submit_command "qsub -P batch -l mem_free=30G" -max_cores 8
```

```
icc2_shell> place_macro_ml -host_options mlmp -style hybrid
```

- Perform machine learning-based hybrid macro placement with medium effort to reduce congestion, total negative slack, and power.

```
icc2_shell> set_host_options -name mlmp \
-submit_command "qsub -P batch -l mem_free=30G" -max_cores 8
```

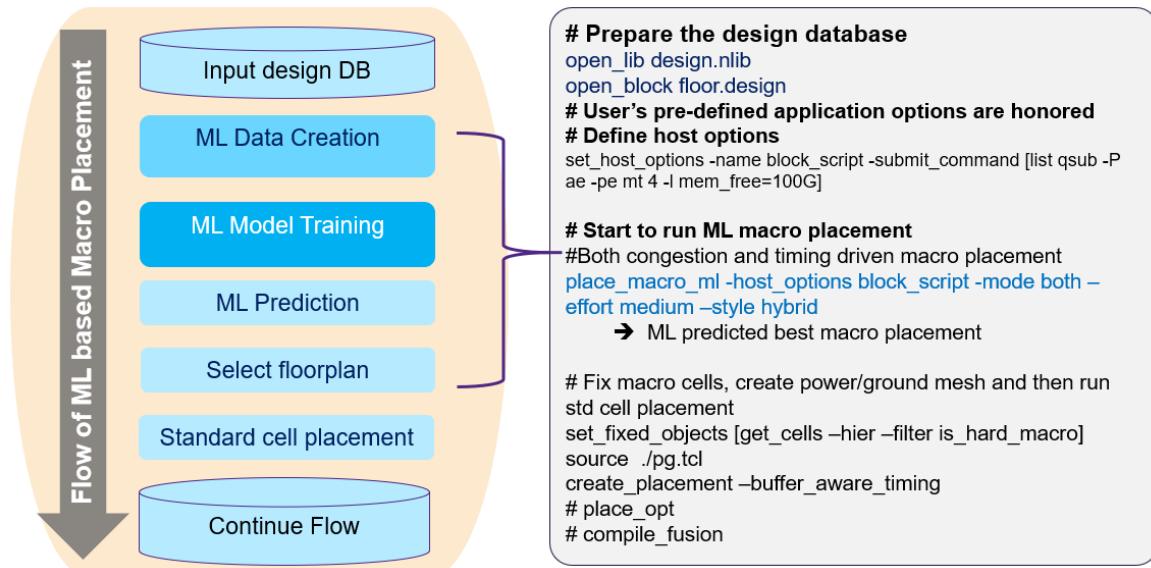
```
icc2_shell> place_macro_ml -host_options mlmp -style hybrid \
-mode all
```

- Store machine language-based macro placement data in "ml\_dir" directory.

```
icc2_shell> set_host_options -name mlmp \
-submit_command "qsub -P batch -l mem_free=30G" -max_cores 8
```

```
icc2_shell> place_macro_ml -host_options mlmp \
-work_directory "ml_dir"
```

### Command Usage Example



### Creating and Reusing Machine Learning Macro Placement Data

You can create machine learning-based macro placement data, which you can reuse to improve QoR of other designs.

The ways to create this data from your design are as follows:

| Option           | Example                                                                                                                                                                                                                        | Description                                                                                                                                                                                                                                                                                                                      |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| create_mlmp_data | icc2_shell> create_mlmp_data \     -output /     ML_design1/hybrid_ml_macro.csv                                                                                                                                                | In this example, the tool creates two files in the "ML_design1" directory: <ul style="list-style-type: none"> <li>hybrid_ml_macro.csv - Contains data for the current floorplan</li> <li>.hybrid_ml_macro.csv.signature - Contains information for decoding the data file (hybrid_ml_macro.csv). This file is hidden.</li> </ul> |
| place_macro_ml   | icc2_shell> set_host_options -name mlmp \     -submit_command "qsub -P batch -l mem_free=30G" -max_cores 8     icc2_shell> place_macro_ml \     -host_options mlmp \     -style "hybrid" \     -work_directory mlmp_dir_medium | In this example, the tool creates two files in the "mlmp_dir_medium/design_local_data/" directory: <ul style="list-style-type: none"> <li>hm_local.csv</li> <li>.hm_local.signature</li> </ul> Here, "mlmp_dir_medium" is the directory set by using the -work_directory option.                                                 |

## Placing Macros Island Style

The default macro placement is along the edge of blocks, voltage areas, and move bounds. For some designs, macro placement might be improved by placing groups of macros away from the edges of these objects or placing the macros among standard cells. Island-style placement can improve placement quality in designs with smaller macros that need to be relatively close to objects on different edges of the design.

By default, the tool uses timing and connectivity information to determine which macros to group together. You can specify the macro grouping by creating a bound to contain the group of macros as shown in steps 2 and 3 in the following example.

To place macros away from block, voltage area, and move bound edges during macro placement,

1. Set the `plan.macro.style` application option to `islands`.

```
icc2_shell> set_app_options -list {plan.macro.style islands}
```

2. (Optional) Create a group bound for the collection of island macros with the `create_bound` command.

```
icc2_shell> create_bound -effort high \
 -name bound1 { I_ORCA_TOP/I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_7
 I_ORCA_TOP/I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_6
 I_ORCA_TOP/I_PCI_TOP/I_PCI_WRITE_FIFO/PCI_FIFO_RAM_5 }
```

3. (Optional) If you created a bound to specify the macros to group together, set the `is_off_edge` attribute on the group bound to `true`.

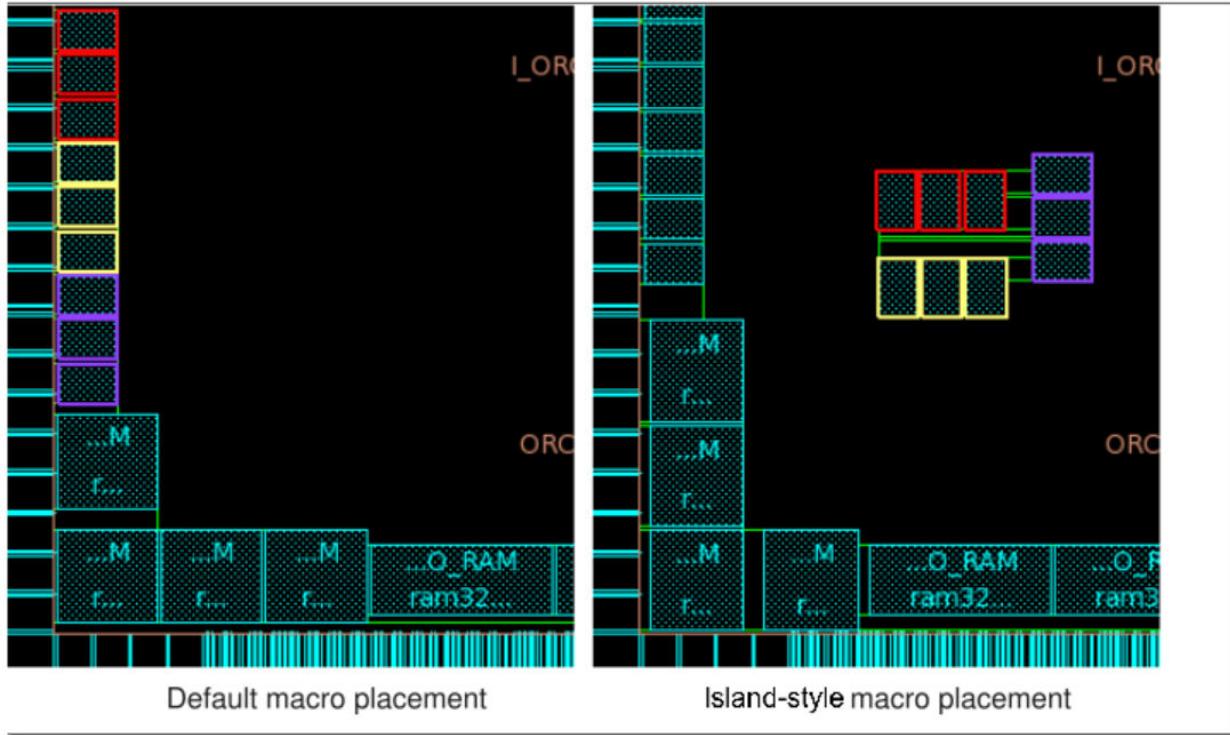
```
icc2_shell> set_attribute -objects [get_bounds bound1] \
 -name is_off_edge -value true
```

4. Perform global macro placement with the `create_placement -floorplan` command.

```
icc2_shell> create_placement -floorplan
```

In the following figure, the default on-edge macro placement is shown on the left. On the right, three group bounds are created and the preceding flow is used to create island-style macro placement for the design.

Figure 76 Default and Island Macro Placement



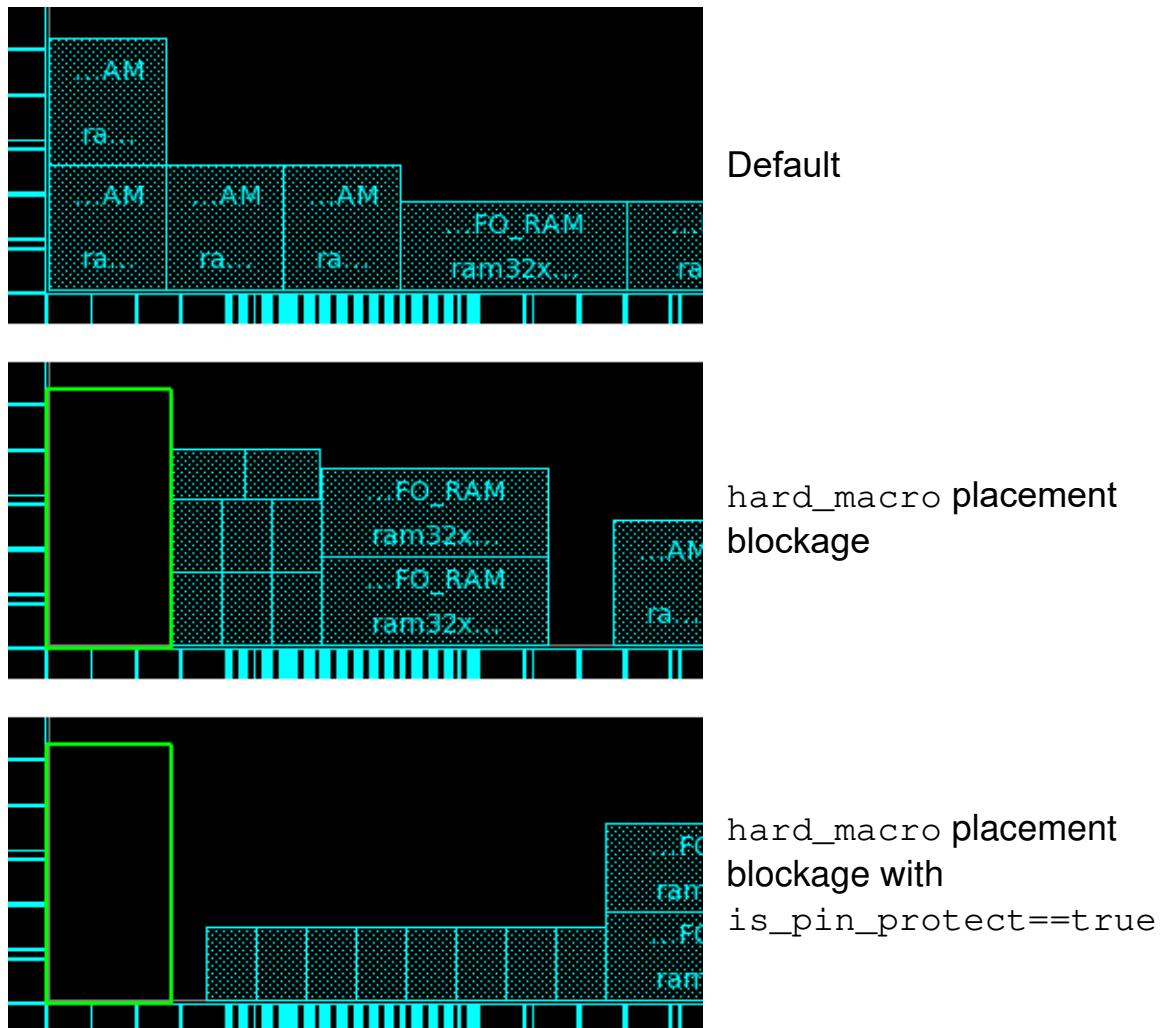
## Placing Macros Away From Block Edges

To avoid placing macros over a block, use the `create_placement_blockage -type hard_macro` command to create a hard macro placement blockage over the block. The tool avoids placing macros within the blockage, but might place the macros abutting the blockage. To reserve additional space around the blockage, set the `is_pin_protect` attribute on the hard macro placement blockage to `true`. This setting allows the tool to reserve space around the hard macro placement blockage area for pin placement. By default, the `is_pin_protect` attribute is `false` and the macro placer only avoids placing macro cells over the hard macro placement blockage area; macro cell placement is allowed at the edge of the placement blockage. For example, the following commands create a placement blockage and set the `is_pin_protect` attribute to `true` on the blockage:

```
icc2_shell> set pb [create_placement_blockage -type hard_macro \
 -boundary {{350 350} {500 500}}]
icc2_shell> set_attribute -objects $pb -name is_pin_protect \
 -value true
```

The following figure shows the layout results with no placement blockage, a `hard_macro` type placement blockage, and a `hard_macro` type placement blockage with the `is_pin_protect` attribute set to `true`.

Figure 77 Macros Placed Away From Block Edges

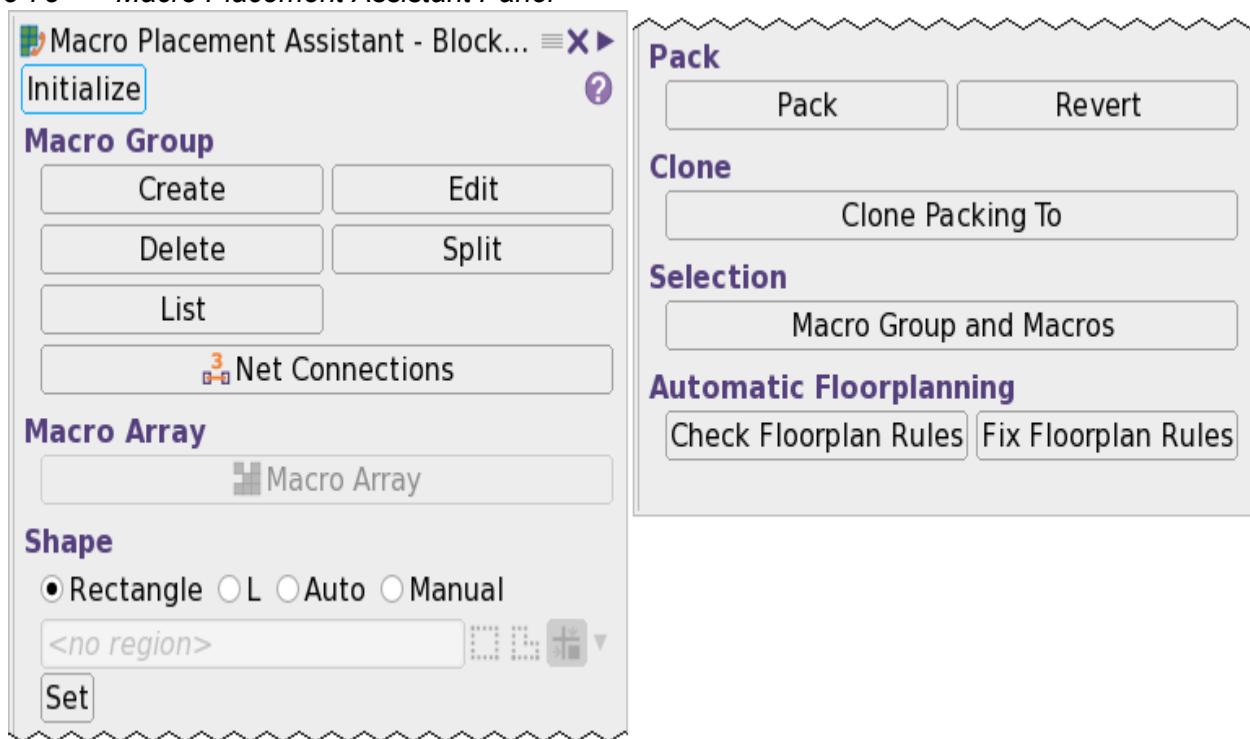


## Placing Macros Among Standard Cells

### Using the Macro Placement Assistant

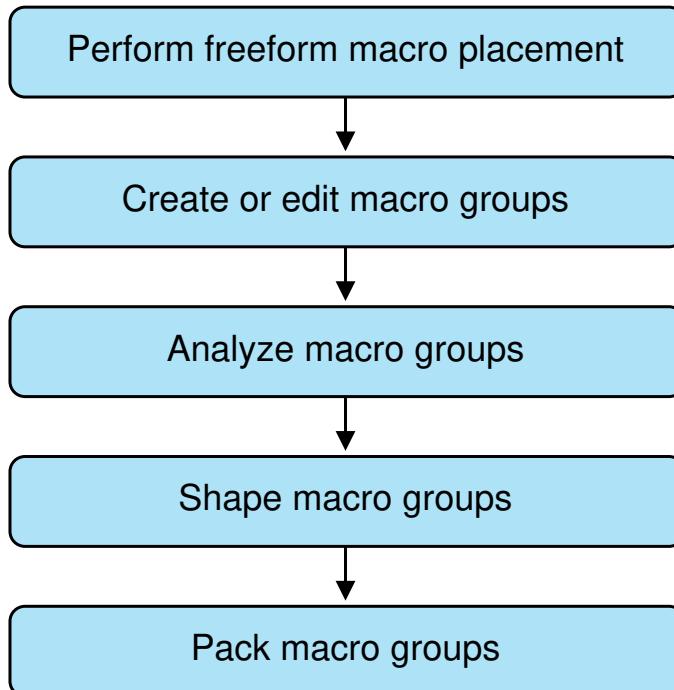
To improve the quality of your placement results when placing macros by hand, you can use the Macro Placement Assistant to group macros and experiment with different macro placements. Use this tool to reduce the number of narrow channels in your layout and improve routing congestion. To start the Macro Placement Assistant, choose View > Assistants > Macro Placement Assistant from the menu, or click the Macro Placement Assistant button on the toolbar. After starting the Macro Placement Assistant, click the Initialize button before proceeding.

Figure 78 Macro Placement Assistant Panel



You can use the following flow to create the initial macro placement and refine the macro placement with the Macro Placement Assistant.

Figure 79 Macro Placement Flow



## Creating Macro Groups

Macro groups are a collection of macro cells and macro arrays that should be placed together in the layout. When you create a macro group with the Macro Placement Assistant, the tool applies the `create_macro_groups` command to create a bound. The Macro Placement Assistant supports the following methods for creating groups of macros; to access these methods, click the Create button to open the Create Macro Groups dialog box.

### Auto

Creates one or more groups to produce the best grouping based on hierarchy, connectivity, and coarse placement. In the GUI, click Auto and then click OK. Alternatively, use the following Tcl command to create the macro groups:

```
create_macro_groups -method auto -macros
```

### By connectivity

Creates one or more groups based on the connections between the macros and flip-flops in the layout. In the GUI, click “By connectivity” and then click OK. To specify names for

the new macro groups, enter the names in the “Group name” box separated by spaces. Alternatively, use the following Tcl command to create the macro groups:

```
create_macro_groups -method by_connectivity -names {M1 M2 M3}
```

### **By hierarchy**

Creates one group for each level of hierarchy that contains macros. In the GUI, click “By hierarchy” and then click OK. To specify names for the new macro groups, enter the names in the “Group name” box separated by spaces. Alternatively, use the following Tcl command to create the macro groups:

```
create_macro_groups -method by_hierarchy -names {M1 M2 M3}
```

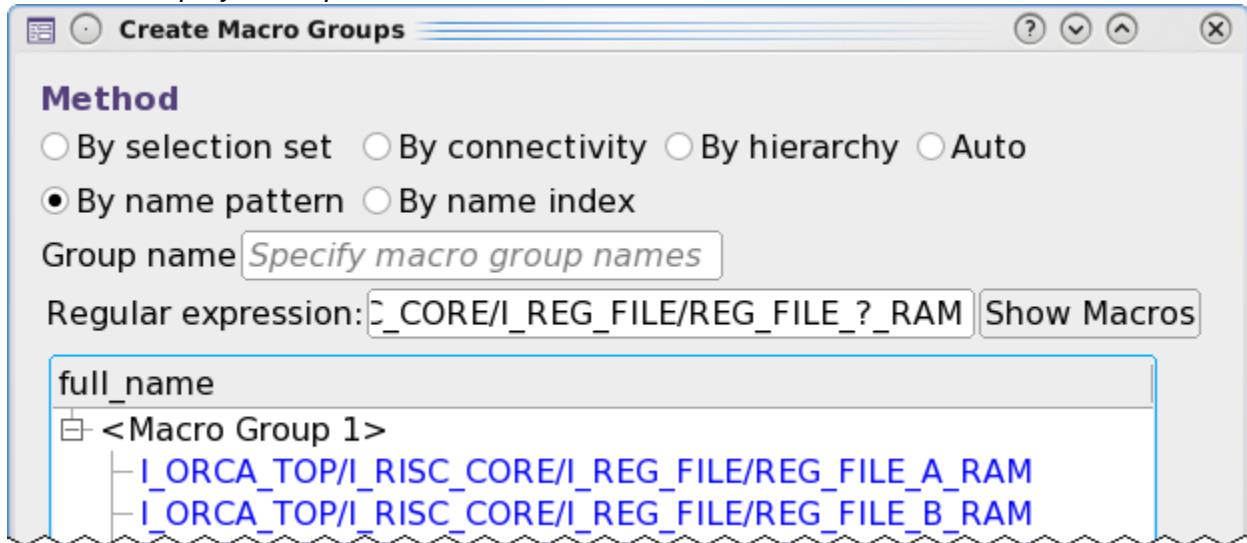
### **By name pattern**

Creates a macro group based on a regular expression. In the GUI, select a macro in the layout, click “By name pattern” and adjust the regular expression pattern as needed. The tool derives a regular expression based on the instance name of the selected macro. To specify names for the new macro groups, enter the names in the “Group name” box separated by spaces. Alternatively, enter a pattern string in the regular expression box; the pattern should contain the full path name and use “?” or “.” characters to allow matching of more than one macro. Click “Show Macros” to populate the macro list based on the pattern. Click OK to create the macro groups. Alternatively, use the following Tcl command:

```
create_macro_groups -method by_pattern \
 -pattern {I_ORCA_TOP/I_RISC_CORE/I_REG_FILE/REG_FILE_?_RAM}
```

In the following example, the pattern string `I_ORCA_TOP/I_RISC_CORE/I_REG_FILE/REG_FILE_?_RAM` matches macros `REG_FILE_A_RAM`, `REG_FILE_B_RAM`, `REG_FILE_C_RAM`, and `REG_FILE_D_RAM` under `I_ORCA_TOP/I_RISC_CORE/I_REG_FILE`.

Figure 80 Group by name pattern



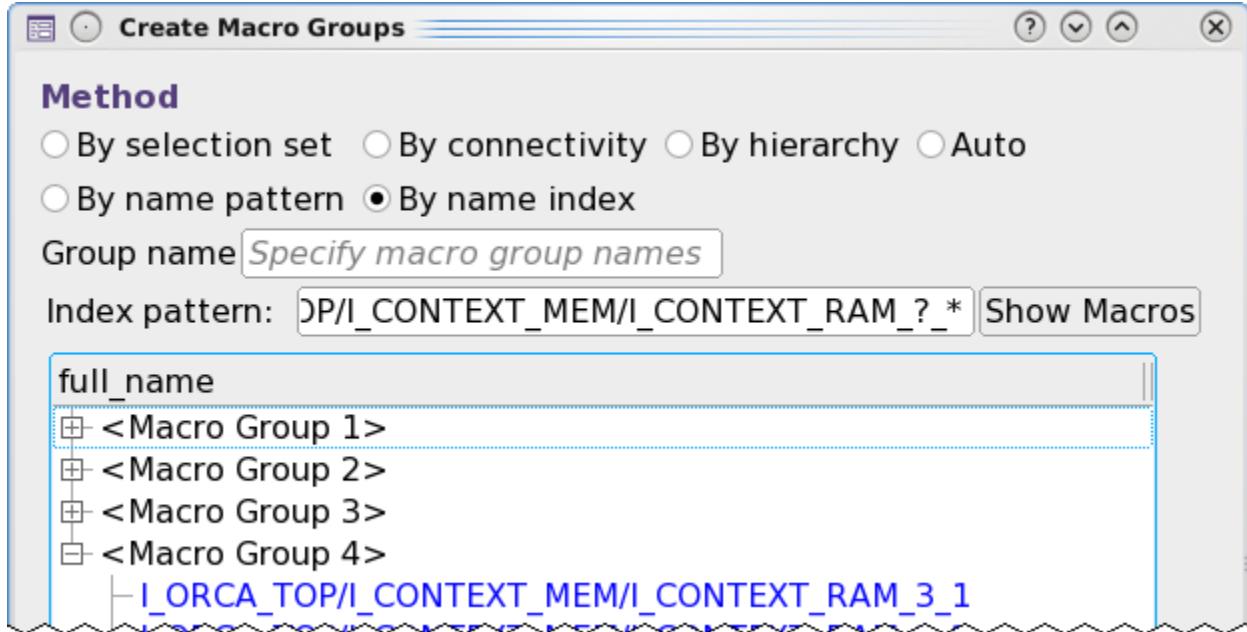
### By name index

Creates one or more macro groups based on a regular expression for the macro indexes. In the GUI, select a macro in the layout and click “By name index”. The tool tries to derive an index pattern based on the indexes at the end of the instance name of the selected macro. Alternatively, enter a pattern string in the index pattern box; the pattern should contain the full path name and use “?” or “.” characters to allow matching of more than one macro. Click “Show Macros” to populate the macro list based on the pattern. Alternatively, use the following Tcl command to create the macro groups:

```
create_macro_groups -method by_index_pattern \
 -index_pattern {I_ORCA_TOP/I_CONTEXT_MEM/I_CONTEXT_RAM_?_*}
```

In the following example, the pattern string `I_ORCA_TOP/I_CONTEXT_MEM/I_CONTEXT_RAM_?_*` matches 16 macros in the layout. The tool creates four groups of four macros each with the following index partitioning: {0\_1 0\_2 0\_3 0\_4}, {1\_1 1\_2 1\_3 1\_4}, {2\_1 2\_2 2\_3 2\_4}, and {3\_1 3\_2 3\_3 3\_4}.

Figure 81 Group by name index

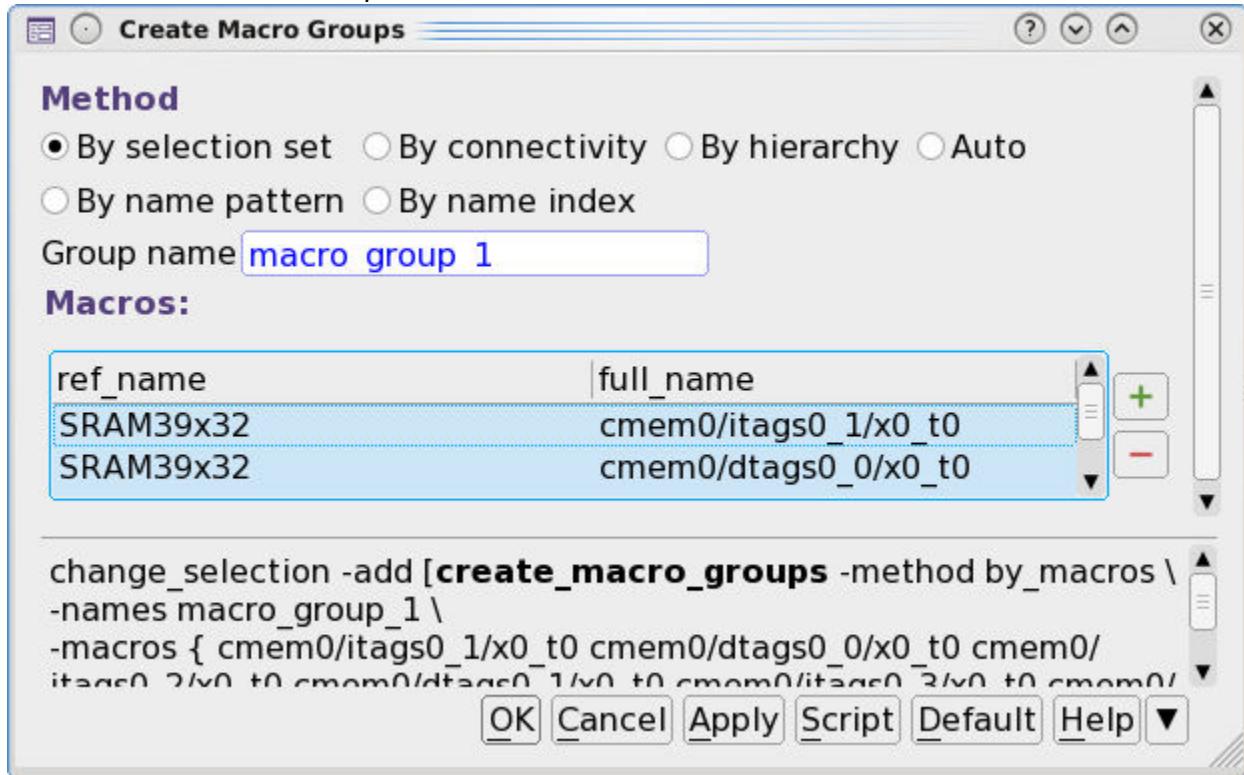


#### By selection set

Creates a single group that contains the currently selected macros. In the GUI, select the macros, click “By selection set”, and click OK. Add a new macro to the list by selecting it in the layout and clicking the [+] button. If the macro already belongs to a group, it is moved to a new group. Remove one or more macros from the list by selecting them in the list and clicking the [-] button. Alternatively, use the following Tcl command to create the macro groups:

```
create_macro_groups -method by_macros -macros [get_selection]
```

Figure 82 Create Macro Groups



## Performing Other Tasks on Macros

The Macro Placement Assistant supports other tasks that you can perform on macros and macro groups:

### Delete

Select one or more macro groups and click the Delete button to remove the currently selected macro groups.

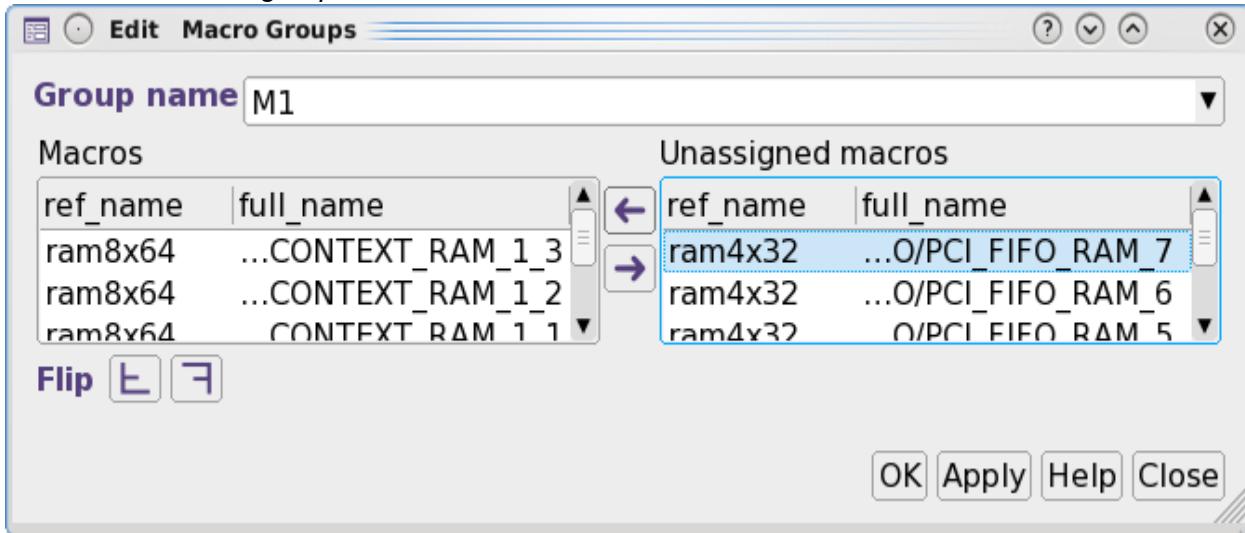
### Edit

Select one macro group in the layout and click Edit to add or delete macros from the group. You can perform the following tasks in this tool:

- Select one or more macros in the left or right boxes, then use the arrows to move the selected macros from one side to the other.
- Select one or more macros in the left or right boxes and click the orientation buttons in the Flip section to change the orientation of the selected macros to MX or MY.

Click OK when finished.

Figure 83 Edit macro groups



### List

Click the List button to show a list of the current macro groups. You can select macro groups, collections of macros, or individual macros in the layout by clicking their instance name in the list.

### Net Connections

Click the Net Connections button to open the Net Connections panel.

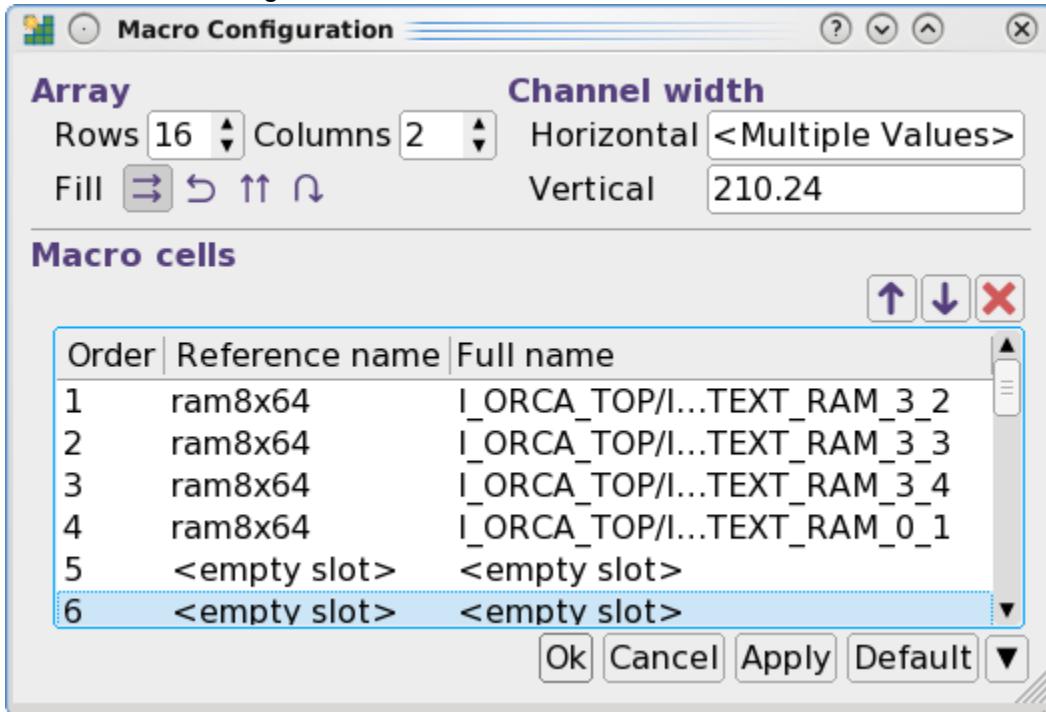
### Split

Create a new group and move some macros from the currently selected group into the new group based on connectivity.

### Macro Array

Select two or more macros in the layout and click the Macro Array button to start the Macro Array tool. Alternatively, choose Edit > Macro Array from the menu to open the Macro Configuration dialog box.

Figure 84 Macro configuration



### Macro Group Shape

Select one or more macro group shapes in the layout, click Rectangle, L, Auto, or Manual, then click Set to set the shape. Alternatively, use the `set_macro_group_shape` command to set the shape.

### Pack

Select one or more macro group shapes in the layout and click the Pack button to place the macros into the macro group shape. Alternatively, use the `pack_macro_group` command.

### Revert

Click the Revert button to undo the most recent pack operation. You can click the Revert button multiple times to perform multiple undos.

### Clone

Click the Clone button to copy the macro placement from one macro group to another. First, select the source macro placement group. Next, click the Clone Packing to button, choose the destination group from the list, and click OK.

### Macro Group and Macros

Select a macro in the layout and click the Macro Group and Macros button to select the corresponding macro group. Alternatively, select a macro group and click the Macro Group and Macros button to select the macros in the group.

## Performing Timing-Driven and Congestion-Driven Placement

The `create_placement` command supports both timing-driven and congestion-driven placement of macros and standard cells. Use command options with the `create_placement` command and the `plan.place.timing_driven_mode` and `plan.place.congestion_driven_mode` application options to control how timing-driven and congestion-driven placements are performed.

### Congestion-Driven Placement

To create a congestion-driven placement, use both the `-congestion` and `-floorplan` options with the `create_placement` command. When you use the `-congestion` option, you can also control the level of effort applied to congestion-driven placement with the `-congestion_effort low | medium | high` option. You can specify both timing-driven placement and congestion-driven placement within the same `create_placement` command.

Congestion-driven placement can perform three types of congestion reduction, based on the setting of the `plan.place.congestion_driven_mode` application option:

- `macro`: Reduces the congestion over and between the macros only
- `std_cell`: Reduces congestion between standard cells
- `both`: Reduces congestion both between macros and between standard cells

The following example applies congestion reduction only between hard macros, and performs coarse placement with high effort:

```
icc2_shell> set_app_options -name plan.place.congestion_driven_mode \
 -value macro
icc2_shell> create_placement -floorplan -congestion \
 -congestion_effort high
```

### Timing-Driven Placement

To create a timing-driven placement, use the `-timing_driven` and `-floorplan` options with the `create_placement` command. You can specify both timing-driven placement and congestion-driven placement within the same `create_placement` command.

Timing-driven placement can perform three types of placement, based on the setting of the `plan.place.timing_driven_mode` application option:

- `macro` : Performs timing-driven placement only for macros
- `std_cell` : Performs timing-driven placement only for standard cells
- `both` : Performs timing-driven placement for both macros and standard cells

The following example applies timing-driven placement only to standard cells; macros are placed normally.

```
icc2_shell> set_app_options -name plan.place.timing_driven_mode \
 -value std_cell
icc2_shell> create_placement -floorplan -timing_driven
```

### Combined Congestion-Driven and Timing-Driven Placement

To perform congestion-driven placement and timing-driven placement with the same `create_placement` command, use both the `-congestion` and `-timing_driven` options and specify the `plan.place.congestion_driven_mode` and `plan.place.timing_driven_mode` application options. The following example performs congestion-driven and timing-driven placement on both macros and standard cells.

```
icc2_shell> set_app_options -name plan.place.congestion_driven_mode \
 -value both
icc2_shell> set_app_options -name plan.place.timing_driven_mode \
 -value both
icc2_shell> create_placement -floorplan -congestion -timing_driven
```

# 10

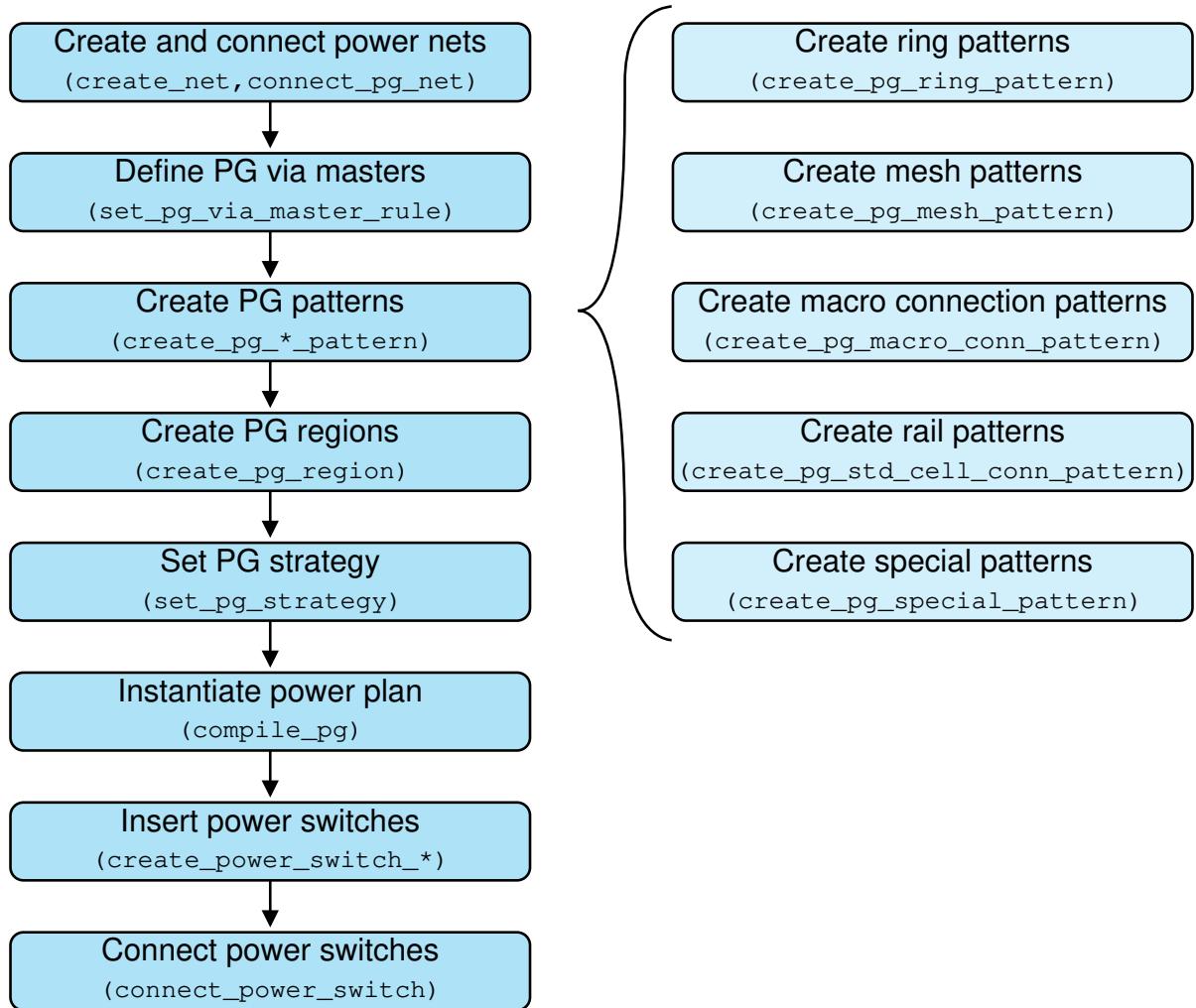
## **Performing Power Planning**

---

Power planning, which includes power network routing and power network analysis, is required to create a design with good power integrity. A design with a robust power and ground (PG) grid reduces 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 command option settings and regenerating the power plan.

The flow to create a power plan for your design is shown in [Figure 85](#).

*Figure 85 Pattern-Based Power Planning Flow*



The pattern-based power planning flow separates the physical implementation details (layer, width, spacing) of the power ring and mesh from the regions of the design where the structures are inserted. In a typical design, you run the flow multiple times. The first pass defines and creates the power rings, the second pass defines and creates the power mesh, and so on.

The `create_pg_ring_pattern`, `create_pg_mesh_pattern`, `create_pg_macro_conn_pattern`, and `create_pg_std_cell_conn_pattern` commands create pattern specifications that define the physical implementation details of the power plan and associate a width, spacing, offset, and via rule specification with the metal layers

in your design. After defining the patterns, they can be easily reused among different power regions in the design.

The `set_pg_strategy` command assigns a pattern specification to specific power nets and regions in the design. You can also define the offset for the start of the pattern, the power strap extension specification, and the blockage specification with this command.

You can create and implement complex via rules during pattern-based power planning. The `set_pg_via_master_rule` command defines the contact code, offset for the contact code within the via structure, cut spacing, and other parameters for the custom via structure. Power via masters are associated with a specific power plan strategy by specifying the `-via_rule` option with the appropriate `create_pg_*_pattern` command.

The `create_pg_*_pattern` and `set_pg_strategy` commands specify the power plan, but they do not modify the design. After creating the pattern and strategy definitions, use the `compile_pg` command to instantiate the power structures into the design. To make changes to the power plan after creating the power rings and meshes, remove the rings and meshes with the `compile_pg -undo` command and make changes to the via rules, pattern specifications, or power plan strategies.

After instantiating the power plan, you can validate the integrity of the power plan by checking for DRC and connectivity violations. Use the `check_pg_drc` command to report any power structures that violate the routing design rules. Use the `check_pg_connectivity` command to check the physical connectivity of the power network.

These topics are described in the following sections:

- [Creating and Connecting Power Nets](#)
- [Defining PG Via Masters](#)
- [Creating Power and Ground Ring Patterns](#)
- [Creating Power and Ground Mesh Patterns](#)
- [Creating Power and Ground Macro Connections](#)
- [Creating Power and Ground Standard Cell Rails](#)
- [Creating Channel and Alignment Power Straps](#)
- [Creating Complex Composite Patterns](#)
- [Defining Power Plan Regions](#)
- [Setting the Power Plan Strategy](#)
- [Creating Via Rules Between Different Strategies](#)
- [Instantiating the Power Plan](#)

- Pattern-Based Power Network Routing Example
- PG Script Generator
- Manually Creating Power Straps and Vias
- Using Secondary PG Constraints
- Honoring Metal and Via Spacing Rules to Blockages
- Using PG Pattern Shapes
- Using Via Matrixes
- Debugging Vias and PG Straps Removed Due to DRC Violations
- Merging Shapes in the PG Mesh
- Inserting Stapling Vias Into an Existing Power Mesh
- Connecting Via Ladders to Power and Ground
- Inserting and Connecting Power Switches
- Trimming the Power and Ground Mesh
- Checking Power Network Integrity
- Analyzing the Power Plan
- Performing Distributed Power Network Routing

---

## Creating and Connecting Power Nets

To begin pattern-based power planning, define the power and ground nets in your design and connect them to the power and ground pins. If you use UPF to describe your power network, you can skip these steps.

1. Define the power and ground nets with the `create_net` command.

```
icc2_shell> create_net -power VDD
{VDD}
icc2_shell> create_net -ground VSS
{VSS}
```

2. Connect the power and ground nets to power and ground pins with the `connect_pg_net` command.

```
icc2_shell> connect_pg_net -net VDD [get_pins -physical_context *VDD]
icc2_shell> connect_pg_net -net VSS [get_pins -physical_context *VSS]
```

If your design has a UPF description of the power network, you can specify `connect_pg_net -automatic` to derive the power and ground nets directly from the power domain specification and perform the connection.

## Defining PG Via Masters

If your design requires a via master that is not defined in the technology file, you can create a via master for your power network with the `set_pg_via_master_rule` command. You can use the new via master in the power network by referencing it with the `-via_rule` option of the `create_pg_composite_pattern`, `create_pg_macro_conn_pattern`, `create_pg_mesh_pattern`, or `create_pg_ring_pattern` command.

To create a new via master for the power network, use the `set_pg_via_master_rule` command as shown in the following example.

```
icc2_shell> set_pg_via_master_rule VIA78_2x2 -contact_code VIA78 \
-via_array_dimension {2 2} -offset {3 1}
```

To create an offset from the origin of the via bounding box, use the `-offset {x y}` option. To specify a via master or contact code to use for this via, use the `-contact_code code_name` option. To specify the dimensions of the via array for this rule, use the `-via_array_dimension {column row}` option. To specify the horizontal and vertical pitch to use when filling a via rule with multiple repetitions of the contact code or via array, use the `-allow_multiple {x_pitch y_pitch}` option. To specify the horizontal and vertical spacing values between cuts in a simple array via, use the `-cut_spacing {horizontal_spacing vertical_spacing}` option.

To use the new rule in your design, specify the via rule name after the `via_master` keyword for the `create_pg_composite_pattern`, `create_pg_macro_conn_pattern`, `create_pg_mesh_pattern`, and `create_pg_ring_pattern` commands. For example, the following command uses the `VIA78_2x2` via rule defined by the previous command.

```
icc2_shell> create_pg_mesh_pattern mesh_pat \
-layers {layer_spec} \
-via_rule {
 {{layers: M6} {layers: M7} {via_master: default}} \
 {{layers: M7} {layers: M8} {via_master: VIA78_2x2}}}
```

### Using the Via Master With the Maximum Total Cut Area

To use the via master or ContactCode with the maximum total cut area based on the bounding box for the route intersection, set the

`plan.pgroute.maximize_total_cut_area` application option to one of the following values:

- `all` : Consider all possible ContactCodes.
- `default` : Use the default ContactCode.
- `Vs` : Use a ContactCode with square cuts
- `Vh` : Use a ContactCode with horizontal bar shape
- `Vv` : Use a ContactCode with vertical bar shape

## Creating Power and Ground Ring Patterns

The ring pattern specifies the horizontal and vertical layer names, ring width values, spacing values, vias, and corner bridging to use to create the power and ground ring. You can create ring patterns around the core, design blocks, macros, power and ground regions, or rectilinear polygons that you specify. To define the ring pattern, use the `create_pg_ring_pattern` command as follows:

```
icc2_shell> create_pg_ring_pattern ring_pat -horizontal_layer M7 \
 -horizontal_width {10} -horizontal_spacing {2} \
 -vertical_layer M8 -vertical_width {10} \
 -vertical_spacing {2} -corner_bridge false
```

Alternatively, use the Task Assistant to define the ring pattern.

1. Select Task > Task Assistant in the GUI.
2. Select PG Planning > Ring/Hard Macro/Std Cell Pattern in the Task Assistant.
3. Click the Ring tab.
4. Enter the pattern name, layer names, widths, and other ring parameters in the Task Assistant.
5. Click Apply to create the pattern.

You can also define the width, spacing, and other values for the `create_pg_ring_pattern` command by using parameters. To create a parameter, include the `-parameters` option followed by the parameter name, and replace the value for the command option with a parameter name. Parameters are identified by the parameter name preceded by the (@) character.

The values for the parameters remain undefined until they are set with the `set_pg_strategy` command. Parameters are set in the same order they are specified by the `-parameters` option. Using this approach, you can reuse the pattern specified by the command and apply different parameter values with the `set_pg_strategy` command.

The following example uses parameters to define the metal layer, width, spacing, and corner bridge settings for the ring pattern.

```
icc2_shell> create_pg_ring_pattern ring_pat -horizontal_layer @hlayer \
 -horizontal_width {@hwidth} -horizontal_spacing {@hspace} \
 -vertical_layer @vlayer -vertical_width {@vwidth} \
 -vertical_spacing {@vspace} -corner_bridge @cbridge \
 -parameters {hlayer hwidth hspace
 vlayer vwidth vspace cbridge}

icc2_shell> set_pg_strategy ring_strat -core \
 -pattern {{name: ring_pat} {nets: {VDD VSS}}
 {offset: {3 3}} {parameters: {M7 10 2 M8 10 2 true}}} \
 -extension {{stop: design_boundary}}
```

## Creating Power and Ground Mesh Patterns

The mesh pattern specifies the horizontal and vertical layer names, metal width values, metal spacing values, metal pitch, vias, and wire trimming to use to create the power and ground mesh. To define the mesh pattern, use the `create_pg_mesh_pattern` command as follows:

```
icc2_shell> create_pg_mesh_pattern mesh_pat -layers {
 {{vertical_layer: M8} {width: 5}
 {spacing: interleaving} {pitch: 32}}
 {{vertical_layer: M6} {width: 2}
 {spacing: interleaving} {pitch: 32}}
 {{horizontal_layer: M7} {width: 5}
 {spacing: interleaving} {pitch: 28.8}}} \
-via_rule {
 {{layers: M6} {layers: M7} {via_master: default}}
 {{layers: M8} {layers: M7} {via_master: VIA78_3x3}}}
```

Alternatively, use the Task Assistant to define the mesh pattern.

1. Select Task > Task Assistant in the GUI.
2. Select PG Planning > Mesh Pattern in the Task Assistant.
3. Enter the pattern name and define the via rule.
4. Select the layer, direction, width, spacing, pitch, and offset values.
5. Click the entry in the table row to edit the setting.
6. Click Preview to view the Tcl command in the Tcl Command box.
7. Click Run Tcl Command to create the mesh pattern.

You can also define the settings for the power mesh by using parameters as described in [Creating Power and Ground Ring Patterns](#). The following example uses parameters to define the metal width values for the `create_pg_mesh_pattern` command.

```
icc2_shell> create_pg_mesh_pattern mesh_pat -layers {
 {{vertical_layer: M8} {width: @width8}
 {spacing: interleaving} {pitch: 32}}
 {{vertical_layer: M6} {width: @width6}
 {spacing: interleaving} {pitch: 32}}
 {{horizontal_layer: M7} {width: @width7}
 {spacing: interleaving} {pitch: 28.8}}} \
-via_rule {
 {{layers: M6} {layers: M7} {via_master: default}}
 {{layers: M8} {layers: M7} {via_master: VIA78_3x3}}}} \
-parameters {width6 width7 width8}

icc2_shell> set_pg_strategy mesh_strat -core -pattern {
 {name: mesh_pat} {nets: {VDD VSS}}
 {parameters: {32 28.8 32}}}
```

## Creating Power and Ground Macro Connections

The macro connection pattern specifies the power and ground nets, routing direction, metal layers, layer width, layer spacing, metal pitch and other information to create connections to macros. To define the macro connection pattern, use the `create_pg_macro_conn_pattern` as in the following example.

```
icc2_shell> create_pg_macro_conn_pattern macro_pat -nets {VDD VSS} \
 -direction horizontal -width 1 -layers M5 -spacing minimum \
 -pitch 5 -pin_conn_type long_pin
```

Alternatively, use the Task Assistant to define the standard cell rail pattern.

1. Select Task > Task Assistant in the GUI.
2. Select PG Planning > Ring/Hard Macro/Std Cell Pattern.
3. Click the HM scattered pin, HM long pin, or HM ring pin tab depending on the type of hard macro pin connection.
4. Enter the pattern name, layer, width, and other information in the form.
5. Click Apply to create the pattern.

You can also define the settings for the standard cell rails by using parameters as described in [Creating Power and Ground Ring Patterns](#). The following example uses parameters to define the metal width for the standard cell rail pattern, and assigns values to the parameters with the `set_pg_strategy` command.

```
icc2_shell> create_pg_macro_conn_pattern macro_pat -nets {VDD VSS} \
 -direction horizontal -width @width -layers M5 -spacing minimum \
 -pitch @pitch -pin_conn_type long_pin -parameters {width pitch}

icc2_shell> set_pg_strategy macro_strat -core \
 -pattern {{pattern: macro_pat} {nets: {VDD VSS}} \
 {parameters: {1 5}}}
```

## Creating Power and Ground Standard Cell Rails

The standard cell rail pattern specifies the metal layers, rail width, and rail offset to use to create the power and ground rails for the standard cell rows. To define the standard cell rail pattern, use the `create_pg_std_cell_conn_pattern` as in the following example.

```
icc2_shell> create_pg_std_cell_conn_pattern rail_pat -layers {M1} \
 -rail_width {0.2 0.2}
```

Alternatively, use the Task Assistant to define the standard cell rail pattern.

1. Select Task > Task Assistant in the GUI.
2. Select PG Planning > Ring/Hard Macro/Std Cell Pattern in the Task Assistant.
3. Click the Std cell tab.
4. Enter the pattern name, layer name, rail width, and offset in the Task Assistant.
5. Click Apply to create the pattern.

If you do not define a rail width, the command uses the width of the standard cell power and ground pins as the width.

You can also define the settings for the standard cell rails by using parameters as described in [Creating Power and Ground Ring Patterns](#). The following example uses parameters to define the metal width for the standard cell rail pattern, and assigns values to the parameters with the `set_pg_strategy` command.

```
icc2_shell> create_pg_std_cell_conn_pattern rail_pat -layers {M1} \
 -rail_width {@wtop @wbottom} -parameters {wtop wbottom}

icc2_shell> set_pg_strategy rail_strat -core \
 -pattern {{name: rail_pat} {nets: VDD VSS} \
 {parameters: {0.2 0.2}}}
```

---

## Creating Channel and Alignment Power Straps

The special pattern defines the structure for the following types of power ground straps:

- Straps placed in channels
- Straps and vias inserted to connect and align terminals
- Straps inserted to align power switches
- Straps inserted to align physical-only and filler cells

Only one pattern type can be specified within a single `create_pg_special_pattern` command. To define these patterns, use the `create_pg_special_pattern` command as in the following example.

```
icc2_shell> create_pg_special_pattern channel_pattern \
 -insert_channel_straps {{layer: M6} {direction: vertical} \
 {width: 2} \
 {channel_between_objects: {macro placement_blockage voltage_area} \
 }}
```

Alternatively, use the Task Assistant to define the special pattern.

1. Select Task > Task Assistant in the GUI.
2. Select PG Planning > Special Pattern in the Task Assistant.
3. Click the Channel straps tab. The Switch alignment, Terminal alignment, Physical cell alignment, and Max stdcell distance tabs are also available.
4. Enter the information for the form, such as the layer name, direction, width, spacing, and other settings.
5. Click Apply to create the pattern.

---

## Creating Complex Composite Patterns

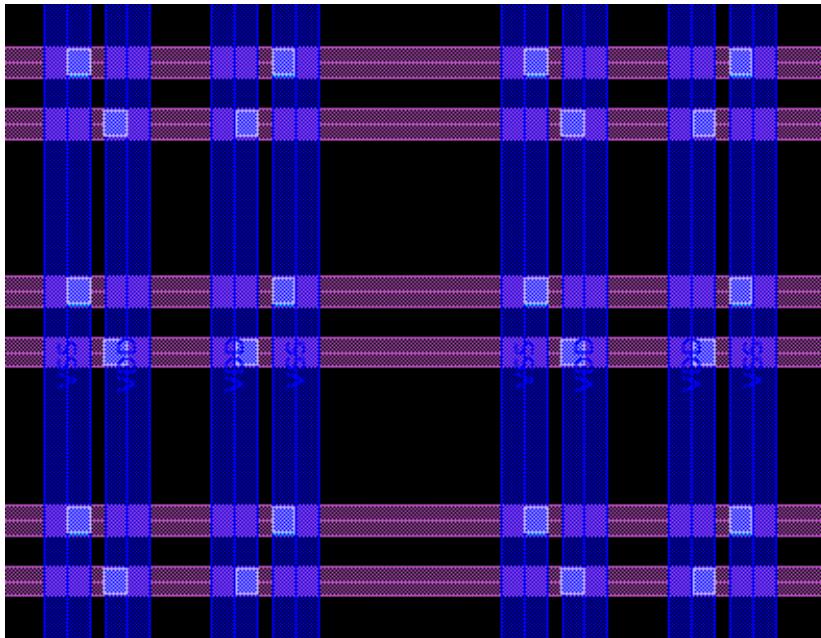
This section uses examples to create complex PG patterns using the `create_pg_composite_pattern` command.

- [Creating Center-Aligned Vias](#)
- [Creating Bridging Straps](#)
- [Creating Via Bridging Straps](#)
- [Creating Tapering Straps](#)
- [Creating a Checkerboard Via Pattern](#)

## Creating Center-Aligned Vias

The following example creates horizontal power straps on layer M3 and vertical power straps on layer M8. Stacked vias connect the horizontal and vertical straps, and the vias are pushed toward the center of each VDD/VSS vertical power strap pair to save routing resources beneath the straps. The `create_pg_wire_pattern` command defines the base pattern, and the `create_pg_composite_pattern` command defines the parameters for the horizontal and vertical PG routes.

*Figure 86 Power Straps With Center-Aligned Vias*



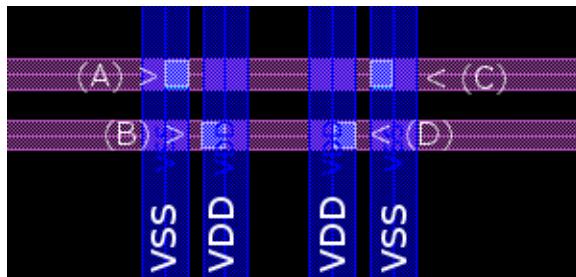
This example defines two via master rules: `via_5x6_shift_right` and `via_5x6_shift_left`. For each via, the `-offset` option is applied to shift the via from center by 0.8 microns.

The `-via_rule` option of the `create_pg_composite_pattern` command specifies which via is placed at each intersection of layers M3 and M8 as shown in [Figure 87](#). In this example, the `-add_patterns` option instantiates two `create_pg_wire_pattern` definitions: the vertical power strap pattern on layer M8 (`pattern_id: 1`), and the horizontal power strap pattern on layer M3: (`pattern_id: 2`). The `create_pg_composite_pattern` also refers to four net ids for the two PG nets: VSS (`net_id: 1` and `4`) and VDD (`net_id: 2` and `3`).

The `via_5x6_shift_right` via is inserted at the intersections of `pattern_id: 1` and `net_id: 1` of the first pattern (layer M8 and net VSS), and `pattern_id: 2` and `net_id: 2` of the second pattern (layer M3 and net VSS); location (A) in the figure. The `via_5x6_shift_right` via is also inserted at `pattern_id: 1` and `net_id: 3` (layer M8 and net VDD), and

pattern\_id: 2 and net\_id: 1 (layer M3 and net VDD); location (D) in the figure.  
 Similarly, the remaining statements define vias at locations (B) and (C).

**Figure 87 Annotated Center-Aligned Vias**



The commands to create this power plan are as follows:

```
set_pg_via_master_rule via_5x6_shift_right \
 -via_array_dimension {5 6} -offset {0.8 0}

set_pg_via_master_rule via_5x6_shift_left \
 -via_array_dimension {5 6} -offset {-0.8 0}

create_pg_wire_pattern wire_base -layer @l -direction @d \
 -width @w -spacing @s -pitch @p -parameters {l d w p s}

For the vertical straps on M8 defined by pattern_id: 1
- net_id: 1 is VSS
- net_id: 2 is VDD
- net_id: 3 is VDD
- net_id: 4 is VSS
#
For the horizontal straps on M3 defined by pattern_id: 2
- net_id: 1 is VDD
- net_id: 2 is VSS

create_pg_composite_pattern center_aligned_vias -nets {VDD VSS} \
 -add_patterns { \
 {{pattern: wire_base} {nets: VSS VDD VDD VSS} \
 {parameters: {M8 vertical 3 30 {1 4 1}}}{offset: 4}} \
 {{pattern: wire_base} {nets: VDD VSS} \
 {parameters: {M3 horizontal 2 15 2}}{offset: 4}} \
 } \
 -via_rule { \
 {{pattern_id: 1}{net_id: 1}} \
 {{pattern_id: 2}{net_id: 2}} \
 {via_master: via_5x6_shift_right} \
 {{pattern_id: 1}{net_id: 2}} \
 {{pattern_id: 2}{net_id: 1}} \
 {via_master: via_5x6_shift_left} \
 {{pattern_id: 1}{net_id: 3}} \
 {{pattern_id: 2}{net_id: 1}} \
 }
```

```

{via_master: via_5x6_shift_right}} \
{{pattern_id: 1}{net_id: 4}} \
{{pattern_id: 2}{net_id: 2}} \
{via_master: via_5x6_shift_left}} \
{{intersection: undefined}{via_master: NIL}} \
}
set_pg_strategy core_patterns \
-pattern {{name: center_aligned_vias} \
{nets: VDD VSS } {offset: {5 5}}} -core
compile_pg -strategies core_patterns

```

## Creating Bridging Straps

The following example creates bridging straps to connect parallel vertical power straps. This power plan contains the following features:

- Three VDD power straps of width 2 microns, 4 microns, and 2 microns form a group of vertical straps on layer M6
- Three M6 VSS power straps of width 2 microns, 4 microns, and 2 microns also form a group of vertical straps on layer M6
- The groups of three vertical VDD straps are connected together with a 2 micron wide strap on layer M3 with a pitch of 15 microns, length of 12 microns, and 5x1 stacked via arrays between M6 and M3
- The groups of three vertical VSS straps are connected together with a 2 micron wide strap on layer M5 with a pitch of 15 microns, length of 12 microns, and 5x3 via arrays between M5 and M6

The bridging straps and via structures used to connect the straps are shown in [Figure 88](#) and [Figure 89](#).

*Figure 88 Bridging Straps*

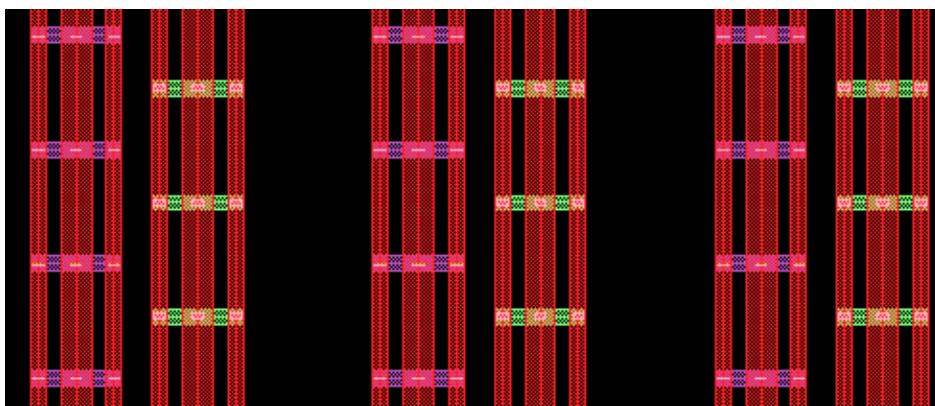
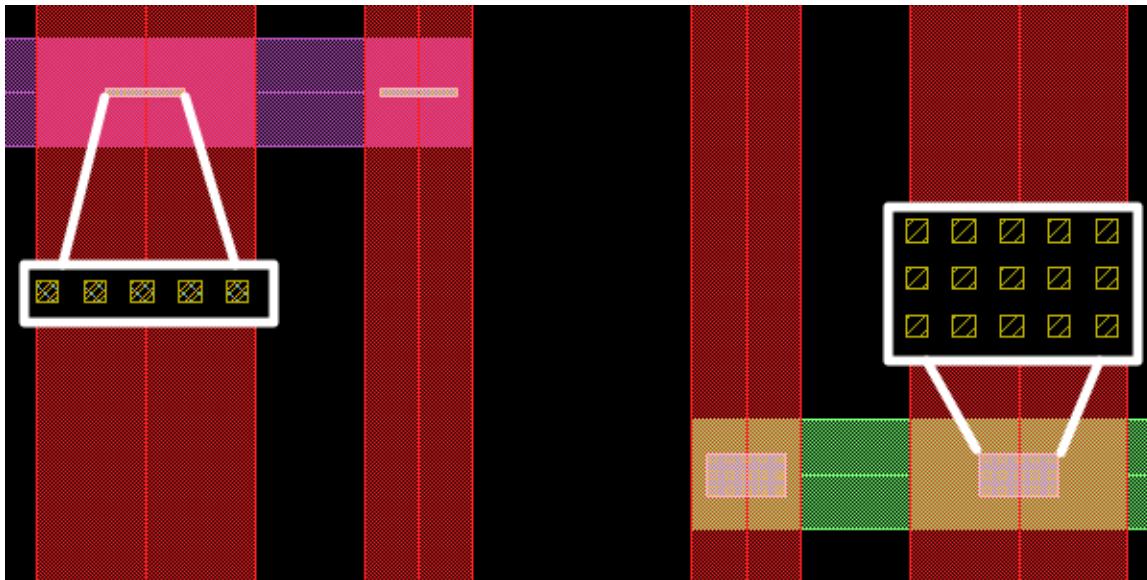


Figure 89 5x1 and 5x3 Vias



The `set_pg_via_master_rule` command defines the via rules for the 5x1 and 5x3 via arrays. The `create_pg_composite_pattern` command defines a power plan that uses two distinct wire patterns: one pattern called `wire_base` used to define the VDD and VSS vertical straps, one pattern called `segment_base` used to define both the M3 layer straps that connect VDD and the M5 layer straps that connect VSS. The wire patterns are referenced using the `{pattern: wire_base}` and `{pattern: segment_base}` statements.

The `-via_rule` option to `create_pg_composite_pattern` defines what vias are inserted at different metal intersections in the power plan. The `{}{layers: M6} {}{layers: M5} {via_master: via65_5x3}` statement inserts a via65\_5x3 via master between any intersection of layers M6 and M5 in the power plan.

The `-via_rule` statement `{}{intersection: undefined} {via_master: NIL}` ensures that only the specified vias are inserted in the power plan. For designs that contain many wire patterns, this statement ensures that no vias are inserted between undefined layer pairs; by default, vias are created for undefined layer pairs.

The commands used to implement this power plan are as follows:

```
set_pg_via_master_rule via63_5x1 \
 -contact_code {VIA34 VIA45 VIA56} \
 -via_array_dimension {5 1}

set_pg_via_master_rule via65_5x3 \
 -contact_code {VIA56} \
 -via_array_dimension {5 3}

create_pg_wire_pattern wire_base -layer @1 -direction @d \
```

```

 -width @w -spacing @s -pitch @p -parameters {l d w p s}

create_pg_wire_pattern segment_base -layer @l -direction @d \
 -width @w -low_end_reference_point 0 \
 -high_end_reference_point @len -pitch @p \
 -parameters {l d w len p}
create_pg_composite_pattern bridging_straps -nets {VDD VSS} \
 -add_patterns { \
 {{pattern: wire_base} {nets: VDD VSS} \
 {parameters: {M7 horizontal 2 60 2}}{offset: 13}} \
 {{pattern: wire_base} {nets: VDD VDD VDD VSS VSS VSS} \
 {parameters: {M6 vertical {2 4 2 2 4 2} 45 {2 2 4 2 2}}}} \
 {offset: 4}} \
 {{pattern: segment_base} {nets: VDD} \
 {parameters: {M3 horizontal 2 12 {45 15}}}} {offset: {3 2}}}} \
 {{pattern: segment_base} {nets: VSS} \
 {parameters: {M5 horizontal 2 12 {45 15}}}} {offset: {19 10}}}} } \
 -via_rule { \
 {{layers: M7}{layers: M6} {via_master: default}} \
 {{layers: M6}{layers: M5} {via_master: via65_5x3}} \
 {{layers: M6}{layers: M3} {via_master: via63_5x1}} \
 {{intersection: undefined} {via_master: NIL}} \
 }
}

set_pg_strategy core_patterns \
 -pattern {{name: bridging_straps} \
 {nets: VDD VSS } {offset: {5 5}}} -core

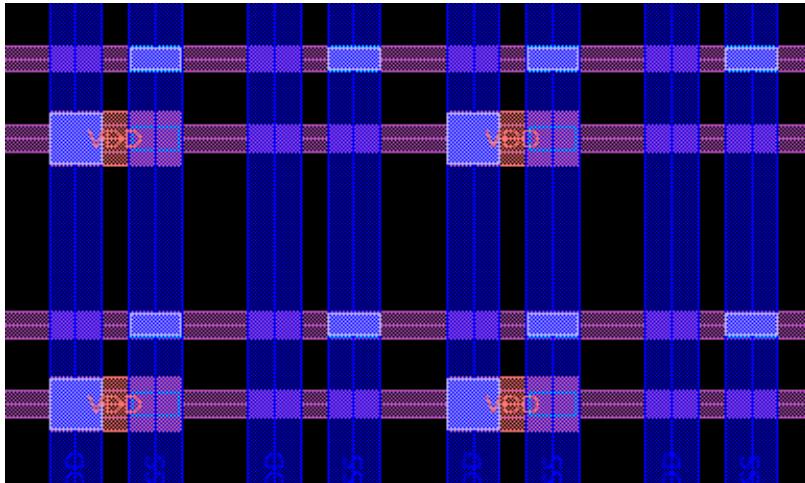
compile_pg -strategies core_patterns

```

## Creating Via Bridging Straps

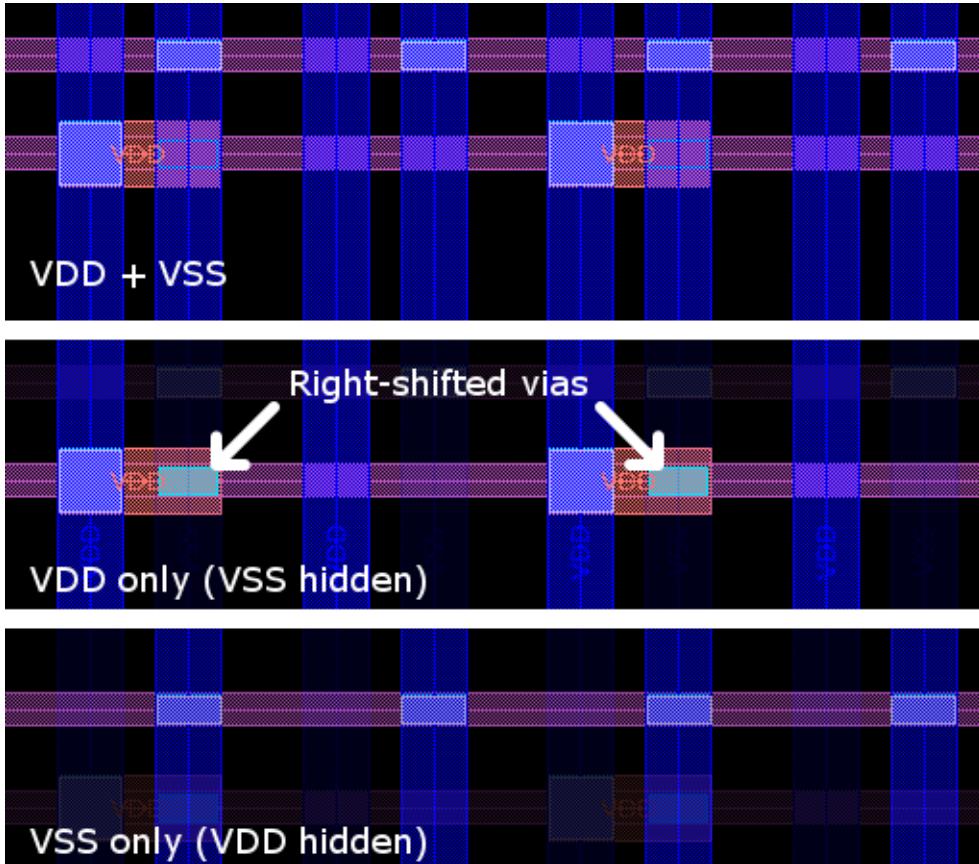
The following example creates a complex, shifted via bridging structure to connect the vertical VDD straps on layer M8 to the horizontal VDD straps on layer M3. The horizontal and vertical VSS straps use a standard stacked via to connect layers M8 and M3.

Figure 90 Composite Bridging Patterns



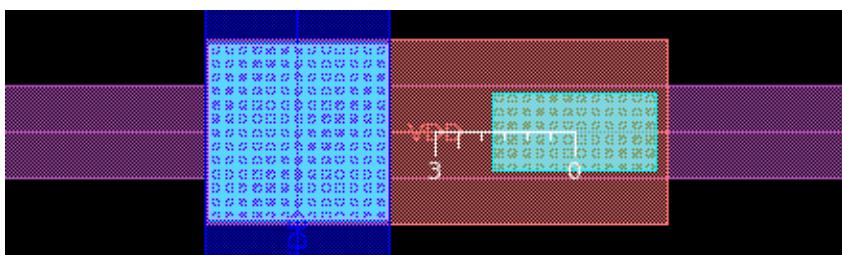
For the VDD power straps, a 13x13 via array connects the vertical straps on layer M8 with the short horizontal metal segment on layer M7. On the same M7 segment, a smaller 6x12 stacked via array connects M7 with the horizontal VDD strap on layer M3.

Figure 91 Composite Bridging Pattern Structure



The composite pattern technique allows you to shift the via locations and open additional vertical routing tracks to reduce routing jogs and improve timing. The via73\_12x6\_shift\_right via array is typically inserted in the center of intersection between the M7 layer metal segment and the M3 layer horizontal strap. In this example, the via is shifted to the right with the `-offset {3 0}` option of the `set_pg_via_master_rule` command. The offset is shown in Figure 92. Vertical routing tracks on layers M6 and M4 that are beneath the M8 layer VDD straps are not used for PG and can be used for signal routing.

Figure 92 Via Offset From Center



The `set_pg_via_master_rule` command defines the via rule for the 12x6 stacked via array. The `create_pg_composite_pattern` command defines a composite pattern that uses two distinct wire patterns: `wire_base` and `segment_base`. The `wire_base` pattern specifies the vertical straps on M8 and the horizontal straps on M3 for VDD and VSS. The `segment_base` pattern specifies the short 10 micron wide horizontal segments for net VDD on layer M7. The wire patterns are referenced using the `{pattern: wire_base}` and `{pattern: segment_base}` statements.

The `-via_rule` option specifies where vias are inserted for each layer intersection. Consider the following definition for the right-shifted via73\_12x6\_shift\_right stacked via:

```
{{{pattern_id: 2}{net_id: 1}}
 {{pattern_id: 3}{net_id: 1}}
 {via_master: via73_12x6_shift_right} {between_parallel: true}}
```

In the previous definition,  `{{pattern_id: 2}{net_id: 1}}` refers to the second pattern specified by the `-add_patterns` option: `{pattern: wire_base} ... {parameters: {M3 horizontal ...}}`, and the first net specified by the `-nets` option: VDD. Likewise,  `{{pattern_id: 3}{net_id: 1}}` refers to the third pattern definition: `{pattern: segment_base} ... {parameters: {M7 horizontal ...}}`, and the first net: VDD. Based on this definition, the tool inserts a via73\_12x6\_shift\_right stacked via at the intersection of the M3 and M7 straps. The `{between_parallel: true}` statement allows the tool to insert the vias between parallel straps.

The commands to create this power plan, including the via master rules, are as follows:

```
set_pg_via_master_rule via73_12x6_shift_right \
 -contact_code {VIA34 VIA45 VIA56 VIA67} \
 -via_array_dimension {12 6} -offset {3 0}

create_pg_wire_pattern wire_base -layer @l -direction @d \
 -width @w -spacing @s -pitch @p -parameters {l d w p s}

create_pg_wire_pattern segment_base -layer @l -direction @d \
 -width @w -low_end_reference_point 0 \
 -high_end_reference_point @len -pitch @p \
 -parameters {l d w len p}

create_pg_composite_pattern via_bridging -nets {VDD VSS} \
 -add_patterns { \
 {{pattern: wire_base} {nets: VDD VSS} \
 {parameters: {M8 vertical 4 15 2}}{offset: 4}} \
 {{pattern: wire_base} {nets: VDD VSS} \
 {parameters: {M3 horizontal 2 20 4}}{offset: 4}} \
 {{pattern: segment_base} {nets: VDD} \
 {parameters: {M7 horizontal 4 10 {30 20}}}{offset: {2 4}}} \
 } \
 -via_rule { \
 {{{pattern_id: 1}{net_id: 1}} \
 {{pattern_id: 2}{net_id: 1}} {via_master: NIL}} \
 {{{pattern_id: 1}{net_id: 2}}}
```

```

 {{pattern_id: 2}{net_id: 2}} {via_master: default} } \
 {{pattern_id: 1}{net_id: 1}} \
 {{pattern_id: 3}{net_id: 1}} {via_master: default} } \
 {{pattern_id: 2}{net_id: 1}} \
 {{pattern_id: 3}{net_id: 1}} \
 {via_master: via73_12x6_shift_right} {between_parallel: true} } \
 {{intersection: undefined} {via_master: NIL} } \
 }

set_pg_strategy core_patterns \
 -pattern {{name: via_bridging} \
 {nets: VDD VSS } {offset: {5 5}}} -core

compile_pg -strategies core_patterns

```

## Creating Tapering Straps

The following example creates a power structure with narrowed power straps at the center of the die. Vertical VDD and VSS straps are created on layer M6 and horizontal VDD and VSS straps are created on layer M7. At the periphery of the die, horizontal M7 straps are 10 microns wide and vertical M6 straps are 8 microns wide. At the center of the die, the M7 straps narrow to 5 microns and M6 straps narrow to 4 microns. The green square annotates the bounding box {{500 500} {1000 1000}}.

*Figure 93 Tapering Straps*

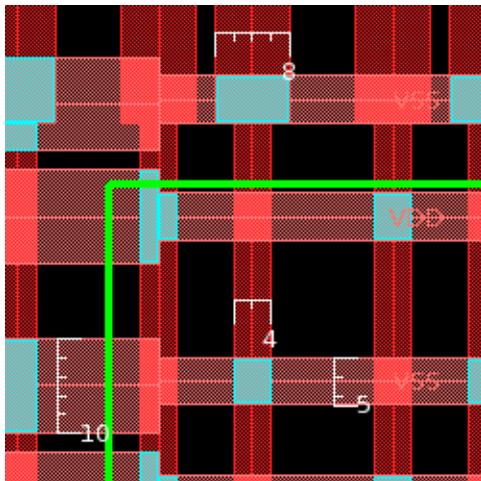
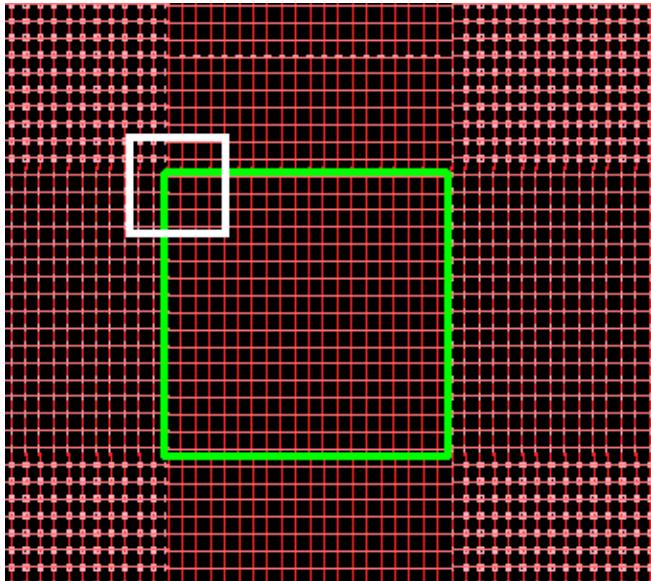


Figure 94 Tapering Straps Zoomed Out View



The power plan begins with three `create_pg_wire_pattern` commands to create three wire pattern definitions. These commands define the pattern layer, direction, width, and spacing in a particular region of the routing boundary. The `first_wire` wire pattern defines a pattern from the left and bottom of the routing boundary as specified by the `-extend_low` boundary option, and extends to the point specified by the `-high_end_reference_point @pt1` argument. The `center_wire` wire pattern extends from the `-low_end_reference_point @pt1` and ends at the `-high_end_reference_point @pt2` points. The `last_wire` pattern starts from the point specified by `-low_end_reference_point @pt2` and extends to the `-extend_high` boundary, which is the top or right edge of the routing boundary.

The two `create_pg_composite_pattern` commands are used to define two different relationships between the patterns. The `create_pg_composite_pattern tapering_base` command associates the previous wire patterns with the left, center, and right areas of the routing boundary, and the bottom, middle, and top areas respectively. The second command, `create_pg_composite_pattern tapering_straps`, specifies the actual values for the PG route layer, direction, side width, center width, side spacing, and center spacing using the previous `create_pg_composite_pattern tapering_base` definition to create horizontal and vertical PG routes.

Note that the `tapering_straps` composite pattern uses another composite pattern called `tapering_base`. Composite pattern can be defined hierarchically to create more complex patterns for a demanding design requirement.

The `set_pg_strategy` command passes the actual x- and y-locations where the tapered straps should begin and end, together with the signal names for the PG straps. These values are specified by using Tcl variables.

The commands used to implement this power plan are as follows.

```

create_pg_wire_pattern first_wire -layer @l -direction @d \
 -width @w -spacing @s -extend_low_boundary \
 -high_end_reference_point @pt1 -parameters {l d w s pt1}

create_pg_wire_pattern center_wire -layer @l -direction @d \
 -width @w -spacing @s -low_end_reference_point @pt1 \
 -high_end_reference_point @pt2 -parameters {l d w s pt1 pt2}

create_pg_wire_pattern last_wire -layer @l -direction @d \
 -width @w -spacing @s -low_end_reference_point @pt2 \
 -extend_high_boundary -parameters {l d w s pt2}

create_pg_composite_pattern tapering_base \
 -parameters {layer dir side_w center_w side_s center_s pt1 pt2} \
 -add_patterns { \
 {{pattern: first_wire} \
 {parameters: {@layer @dir @side_w @side_s @pt1}}} \
 {{pattern: center_wire} \
 {parameters: {@layer @dir @center_w @center_s @pt1 @pt2}}} \
 {{pattern: last_wire} \
 {parameters: {@layer @dir @side_w @side_s @pt2}}} \
 }
}

create_pg_composite_pattern tapering_straps \
 -parameters {h_pt1 h_pt2 v_pt1 v_pt2} \
 -add_patterns { \
 {{pattern: tapering_base} \
 {parameters: {M7 horizontal 10 5 2 7.5 @h_pt1 @h_pt2}} \
 {pitch: {0 30}}} \
 {{pattern: tapering_base} \
 {parameters: {M6 vertical 8 4 2 6 @v_pt1 @v_pt2}} \
 {pitch: {25 0}}} \
 }
}

set x_start 500
set x_end 1000
set y_start 500
set y_end 1000
set c_offset 5
set_pg_strategy core_patterns -core \
 -pattern [list {name: tapering_straps} \
 {nets: VDD VSS} \
 {parameters: $x_start $x_end $y_start $y_end} \
 [list offset: [list 5 $c_offset]]]

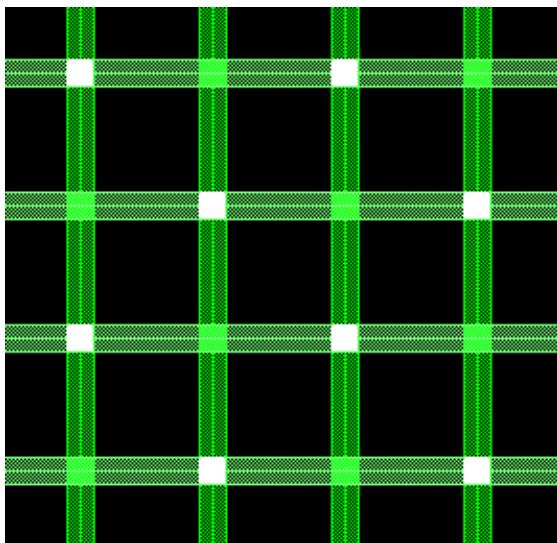
compile_pg -strategies core_patterns

```

## Creating a Checkerboard Via Pattern

The following example creates horizontal power straps on layer M5 and vertical power straps on layer M4 for net VDD. Vias are inserted at alternating locations and form a checkerboard pattern. The `create_pg_wire_pattern` command defines the base pattern, and the `create_pg_composite_pattern` command defines the parameters for the horizontal and vertical PG routes.

*Figure 95      Checkerboard Vias*



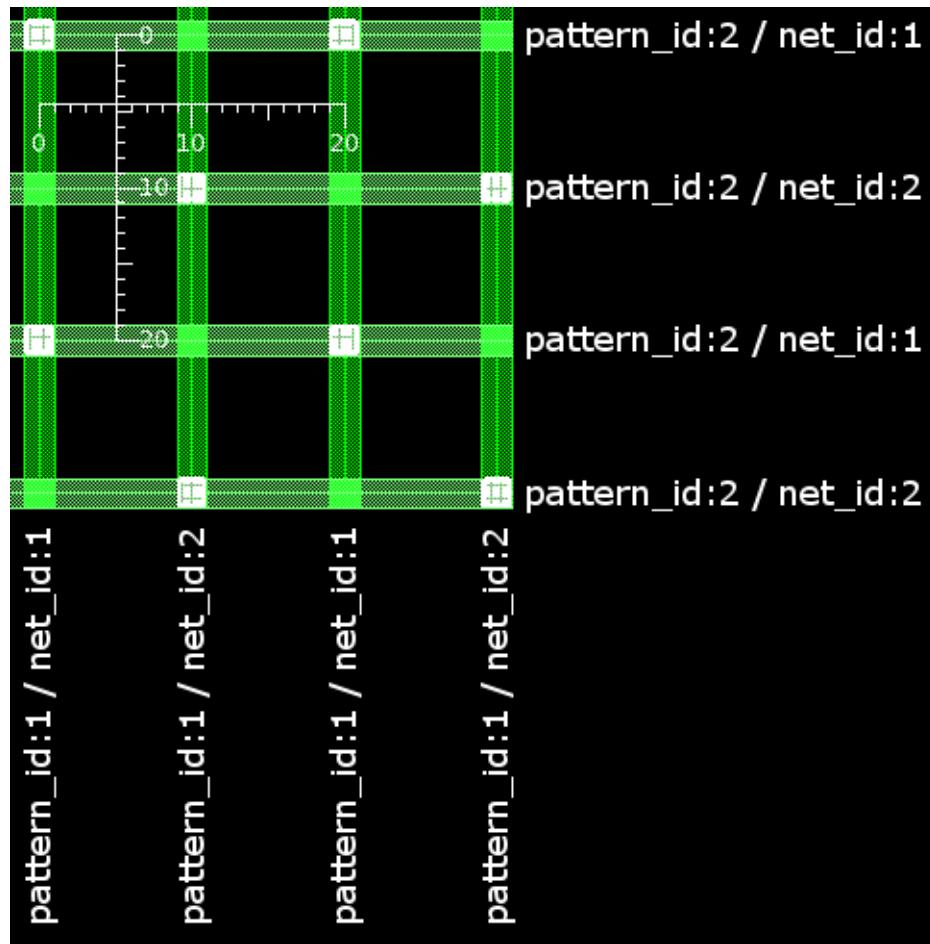
The single `create_pg_wire_pattern` command creates a wire pattern named `wire_base`. The `create_pg_composite_pattern` command instantiates the `wire_base` wire pattern two times; one time for the vertical straps on layer M4 and one time for the horizontal straps on layer M5. Each pattern defines a pair of nets, {VDD VDD}. The pitch between the pairs of nets is 20 microns; the two nets are evenly distributed between the pitch due to the `-spacing interleaving` option of the `create_pg_wire_pattern` command. Note that interleaving is passed as a parameter by the `{parameters: {M4 vertical 2 20 interleaving}}` statement in the `create_pg_composite_pattern` command.

The `-via_rule` option specifies which via is placed at each intersection of layers M4 and M5. The following figure is annotated with the `pattern_id` and `net_id` numbering for this power plan. In this example, `{pattern_id: 1}` refers to the first pattern in the `-add_patterns` option: the vertical power straps on layer M4. `{pattern_id: 2}` refers to the second pattern: the horizontal power straps on layer M5. `{pattern_id: 1}` `{net_id: 1}` refers to the first net following the `nets:` keyword listed for the first pattern specification.

Based on this specification, vias are placed at the intersections of `{pattern_id: 1}` `{net_id: 1}` and `{pattern_id: 2}` `{net_id: 1}`. Vias are also placed at the

intersections of {pattern\_id: 1} {net\_id: 2} and {pattern\_id: 2} {net\_id: 2}. At other locations, {via\_master: NIL} specifies that no via is placed at those intersections. The {intersection: undefined} {via\_master: NIL} statement specifies that no other vias should be inserted anywhere in the power plan.

Figure 96 Annotated Checkerboard Pattern



The commands to create this power plan are as follows:

```
create_pg_wire_pattern wire_base -layer @1 -direction @d \
 -width @w -spacing @s -pitch @p -parameters {l d w p s}

create_pg_composite_pattern checker_board_vias -nets {VDD} \
 -add_patterns {
 {{pattern: wire_base} {nets: VDD VDD} \
 {parameters: {M4 vertical 2 20 interleaving}} \
 {offset: 4}} \
 {{pattern: wire_base} {nets: VDD VDD} \
 {parameters: {M5 horizontal 2 20 interleaving}} \
 {offset: 4}}}
```

```

 {offset: 4}}) \
-via_rule { \
 {{pattern_id: 1} {net_id: 1}} \
 {{pattern_id: 2} {net_id: 1}} {via_master: default} \
 {{pattern_id: 1} {net_id: 2}} \
 {{pattern_id: 2} {net_id: 2}} {via_master: default} \
 {{pattern_id: 1} {net_id: 2}} \
 {{pattern_id: 2} {net_id: 1}} {via_master: NIL} \
 {{pattern_id: 1} {net_id: 1}} \
 {{pattern_id: 2} {net_id: 2}} {via_master: NIL} \
 {intersection: undefined} {via_master: NIL}}}

set_pg_strategy core_patterns -core \
 -pattern {{name: checker_board_vias} {nets: VDD VSS} \
 {offset: {5 5}}}

compile_pg -strategies core_patterns

```

## Defining Power Plan Regions

In the pattern-based power network routing flow, power plan regions define areas of the design used to create the power rings and power straps to form the power network. To create a power plan region, use the `create_pg_region` command as shown in the following example.

```
icc2_shell> create_pg_region region1 \
 -exclude_macros {macro1 macro2} -core -expand -35 \
 -macro_offset 35
{region1}
```

Alternatively, use the Task Assistant to define the power plan region.

1. Select Task > Task Assistant in the GUI.
2. Select PG Planning > PG Region in the Task Assistant.
3. Enter the region name, region area, expansion parameters, and other information in the Task Assistant.
4. Click Preview to display the Tcl command you specified.
5. Click Apply to create the power plan region.

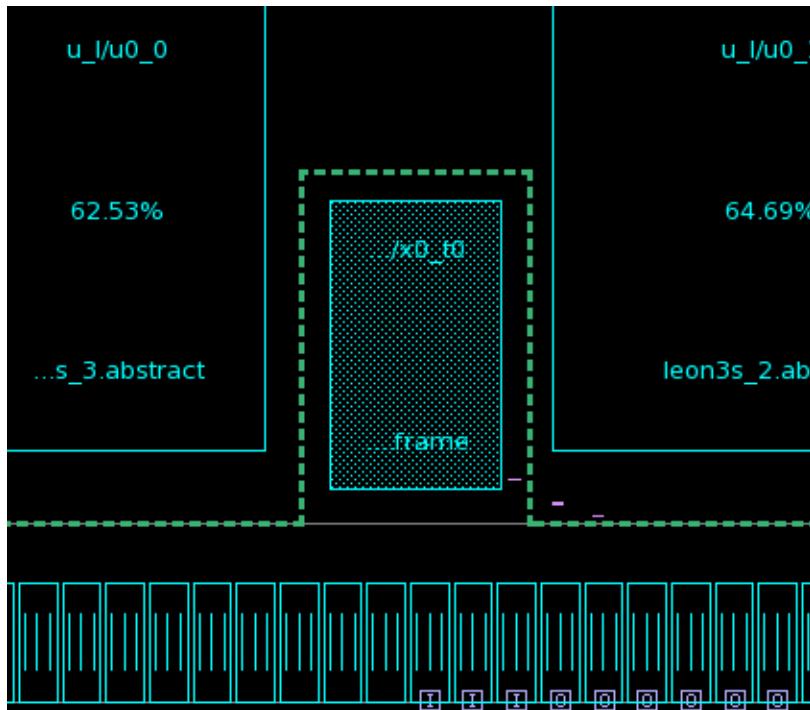
To create a power plan region that is bounded by a polygon, specify the `-polygon` option followed by the coordinates of the polygon. The following example creates a power plan region named `region_polygon` bounded by the coordinates (2000,2000), (4000,2000), (4000,4000), and (2000,4000).

```
icc2_shell> create_pg_region region_polygon \
 -polygon {{2000 2000} {4000 2000} {4000 4000} {2000 4000}}
{region_polygon}
```

To create a power plan region that excludes specified macros, use the `-exclude_macros` option and specify the list of macros to exclude. You can create an offset around the macro by including the `-macro_offset` option. In the following example, the tool creates an indentation in the region to exclude the specified macro.

```
icc2_shell> create_pg_region region_exclude -core \
 -exclude_macros {u_1/u_m/ahbram0/aram_1/x0_t0} -macro_offset 50
{region_exclude}
```

Figure 97 Region Excluding a Macro



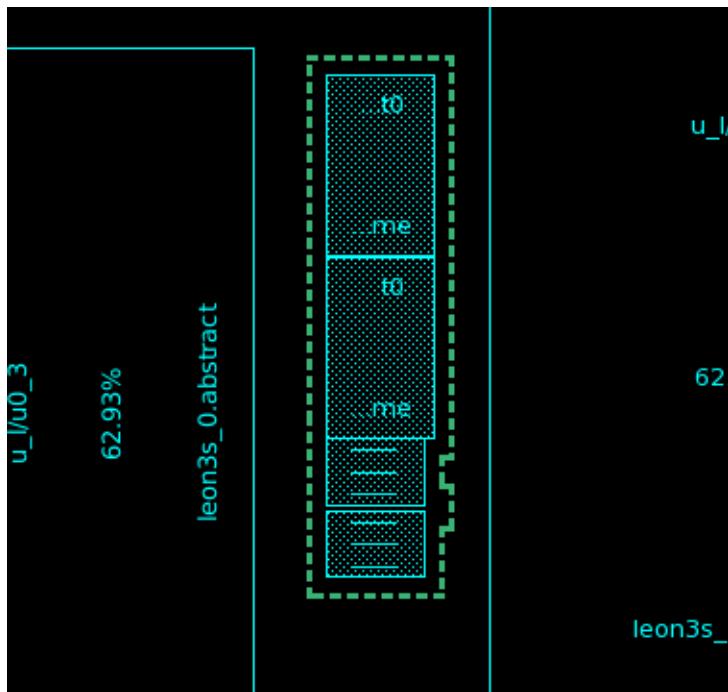
To create a PG region that surrounds macros in the design, use the `-group_of_macros` option. You can create an offset by including the `-expand` option. In the following example, the tool creates a region around the specified macros and expands the region by 50 microns.

```
icc2_shell> set macros {u_1/u_m/ahbram0/aram_0/x0_t0
 u_1/u_m/dsu0/x0/mem0_ram0_0_x0_x0_t0
 u_1/u_m/ahbram0/aram_2/x0_t0
 u_1/u_m/dsu0/x0/mem0_ram0_0_x1_x0_t0}

icc2_shell> create_pg_region region_surround -group_of_macros $macros \
```

```
-expand 50
{region_surround}
```

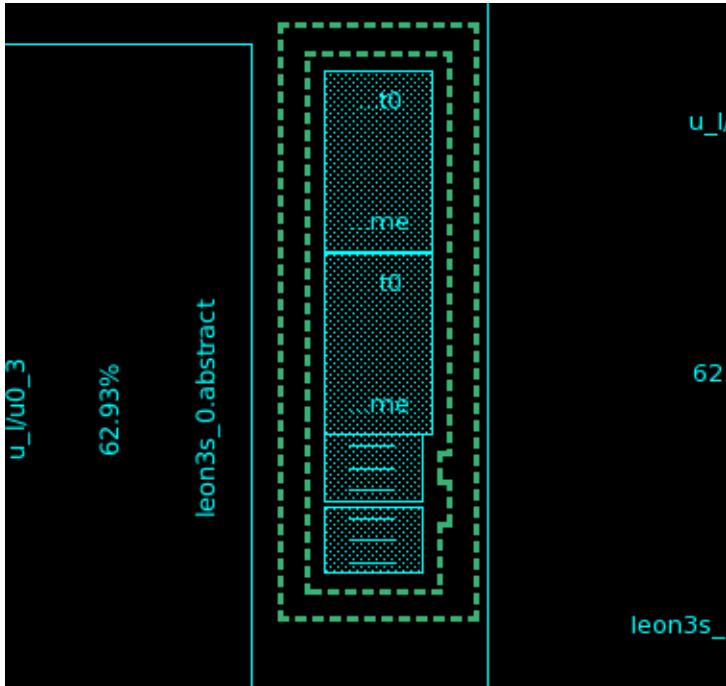
Figure 98 Region Surrounding a Group of Macros



If you run the `create_pg_region` command and specify an existing region name, the command deletes the existing region, reuses the region name, and creates the new region definition. You can also create a copy of an existing region and make modifications to the region copy by using the `-update` option. The next example copies the `region_surround` power plan region from the previous example, expands the region, and removes the jogs.

```
icc2_shell> create_pg_region region_copy -update region_surround \
 -expand 75 -remove_jog {expand: 50}
```

*Figure 99 Copy Region and Expand*



To list all the power plan regions in the design, use the `report_pg_regions` command. To list only a single region, specify the region name after the command. The following example reports the currently defined power plan regions in the design.

```
icc2_shell> report_pg_regions
Region: region_surround
Points: {{2202.360 3092.270} {2202.360 3575.890} ... }
Region: region_copy
Points: {{2127.360 3017.270} {2127.360 4647.760} ... }
```

To delete a power plan region, use the `remove_pg_regions` command followed by the region name. To remove all regions, use the `remove_pg_regions -all` command. The following example removes the power plan region named `region_copy`.

```
icc2_shell> remove_pg_regions region_copy
PG region region_copy is removed.
```

## Setting the Power Plan Strategy

After defining the ring, mesh, macro connection, and standard cell rail patterns for your design, you can associate the patterns with the power plan regions or other areas of the design by using the `set_pg_strategy` command. The command also defines the pattern offset, how the patterns extend past the boundaries of the power plan region or area, and routing blockages where the pattern is not created.

To define the power plan strategy, use the `set_pg_strategy` command as in the following example.

```
icc2_shell> set_pg_strategy ring_strat -core \
 -pattern {{name: ring_pattern} {nets: {VDD VSS}} \
 {offset: {3 3}} {parameters: {M7 10 2 M8 10 2 true}}} \
 -extension {{stop: design_boundary}}
```

Alternatively, use the Task Assistant to define the standard cell rail pattern.

1. Select Task > Task Assistant in the GUI.
2. Select PG Planning > Strategy in the Task Assistant.
3. Enter the strategy name, PG nets, pattern name, routing area, and other settings in the Task Assistant.
4. Click Apply to apply the strategy.

In the previous example, the `set_pg_strategy` command defined a strategy that associated the `ring_pattern` pattern with the `ring_strat` strategy. The following example defines a strategy that associates the `mesh_pattern` pattern for the power mesh defined in [Creating Power and Ground Mesh Patterns](#) with the `mesh_strat` strategy. The `-extension` option extends the power mesh to the outermost ring.

```
icc2_shell> set_pg_strategy mesh_strat -core \
 -extension {{stop: outermost_ring}} \
 -pattern {{pattern: mesh_pattern} {nets: {VDD VSS}}}
```

The following example defines a strategy that associates the `macro_pattern` macro connection pattern defined in [Creating Power and Ground Macro Connections](#) with the `macro_strat` strategy.

```
icc2_shell> set_pg_strategy macro_strat -core \
 -pattern {{pattern: hm_pattern} {nets: {VDD VSS}}}
```

The following example defines a strategy that associates the `rail_pattern` standard cell connection pattern defined in [Creating Power and Ground Standard Cell Rails](#) with the `rail_strat` strategy.

```
icc2_shell> set_pg_strategy rail_strat -core \
 -pattern {{pattern: std_cell_rail} {nets: VDD VSS}}
```

The following sections summarize the specification keywords and their function for the `-pattern`, `-blockage` and `-extension` options.

## Strategy Settings for the -pattern Option

Arguments to the `-pattern` option define how the specified pattern is created in the design. The following list describes the available arguments. Note that not all arguments are valid for all pattern types.

- The `{name: pattern_name}` argument assigns the pattern name to use with this strategy. The name is the pattern name you created with the `create_pg_*_pattern pattern_name` command.
- The `{nets: {net1 net2 ...}}` argument specifies the power and ground net names for the strategy. You can also define placeholder names in the `create_pg_*_pattern` command and set the actual net names using this argument.

To specify a null or empty slot for a net, replace the net name with the dash (-) character. For example, `{nets: {VDD - VSS -}}`.

- The `{offset: {x_offset y_offset}}` argument specifies the x- and y-offset to apply when creating the first pattern. This argument is not used for scattered pin macro connection patterns or standard cell rail patterns.
- The `{offset_start: boundary | {x y}}` argument specifies the starting point for the pattern offset. This argument is not used for scattered pin macro connection patterns or standard cell rail patterns.
- The `{parameters: value1 value2 ...}` argument specifies the ordered list of values for the parameters defined with the `-parameters` option of the `create_pg_*_pattern` command.
- The `{skip_sides: {side1 side2 ...}}` argument defines the sides to omit when creating the power ring. This argument is valid only for ring patterns.
- The `{side_offset: {side: number} {offset: offset}}` argument defines the offset for each specified side in the ring pattern. This argument is valid only for ring patterns.

## Strategy Settings for the -blockage Option

Arguments to the `-blockage` option define areas where the pattern should not be created. A blockage is defined with three arguments: `nets`, `layers`, and the area specification (`voltage_areas`, `macros`, `polygon`, `blocks`, or `pg_regions`). You can provide multiple blockage specifications within a single `set_pg_strategy` command. The following list

describes the available arguments. Note that not all arguments are valid for all pattern types.

- The `{layers: {layer1 layer2 ...}}` argument specifies the layers to avoid routing over the blockage.
- The `{nets: {net1 net2 ...}}` argument specifies the power and ground net names to avoid routing over the blockage. This list of nets is a subset of the nets specified by the `-pattern` option.
- The area is defined by the `pg_regions`, `blocks`, `macros`, `pg_regions`, `polygon`, or `voltage_areas` argument, followed by a list of elements or points.

## Strategy Settings for the -extension Option

Arguments to the `-extension` option define how the power mesh extends beyond the specified power plan region or design area. The extension specification enables a connection between the power structure and other power structures in the design. An extension is defined with the `direction`, `layers`, `nets`, `side`, and `stop` arguments. You can define multiple extensions in the same `set_pg_strategy` command. The following list describes the available arguments. Note that some arguments are invalid for some commands.

- The `{direction: {L R T B}}` argument limits the direction to extend the power mesh to only the specified directions.
- The `{layers: {layer1 layer2 ...}}` argument limits the extension to only the specified layers.
- The `{nets: {net1 net2 ...}}` argument specifies the power and ground net names to avoid routing over the blockage. This list of nets is a subset of the nets specified by the `-pattern` option.
- The `{side: {side_a side_b ...}}` argument limits power ring or macro connection extension to only the specified sides.
- The `{stop: first_target | innermost_ring | outermost_ring | pad_ring | design_boundary | design_boundary_and_generate_pin | distance_in_microns}` argument defines the stopping point for the extension.

## Creating Via Rules Between Different Strategies

A complex power plan might require that you connect combinations of several different power rings, meshes, power rails, and other structures. Use the `set_pg_strategy_via_rule` command to set the via insertion rules when inserting vias between different power plan strategies.

```
icc2_shell> set_pg_strategy_via_rule via_rule1 \
-via_rule {
 {{strategies: strat1} {layers: M2}}
 {{strategies: strat2} {layers: M3}}
 {via_master: VIA23_FAT}
 {{intersection: undefined} {via_master: nil}}
}
```

The preceding example creates a via rule that inserts the VIA23\_FAT via between shapes on layer M2 defined by strategy strat1 and shapes on layer M3 defined by strategy strat2. New vias are omitted between other metal layer intersections.

Alternatively, use the Task Assistant to define the via rule.

1. Select Task > Task Assistant in the GUI.
2. Select PG Planning > Create PG in the Task Assistant.
3. Click the Via rule tab.
4. Enter the rule name.
5. Select Advanced, click Define and define the details of the via rule.
6. Click Apply to create the via strategy.

## Instantiating the Power Plan

The `create_pg_*_pattern` and `set_pg_strategy` commands define the power plan, but do not add power straps or rails to the design. To instantiate the power plan, use the `compile_pg` command with the `-strategies` option and specify the strategy name as shown in the following example:

```
icc2_shell> compile_pg -strategies ring_strat
...
Successfully compiled PG.
1
```

The command instantiates the power plan defined by the specified strategy and checks for any DRC violations created by the power plan. In a typical flow, you create the ring, mesh, rail, or macro pattern, associate the pattern with a power plan strategy by using the `set_pg_strategy` command, instantiate the pattern with the `compile_pg` command, and then repeat the flow for the next pattern type in the power plan.

If you specified the power plan incorrectly, or if the power plan contains an error, you can remove the power plan with the `compile_pg -undo` command as shown in the following example. The `compile_pg -undo` command removes only the power and ground network created by the most recent `compile_pg` command.

```
icc2_shell> compile_pg -undo
```

To ignore DRC violations while creating the power network, include the `-ignore_drc` and `-ignore_via_drc` options. The `compile_pg -ignore_drc -ignore_via_drc` command creates the power network and reports DRC violations created by the power plan, but does not remove power straps or vias that create DRC violations.

## Handling Design Data During Design Planning Flow

The absence of data or inconsistent data in the design prevents the design flow from proceeding further in the flow. The Early Data Check Manager allows you to check designs for such issues early in the design cycle.

The general flow is as follows:

1. Use the `set_early_data_check_policy` command to define the violation handling policy for data checks.
2. Proceed with your tool flow. The tool detects and responds to violations throughout the flow.
3. Use the `report_early_data_checks` command to obtain a report about all violations or specific data checks.
4. Use the `get_early_data_check_records` command to get a Tcl collection of violations that can be used in other commands.
5. Use the `write_early_data_check_config` command to save the settings in a file for future use.
6. Use the `remove_early_data_check_records` command to clear the data in preparation for another iteration.

For more information on these commands and Early Data Check Manager, see the respective man pages and the “Handling Design Data Using the Early Data Check Manager” topic in the *IC Compiler II Data Model User Guide*.

For design planning, you can set policies and strategy configurations for the predefined checks to allow, tolerate, or repair data.

*Table 8      Design Planning Checks, Policies, and Strategies*

| Check                                  | Supported policies                         | Description                                                                                         |
|----------------------------------------|--------------------------------------------|-----------------------------------------------------------------------------------------------------|
| <code>plan.pg.strategy_via_rule</code> | <code>error</code> , <code>tolerate</code> | Checks whether the strategy via rule specified in the <code>compile_pg</code> command exists or not |

**Table 8 Design Planning Checks, Policies, and Strategies (Continued)**

| Check                                                    | Supported policies      | Description                                                             |
|----------------------------------------------------------|-------------------------|-------------------------------------------------------------------------|
| plan.floorplan.<br>missing_layer_<br>routing_direction   | error, repair           | Checks whether the routing directions of metal layers are missing       |
| plan.floorplan.core_offset<br>_conflict_enclosure_rules  | error, repair, tolerate | Checks whether the floorplan core offset conflicts with enclosure rules |
| plan.floorplan.core_area_l<br>ength_conflict_width_rules | error, repair, tolerate | Checks whether the floorplan core area conflicts with width rules       |
| plan.floorplan.<br>missing_block_<br>boundary            | error, repair           | Checks whether the block boundary is missing                            |

In the following examples, the reports show what check was performed, what action was taken, and what object is causing the issue.

- Check – The check that is performed
- Policy
  - tolerate – The tool makes predictable and explainable assumptions to mitigate the violation. No changes are made to the design or setup.
  - error – The tool does not mitigate the violation, but issues an error message.
- Strategy – Specifies the strategy to apply for a check.
- Checked Objects – Objects causing the issue.

#### *Example 16 report\_early\_data\_checks With Tolerate Policy*

```
report_early_data_checks -verbose

Report : report_early_data_checks
Design : abc
Version: T-2022.03
Date : Wed Mar 23 11:22:12 2022

Check Policy Strategy
 Checked Objects Comment

plan.pg.strategy_via_rule tolerate skip_via_rule_check
 abc.nlib:abc_nwcopt.design
```

-----  
-----  
1

*Example 17 report\_early\_data\_checks With Error Policy*

```
report_early_data_checks -verbose

Report : report_early_data_checks
Design : abc
Version: T-2022.03
Date : Wed Mar 23 11:22:12 2022

Check Policy Strategy
 Checked Objects Comment

plan.pg.strategy_via_rule error
 stop_compile_pg_execution abc.nlib:abc_nwcopt.design

1
```

---

## Improving compile\_pg Runtime in Large Designs

For large designs, the `compile_pg` command might take a long time to insert all the vias required by the power mesh. You can use the following application options to change how DRC rules are applied around vias and apply partitioning to reduce runtime.

- `plan.pgroute.high_capacity_mode`: Set this application option to `true` to enable high-capacity mode for the `compile_pg` and `create_pg_vias` PG route commands. High-capacity mode reduces the memory required by the multithreading PG router when performing DRC checking and fixing on the vias.
- `plan.pgroute.drc_check_fast_mode`: Set this application option to `true` to enable a low-effort or fast-mode DRC checking to improve runtime. Specifically, the DRC checking engine skips the top and bottom layers of stacked vias and checks only the layers between them.

---

## Application Options for PG Routing

The PG routing commands supports additional application options to control power and ground object creation and checking. Unless otherwise noted, the following descriptions describe the tool behavior when the application option is `true`.

**Table 9 PG Routing Application Options**

| To do this                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Use this application option                                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| <p>Specifies how PG routing commands fix rectangle only rule violation when via is created.</p> <p>When set to <code>true</code>, which is the default, the tool discards the via. When <code>false</code>, the tool tries to patch the DRC violation.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <code>plan.pgroute.no_patch_fix_rectangle_only_rule</code>   |
| <p>Specify whether the <code>create_pg_vias</code> command should consider routing blockage at the wire's intersection - the wire's intersection is trimmed by routing blockage.</p> <p>When <code>true</code>, which is the default, the command considers routing blockage, which results in more accurate intersection for via creation. When <code>false</code>, the command does not create any via where there is routing blockage overlapping with intersection region.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | <code>plan.pgroute.trim_by_blockage</code>                   |
| <p>Specify whether to automatically color wires and cuts internally before DRC checking and fixing.</p> <p>You can set this application option to either one of the following:</p> <ul style="list-style-type: none"> <li><code>metal</code> - Color PG wires based on track color<br/>The color is based on the <code>plan.pgroute.always_align_track_color_layer</code>s application option setting.<br/>When the wire width is the default width and the center of the wire aligns with the track, the PG wire color follows the track color. Non-default width wires have opposite color of the track color.</li> <li><code>cut</code> - Color PG wires using mask one at first and PG DRC fixing engine recolors as necessary to fix any DRC violations. If there is violation, the tool tries mask two. If none of the masks are DRC clean, the tool continues with regular PG DRC fixing.</li> <li><code>none</code> - Do not colour metal or cut anything before DRC checking. This is the default.</li> </ul> | <code>plan.pgroute.color_metal_cut_internally_for_drc</code> |
| <p>Control whether wire color follows the track color in a specified metal layer when the <code>plan.pgroute.color_metal_cut_internally_for_drc</code> application option is set to <code>metal</code>.</p> <p>By default, all wires follow the track color on all layers.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <code>plan.pgroute.always_align_track_color_layers</code>    |

**Table 9 PG Routing Application Options (Continued)**

| To do this                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Use this application option                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|
| <p>Assign color mode to uncolored shapes during design rule checking. This approach helps you to relax or tighten the spacing requirements during checking. The valid values are -1, 0, and 1.</p> <p>The default is -1, where the tool ignores any uncolored shapes or treats them as a certain color during checking.</p> <p>When set to 0, the design rule checker treats an uncolored shape as a different color from neighboring shapes.</p> <p>When set to 1, the design rule checker treats an uncolored shape as the same color as neighboring shapes.</p> <p>For more information, see the application option man page.</p> | <code>plan.pgroute.assign_drc_color_mode</code>               |
| <p>Invoke the <code>connect_pg_net</code> command to update the PG net-to-pin logical connectivity. By default, the <code>connect_pg_net</code> command is not called.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                           | <code>plan.pgroute.auto_connect_pg_net</code>                 |
| <p>Create vias between new PG shapes and existing shapes with a <code>shape_type</code> attribute of <code>user_route</code>. By default, <code>user_route</code> shapes are ignored.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                            | <code>plan.pgroute.connect_user_route_shapes</code>           |
| <p>Determine the rail width based on the PG pins of routed cells in the routing area when running the <code>compile_pg</code> command. The routing area is specified by the PG strategy. By default, the tool determines the rail width from cells in a design independent of cell location.</p>                                                                                                                                                                                                                                                                                                                                     | <code>plan.pgroute.decide_rail_width_from_routing_area</code> |
| <p>Prevent the <code>compile_pg</code> command from removing floating wires and vias. Use this application option in designs with multiple levels of physical hierarchy to ensure connectivity between physical hierarchies. By default, floating wires and vias are removed.</p>                                                                                                                                                                                                                                                                                                                                                    | <code>plan.pgroute.disable_floating_removal</code>            |
| <p>Ignore DRC repair for stapling vias and do not create the via. Use this application option to improve runtime at the expense of additional DRC errors. By default, the command tries to repair the DRC violation.</p>                                                                                                                                                                                                                                                                                                                                                                                                             | <code>plan.pgroute.disable_stapling_via_fixing</code>         |
| <p>Prevent the <code>compile_pg</code> from reevaluating Tcl variables used in the PG strategy. By default, the <code>compile_pg</code> command reevaluates all Tcl variables defined in PG strategy and returns updated values for the variables.</p>                                                                                                                                                                                                                                                                                                                                                                               | <code>plan.pgroute.disable_strategy_evaluation</code>         |

**Table 9 PG Routing Application Options (Continued)**

| To do this                                                                                                                                                                                                                                                                                                                                                         | Use this application option                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| Prevent the <code>compile_pg</code> command from trimming dangling wires. Use this application option in designs with multiple levels of physical hierarchy to ensure connectivity between physical hierarchies. By default, dangling wires are trimmed.                                                                                                           | <code>plan.pgroute.disable_trimming</code>               |
| Prevent the <code>compile_pg</code> command from creating vias. By default, the <code>compile_pg</code> command creates vias.                                                                                                                                                                                                                                      | <code>plan.pgroute.disable_via_creation</code>           |
| Avoid DRC repair on stacked vias and removes the stacked via when any via in the stack is not DRC-clean. Use this application option to decrease runtime at the expense of fewer stacked vias. By default, the tool performs DRC repair for stacked vias which are not DRC-clean.                                                                                  | <code>plan.pgroute.discard_stackvia_with_drc</code>      |
| Enable the PG router commands to try all contact codes specified by the <code>set_pg_via_master_rule</code> command. Use this application option to allow the tool to try additional via masters to create a DRC-clean power plan at the expense of longer runtime. By default, the tool selects a via master and uses that via for all subsequent via insertions. | <code>plan.pgroute.fix_via_drc_multiple_viadef</code>    |
| Specify a restricted set of PG routing layers to use when connecting to hard macro power pins.                                                                                                                                                                                                                                                                     | <code>plan.pgroute.hmpin_connection_target_layers</code> |
| Honor routing blockage inside cover cell.                                                                                                                                                                                                                                                                                                                          | <code>plan.pgroute.honor_cover_cell</code>               |
| Honor net-based nondefault routing rule spacing rules when checking DRCs on PG structures. By default, nondefault routing rule spacing rules are not honored.                                                                                                                                                                                                      | <code>plan.pgroute.honor_ndr_spacing_drc</code>          |
| Consider existing signal routes when performing a DRC check. By default, the signal routes are ignored for DRC check.                                                                                                                                                                                                                                              | <code>plan.pgroute.honor_signal_route_drc</code>         |
| Consider shapes within standard cells when performing DRC checks. By default, shapes within standard cells are ignored.                                                                                                                                                                                                                                            | <code>plan.pgroute.honor_std_cell_drc</code>             |
| Load the frame view of macro, block, and standard cell instances when performing DRC checking. By default, the frame views are not loaded.                                                                                                                                                                                                                         | <code>plan.pgroute.load_frame_view</code>                |

**Table 9 PG Routing Application Options (Continued)**

| To do this                                                                                                                                                                                                                                                                                                                                                                                               | Use this application option                                 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| Define the maximum distance to consider when extending a PG strap. If the distance to the target extension object is greater than the specified value, the extension is dropped. By default, the tool creates extensions to the specified objects regardless of distance.                                                                                                                                | plan.pgroute.max_extension_distance                         |
| Specify the maximum number of undo steps for the <code>compile_pg</code> command. By default, the maximum number of undo steps is five.                                                                                                                                                                                                                                                                  | plan.pgroute.max_undo_steps                                 |
| Specify the type of via master or ContactCode to consider when inserting a via. Valid values for this application option are: <code>all</code> , <code>default</code> , <code>Vs</code> , <code>Vh</code> , or <code>Vv</code> . See the man page for more information.                                                                                                                                  | plan.pgroute.maximize_total_cut_area                        |
| Create a separate alignment strap between neighboring cells when the distance between the cells is greater than the specified value. This application option is valid when using the <code>compile_pg</code> command on a design with a power switch or physical cell alignment pattern.                                                                                                                 | plan.pgroute.maximum_cell_gap_for_alignment_strap           |
| Merge metal shapes in specified pad cell types while performing DRC checks. Use this application option to avoid DRC errors when the internal metal shapes in pad cells are rectilinear polygons rather than rectangles. Valid values are one or more of <code>io_pad</code> , <code>corner_pad</code> , and <code>flip_chip_pad</code> . By default, the tool does not merge metal shapes in pad cells. | plan.pgroute.merge_shapes_in_pad_cell                       |
| When set to <code>true</code> , the <code>create_pg_vias</code> command merges overlapping shapes for via connection. By default, the tool does not merge the overlapping shapes.                                                                                                                                                                                                                        | plan.pgroute.merge_shapes_for_via_creation                  |
| Allow the <code>compile_pg</code> command to create wires that partially overlap the routing area boundary. By default, overlapping wires are not created.                                                                                                                                                                                                                                               | plan.pgroute.overlap_route_boundary                         |
| When set to <code>true</code> , the <code>compile_pg</code> command creates wires that partially overlap the routing area boundary and that excludes the design boundary. By default, overlapping wires are not created.                                                                                                                                                                                 | plan.pgroute.overlap_route_boundary_exclude_design_boundary |
| Include standard cell rail and PG strap intersections for snapping. By default, intersected standard cell rails are not considered for snapping.                                                                                                                                                                                                                                                         | plan.pgroute.snap_stdcell_rail                              |

**Table 9 PG Routing Application Options (Continued)**

| To do this                                                                                                                                                                                                                                                                                                                                                                                                                     | Use this application option                               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| Ignore DRC checking between neighboring stapling vias stacks to improve runtime. Use this application option to improve runtime at the expense of possible DRC errors. By default, the tool checks neighboring stacked vias.                                                                                                                                                                                                   | <code>plan.pgroute.stapling_via_fast_mode</code>          |
| Specify a library cell reference name to use when calculating width and offset of standard cell rails. By default, the tool selects the standard cell reference.                                                                                                                                                                                                                                                               | <code>plan.pgroute.std_rail_from_refname</code>           |
| Specify a cell type (lib_cell, filler, well_tap, and end_cap) on which to create vias to cell pins. Use this application option to create rail-to-cell connections for cells specified by <code>create_pg_macro_conn_pattern</code> command. By default, vias are not created on pins of any nonmacro cells.                                                                                                                   | <code>plan.pgroute.treat_cell_type_as_macro</code>        |
| When set to <code>true</code> , which is the default, honor routing blockages as real metal or cut shapes when performing DRC checks for blockages in physical blocks, macros, or standard cells. When set to <code>false</code> , only overlap and minimum spacing rules per layer are checked for routing blockages.                                                                                                         | <code>plan.pgroute.treat_fat_blockage_as_fat_metal</code> |
| Enable the <code>create_pg_vias</code> command to create vias between PG rails and pins of fixed standard cells. In addition, use this application option to create vias on PG pins of cells specified by <code>create_pg_macro_conn_pattern</code> command. By default, vias are not created on pins of any standard cells.                                                                                                   | <code>plan.pgroute.treat_fixed_stdcell_as_macro</code>    |
| Write out additional debugging information messages to the console.                                                                                                                                                                                                                                                                                                                                                            | <code>plan.pgroute.verbose</code>                         |
| Maximize the number of cuts in the via array based on the specified value. Valid values are <code>force_both_enclosures_in_intersection</code> , <code>try_both_enclosures_in_intersection</code> , <code>try_one_enclosure_in_intersection</code> , <code>try_cuts_in_intersection</code> , and <code>enforce_via_rule_dimensions</code> . By default, the tool fits the entire via array bounding box into the intersection. | <code>plan.pgroute.via_array_size_control</code>          |

**Table 9 PG Routing Application Options (Continued)**

| To do this                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Use this application option                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| Enable the PG routing commands to create vias between partially overlapped PG straps or between partially overlapped PG wires and macro pins. This application option accepts an overlap ratio between 0 and 1, where a value of 1 requires that shapes must be fully overlapped.                                                                                                                                                                                                                                                                                                                             | plan.pgroute.via_site_threshold                |
| Specify the default tag in all PG routing commands even when the <code>-tag</code> option is not used in PG routing commands. For example, when the <code>-tag</code> option is not specified in the <code>create_pg_vias</code> command, all vias created have a tag value of <code>myTag</code> . If the <code>compile_pg</code> PG routing command specifies a tag as <code>secondTag</code> , then the default tag value is overwritten with <code>secondTag</code> and the PG shapes and vias generated using the <code>compile_pg</code> PG routing command has a tag value of <code>secondTag</code> . | plan.pgroute.default_tag                       |
| Enable the <code>create_pg_vias</code> command to optimize the track usage. It does not indicate to reduce track usage, but it generates the wire closer to the same direction wires in different layers. This makes the tracks closer in different layers and makes future routing easier..                                                                                                                                                                                                                                                                                                                  | plan.pgroute.optimize_track_alignment          |
| Enable the <code>create_pg_special_pattern</code> command to control the behavior of the <code>-insert_power_switch_alignment_straps</code> and <code>-insert_physical_cell_alignment_straps</code> options. The <code>number</code> keyword in the two options indicates the number of straps that connect to each pin. With this option set to <code>true</code> , all pins are treated as one target. That is, it indicates the number of straps that go through the bounding box of all pins.                                                                                                             | plan.pgroute.treat_multiple_pins_as_one_target |
| Enable <code>derive_pg_mask_constraint</code> to use new algorithm to derive color of wire and via enclosure.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | plan.pgroute.derive_pg_mask_engine_version     |
| Enable the tool to derive net information for cut without net information.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | plan.pgroute.derive_cut_net_from_pin           |
| Enable the creation of vias on 45-degree RDL shapes. When this option is set to true, the PG router uses rectangle shapes as input and calculates the intersection area with 45-degree RDL shapes. If an intersection area is a valid rectangle, the via is created.                                                                                                                                                                                                                                                                                                                                          | plan.pgroute.enable_rdl_45_shapes              |

## Reducing the Memory Size of the Design Database

To reduce the memory of the design database that results in smaller peak memory when loading the design database after PG insertion, you can use the following application options:

- `plan.pgroute.use_via_matrix`: Set this application option to `true` to enable the `compile_pg` and `create_pg_vias` PG route commands to commit `via_matrix` objects rather than individual vias to the database. This approach results in significantly fewer database objects, thereby reducing the memory of the resulting database.

When this application option is set to `true`, the PG routing commands analyze and create via matrixes from individual vias that match a regular pattern. The result is a number of via matrices and left-over individual vias, which are then committed to the database.

Note that this via matrix inference by the PG route commands might come at a slight runtime increase and that the peak memory of the PG routing commands is not reduced by this option.

- `plan.pgroute.use_shape_pattern`: Set this application option to `true` to enable the `compile_pg` and `create_pg_strap` PG route commands to commit `shape_pattern` objects rather than individual shapes to the database. This approach results in significantly fewer database objects, thereby reducing the memory of the resulting database.

When this application option is set to `true`, the PG routing commands analyze and create shape patterns from individual shapes that match a regular pattern. The result is a number of `shape_pattern` objects and left-over individual shapes, which are then committed to the database.

Note that this `shape_pattern` object inference by the PG route commands might come at a slight runtime increase and that the peak memory of the PG routing commands is not reduced by this option.

## Pattern-Based Power Network Routing Example

The following example is a complete pattern-based power network routing script.

```
Create the power and ground nets and connections
create_net -power VDD
create_net -ground VSS
connect_pg_net -net VDD [get_pins -physical_context *VDD]
connect_pg_net -net VSS [get_pins -physical_context *VSS]

Create the power and ground ring pattern
create_pg_ring_pattern ring_pattern -horizontal_layer @hlayer \
```

Chapter 10: Performing Power Planning  
 Pattern-Based Power Network Routing Example

```

-horizontal_width {@hwidth} -horizontal_spacing {@hspace} \
-vertical_layer @vlayer -vertical_width {@vwidth} \
-vertical_spacing {@vspace} -corner_bridge @cbridge \
-parameters {hlayer hwidth hspace
 vlayer vwidth vspace cbridge}

Set the ring strategy to apply the ring_pattern
pattern to the core area and set the width
and spacing parameters
set_pg_strategy ring_strat -core \
 -pattern {{name: ring_pattern} {nets: {VDD VSS}}
 {offset: {3 3}} {parameters: {M7 10 2 M8 10 2 true}}} \
 -extension {{stop: design_boundary}}

Create the ring in the design
compile_pg -strategies ring_strat

Define a new via rule, VIA78_3x3, for the power mesh
set_pg_via_master_rule VIA78_3x3 -contact_code VIA78 \
 -via_array_dimension {3 3}

Create the power and ground ring mesh pattern
create_pg_mesh_pattern mesh_pattern -layers {
 {{vertical_layer: M8} {width: 5}
 {spacing: interleaving} {pitch: 32}}
 {{vertical_layer: M6} {width: 2}
 {spacing: interleaving} {pitch: 32}}
 {{horizontal_layer: M7} {width: 5}
 {spacing: interleaving} {pitch: 28.8}}} \
-via_rule {
 {{layers: M6} {layers: M7} {via_master: default}}
 {{layers: M8} {layers: M7} {via_master: VIA78_3x3}}}

Set the mesh strategy to apply the mesh_pattern
pattern to the core area. Extend the mesh
to the outermost ring
set_pg_strategy mesh_strat -core -pattern {{pattern: mesh_pattern}
 {nets: {VDD VSS}}} \
 -extension {{stop: outermost_ring}}

Create the mesh in the design
compile_pg -strategies mesh_strat

Create the power and ground ring mesh pattern
create_pg_mesh_pattern mesh_pattern -layers {
 {{vertical_layer: M8} {width: @width8}
 {spacing: interleaving} {pitch: 32}}
 {{vertical_layer: M6} {width: @width6}
 {spacing: interleaving} {pitch: 32}}
 {{horizontal_layer: M7} {width: @width7}
 {spacing: interleaving} {pitch: 28.8}}} \
-via_rule {
 {{layers: M6} {layers: M7} {via_master: default}}}
```

```

{{layers: M8} {layers: M7} {via_master: VIA78_3x3}}} \
-parameters {width6 width7 width8}

Set the mesh strategy to apply the mesh_pattern
pattern to the core area. Extend the mesh
to the outermost ring
set_pg_strategy mesh_strat -core -pattern {
 {pattern: mesh_pattern} {nets: {VDD VSS}}
 {parameters: {32 28.8 32}}} \
 -extension {{stop: outermost_ring}}

Create the mesh pattern in the design
compile_pg -strategies mesh_strat

Create the connection pattern for macro
power and ground pins
create_pg_macro_conn_pattern macro_pattern \
 -pin_conn_type scattered_pin

Set the macro connection strategy to
apply the macro_pattern pattern to
the core area
set_pg_strategy macro_strat -core \
 -pattern {{pattern: macro_pattern}
 {nets: {VDD VSS}}}

Connect the power and ground macro pins
compile_pg -strategies macro_strat

Create a new 1x2 via
set_pg_via_master_rule via16_1x2 -via_array_dimension {1 2}

Create the power and ground rail pattern
create_pg_std_cell_conn_pattern rail_pattern -layers {M1}

Set the power and ground rail strategy
to apply the rail_pattern pattern to the
core area
set_pg_strategy rail_strat -core \
 -pattern {{pattern: rail_pattern} {nets: VDD VSS}}

Define a via strategy to insert via16_1x2 vias
between existing straps and the new power rails
specified by rail_strat strategy on the M6 layer
set_pg_strategy_via_rule rail_rule -via_rule {
 {{existing: strap} {layers: M6}}
 {strategies: rail_strat} {via_master: via16_1x2}}
 {{intersection: undefined} {via_master: nil}}}

Insert the new rails
compile_pg -strategies rail_strat -via_rule rail_rule

```

## PG Script Generator

Creating a pattern-based PG network can be time-consuming due to the complex Tcl syntax. To simplify the process, you can convert the configuration file that contains the pattern specifications in PG syntax to a Tcl script that includes all the commands you need to create the PG structure.

To generate the PG Tcl script,

1. Open or create the configuration file that contains your pattern-based power network specifications.

If you do not have a configuration file, use the following command to create a default configuration file that includes the syntax for all the functionality available in pattern-based PG creation, including power strap and via configurations:

```
icc2_shell> generate_pg_script -template
```

2. Modify the keyword syntax and values in the configuration file to meet your power network requirements.
3. Run the `generate_pg_script` command, specifying the configuration file as your input file:

```
icc2_shell> generate_pg_script -input dp_pgConfig.txt \
 -output pg_script.tcl
```

If you do not specify the `-output` option, the tool creates a script file named `ppns_script.tcl`. You can edit and reuse the Tcl script as needed.

4. Source the Tcl script in the IC Compiler II tool:

```
icc2_shell> source pg_script.tcl
```

The keywords in the configuration file correspond to the keywords in the Tcl syntax. See the command man pages for detailed usage.

The following example shows an excerpt of the PG syntax in a configuration file:

```
StrapConfig: PG_TOP_MESH1 {
 # required
 target_area: core
 category: mesh
 nets: "VDD VSS"
 # optional
 tag: M2_M3_MESH
 blockage: macros:{ u_macro/mem1 u_macro/mem2}
 extension: {{stop: 10}{layers: M1} }

 layer: M2 {
 # required
 }
}
```

```

 widths: 0.4
 direction: horizontal
 spacings: interleaving
 pitch: 7
 # optional
 offset: 0
 alignTrack: track
 trim: false
 }
 layer: M3 {
 # required
 widths: 0.8
 direction: vertical
 spacings: 1.2
 pitch: 8
 # optional
 offset: 0
 alignTrack: track
 trim: false
 }
}

```

The following example shows the corresponding Tcl commands written out by the `generate_pg_script` command:

```

set_pg_strategy_via_rule NO_VIA \
 -via_rule { {{intersection: undefined}{via_master: NIL}} }

create_pg_mesh_pattern PG_TOP_MESH1_PTRN \
 -layers { \
 {{horizontal_layer:M2}{width:0.4 }} \
 {spacing:interleaving}{pitch:7} \
 {track_alignment: track}{trim: false}{offset:0} } \
 {{vertical_layer:M3}{width:0.8 }} \
 {spacing:1.2}{pitch:8}{track_alignment: track} \
 {trim: false}{offset:0} } \
 }

 -via_rule {{intersection : all}{via_master:NIL} }

set_pg_strategy PG_TOP_MESH1_STR \
 -extension {{stop: 10}{layers: M1}} \
 -blockage {macros:{ u_macro/mem1 u_macro/mem2}} \
 -pattern { {name: PG_TOP_MESH1_PTRN}{nets: VDD VSS}} \
 -core

compile_pg -strategies PG_TOP_MESH1_STR -via_rule NO_VIA -tag M2_M3_MESH

```

## Manually Creating Power Straps and Vias

For most designs, pattern-based power planning is the most effective methodology for generating a power mesh. Alternatively, to create individual power straps or a power mesh for a specific region of the design, use the `create_pg_strap` and `create_pg_vias` commands. These commands create components of the power mesh and do not require a pattern or strategy as part of the pattern-based power planning methodology.

### PG Straps

The `create_pg_strap` command creates power ground straps on specified layers in a specified region of the design. The command also inserts vias between the new strap shapes and existing shapes. The following `create_pg_strap` command creates a power strap with these characteristics:

- The strap is created for net VDD on the ME3 layer and arranged vertically.
- The center of the strap is placed at x=100 with a width of 4.
- The strap extends from y=100 to y=400.
- To improve runtime, no DRC is performed when the tool creates the strap.

```
icc2_shell> create_pg_strap -layer ME3 -direction vertical \
 -width 4 -net VDD -drc no_check -start 100.0 \
 -low_end 50 -high_end 400
```

### PG Vias

The `create_pg_vias` command inserts vias at metal intersections on adjacent layers. The following examples show different applications of the `create_pg_vias` command:

- Create vias at all intersections.

```
icc2_shell> create_pg_vias -nets VDD
Committed 288 vias.
```

- Insert vias on the VDD net to connect layers M3 to M4, set the `shape_use` attribute for the via as `stripe`, and do not perform a DRC when inserting the via.

```
icc2_shell> create_pg_vias -nets {VDD} -from_layers M3 \
 -to_layers M4 -drc no_check -mark_as stripe
Committed 288 vias.
```

- Insert vias within the specified bounding box for layers from M6 to M8 and connect macro pins to the PG straps.

```
icc2_shell> create_pg_vias -nets VDD -drc no_check \
 -within_bbox [get_attribute [current_block] bbox] \
```

```
-from_layers M6 -to_layers M8 -from_types stripe \
-to_types macro_pin
```

- Create a via matrix for VDD and VSS net shapes with a `shape_use` attribute set of stripe. Use the VIA34f via master and set the `tag` attribute on the via matrix to `pg_via_matrix`.

```
icc2_shell> set bbox [get_attribute [get_core_area] bbox]
icc2_shell> create_pg_vias -nets {VDD VSS} -from_types stripe \
 -to_types stripe -via_masters {VIA34f} -within_bbox $bbox \
 -start {{741 735} {708 720}} -pitch {{66 30} {66 30}} \
 -create_via_matrix -tag pg_via_matrix
```

## Using Secondary PG Constraints

For designs with power domains that have the primary supplies being shutdown, but still have logic that needs to operate in some power states, you can consider having dual-rail cells in your design to meet this requirement. For optimal performance, these cells should be placed near the secondary power straps.

By default, the tool assumes that the secondary PG straps are available in all regions of a voltage area. However, to save routing resources, you might choose to have secondary PG straps available in only a subset of the voltage area.

Secondary PG constraints help the tool to identify the location of straps that have limited physical availability in the voltage area. The tool honors these constraints throughout the flow.

For more information, see the “Specifying Secondary PG Constraints” topic in the *IC Compiler II Multivoltage User Guide*.

## Hierarchical Secondary PG Aware Placement Constraints

The hierarchical design planning flow is used to accommodate limited physical availability of secondary PG straps with less effort on the design. By using secondary PG-aware placement constraints, certain secondary supplies can be planned to use less routing resources.

Specify secondary PG placement constraints at both the individual block levels and the top levels. However,

- PG constraints cannot be created at top and pushed down to the blocks.
- PG constraints spanning top/block boundary must be applied separately at the specific levels.

The general flow is as follows in which the secondary PG placement constraints can be specified as part of this flow,

1. Use the `split_constraints` and `commit_block` commands to create the subblocks.
2. Create voltage area shapes by using the `shape_blocks` command.
3. Create PG straps at the top level, then use the `characterize_block_pg` command to create the PG strategies for the subblocks.
4. Create power network as follows:
  - a. Read the UPF power intent by using the `load_upf` command.
  - b. Create the PG straps at the block level by using the `compile_pg` command.

**Note:**

After power planning and secondary PG placement constraints are committed at the blocks and at the top level, then a second placement is necessary. This places the dual-rail cells based on the secondary PG strap locations.

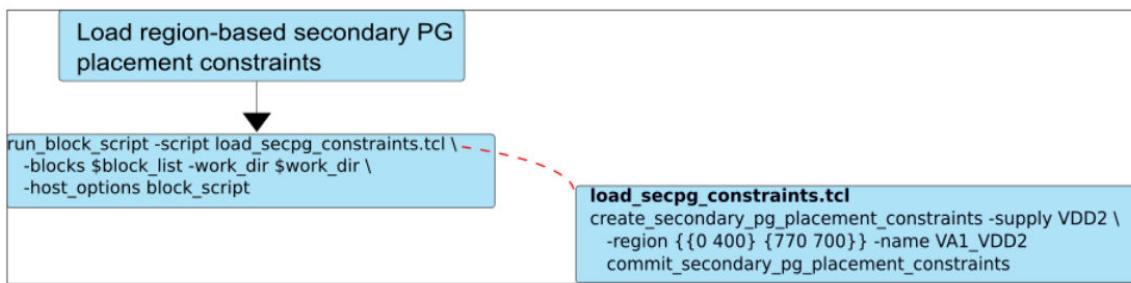
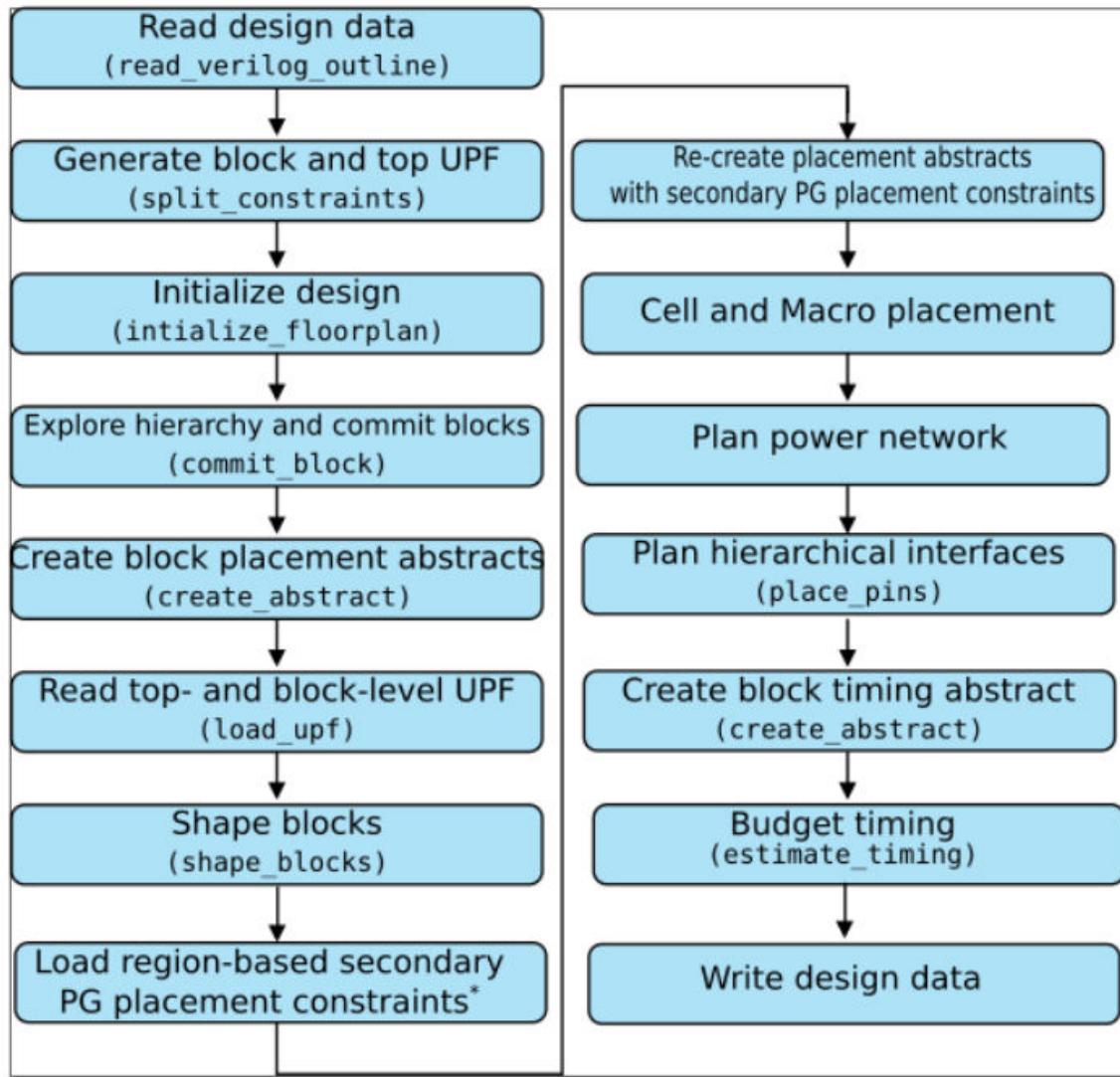
5. Create a new corner, `estimated_corner`, and optimize the top-level interface logic by using the `estimate_timing` command.

When you commit the constraints at the top level of the design, use the `commit_secondary_pg_placement_constraints -commit_subblocks` command. The `-commit_subblocks` option checks all block-level secondary PG constraints and commits any uncommitted constraints. The command commits the secondary PG placement constraints at both top and linked subblocks.

To display the associated block for each constraint, use the `report_secondary_pg_placement_constraints -all_blocks` command. To check for consistency and error conditions in specific blocks, use the `check_secondary_pg_placement_constraints -blocks` command with a list of blocks.

To define secondary PG constraints manually, use the `create_secondary_pg_placement_constraints` command and to define secondary PG constraints automatically, use the `derive_secondary_pg_placement_constraints` command.

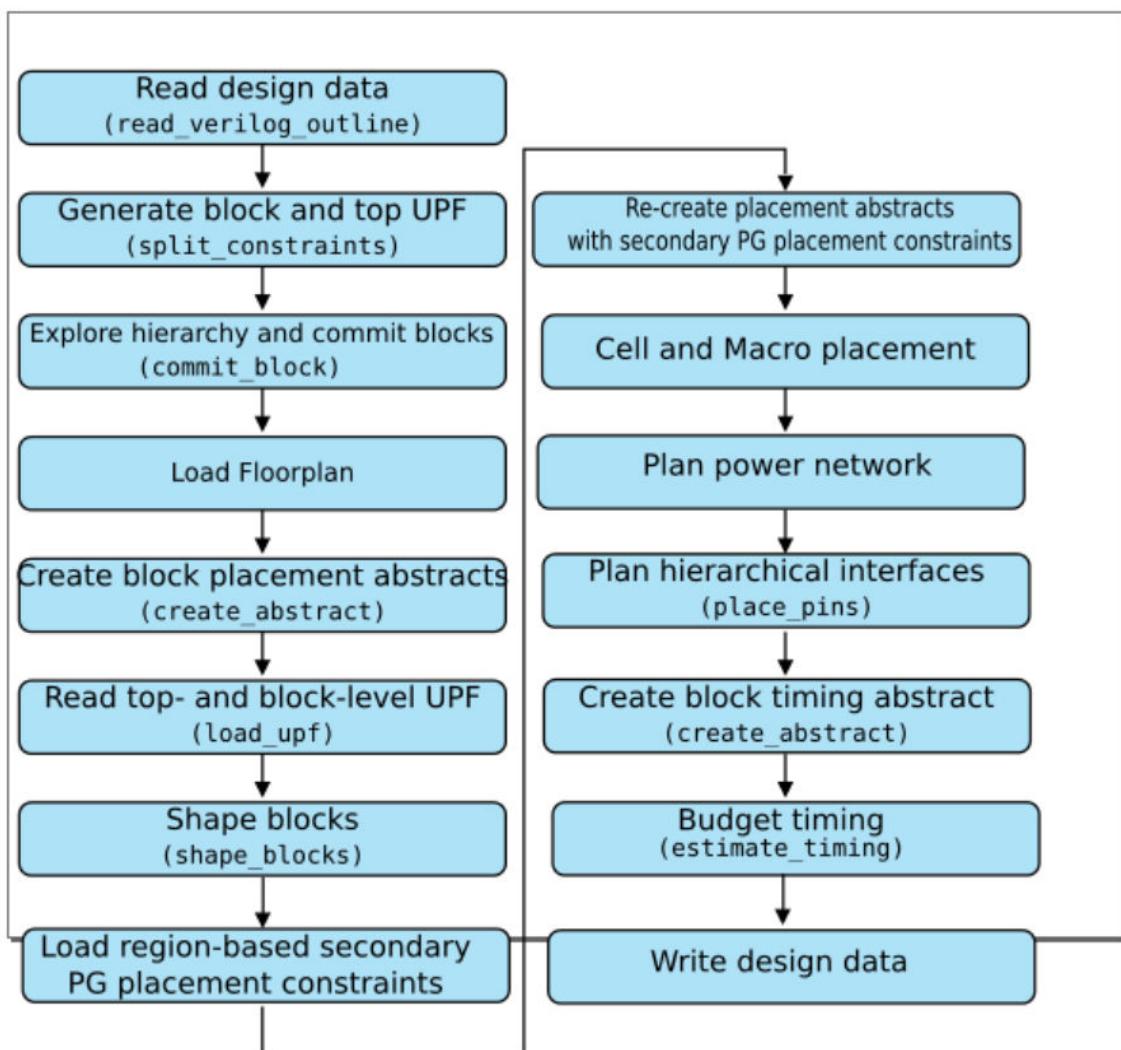
The flow to implement a design plan iteration after the initial design netlist is generated in the tool topographical mode is shown in the following figure:



**Note:**

The `load_block_constraints` command that loads constraints for child and top blocks is not supported by the tool. However, you can use the `run_block_script` command to run the specified Tcl script on a set of blocks. In the following example, the `load_secpg_constraints.tcl` script runs the `create_secondary_pg_placement_constraints` command, which then creates the relevant constraints.

The flow to implement a design plan iteration after the partial floorplan is generated in the tool topographical mode is shown in the following figure:



## Support for Abstract View

To copy the pin information into the design view, use the following commands:

*Table 10 Support for Abstract View*

| To do this                                                                                                        | Use this                                                                      |
|-------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| To copy the secondary PG constraints between design view and abstract view                                        | Use the <code>create_abstract</code> or <code>merge_abstract</code> commands. |
| To render the secondary PG constraints stale after the UPF data is updated by the <code>commit_upf</code> command | Use the <code>change_view</code> and <code>change_abstract</code> commands    |

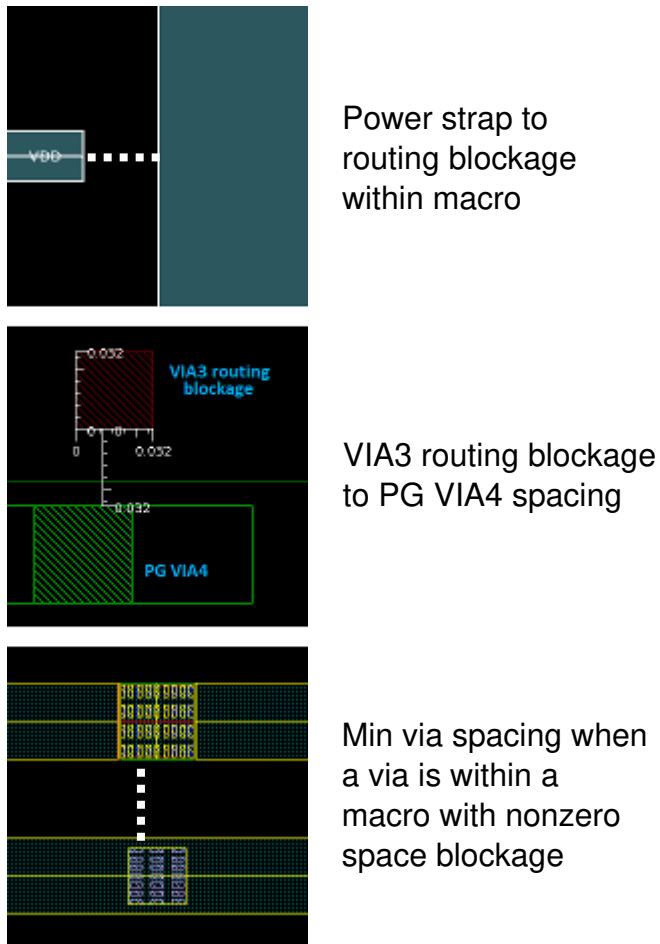
---

## Honoring Metal and Via Spacing Rules to Blockages

When routing near blockages, you can guide the tool to honor fat metal spacing rules or via cut rules for routes and vias. [Figure 100](#) shows some layout topologies where this might be necessary. Set the `plan.pgroute.treat_fat_blockage_as_fat_metal` application option to `true` as follows to honor routing blockages as real metal or cut shapes when performing DRC checks for blockages in physical blocks, macros, or standard cells. By default, this application option is `false` and the tool only checks overlaps and minimum spacing rules per layer for routing blockages.

```
icc2_shell> set_app_options \
 -name plan.pgroute.treat_fat_blockage_as_fat_metal -value true
```

*Figure 100 Minimum Spacing Violations to Blockages*



## Using PG Pattern Shapes

PG pattern shapes are compact array representations of individual PG wire shapes. PG pattern shapes provide an efficient method for instantiating large, regular patterns of PG wire shapes in a design, reducing the size of the database. The commands in the following table perform operations on PG pattern shapes as described in the following topics.

| Command                               | Description                                                         |
|---------------------------------------|---------------------------------------------------------------------|
| <code>create_pg_pattern_shapes</code> | Creates a pattern of PG wire shapes in a single, repeatable object. |
| <code>get_shape_patterns</code>       | Returns a collection of PG pattern shapes.                          |

| Command                          | Description                                                                     |
|----------------------------------|---------------------------------------------------------------------------------|
| report_shape_patterns            | Reports information about the specified PG pattern shapes in the current block. |
| remove_shape_patterns            | Removes one or more PG pattern shapes from the current block.                   |
| convert_shape_patterns_to_shapes | Converts PG pattern shapes to PG wire shapes.                                   |
| convert_shapes_to_shape_pattern  | Converts PG wire shapes to a PG pattern shape.                                  |
| decompose_shape_patterns         | Converts PG pattern shapes to more efficient PG pattern shapes.                 |

## Creating PG Pattern Shapes

To create a compact array representation of PG wire shapes in a single object that you can reuse throughout the block, use the `create_pg_pattern_shapes` command. The following example creates a VSS PG pattern shape on the M1 layer. Each wire shape is repeated every 0.57 um in the x-direction and every 0.48 um in the y-direction.

```
icc2_shell> create_pg_pattern_shapes -layer M1 -net VSS -width 0.037 \
 -direction vertical -start 4.788 -low_end {-0.06} -high_end {0.06} \
 -xPitch 0.57 -yPitch 0.48
```

## Retrieving PG Pattern Shapes

To retrieve all PG pattern shapes in the current block or a specific PG pattern shape, use the `get_shape_patterns` command. The following example retrieves all the PG pattern shapes connected to the VDD net.

```
icc2_shell> get_shape_patterns * -of_objects [get_nets VDD]
{PATH_PATTERN_31_0 PATH_PATTERN_31_1}
```

## Reporting PG Pattern Shapes

To report information about one or more PG pattern shapes, use the `report_shape_patterns` command. The following example reports the PG pattern shapes for the VDD net.

```
icc2_shell> report_shape_patterns [get_shape_patterns \
 -of_objects [get_nets VDD]]
Report : report_shape_patterns
Design : blk
...

Shape_pattern Type Dimensions GridSize Displacements

RECT_PATTERN_31_0 rect_pattern {5x4} 1 {250:north}x{150:east}
```

## Removing PG Pattern Shapes

To remove one or more PG pattern shapes, use the `remove_shape_patterns` command. Include the `-verbose` option to report additional information. The following example removes the PG pattern shapes specified by a pattern list.

```
icc2_shell> remove_shape_patterns POLYGON_PATTERN_28_* -verbose
Removed shape_pattern POLYGON_PATTERN_28_173
Removed shape_pattern POLYGON_PATTERN_28_174
```

## Converting PG Pattern Shapes to PG Wire Shapes

To convert PG pattern shapes to individual PG wire shapes, use the `convert_shape_patterns_to_shapes` command. The following example converts a PG pattern shape named `RECT_PATTERN_31_0` to four PG wire shapes.

```
icc2_shell> convert_shape_patterns_to_shapes \
 [get_shape_patterns RECT_PATTERN_31_0] \
{RECT_31_1829 RECT_31_1830 RECT_31_1831 RECT_31_1832}
```

## Converting PG Wire Shapes to a PG Pattern Shape

To convert PG wire shapes to a new PG pattern shape, use the `convert_shapes_to_shape_pattern` command. The following example converts PG wire shapes to a new PG pattern shape.

```
icc2_shell> convert_shapes_to_shape_pattern [get_shapes \
 -filter {shape_type==rect && layer_name==M1}]
{RECT_PATTERN_31_0}
```

To automatically detect shapes that can be converted to PG pattern shapes, use the `-layers`, `-region`, or `-shapes` option with the `convert_shapes_to_shape_pattern` command instead of listing the shapes with the `shape_list` argument. When you enable this feature, the tool creates PG pattern shapes based on the layers, regions, or shapes you specify.

## Optimizing PG Pattern Shapes

To optimize PG pattern shapes to more efficient PG pattern shapes, use the `decompose_shape_patterns` command. If the specified PG pattern shapes are suboptimal, the tool replaces them with more efficient, nearby PG pattern shapes.

```
icc2_shell> decompose_shape_patterns [get_shape_patterns]
{RECT_PATTERN_0 RECT_PATTERN_1}
```

## Using Via Matrixes

Via matrixes are compact array representations of individual via definitions. Via matrixes provide an efficient method for instantiating large, regular patterns of vias in a design,

resulting in a reduction in database size and faster processing by the tool. The commands in the following table perform operations on via matrixes as described in the following topics.

| Command                      | Description                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------|
| create_via_matrix            | Creates a via matrix with a specific via definition, pitch, size, and other attributes. |
| get_via_matrixes             | Retrieves all via matrixes in the current block or retrieves a specific via matrix.     |
| report_via_matrixes          | Reports basic information for one or more via matrixes.                                 |
| edit_via_matrix              | Adds or subtracts vias from a specific region in an existing via matrix.                |
| convert_vias_to_via_matrix   | Converts a collection of vias into a single via matrix.                                 |
| convert_via_matrixes_to_vias | Converts a via matrix to individual vias.                                               |
| decompose_via_matrixes       | Converts existing via matrixes to more efficient via matrixes.                          |

## Inserting Via Matrixes

Use the `create_via_matrix` command to create a via matrix with a specific via definition, pitch, size, and other attributes. The following example creates a via matrix of size 20 by 30 by using the VIA56f via definition. The via matrix starts at (475, 475), uses a pitch of 100 microns, and connects to the VSS net.

```
icc2_shell> create_via_matrix -via_def VIA56f -pitch {100 100} \
 -size {20 30} -origin {475 475} -net VSS
```

Some power planning commands, such as `create_pg_vias`, support options to create via matrixes instead of individual vias. The following `create_pg_vias` command uses the `-create_via_matrix` option to create via matrixes for VSS net intersections:

```
icc2_shell> create_pg_vias -nets VSS -pitch {{100 100}} \
 -within_bbox {{0 0} {1800 1800}} -via_masters VIA56f \
 -create_via_matrix
```

## Retrieving Via Matrixes

Use the `get_via_matrixes` command to retrieve all via matrixes in the current block or retrieve a specific via matrix. The following example retrieves via matrixes associated with the VSS net.

```
icc2_shell> get_via_matrixes -of_objects [get_nets VSS]
{VIA_MATRIX_0}
```

Use the `get_attribute` command with the `get_via_matrixes` command to return properties of the via matrix.

```
icc2_shell> get_attribute [get_via_matrixes \
 -of_objects [get_nets VSS]] number_of_columns
7
```

### Reporting Via Matrixes

Use the `report_via_matrixes` command to report basic information for one or more via matrixes. Include the `-verbose` option to report additional information.

```
icc2_shell> report_via_matrixes
Via_matrix Via_def Origin

VIA_MATRIX_0 VIA56f {476 476}
VIA_MATRIX_1 VIA56f {576 576}
1

icc2_shell> report_via_matrixes -verbose
Via_matrix Via_def Origin Description

VIA_MATRIX_0 VIA56f {476 476} Orientation: R0
 Base Pitch: NA
 Base Size: {1 1}
 Pitch: {100 100}
 ...
VIA_MATRIX_1 VIA56f {576 576} Orientation: R0
 Base Pitch: NA
 Base Size: {1 1}
 Pitch: {100 100}
 ...
```

### Adding and Removing Vias in a Via Matrix

The `create_via_matrix` command creates via matrixes with a rectangular shape. In some cases based on design requirements, matrix points within the rectangle should not receive a via, or the matrix should be modified to add a missing via. Use the `edit_via_matrix` command to add or subtract vias from a specific region in an existing via matrix.

The following example creates an “empty” via matrix by setting the initial pattern to 0 with the `-pattern` option. Next, the `edit_via_matrix` command is used to add vias in the region (900, 900) to (1100, 1100). The `get_attribute` command displays the updated pattern after running the `edit_via_matrix` command. Note that a “1” in the pattern attribute represents a via at that location, while “0” represents a location with no via.

```
icc2_shell> create_via_matrix -via_def VIA56f -pitch {200 200} \
 -size {9 9} -origin {475.565 475.565} -pattern "0" -net VSS
{VIA_MATRIX_0}
```

```
icc2_shell> get_attribute [get_via_matrixes {VIA_MATRIX_0}] pattern
0
icc2_shell> edit_via_matrix \
 -add {{900 900} {1100 1100}} [get_via_matrixes]
1
icc2_shell> get_attribute [get_via_matrixes {VIA_MATRIX_0}] pattern
000000000 000000000 000000000 000100000 ...
```

### Converting Vias to Via Matrixes

To convert a collection of vias into a single via matrix, use the `convert_vias_to_via_matrix` command. Note that the command does not create a via matrix if the vias within a collection have a different number of rows or columns.

```
icc2_shell> convert_vias_to_via_matrix [get_vias -of_objects [get_nets
 VDD]]
{VIA_MATRIX_0}
```

To automatically detect vias that can be converted to efficient via matrixes, use the `-via_defs`, `-region`, or `-vias` option with the `convert_vias_to_via_matrix` command instead of listing the vias. When you enable this feature, the tool creates via matrixes based on the via definitions, regions, or vias you specify. The command can convert vias into multiple via matrixes with different dimensions.

### Converting Via Matrixes into Vias

In some cases you might need to convert a via matrix to individual vias to manage specific via locations. To convert one or more via matrixes into individual vias, use the `convert_via_matrixes_to_vias` command.

```
icc2_shell> convert_via_matrixes_to_vias \
 [get_via_matrixes {VIA_MATRIX_0 VIA_MATRIX_1}]
{VIA_SA_0 VIA_SA_1 ...}
```

### Optimizing Via Matrixes

To convert existing via matrixes to more efficient via matrixes, use the `decompose_via_matrixes` command. If the specified via matrixes are suboptimal, the tool replaces them with more efficient, nearby via matrixes.

```
icc2_shell> decompose_via_matrixes [get_via_matrixes]
{VIA_MATRIX_0 VIA_MATRIX_1}
```

## Debugging Vias and PG Straps Removed Due to DRC Violations

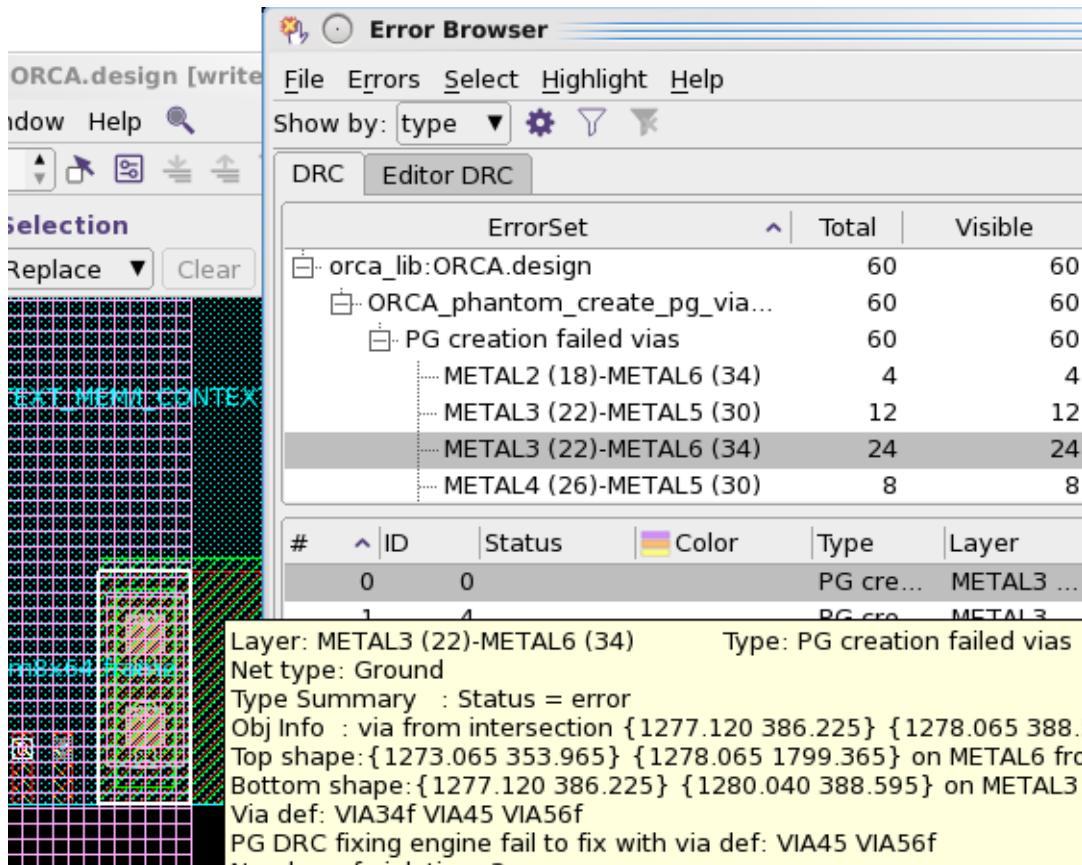
In some cases, the `compile_pg`, `create_pg_strap`, and `create_pg_vias` commands might skip via insertion on shape intersections when the via creates a DRC violation. These vias that are removed by the tool are known as “phantom vias” and require additional analysis to determine why the via was not inserted. To assist in debugging

the missing vias, the `compile_pg`, `create_pg_strap`, and `create_pg_vias` commands support the `-show_phantom` option to save information about the missing via or PG strap and make that information available to the Error Browser. When this option is specified, the tool writes out an EMS error database that contains location and other information about the missing via or missing PG strap. Use the Error Browser in the GUI to review the errors and diagnose the problem. Note that phantom vias are not reported when the `-ignore_drc` or `-ignore_via_drc` option is specified.

For PG straps, only DRC violations that are repaired by cutting part of the wire shape are listed. Abutted phantom PG straps are merged and reported as part of the same error. When a phantom via is created, information about the via and the DRC violation caused by the via are saved. The information includes data about the top and bottom wire shapes, via def, and net name. The error information shows the initial violation before fixing.

The following figure shows a missing via error displayed in the Error Browser.

Figure 101 Phantom Via



## Merging Shapes in the PG Mesh

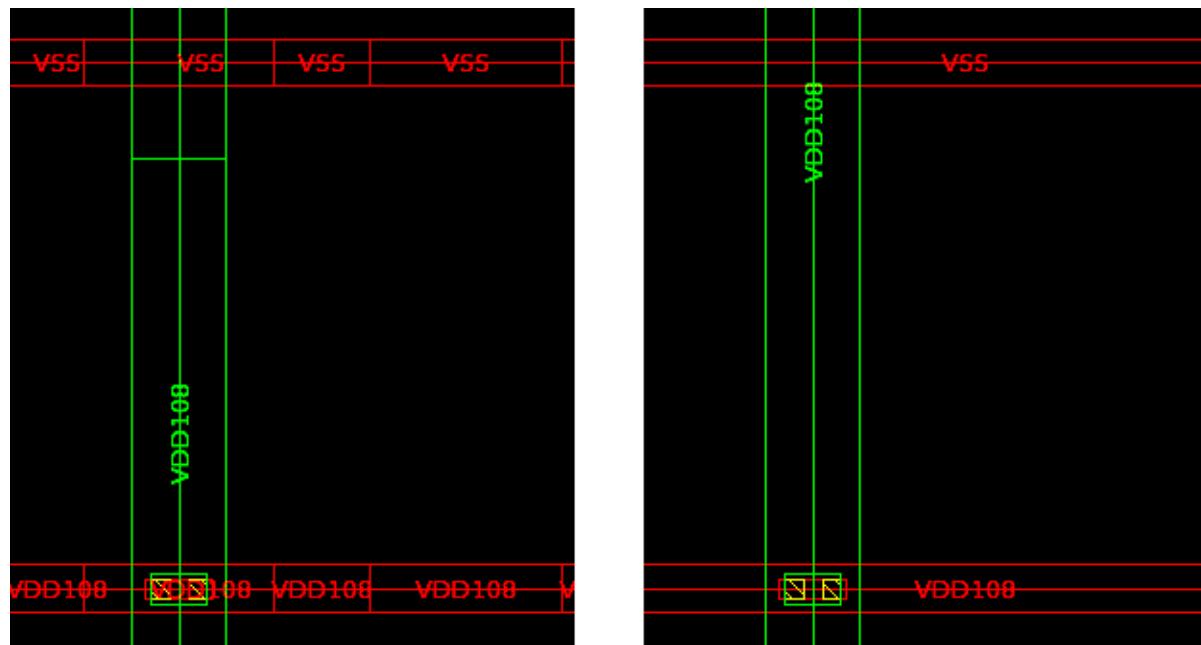
Some designs might contain multiple overlapping PG straps that should be merged together. The `merge_pg_mesh` command merges shapes based on layer, net name, shape, or type based on command options.

The following example first merges PG mesh shapes for the VDD100 net, then merges shapes for the VSS net.

```
icc2_shell> merge_pg_mesh -nets {VDD100 VSS}
...
Net VSS has 4028 overlapped wires.
Net VSS created 37 ndmshape
Net VSS deleted 4028 ndmshape
```

The following figure shows a portion of the design before and after running the `merge_pg_mesh` command.

*Figure 102 Before and After Merging Shapes in the PG Mesh*



Use options with the `merge_pg_mesh` command to control how the shapes are merged:

- Limit the merge to specific shape types. The allowed shape types are: `stripe`, `ring`, `macro_pin_connect`, `lib_cell_pin_connect`, `follow_pin`, `core_wire`, and `user_route`.

```
icc2_shell> merge_pg_mesh -types {core_wire stripe}
```

- Limit the checking to specific layers with the `-layers` option.

```
icc2_shell> merge_pg_mesh -layers {M3 M4}
```

- Merge only specified shapes with the `-shapes` option.

```
icc2_shell> merge_pg_mesh \
 -shapes {PATH_50_0 PATH_50_1 PATH_50_2 PATH_50_3}
```

- Limit the merge to only the specified nets with the `-nets` option as shown in the previous example.
- Undo the previous `merge_pg_mesh` command with the `-undo` option.

## Inserting Stapling Vias Into an Existing Power Mesh

By default, the `compile_pg` command inserts vias into the power mesh as specified by the `set_pg_strategy_via_rule` command. However, some designs might require stapling vias that are inserted between overlapping metal stripes on different design layers. To insert stapling vias into a power mesh, use the `create_pg_stapling_vias` command. The following example creates stapling vias between layers M1 and M2 throughout the core area:

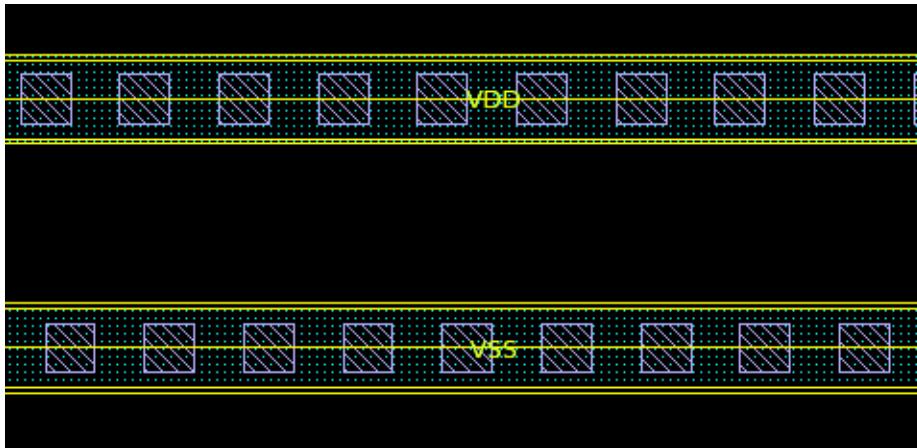
```
icc2_shell> create_pg_stapling_vias -nets {VDD VSS} \
 -from_layer M1 -to_layer M2 \
 -from_shapes [get_shapes -filter "layer_name == M1"] \
 -to_shapes [get_shapes -filter "layer_name == M2"]
```

The `create_pg_stapling_vias` command inserts stapling vias and sets the `is_via_staple` attribute to `true` on the vias inserted by the command. Create a collection of stapling vias by using the `get_vias` command together with the `is_via_staple` attribute as follows:

```
icc2_shell> get_vias -filter is_via_staple
{VIA_SA_0 VIA_SA_1 ...}
```

The following figure shows the stapling vias created by the `create_pg_stapling_vias` command.

*Figure 103 Stapling Vias*



Specify options with the `create_pg_stapling_vias` command to control how the stapling vias are inserted.

- Specify a region in which to insert vias with the `-regions` option

```
icc2_shell> set core [get_attribute [get_core_area] bbox]
icc2_shell> create_pg_stapling_vias -regions $core -nets {VDD VSS} ...
```

- Specify a tag name for the vias inserted by a specific `create_pg_stapling_vias` command with the `-tag` option. The tag can be used as a filter with the `get_vias` command or extracted as an attribute.

```
icc2_shell> create_pg_stapling_vias -nets {VDD VSS} \
 -from_layer ME1 -to_layer M2 \
 -from_shapes [get_shapes -filter "layer_name == M1"] \
 -to_shapes [get_shapes -filter "layer_name == M2"] \
 -tag pg1
icc2_shell> remove_vias [get_vias -filter tag==pg1]
136
```

- Create a via matrix of stapling vias instead of individual vias by specifying the `-create_via_matrix` option. Other settings are the same as the previous example. Note that this example creates 136 via matrixes, one for each horizontal row of overlapping metal shapes on layers ME1 and ME2.

```
icc2_shell> create_pg_stapling_vias \
 -nets {VDD VSS} -from_layer ME2 -to_layer ME1 \
 -from_shapes [get_shapes -filter "layer_name == ME2"] \
 -to_shapes [get_shapes -filter "layer_name == ME1"] \
 -create_via_matrix -tag pg1
```

```
icc2_shell> sizeof_collection [get_via_matrixes -filter tag==pg1]
136
```

- Specify the offset in both the x- and y-directions with the `-offset {x_offset y_offset}` option. The `x_offset` value specifies the amount to shrink the layer intersection in the x-direction. The `y_offset` value specifies the amount of vertical displacement to apply.
- Limit the number of vias in an inserted via array with the `-max_array_size` option.
- Specify contact codes to use when inserting vias with the `-contact_code` option.
- Specify custom PG via masters to form a stacked via by using the `set_pg_via_master_rule` command and the `-via_masters` option with the `create_pg_stapling_vias` command. In this case, via pitch is defined within the `set_pg_via_master_rule` command; the `-pitch` option is not used with the `create_pg_stapling_vias` command.

```
icc2_shell> set_pg_via_master_rule vial_staple \
 -contact_code vial -via_array_dimension {2 1} \
 -allow_multiple {5 0} -snap_reference_point {0 0}

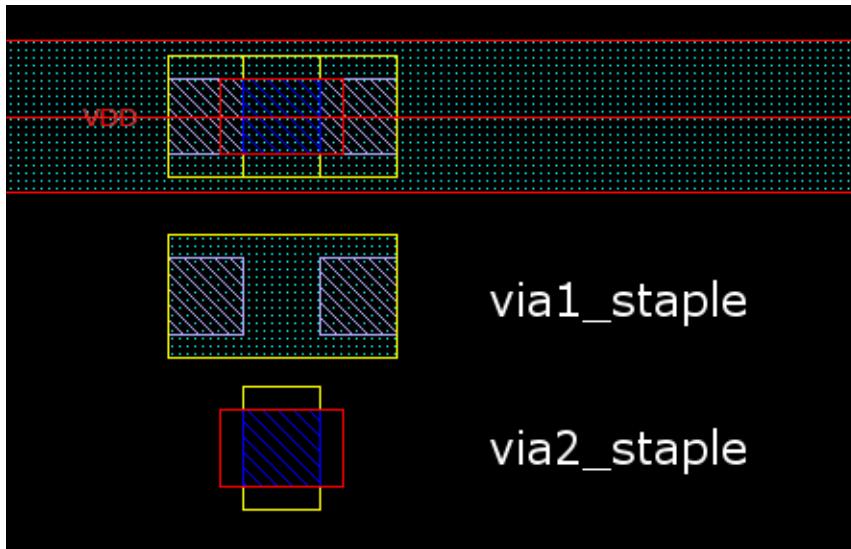
icc2_shell> set_pg_via_master_rule via2_staple \
 -contact_code via2 -via_array_dimension {1 1} \
 -allow_multiple {5 0} -snap_reference_point {0 0}

icc2_shell> create_pg_stapling_vias -nets {VDD VSS} \
 -from_layer ME3 -to_layer ME1 \
 -from_shapes [get_shapes -filter "layer_name == ME3"] \
```

```
-to_shapes [get_shapes -filter "layer_name == ME1"] \
-via_masters {via1_staple via2_staple}
```

[Figure 104](#) shows a via inserted by the previous command. The via1\_staple and via2\_staple via masters are shown below the inserted via for reference.

*Figure 104 Stapling Via With Multiple Masters*



## Connecting Via Ladders to Power and Ground

A via ladder, or via pillar, is a stacked via that starts from the pin layer and extends to an upper layer where the route is connected. Via ladders reduce via resistance and can improve performance and electromigration robustness.

Use the `connect_pg_via_ladders` command to connect an available via ladder to a specified cell and net. The following example creates a connection between the pin connected to net n1 in over cell u1 and the via ladder over the same cell. The tool applies the tag named `via_ladder` to the connection.

```
icc2_shell> connect_pg_via_ladders -cells u1 -nets n1 -tag via_ladder
```

## Inserting and Connecting Power Switches

In a design that contains shut-down power domains, you can insert power switches to control the power within the power domain. To create a power switch ring,

1. Create a voltage area with the `create_voltage_area` command.

```
icc2_shell> create_voltage_area -power_domains PD1 \
 -region {{100 100} {300 300}}
```

2. Create the power switch ring with the `create_power_switch_ring` command.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw \
 -x_pitch 20 -y_pitch 30 -orient r180 -snap_to_site_row false
```

To create a power switch array,

1. Create a voltage area with the `create_voltage_area` command.
2. Define a placement pattern with the `set_power_switch_placement_pattern` command.

```
icc2_shell> set_power_switch_placement_pattern -name p1 \
 -direction horizontal \
 -pattern {HEAD16DM {SPACE 5} {{HEAD8DM {SPACE 5}} 3}} \
 -connect_mode hfn -driver HEAD16DM
```

3. Create the power switch array with the `create_power_switch_array` command.

```
icc2_shell> create_power_switch_array -power_switch inst_sw \
 -pattern p1 -x_pitch 50 -y_pitch 20 -checkerboard even \
 -prefix "PS_"
```

Options to the `create_power_switch_array` and `create_power_switch_ring` commands control the spacing, orientation, naming, placement area, and other features of the ring or array. The following sections describe the options available for these commands.

To create a power switch placement pattern, use the `set_power_switch_placement_pattern` command as follows:

```
icc2_shell> set_power_switch_placement_pattern -name pattern1 \
 -driver buf1 -direction vertical \
 -placement_type array \
 -pattern {buf1 {SPACE 10} {buf2 {SPACE 5} 3}} \
 -connect_mode daisy
```

Options to the `set_power_switch_placement_pattern` command control how the pattern is defined. Use the `-name` option to specify the name of the pattern. Use the `-placement_type ring | array` option to specify whether the pattern applies to a power

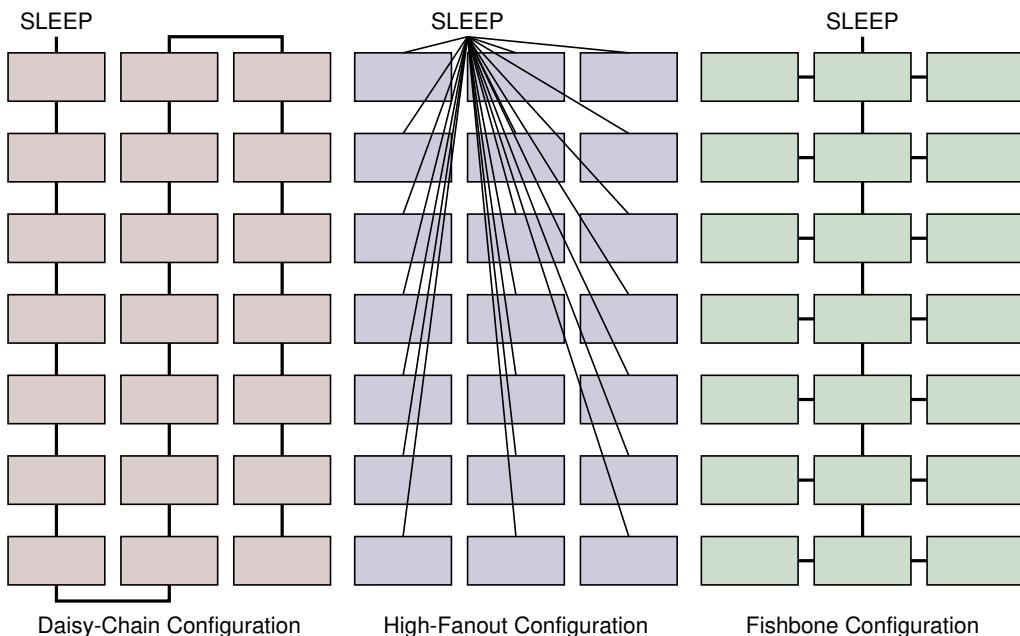
switch ring or array. Use the `-driver` option to assign a driver cell for the pattern. Use the `-direction horizontal | vertical` option to specify the placement direction. Use the `-connect_mode hfn | daisy` option to specify whether the power switches are connected in high-fanout mode (hfn) or daisy-chain mode. Use the `-pattern` option to specify the actual placement pattern. Use the `-port_net_name` option to set the net name prefix for the nets in the connection pattern.

After inserting the power switch ring or array, you can connect the control pins of the power switches with the `connect_power_switch` command as follows:

```
icc2_shell> connect_power_switch \
 -source [get_pins u0_2/pd_switchable/sw_ctl] -mode hfn \
 -port_name p2_sd -object_list [get_cells u0_2/.* -regexp \
 -filter "ref_name == HEAD2X16_HVT"]
```

Options to the `connect_power_switch` command control how the power switches are connected. Use the `-source` option to specify the control signal within the power domain for the power switch. Use the `-mode hfn | daisy | fishbone` option to specify the connection strategy for the power switches as shown in [Figure 105](#). Use the `-port_name` option to specify the base name of the switch ports when a new port is created. Use the `-object_list` option to specify the switch cells to connect. See the man page for additional options.

*Figure 105 Power Switch Fanout Configurations*



## Power Switch Ring and Array Options

Use the following options with the `create_power_switch_array` and `create_power_switch_ring` commands to control how the power switches are inserted. The options described in this section can be used with either command.

- Specify the power switch strategy name with the `-power_switch` option if the `-lib_cell` option is not specified. The `-power_switch` option uses the power switch defined in the UPF file.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw
```

- Specify the name of the power switch library cell with the `-lib_cell` option.

```
icc2_shell> create_power_switch_ring -lib_cell HEAD16DM \
-voltage_area G/PD
```

- Specify an instance name prefix for the switch cells with the `-prefix` option.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw \
-prefix POWERSWITCH_
icc2_shell> get_cells {*/POWERSWITCH_*}
{G/POWERSWITCH_snps_snps_HEAD16DM_H0_1477 ...}
```

- Define the shape or bound to place the switch cells with the `-voltage_area`, `-voltage_area_shape`, or `-boundary` option.

```
icc2_shell> create_power_switch_ring -lib_cell HEAD16DM \
-voltage_area G/PD
icc2_shell> create_power_switch_ring -lib_cell HEAD16DM \
-voltage_area_shape {VOLTAGE_AREA_SHAPE_2}
icc2_shell> create_power_switch_ring -power_switch inst_sw \
-boundary {{100 100} {200 200}}
```

- Specify the initial offset and spacing between each power switch with the `-x_pitch`, `-y_pitch`, `-x_offset`, and `-y_offset` options.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw \
-x_pitch 10 -y_pitch 10 -x_offset 20 -y_offset 20
```

- Specify the placement pattern and orientation of the switch cells with the `-pattern` and `-orient` options. In a power switch array, the same orientation is applied throughout the array. In a power switch ring, switch cells placed on the top and right edges are rotated accordingly.

```
icc2_shell> set_power_switch_placement_pattern -placement_type ring \
-name pattern1 -pattern {HEAD32DM {HEAD16DM 2} {HEAD8DM 3}}
icc2_shell> create_power_switch_ring -power_switch mult_sw \
-pattern pattern1 -orient r180
```

The pattern is defined by the `set_power_switch_placement_pattern` command and is replicated across the power switch ring or throughout the power switch array. For power switch arrays, pattern-based power switch insertion automatically connects SLEEP and ACKNOWLEDGE pins within the pattern.

- Allow power switch cells to avoid snapping to site rows with the `-snap_to_site_row false` option.

```
icc2_shell> create_power_switch_ring -power_switch mult_sw \
-snap_to_site_row false
```

### Power Switch Ring Options

Use the following options with the `create_power_switch_ring` command to control how the power switches are inserted in a power switch ring.

- Continue the pattern specified by the `-pattern` option and avoid restarting the placement pattern on adjacent edges with the `-continue_pattern` option.

```
icc2_shell> create_power_switch_ring -power_switch mult_sw \
-pattern pattern1 -continue_pattern
```

- Specify the starting and ending points of a partial ring with the `-start_point` and `-end_point` options.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw \
-start_point {100 70} -end_point {388 175} -x_pitch 20 \
-y_pitch 30 -orient R180
```

- Specify the list of filler cells to be placed into gaps in the power switch ring with the `-filler_cells` option. The tool inserts the largest possible filler cell into the gap between power switch cells.

```
icc2_shell> create_power_switch_ring -power_switch mult_sw \
-filler_cells {FILL8 FILL4 FILL2} -snap_to_site_row false
```

- Specify the corner library cell name and orientation to use when placing cells at the inner (concave) and outer (convex) corners of the ring with the `-inner_corner_cell`, `-inner_corner_cell_orient`, `-outer_corner_cell`, and `-outer_corner_cell_orient` options.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw \
-inner_corner_cell {*/HEAD16DM} -outer_corner_cell {*/HEAD16DM}
```

- Specify the library cell to be inserted along vertical edges with the `-vertical_lib_cell`, `-vertical_lib_cell_orient`, and `-vertical_filler_cells` options.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw \
 -vertical_lib_cell HEAD16DM -vertical_filler_cells FILL8 \
 -vertical_lib_cell_orient R0
```

- Specify a boundary along which to insert power switch cells with the `-through_points` option.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw \
 -through_points {{100 100} {500 100} {500 300}}
```

- Specify the total number of power switches to be inserted and evenly distributed with the `-switch_number` option.

```
icc2_shell> create_power_switch_ring -power_switch inst_sw \
 -switch_number 5
```

## Power Switch Array Options

Use the following options with the `create_power_switch_array` command to control how the power switches are inserted in a power switch array.

- Place power switches based on site rows instead of y-offset and y-pitch with the `-siterow_offset` and `-siterow_pitch` options.

```
icc2_shell> create_power_switch_array -voltage_area InstDecode/PD \
 -power_switch InstDecode/inst_sw -x_pitch 10 -siterow_pitch 5 \
 -siterow_offset 2
```

- Create a checkerboard placement pattern with the `-checkerboard even` or `-checkerboard odd` option. A checkerboard pattern has alternating empty row and column locations in the array. The `even` argument specifies that the tool places a switch cell in the lower-left corner position of the array; the `odd` argument specifies that this location is left open.

```
icc2_shell> set_power_switch_placement_pattern -name pattern2 \
 -direction horizontal \
 -pattern {HEAD16DM {SPACE 5} {{HEAD8DM {SPACE 5}} 3}} \
 -connect_mode hfn -driver HEAD16DM
icc2_shell> create_power_switch_array -power_switch inst_sw \
 -pattern pattern2 -x_pitch 50 -y_pitch 20 -checkerboard even \
 -prefix "POWERSWITCH_"
```

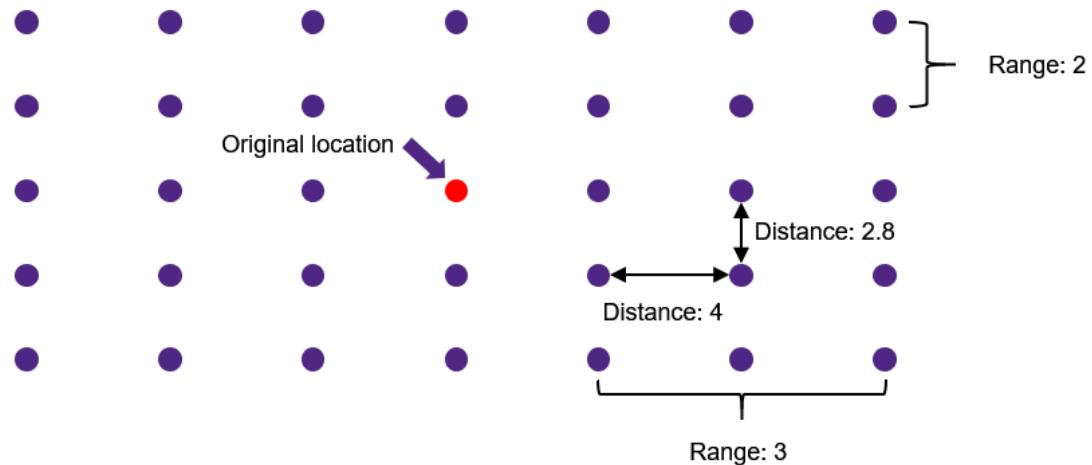
- Specify the vertical power and ground wires with which to align the power switches with either the `-pg_straps` or `-pg_strategy` option.

```
icc2_shell> create_power_switch_array -power_switch mult_sw \
 -pg_straps [get_shapes -of VDD -filter "layer_name==M8"] \
 -y_pitch 10
```

If needed, use the `-align_marker` option together with the `-pg_straps` or `-pg_strategy` option to control the horizontal offset of the power switch cells to the wires. Otherwise, the wires are aligned to corresponding power switch pins.

- Specify the starting point of a uniform power switch grid for disjointed voltage area regions with the `-offset_start{x y}` option.
- Specify the search pitch with the `-fine_grid_pitch` and `-search_range` options to place the power switch when it cannot be inserted in the original location. The tool searches for a location within the specified pitch and places the power switch successfully in that location. The first number in the `-search_range` option specifies the number of grid points in the x-axis. The second number is for the y-axis. Both of these values should be integers. The `-fine_grid_pitch` option specifies the distance from one grid to another and it accepts float values. The combination of `-fine_grid_pitch` and `-search_range` options forms an array grid as possible search locations.

```
icc2_shell> create_power_switch_array -power_switch Multiplier/mult_sw \
 \
 -x_pitch 20.0 -y_pitch 20.0 -fine_grid_pitch {4.0 2.8} \
 -search_range {3 2}
```



---

## Setting Resistance Values for Power Switch Cells

To set resistance values for power switch cells, use the `plan.pna.power_switch_resistance` application option and specify pairs of instance names and resistance values.

The following example sets a resistance value of 10 ohms on cell HEAD2X16\_HVT and a resistance value of 15 ohms on cell HEAD2X16\_LVT.

```
icc2_shell> set_app_options -name plan.pna.power_switch_resistance \
 -value {{HEAD2X16_HVT 10} {HEAD2X16_LVT 20}}
```

Alternatively, you can set the resistance attribute directly on cell instances. The following example sets the `power_switch_resistance` attribute to specify a resistance value of 10 ohms on instances of switch cell SWITCH\_1 and a resistance value of 20 ohms on instances of switch cell SWITCH\_2.

```
icc2_shell> set_attribute {$switch_cell_1} \
 -name power_switch_resistance -value 10
icc2_shell> set_attribute {$switch_cell_2} \
 -name power_switch_resistance -value 20
```

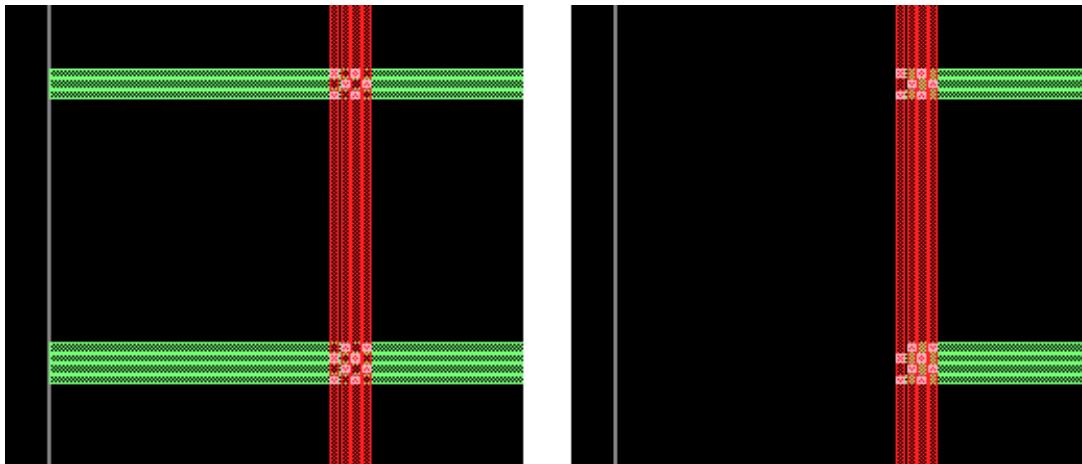
Note that the attribute has a higher priority than the application option.

---

## Trimming the Power and Ground Mesh

Your power plan might contain protruding or dangling shapes after creating the power mesh. To remove these extra shapes, use the `trim_pg_mesh` command. The command trims all power and ground routes up to the closest via metal enclosure on the same layer for the same net. [Figure 106](#) shows a section of a design before and after running the `trim_pg_mesh` command.

Figure 106 PG Mesh Segment Before and After Trimming



Use options with the `trim_pg_mesh` command to control how the mesh is trimmed.

- Trim only the specified net names or layers with the `-nets` and `-layers` options. You can also use the `-shapes` option and specify a collection of net shapes. The following two commands are equivalent.

```
icc2_shell> trim_pg_mesh -nets {VDD} -layers {M5}
icc2_shell> trim_pg_mesh \
 -shapes [get_shapes -filter "layer_name==M5 && owner.name==VDD"]
```

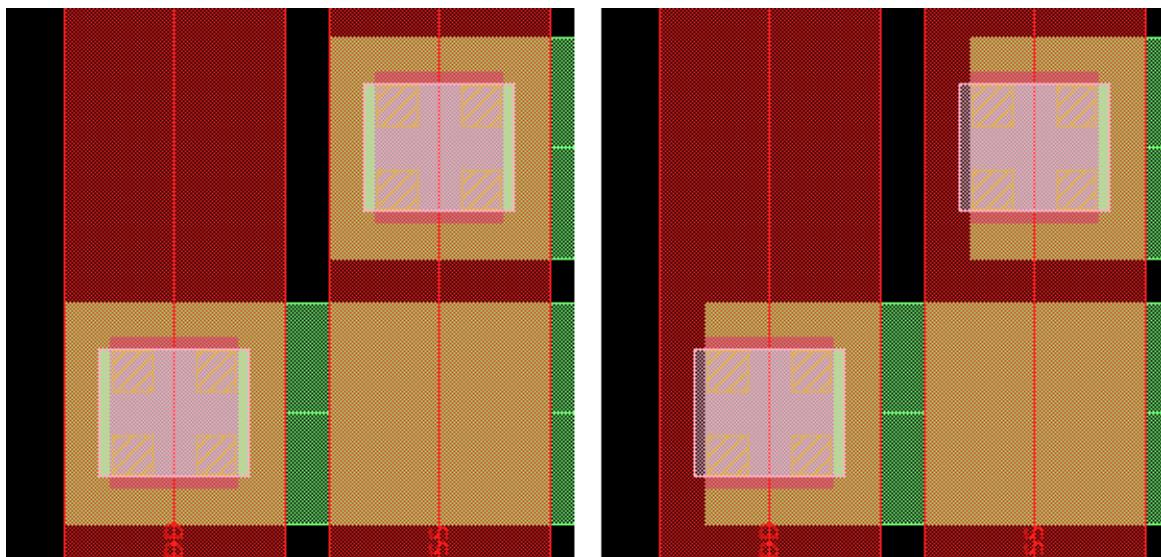
- Trim only nets of the specified type, including ring, stripe, lib\_cell\_pin\_connect, macro\_pin\_connect, and macro\_conn.

```
icc2_shell> trim_pg_mesh -types {ring stripe}
```

- Trim the net to the wire boundary (default) or the via boundary with the `-trim_to_target_wire` | `via` option.

```
icc2_shell> trim_pg_mesh -trim_to target_wire # left image
icc2_shell> trim_pg_mesh -trim_to via # right image
```

*Figure 107 Trim to Wire and Trim to Via*



## Checking Power Network Integrity

The IC Compiler II tool supports commands to identify missing vias, check the connectivity of the power ground network, and verify DRC compliance. The following sections describe the commands used to perform power network validation.

### Checking for Missing Vias

Use the `check_pg_missing_vias` command to check for missing vias between overlapping regions of different metal layers. The command writes information to the console and to an error database. Use the error browser to view the errors in the error database. The following example checks the current design for missing vias.

```
icc2_shell> check_pg_missing_vias -nets VDD
Check net VDD vias...
Number of missing vias on VIA3 layer: 30
Number of missing vias on VIA4 layer: 60
Number of missing vias on VIA5 layer: 40
Total number of missing vias: 130
```

Options to the `check_pg_missing_vias` command control how the check is performed.

- Specify a file that contains the bottommost and topmost metal layers to check, the via pitch, and whether to check stacked vias, vias on macro pins, and vias on parallel routes with the `-via_rule_file` option.

```
icc2_shell> check_pg_missing_vias -nets VDD -via_rule_file viaRule.txt
```

An example via rule file is as follows. Note that the (#) character and comments following the character are for reference only and are not allowed in the file.

```
viaRule1 {
 bottom_layer: METAL # Bottom metal layer to check
 top_layer: METAL6 # Top metal layer to check
 stack_via: true # Enable/disable stacked via check
 parallel: false # (Optional) Enable/disable via check
 # between parallel wires
 macro_pin: true # (Optional) Enable/disable via check
 # to macro and power switch pins
 pitch: 10 # (Optional) Specifies the via pitch
 # for parallel wires
}
```

- Write a report that contains the layer and location of each missing via with the `-output_file` option.

```
icc2_shell> check_pg_missing_vias -nets VDD -output_file viacheck.rpt
Check net VDD vias...
Number of missing vias on VIA5 layer: 72
Total number of missing vias: 72
...
icc2_shell> sh cat viacheck.rpt
Total number of missing vias: 72
1: Missing via of net VDD on layer VIA5
 bbox: {1773.06, 1573.06}, {1778.06, 1578.06}
2: Missing via of net VDD on layer VIA5
 bbox: {1773.06, 364.565}, {1778.06, 374.565}
...
```

- Write out the default via rules file to `defaultViaRule.txt` with the `-write_default_via_rule_file` option.

```
icc2_shell> check_pg_missing_vias -nets VDD \
 -write_default_via_rule_file
Successfully write default via rule to defaultViaRule.txt
```

After writing the `defaultViaRule.txt` file, you can modify it and load the file with the `-via_rule_file` option to create a custom via check rule file.

- Ignore shielding routes with the `-ignore_shield_route` option.

Note that shield routes have the `shape_use` attribute set to `shield_route`.

- Avoid reporting an error if the via site is smaller than the minimum contact code for the layer with the `-ignore_small_intersections` option.
- Avoid reporting an error if the error location is covered by a routing blockage with the `-honor_routing_blockage` option.

## Checking Power Network Connectivity

Use the `check_pg_connectivity` command to perform a detailed connectivity check and ensure that each block, macro, pad, and standard cell is connected to the power ground network as shown in the following example:

```
icc2_shell> check_pg_connectivity
Loading cell instances...
Number of Standard Cells: 16590
Number of Macro Cells: 4
Number of IO Pad Cells: 893
Number of Blocks: 4
Loading P/G wires and vias...
Number of VDD Wires: 4359
Number of VDD Vias: 263425
Number of VDD Terminals: 0
...
*****Verify net VDD connectivity*****
Number of floating wires: 0
Number of floating vias: 0
Number of floating std cells: 3633
...
```

Use options with the `check_pg_connectivity` command to control how the check is performed.

- Use the `-nets` option to limit the check to a specific collection of power and ground nets.

```
icc2_shell> check_pg_connectivity -nets {VDD VSS}
```

- Use the `-write_connectivity_file` option to write out the results of the connectivity check to a text file.

```
icc2_shell> check_pg_connectivity -write_connectivity_file conn.txt
```

```
icc2_shell> sh head conn.txt

*****Net VDD Connectivity Stats*****

```

```
Number of disjoint networks: 369
...
```

- Use the `-check_macro_pins one | all | none` option to check the power ground connections to macro cells. The `one` argument checks for at least one connection, the `all` argument specifies that all power ground pins must be connected, and the `none` argument disables this check.

```
icc2_shell> check_pg_connectivity -check_macro_pins none
...
Number of Macro Cells: 0
```

- Use the `-check_block_pins one | all | none` option to check the power ground connection for block pins. The `one`, `all`, and `none` arguments have the same meaning as the arguments of the `-check_macro_pins` option.

```
icc2_shell> check_pg_connectivity -check_block_pins all
```

- Use the `-check_pad_pins one | all | none` option to check the power ground connection for pad pins. The `one`, `all`, and `none` arguments have the same meaning as the arguments of the `-check_macro_pins` option.

```
icc2_shell> check_pg_connectivity -check_pad_pins all
```

- Use the `-check_std_cell_pins one | all | none` option to enable or disable power ground network checking for standard cells. The `one`, `all`, and `none` arguments have the same meaning as the arguments of the `-check_macro_pins` option.

```
icc2_shell> check_pg_connectivity -check_std_cell_pins all
```

By default, the `check_pg_connectivity` command does not consider all terminals as potential power sources. To change the default behavior and enable the tool to consider all terminals as power sources, set the `plan.pgcheck.treat_terminal_as_voltage_source` application option to `true`.

To create a collection of floating objects in the power network, save the value returned by the `check_pg_connectivity` command to a variable. The following example runs the `check_pg_connectivity` command, saves the collection of floating objects to the `floating_objects` variable, and selects the floating objects in the GUI.

```
icc2_shell> set floating_objects [check_pg_connectivity]
Loading cell instances...
Number of Standard Cells: 9392
...
icc2_shell> change_selection $floating_objects
```

The `check_pg_connectivity` command uses the `must_join_group` attribute on power and ground terminals to determine if external connections exist between the terminals. If the `must_join_group` attribute is defined for any power or ground terminal, the `check_pg_connectivity` command assumes that terminals with the same attribute setting are externally connected. The following example reduces the number of floating VSS terminals to 0 by setting the `must_join_group` attribute on the terminals.

```
icc2_shell> check_pg_connectivity -nets vss
...
Number of VSS Terminals: 234
...
 Number of floating terminals: 233
...
icc2_shell> set vss_terminals [get_terminals -of_objects [get_ports vss]]
icc2_shell> set_attribute $vss_terminals must_join_group vss_group

icc2_shell> check_pg_connectivity -nets vss
Number of VSS Terminals: 234
...
 Number of floating terminals: 0
...
```

## Validating DRC in the Power Network

To check the DRC compliance of routing objects related to the power and ground network, use the `check_pg_drc` command. For all PG nets, the command checks for DRC violations with respect to the following related objects:

- Pins of nets of any type, including pins that are unassigned
- Routing blockages and keepouts that might restrict power ground routing
- Metal and via routing shapes that are not assigned to any net
- Routing objects associated with other power ground nets that are not connected to the current net being evaluated
- Routing objects of clock nets

The following example uses the `check_pg_drc` command to verify DRC compliance for objects affected by the VDD power net. The command writes out a text report and a database that you can review by using the error browser in the GUI.

```
icc2_shell> check_pg_drc -nets VDD
Total number of errors found: 4375008
 1406 shorts on CO
 25082 shorts on M1
...
Description of the errors can be seen in
gui error set "DRC_report_by_check_pg_drc"
```

You can use multiple threads to shorten the runtime for the `check_pg_drc` command. Use the `set_host_options -max_cores n` command to set the maximum number of processing cores to use when checking power ground DRC.

The `check_pg_drc` command supports options to control which PG nets and which area of the design the tool checks for DRC violations. This command does not check objects of non-PG nets.

- Use the `-nets` option to limit the checking to a specified collection of power and ground nets.
- Use the `-ignore_clock_nets true` option to skip checking of routing objects for clock nets.
- Use the `-coordinates` option to limit the check to a specified bounding box.
- Use the `-bottom_layer` and `-top_layer` options to limit the check to a specified range of layers.
- Use the `-no_gui` option to skip writing the error browser database for the GUI.
- Use the `-output` option to write out a text version of the DRC check result to a file.
- Use the `-load_routing_of_all_nets` option to check the existing routing shapes and vias of all PG nets.

For example, if spacing between a PG-net-routing-shape and a routing-shape of signal net is less than it is required by a rule, an error is reported only if the `-load_routing_of_all_nets` option is true. But a similar error is never reported if the two shapes are assigned to signal nets.

- Use the `-check_detail_route_shapes` option to check the routing shapes and vias of PG nets that have the `detail_route` usage attribute set; if this option is not specified, the `check_pg_drc` command ignores such shapes.

A short example of the DRC text file is as follows:

```
=====
Error type: insufficient space between two same-net via-cuts
Violated rule: Cut-Min_Spacing-Rule
Layer: VIA5
Minimal spacing: 0.07
Actual spacing: 0.05(south - north)
Spacing box: {3408.42 1571.75} {3408.46 1571.8}
Via #1: box = {3408.41 1571.7} {3408.46 1571.75}
 net = VDD
Via #2: box = {3408.42 1571.8} {3408.47 1571.85}
 net = VDD
=====
Error type: illegal overlap of net-shapes
...
```

You can review the errors reported by the `check_pg_drc` command in the GUI. Select Error > Error Browser in the GUI, and select `DRC_report_by_check_pg_drc` as the Data Name. The tool loads the DRC data in the Error Browser.

By default, power network routing ignores signal routes during DRC checking to maximize performance. To enable the `compile_pg`, `create_pg_strap` and `create_pg_vias` commands to validate the DRC correctness of power straps and vias with respect to signal routes, set the `plan.pgroute.honor_signal_route_drc` application option to `true` as follows:

```
icc2_shell> set_app_options -name plan.pgroute.honor_signal_route_drc \
 -value true
plan.pgroute.honor_signal_route_drc true
```

---

## PG Design Checker Rules

For the latest list of PG design checker rules, see the *Synopsys® Technology File and Routing Rules Reference Manual* on SolvNetPlus.

## Analyzing the Power Plan

The IC Compiler II tool supports commands to insert virtual power pads and analyze the quality of the power network. The following sections describe the commands used to set up and use power network analysis (PNA).

### Mapping and Reporting IR Drop

The `analyze_power_plan` command performs different types of analysis on the power plan during design planning. To perform a simple power analysis on your design,

1. Read in the parasitic model data file and set the parasitic parameters.

```
icc2_shell> read_parasitic_tech -tlup max.tluplus \
 -layermap layermap.map
icc2_shell> set_parasitic_parameters -late_spec max.tluplus
```

2. Ensure that the PG nets are associated with the PG network.

```
icc2_shell> resolve_pg_nets
icc2_shell> connect_pg_net -automatic
```

3. Create virtual pad locations for power and ground.

```
icc2_shell> set_virtual_pad -net VDD -coordinate {1000.0 1000.0}
icc2_shell> set_virtual_pad -net VSS -coordinate {1000.0 1000.0}
```

Alternatively, read a file that contains the virtual pad locations by using the `read_virtual_pad_file` command as described in [Using Virtual Pad Files](#). You can also use the existing power pads in the design as pad locations by referencing them with the `-pad_references` option to the `analyze_power_plan` command as shown in the following step.

4. Run power analysis and specify the total power budget.

```
icc2_shell> analyze_power_plan -nets {VDD VSS} -power_budget 1000
```

If your design contains existing power pads, specify the references to the power pads with the `-pad_references` option as follows. This option accepts a list of power net name and cell reference pairs with the format `net_name:pad_reference_name`.

```
icc2_shell> analyze_power_plan -nets {VDD VSS} -power_budget 1000 \
-voltage 1.5 -pad_references {VDD:VDD_NS VSS:VSS_NS}
```

The command generates detailed report files containing instance power values for each cell instance together with resistance and voltage drop values for wires in the power plan.

```
icc2_shell> ls pna_output
leon3mp.VDD.icc2.pna leon3mp.VSS.icc2.pna
leon3mp.VDD.power leon3mp.VSS.power
leon3mp.VDD.pw_hl.pna leon3mp.VSS.pw_hl.pna
```

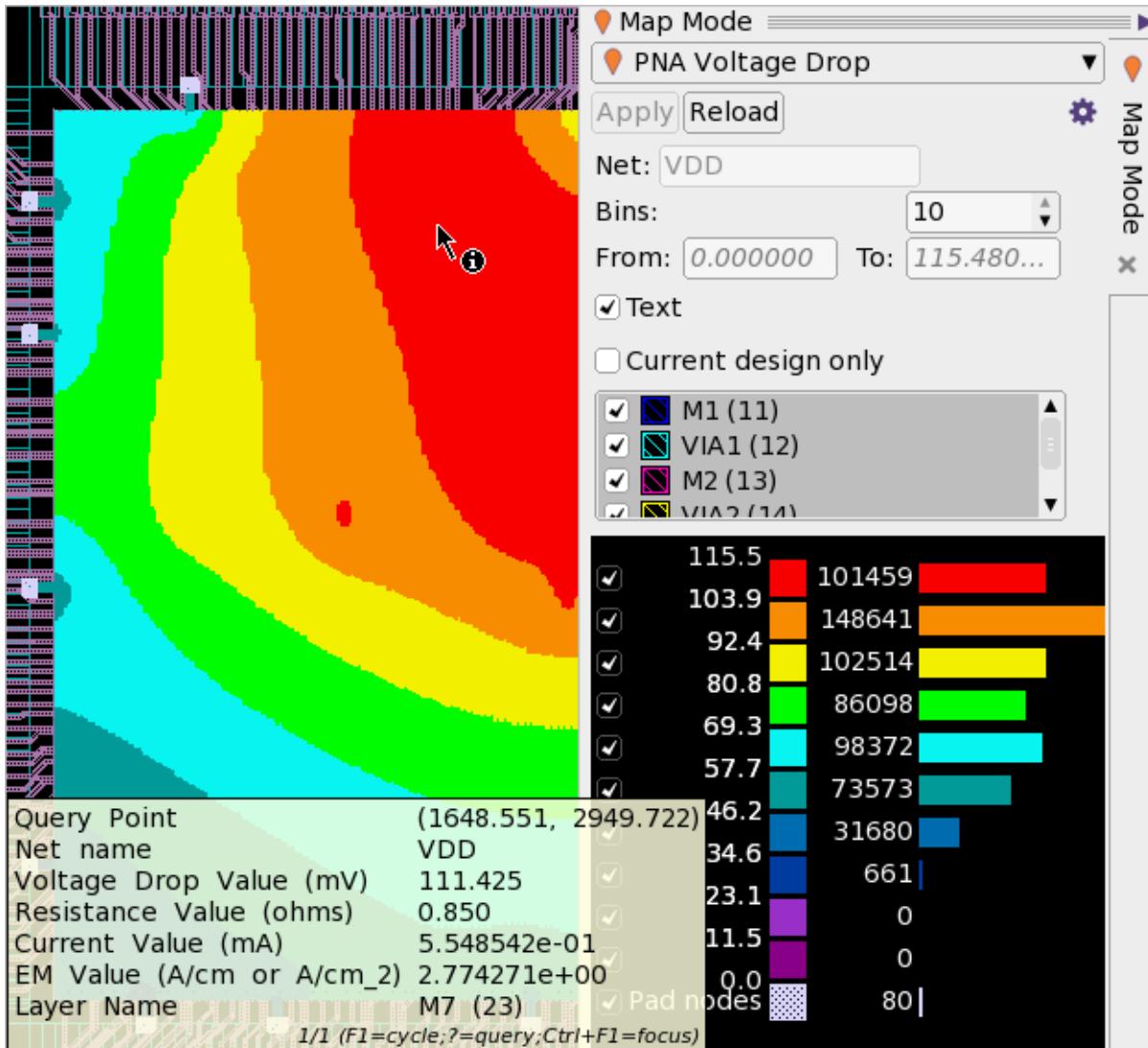
The command also writes a summary report to the console as follows:

```
Primary Net : VDD Secondary Net: VDDS_MIB
Primary Net : VSS Secondary Net:
Number of Standard Cells: 59303
Number of Macro Cells: 76
Number of IO Pad Cells: 345
...
Percentage of routing tracks used by P/G nets
=====
layer | Shape | Spacing | Total
=====
...
M3 | 2.441% | 0.784% | 3.225%
M4 | 0.676% | 0.256% | 0.932%
M5 | 2.837% | 0.934% | 3.771%
M6 | 41.621% | 19.352% | 60.974%
M7 | 33.194% | 20.903% | 54.098%
M8 | 0.382% | 0.043% | 0.425%
M9 | 0.435% | 0.007% | 0.442%
Number of VDD Wires: 22197
...
Total virtual grid current : 75.077 mA
Total current: 666.667 mA
...
Number of extracted resistors: 658947
Number of all nodes: 519143
Number of pad nodes: 80
Number of current sink nodes: 99032
Number of internal nodes: 519063
Pad (687.500 289.777) on Layer M9 Supplies 3.728 mA Current
Pad (669.000 299.600) on Layer M8 Supplies 5.384 mA Current
```

```
...
Maximum IR Drop for Net VDD: 115.480 mV at (2372.000 2763.830)
on layer M6
...
Maximum IR Drop for Net VSS: 69.161 mV at (1805.500 1183.448)
on layer M6
```

The tool reports the amount of current supplied by each virtual pad, the total current in the power grid, the total current for each metal layer, the point of maximum IR drop, and the percentage of routing tracks used by PG nets. The PNA Voltage Drop map panel is populated and the map is displayed on the layout as shown in the following figure. Move the pointer over the map overlay to view the voltage drop, resistance, and current values at that point in the layout.

Figure 108 PNA Voltage Drop



### Using Virtual Pad Files

For designs that do not have power pads, use the `set_virtual_pad` command to create virtual pads in the design. You can save pad locations to a file by using the `write_virtual_pad_file` command and re-create the virtual pads with the `read_virtual_pad_file` command. The virtual pad file contains a list of coordinates for virtual power pads for power network analysis. The following example writes the virtual pad locations to a file named `pads.txt`, removes all virtual pads, and reads in the `pads.txt` file to re-create the pads.

```
icc2_shell> write_virtual_pad_file pads.txt
Writing the virtual pad file pads.txt successfully
```

```
icc2_shell> remove_virtual_pads -all
Removing all virtual pads

icc2_shell> read_virtual_pad_file pads.txt
Added 4 virtual pads
```

The file contains the PG net name followed by the pad locations for the net as shown in the following example. The `auto` keyword specifies that the layer is assigned automatically.

```
icc2_shell> sh cat pads.txt
VDD
1033.275 1860.740 auto
352.915 1116.020 auto
VSS
1033.275 1860.740 auto
352.915 1116.020 auto
```

### Reporting Track Utilization

By default, the `analyze_power_plan` command analyzes the IR drop and reports the track utilization in the design. To skip IR drop analysis and only report track utilization, specify the `-report_track_utilization_only` option as shown in the following example.

```
icc2_shell> analyze_power_plan -report_track_utilization_only
Percentage of routing tracks used by P/G nets
=====
layer | Shape | Spacing | Total
=====
...
M5 | 2.837% | 0.934% | 3.771%
M6 | 41.621% | 19.352% | 60.974%
M7 | 33.194% | 20.903% | 54.098%
M8 | 0.382% | 0.043% | 0.425%
M9 | 0.435% | 0.007% | 0.442%
```

### Specifying a Minimum Layer for Analysis

The `analyze_power_plan` command analyzes all metal layers for IR drop. You can reduce runtime, at the cost of reduced accuracy, by limiting analysis to only upper-level metal layers. To specify the lowest layer for power analysis, set the `plan.pna.ignore_objects_below_layer` application option to the metal layer name as shown in the following example.

```
icc2_shell> set_app_options -name plan.pna.ignore_objects_below_layer \
-value M5
```

In this example, power network analysis analyzes only metal layers M5 and higher. By default, this application option is set to "" and power network analysis considers all metal layers.

## Performing Distributed Power Network Routing

The IC Compiler II tool supports commands to characterize and write out the current power network strategy and pattern settings, then generate the power network through distributed processing. Distributed power network routing reduces the runtime needed to create a power network for your design.

### Preparing for Distributed Power Network Routing

To perform distributed power network routing, you must write out the current power plan constraints with the `characterize_block_pg` command and use the `run_block_compile_pg` command to create the power plan by using a distributed computing network.

The following example writes out the current strategy and pattern settings and defines a script to be run by the `run_block_compile_pg` command.

```
icc2_shell> characterize_block_pg -compile_pg_script pg_script.tcl
...
Characterize block PG constraints.
Characterize PG constraints for block I_ORCA_TOP/I_PCI_TOP in script
 ./pgroute_output/PCI_TOP_pg.tcl.
...
Characterize top-level PG constraints.
..
Characterize PG constraints for top-level design ORCA in script
 ./pgroute_output/ORCA_pg.tcl.
Create PG mapping file ./pgroute_output/pg_mapfile.
```

The `characterize_block_pg` command writes a power plan script for each reference block. Each script contains commands such as `create_pg_mesh_pattern`, `create_pg_ring_pattern`, and so on to create the power ground connection patterns. The scripts also contain `set_pg_strategy` commands to associate the patterns with different areas of the design, and `set_pg_via_master_rule` and `set_pg_strategy_via_rule` commands to define rules to create vias.

If the design contains custom via defs created with the `create_via_def` command, the `characterize_block_pg` command writes out the `create_via_def` commands into each block PG creation script to re-create the custom via defs at the block level.

If you specified the `-compile_pg_script pg_script` option, the tool writes out a map file that associates the power planning scripts with the reference blocks. The `pg_script` file can contain setup commands and application options for creating the power ground routing, but it should not contain via master rules, patterns, strategies, or strategy via rule commands. During distributed power network creation, the map file is associated with the

design by using the `set_constraint_mapping_file` command. In this example, the map file written by the `characterize_block_pg` commands is as follows:

```

ORCA PG_CONSTRAINT ./ORCA_pg.tcl
BLENDER_0 PG_CONSTRAINT ./BLENDER_0_pg.tcl
BLENDER_1 PG_CONSTRAINT ./BLENDER_1_pg.tcl

ORCA COMPILE_PG ./copy_pg_script.tcl
BLENDER_0 COMPILE_PG ./copy_pg_script.tcl
BLENDER_1 COMPILE_PG ./copy_pg_script.tcl

```

---

## Performing Distributed Power Network Routing

After generating the power plan constraint files, use the `run_block_compile_pg` command to create the power plan by using distributed processing. To create the power plan,

1. Assign the map file with the `set_constraint_mapping_file` command.

```
icc2_shell> set_constraint_mapping_file pgout/pg_mapfile
```

2. Set the host options for distributed processing with the `set_host_options` command.

```
icc2_shell> set_host_options -name block_script host_settings
```

3. Run distributed power planning with the `run_block_compile_pg` command.

```

icc2_shell> run_block_compile_pg -host_options block_script
...
Create PG for blocks : BLENDER_0 BLENDER_1
...
found instance I_ORCA_TOP/I_BLENDER_0 for design BLENDER_0
...
Submitting job for block BLENDER_0 ...
...
All tasks created.
...
Successfully ran distributed PG creation.
1

```

The `run_block_compile_pg` command uses distributed processing to create the power plan. If you omit the `-host_options` option, the `run_block_compile_pg` command process each block sequentially.

# 11

## Performing Global Planning

---

To improve global routing between design blocks during the design planning phase, the IC Compiler II tool supports operations to group similar nets, create global routing corridors, and push global route objects into blocks. Global planning enables you to better manage routing resources between blocks. You typically use global bus planning after running block shaping and macro placement.

For more details, see the following topics:

- [Topology Interconnect Planning](#)
  - [Creating Routing Corridors](#)
  - [Creating Global Routes Within Routing Corridors](#)
  - [Adding Repeater Cells](#)
  - [Pushing Down Repeater Cells](#)
- 

### Topology Interconnect Planning

Topology interconnect planning is a methodology for defining and creating a routing topology for large collections of bus signals. The routing topology is defined by creating a topology plan for the bus signals; the plan contains topology nodes and topology edges that describe the routing intent. Designers use topology planning to simplify the task of defining, implementing, and refining the routing topology for the bus signals.

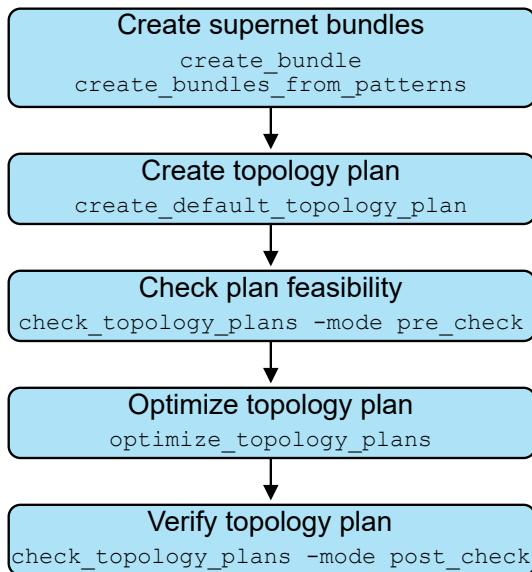
In the traditional flow, top-level interconnect signals are implemented by using EDA point tools to create block pins, insert feedthroughs, insert buffers and repeaters, route bus signals, and so on. Each of these tools has its own constraints methodology and commands to guide the tool; this forces the SoC designer to understand the particular options and constraint settings needed by each tool when implementing the top-level interconnect signals.

In the topology interconnect planning flow, unified topology constraints provide a common constraint language and methodology for creating the routing topology for bus signals in the layout. The unified topology constraints comprise a topology plan, which manages all the constraint requirements; this enables SoC designers to plan, check, optimize, and

verify the top-level interconnections more efficiently. Topology plans are visible in the GUI, where you can use the Design Assistant to create and modify the constraints in the layout.

[Figure 109](#) shows the topology interconnect planning flow.

*Figure 109 Topology Interconnect Planning Flow*



The following topics describe the steps in this flow:

- [Creating Supernet Bundles](#)
- [Creating Net Estimation Rules for Topology Plans](#)
- [Creating Topology Plans](#)
- [Setting the Current Topology Plan](#)
- [Reviewing Topology Plans](#)
- [Refining Topology Plans](#)
- [Optimizing Topology Plans](#)
- [Performing What-If Analysis on the Topology Plan](#)
- [Propagating Topology Plans](#)
- [Writing Topology Plans in Tcl Format](#)
- [Creating Topology Groups](#)

- [Setting the Current Topology Group](#)
  - [Removing Topology Groups](#)
  - [Branching and Ripping Support for Bundles](#)
  - [Bidirectional Support for Bundles](#)
- 

## Creating Supernet Bundles

You begin topology interconnect planning by creating bundles of supernets.

- Supernets define bus signals that might contain feedthroughs, repeaters, and registers, and are named based on the original driver net name. To create a supernet, use the `set_supernet_exceptions` and `create_supernet` commands.
- Bundles are used to group supernets together into a bundle object. Signals within a bundle can be rearranged and sorted to create custom pin ordering. To create a bundle, use the `create_bundle` command and specify the bundle name and the supernets in the bundle as shown in the following example.

```
icc2_shell> create_bundle \
 -name Bundle_z163261 [get_supernets z163261]
```

For more information about creating bundles, see [Creating Bundles and Bundle Constraints](#).

---

## Creating Net Estimation Rules for Topology Plans

Before creating a topology plan, you must create a net estimation rule for the plan. A net estimation rule defines the routing layers, nondefault routing rule, repeater cell name, repeater cell spacing, clock, and other routing- and timing-related constraints. Use the `set_net_estimation_rule` command to define a net estimation rule. Use the `report_net_estimation_rules` command to list the detailed parameter settings for each defined rule.

The following example creates a net estimation rule named M3:

```
icc2_shell> set_net_estimation_rule M3 -parameter "layer" \
 -horizontal_value "M3" -vertical_value "M4"
icc2_shell> set_net_estimation_rule M3 -parameter "buffer_spacing" \
 -value "500.000"
icc2_shell> set_net_estimation_rule M3 -parameter \
 "register_spacing" -value "1200.000"
icc2_shell> set_net_estimation_rule M3 -parameter "buffer" \
 -value "NBUFFX4_RVT"
icc2_shell> report_net_estimation_rules M3
...
```

| parameter      | value(s)              | value from |
|----------------|-----------------------|------------|
| buffer         | NBUFFX4_RVT           | user       |
| buffer_spacing | H: 500.000 V: 500.000 | user       |
| ...            |                       |            |

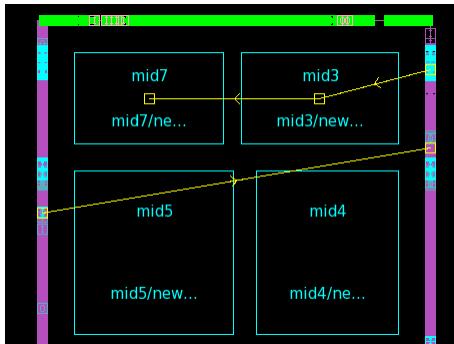
## Creating Topology Plans

A topology plan specifies the routing intent for a supernet bundle. Each plan contains topology nodes and topology edges that define the routing paths and feedthroughs for the route. To create topology plans, use the `create_default_topology_plans` command or use the Design Assistant in the GUI.

The tool creates a base topology plan for each specified bundle (or for all bundles), where each base topology plan contains a start node, end node, and a topology edge that connects the nodes. If the endpoint for the bundle is inside a block, the end node is placed at the center of the block. If a bundle contains feedthroughs, the tool creates nodes inside the feedthrough block. The tool uses the following naming convention for the generated topology plans: `TOPOLOGY_PLAN_<bundle_name>`

Figure 110 shows two base topology plans: the topology plan at the top of the figure contains a feedthrough; the topology plan in the middle of the figure does not.

Figure 110 Topology Plans With and Without Feedthroughs



## Creating Topology Plans From the Command Line

To create base topology plans for one or more supernet bundles from the command line, use the `create_default_topology_plans` command.

The following command creates a base topology plan for each supernet bundle in the block and assigns the M5 net estimation rule to the plan:

```
icc2_shell> create_default_topology_plans -net_estimation_rule M5
{TOPOLOGY_PLAN_B_z161756 TOPOLOGY_PLAN_B_z161763 ...}
```

To create a topology plan for specific supernet bundles, specify the bundles with the `get_bundles` command. The following command creates a topology plan for the supernet bundle named `B_z161756` and assigns the `M3` net estimation rule to the plan:

```
icc2_shell> create_default_topology_plans \
 -net_estimation_rule M3 [get_bundles B_z161756]
{TOPOLOGY_PLAN_B_z161756}
```

To specify layer specific repeater distance, you can create a net estimation rule (NER) for each metal layer and assign these multiple NERs to a topology plan. This approach ensures that the tool applies the relevant NER to the relevant metal layer. If two or more NERs apply to the same layer, the tool assigns the most recent NER. For example, if NER1 is associated with `M3`, NER2 with `M5`, and NER3 with `M5`, then the tool applies NER3 for the `M5` layer.

To associate multiple NERs to a topology plan or a topology edge, use the following commands. In the following example, topology plan `A` is associated with the net estimation rules, `NER1`, `NER2`, and `NER3`.

```
icc2_shell> create_topology_plan -name A \
 -net_estimation_rule {NER1 NER2 NER3}
```

(or)

```
icc2_shell> set_attribute [get_topology_plans A] \
 -name net_estimation_rule - value {NER1 NER2 NER3}
```

Here is an example to associate a topology edge with multiple NERs:

```
icc2_shell> create_topology_edge -plan [get_topology_plan A] \
 -name edge0 -net_estimation_rule {NER1 NER2 NER3}
```

(or)

```
icc2_shell> set_attribute [get_topology_edges A/edge1] \
 -name net_estimation_rule - value {NER1 NER2}
```

If an edge segment has multiple NERs associated with it, then the tool applies the following logic:

- If the segment has no layer shape, then the tool applies the first rule from the NERs list
- If the segment has a layer share, then the tool applies the most recent NER that matches the layer; if no NER matches this criterion, then the first rule from the NERs list is applied

## Creating Topology Plans in the GUI

To create base topology plans in the GUI,

1. Open the Design Assistant by choosing **View > Assistants > Design Assistant**.
2. To create topology plans for
  - All bundles, choose **Create > Create Default Topology Plan**
  - Specific bundles,
    - a. Click the Nets tab and use the `get_bundles` command to populate the list.
    - b. Select the bundles from the list, right-click the selection, and select **Create Default Topology Plans**.
3. Enter information in the **create\_default\_topology\_plans** dialog box, if required, and click **OK**.

The created topology plans are displayed in the block window. You can review the topology plans as described in [Reviewing Topology Plans](#).

You can associate multiple net estimation rules to a topology plan or a topology node from the **Properties Editor > Basic > net\_estimation\_rule** panel. You might want to consider this approach when there is a need to specify layer specific repeater distance. You can create a net estimation rule for each layer and then assign these multiple rules to the topology plan.

---

## Setting the Current Topology Plan

When you create a topology plan, it becomes the current topology plan. To return the current topology plan, use the `current_topology_plan` command without any arguments.

```
icc2_shell> current_topology_plan
```

You can also use the `current_topology_plan` command to set the current topology plan.

```
icc2_shell> current_topology_plan my_topology_plan
```

In the GUI, you can set the current topology plan by choosing **Set Current Topology Plan** from the context menu.

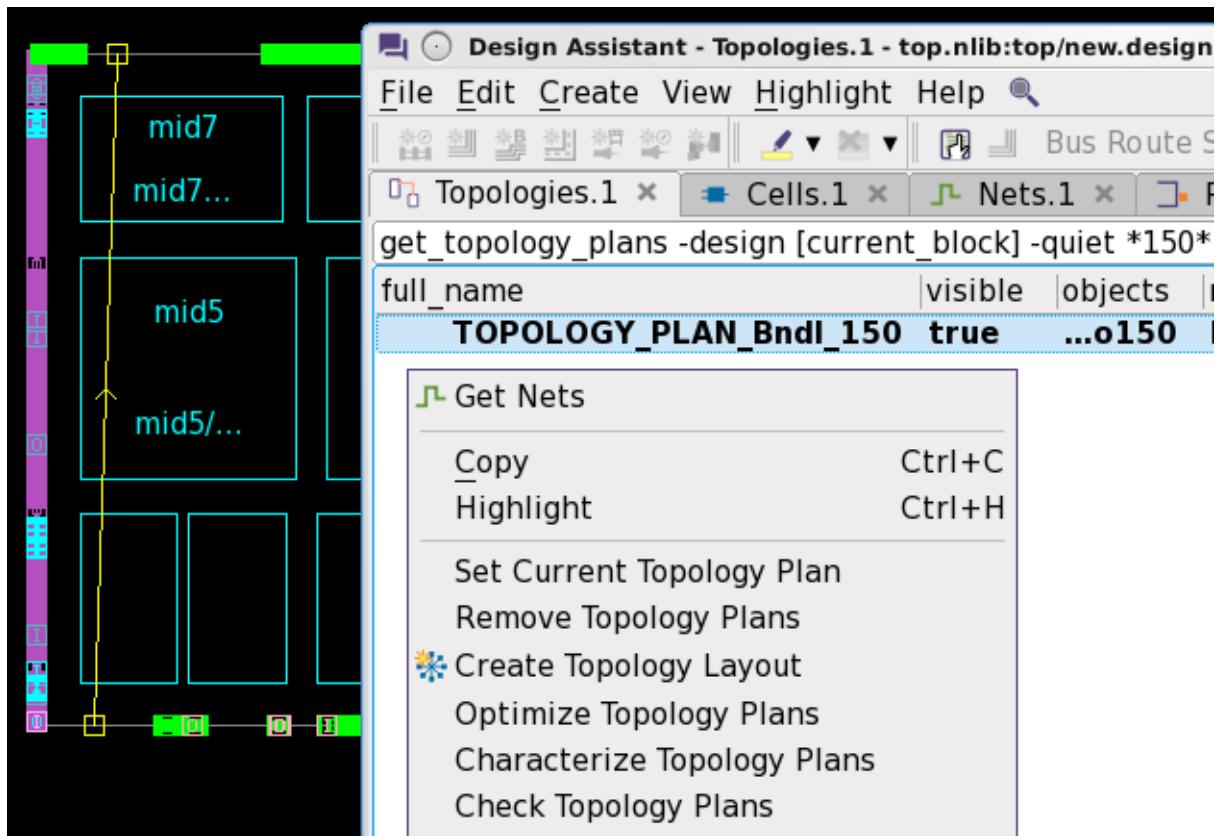
## Reviewing Topology Plans

After creating topology plans, you can examine the plans in the block window and make changes to the plan. Use one of the following methods to view or report topology plans:

- Use the Design Assistant in the GUI to view the topology plan in the block window.

Choose View > New Topologies from the menu to display the Topologies tab, which is shown in [Figure 111](#). By default, the tool shows all topology plans. Enter a command such as `get_topology_plans -design [current_block] -quiet *150*` in the box to filter the list and reduce the number of topology plans that are displayed.

*Figure 111 Design Assistant Topologies Tab*



With a topology plan selected, click the Constraints Editor tab at the right of the Topologies panel to view the list of nodes and edges in the topology plan, as shown in [Figure 112](#). To perform different operations on the nodes and edges, right-click a node or an edge and select an item from the menu.

Figure 112 Constraints Editor

| Constraint           | Owners                 |
|----------------------|------------------------|
| Topology             | TOPOLOGY_PLAN_Bndl_150 |
| Edge: TOPOLOGY_EDGE3 | TOPOLOGY_PLAN_Bndl_150 |
| Edge: TOPOLOGY_EDGE2 | TOPOLOGY_PLAN_Bndl_150 |
| Edge: TOPOLOGY_EDGE1 | TOPOLOGY_PLAN_Bndl_150 |
| Edge: TOPOLOGY_EDGE0 | TOPOLOGY_PLAN_Bndl_150 |
| Node: TOPOLOGY_NODE4 | TOPOLOGY_PLAN_Bndl_150 |
| Node: TOPOLOGY_NODE3 | TOPOLOGY_PLAN_Bndl_150 |
| Node: TOPOLOGY_NODE2 | TOPOLOGY_PLAN_Bndl_150 |
| Node: TOPOLOGY_NODE1 | TOPOLOGY_PLAN_Bndl_150 |

- Use the `report_topology_plans` command to write out information for one or more topology plans. The command reports the net estimation rule, topology nodes, topology edges, and other information about the topology plan.

```
icc2_shell> report_topology_plans TOPOLOGY_PLAN_Bndl_150
...
Topology Plan Description

TOPOLOGY_PLAN_Bndl_150
comment:
stats: 5 nodes, 4 edges, 0 repeaters
net_estimation_rule: M3
objects: 1 bundles
nodes:
 TOPOLOGY_PLAN_Bndl_150/TOPOLOGY_NODE4
 is_reference: false
 num_edges: 1
 num_driving_edges: 1
 objects: 1 ports
 ...
edges:
 TOPOLOGY_PLAN_Bndl_150/TOPOLOGY_EDGE3
 start_node: TOPOLOGY_PLAN_Bndl_150/TOPOLOGY_NODE3
 end_node: TOPOLOGY_PLAN_Bndl_150/TOPOLOGY_NODE4
 net_estimation_rule:
 objects:
 repeaters:
 ...
```

- Use the `check_topology_plans -mode pre_optimization` command to perform a sanity check on the base topology plan.

**Table 11** shows the sanity checks that the `check_topology_plans` command performs. When you run this command, if any of the following warnings or errors are reported, it means that there is a problem with the topology plans. Review these topology plans and revise them, as needed, before you optimize these plans.

**Table 11 Topology Plan Sanity Checks**

| Error/Warning/Information | Message                                                                                                                                                                                                |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DPPA-543                  | Error: Topology node %s contains or, abuts and connects to fixed pins or ports that are placed on layer %s, but this layer is not included in the Net Estimation Rule (NER) associated with this node. |
| DPPA-541                  | Error: Detected overlapping topology nodes on the same layer. The pins or ports on topology node %s are placed interleavingly with topology node %s.                                                   |
| DPPA-499                  | Error: Topology edge %s of block %s overlaps cell %. However, the reported edge is not allowed to create feedthroughs or flyover on the reported cell.                                                 |
| DPPA-498                  | Error: Topology edge %s of block %s is too close to the boundary of cell %s.                                                                                                                           |
| DPPA-497                  | Error: Topology edge %s of block %s is too close to topology edge %s of block %. The collision happens in block %s.                                                                                    |
| DPPA-496                  | Error: The topology node %s has pins or ports placed on layer %s, but this layer is not included in the Net Estimation Rule (NER) that is associated with this node.                                   |
| DPPA-495                  | Warning: The bundle associated with topology plan %s is too wide. Extra repeaters may be needed to compensate the timing difference among the bundle bits.                                             |
| DPPA-494                  | Information: The pin order specified in topology node %s is inconsistent with the net order specified in the bundle associated with the topology plan.                                                 |

## Refining Topology Plans

Before optimizing the topology plans, you can refine the plans to create additional nodes and edges so that routing is done as per your intent. To refine a topology plan, use the `refine_topology_plans` command.

This command creates new nodes and edges for topology edges that have the `shape_layers` attribute defined with Manhattan-style edge segments. If a topology edge does not meet these criteria, no new nodes or edges are created for that topology edge.

One of the use cases where you can use this command is shown in [Figure 113](#). As illustrated, when you create a topology plan for a bundle that has a long route, you need not manually create the feedthrough topology nodes for each and every block in the bundle. Instead, run the `refine_topology_plans` command that automatically creates the necessary feedthrough nodes on these blocks.

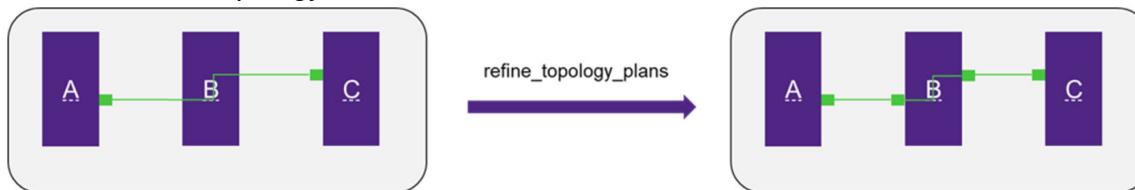
The following command refines all the topology plans within the current design:

```
icc2_shell> refine_topology_plans
```

The following command refines a specific topology plan, A2B:

```
icc2_shell> refine_topology_plans [get_topology_plans A2B]
```

*Figure 113 Refine Topology Plan*



## Optimizing Topology Plans

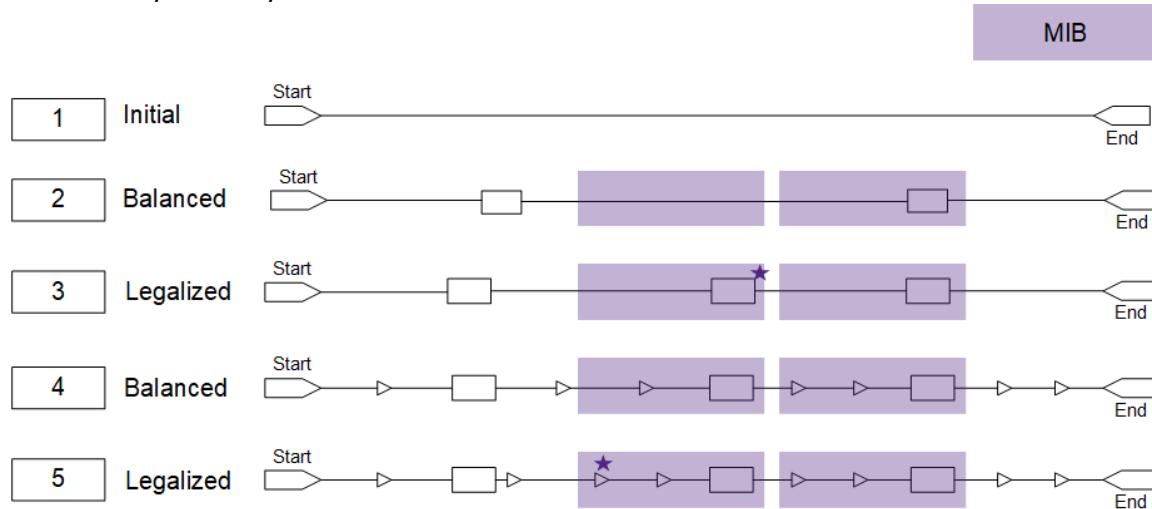
After creating a topology plan, you must optimize the plan to derive the routing topology. The routing topology includes feedthroughs, buffers, retiming flip-flops, and bus routing topologies that honor the topology constraints. To derive the routing topology, use the `optimize_topology_plans` command or use the Design Assistant in the GUI.

The tool performs global routing to create a manhattan route and adds repeaters and sequential elements to the route. Repeater type and spacing is specified by the net estimation rule assigned to the topology plan - while defining the spacing, note that a single spacing rule may be insufficient in constraining all repeaters. If the endpoint for the topology plan is a block or if the block contains feedthroughs, the tool command also creates pins on the block.

The repeater optimization takes place in two passes:

1. The tool places all the repeater (sequential and electrical) along the route and fixes all the spacing violation.
2. The tool performs the legalization operation and inserts additional repeaters where necessary.

*Figure 114 Repeater Optimization*



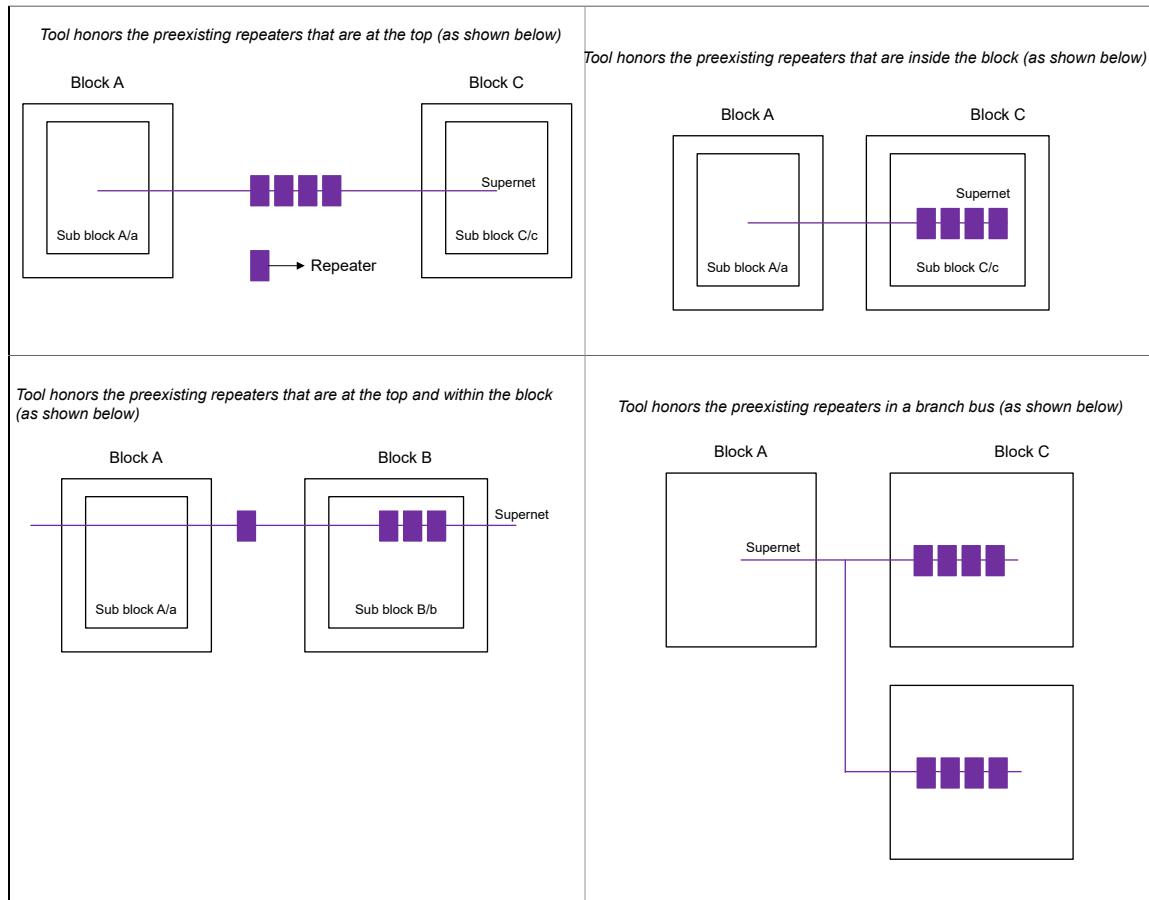
[Figure 114](#) shows repeater optimization in two passes for a straight route by the `optimize_topology_plans` command:

1. Defines a route by specifying the start and end nodes.
2. Places all sequential repeaters equally spaced on the route.
3. Checks that all repeaters are legally placed and adds a repeater to resolve the spacing issues.
4. Add buffers or flip-flops at equal distances to meet the timing constraints.
5. Checks that all buffers are legally placed and adds buffer where necessary.

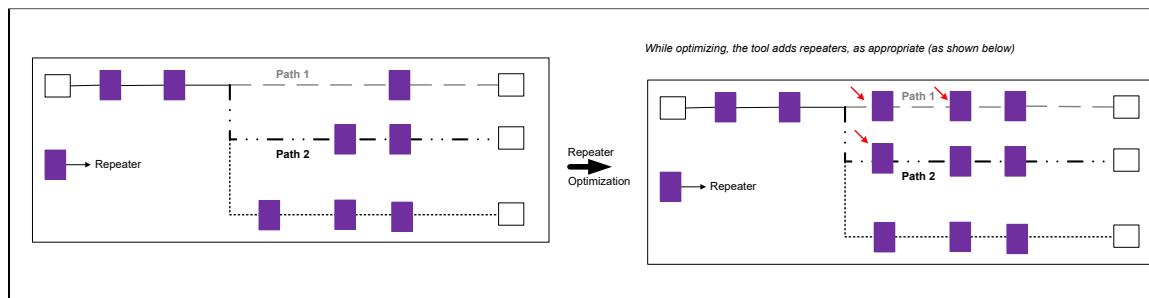
The `optimize_topology_plans` command honors physical and logical information of preexisting repeaters, by default. To disable this feature in the tool, run

```
icc2_shell> set_app_options \
 -list {plan.interconnection.enforce_repeaters_hiers false}
```

**Table 12 Examples Where the Command Honors the Physical Location of Existing Repeaters**

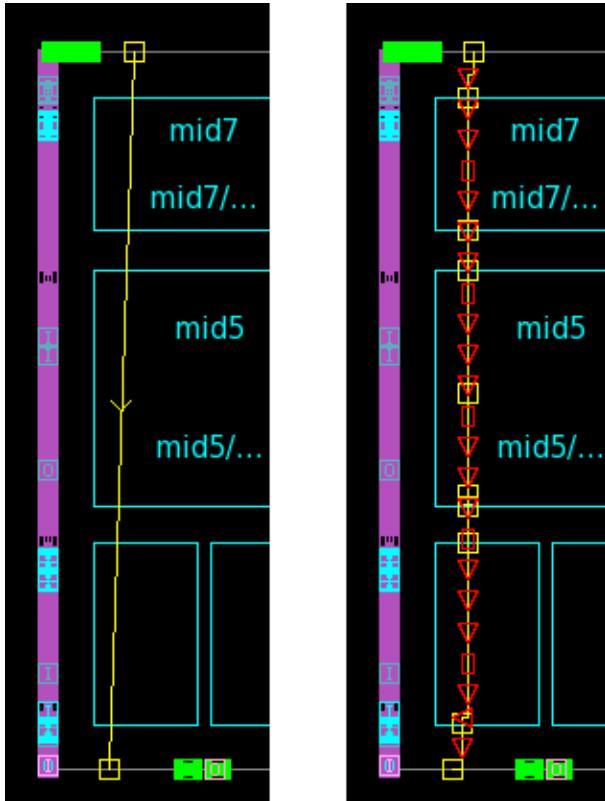


**Table 13 Example Where the Command Honors the Logical Location of Existing Repeaters**



The following figure shows a topology plan before and after optimization. In the figure, the yellow squares represent the block pins, the red triangles represent suggested repeater locations, and the red rectangles represent suggested sequential element locations.

*Figure 115 Optimize Topology Plan*



In the initial stages of your design, where the netlist is not yet finalized, during the optimization phase you might want to ignore pin or feedthroughs (that is, the netlist connectivity) below a certain hierarchy. This approach is useful when you want to look at the implementation results for the parts of the netlist that are complete or finalized, and ignoring the rest of the design. You can achieve this result by using the `stop_hierarchy` user attribute. The flow is as follows:

1. Specify which part of the hierarchy of a bundle you want the tool to ignore.

```
set_attribute bundle_name \
 stop_hierarchy hierarchy_name
```

For example, the following command ensures that the tool ignores the netlist connectivity in block C.

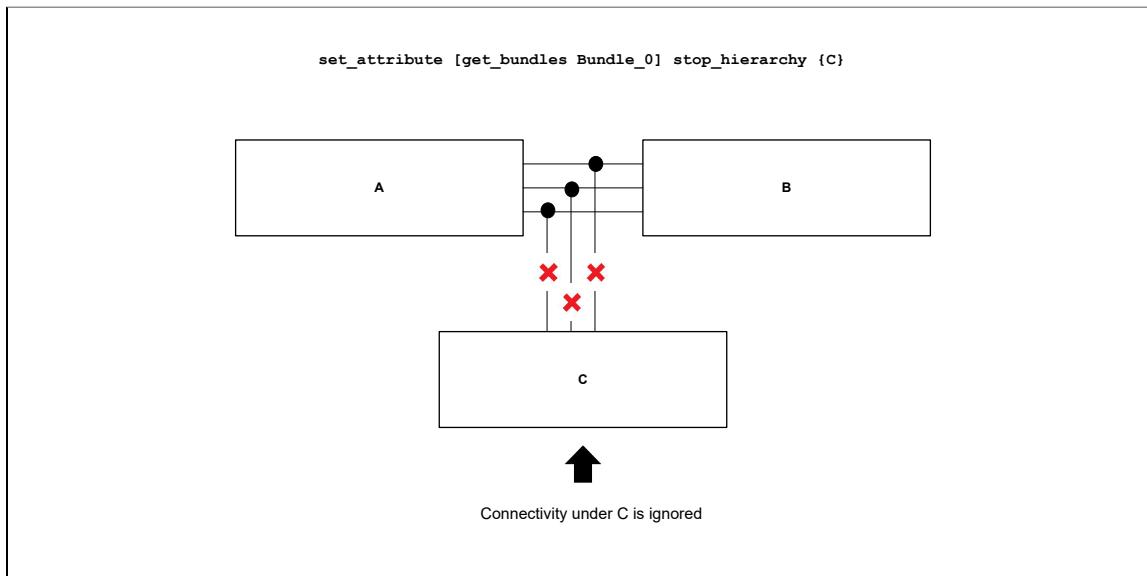
```
icc2_shell> set_attribute [get_bundles Bundle_0] stop_hierarchy {C}
```

The `create_default_topology_plans` command honors the `stop_hierarchy` attribute on the bundle.

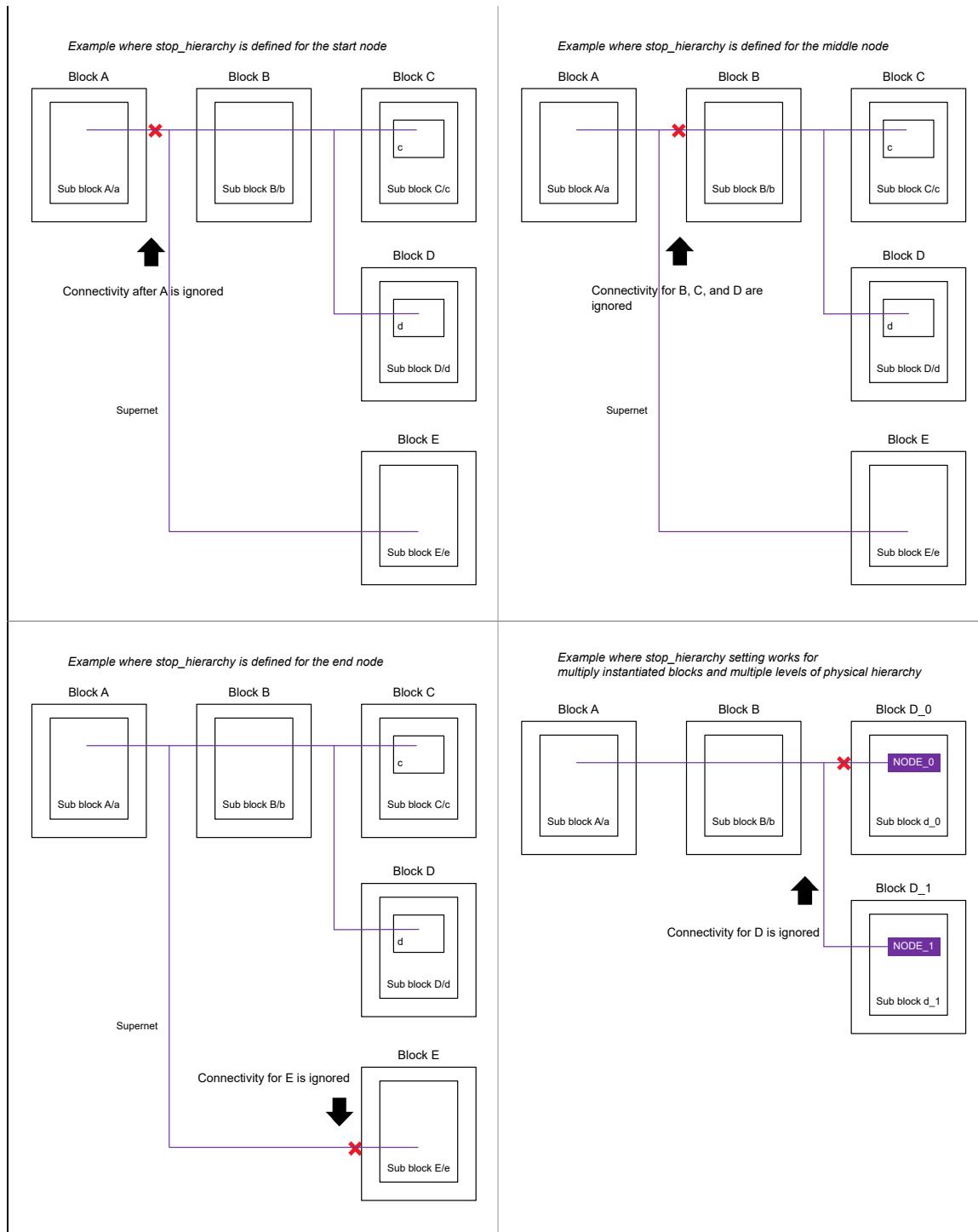
2. Optimize the topology plan.

Here are some examples of how `stop_hierarchy` works.

*Table 14 Examples of How stop\_hierarchy Works*



**Table 14 Examples of How stop\_hierarchy Works (Continued)**



After optimizing the topology plan, you can perform a sanity check by using the `check_topology_plans -mode post_optimization` command.

## Reporting Topology Plan Repeaters

To report the pipeline registers that are part of topology plans after optimization, use the `report_topology_plan_registers` command.

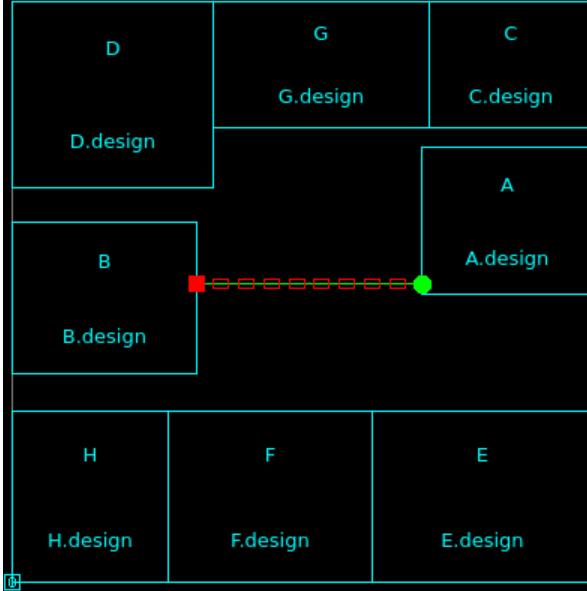
*Table 15 Options to the report\_topology\_plan\_registers Command Control Which Registers Are Reported*

| To do this...                                                                                                                                                                                                   | Use this...                            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
| To specify whether to report all the registers ( <code>-type all</code> ) or only those that are not yet bound to the physician registers ( <code>-type unimplemented</code> ) for the specified topology plans | <code>-type unimplemented   all</code> |
| To specify the list of topology plans whose registers are to be reported<br><br>If no topology plans are specified, the registers that are in all the topology plans within the current design are reported.    | <code>topology_plan_list</code>        |

```
icc2_shell> report_topology_plan_registers -type all
```

In the following example, there are totally eight registers of which four are yet to be bound to physical registers.

Table 16 report\_topology\_plan\_registers Report

| Design                                                                                                                                                                                                                                                                      |  | Report                                                                                                                                                                                                                                                                                                                                                                                            |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
|  <pre> graph TD     D[D.design] --- G[G.design]     G --- C[C.design]     C --- A[A]     A --- B[B]     B --- D     B --- A     H[H.design] --- F[F.design]     F --- E[E.design]   </pre> |  | <pre> fc_shell&gt; report_topology_plan_registers -type all #&lt;topology_plan&gt;,&lt;block_pins_before_PRP_registers&gt;,&lt;block_pins_after_PRP_registers&gt;,&lt;number_of_total_PRP_registers&gt;,&lt;number_of_open_PRP_registers&gt; ===== =====  TOPOLOGY_PLAN_bundle_A_to_B_16,{A/A_to_B_bit_0_0...A/A_to_B_bit_0_15},{B/A_to_B_bit_0_0...B/A_to_B_bit_0_15},8,4 ----- ----- 1   </pre> |  |

To determine the unimplemented repeaters, run the following command,

```
icc2_shell> get_attribute [get_topology_repeater] objects
```

In the following example, TOPOLOGY\_REPEATERO-3 are yet to be implemented.

```
Warning: Attribute 'objects' does not exist on topology_repeater
TOPOLOGY_PLAN_bundle_A_to_B_16/TOPOLOGY_EDGE0/TOPOLOGY_REPEATERO
(ATTR-3)
```

```

Warning: Attribute 'objects' does not exist on topology_repeater
TOPOLOGY_PLAN_bundle_A_to_B_16/TOPOLOGY_EDGE0/TOPOLOGY_REPEATERS1
(ATTR-3)
Warning: Attribute 'objects' does not exist on topology_repeater
TOPOLOGY_PLAN_bundle_A_to_B_16/TOPOLOGY_EDGE0/TOPOLOGY_REPEATERS2
(ATTR-3)
Warning: Attribute 'objects' does not exist on topology_repeater
TOPOLOGY_PLAN_bundle_A_to_B_16/TOPOLOGY_EDGE0/TOPOLOGY_REPEATERS3
(ATTR-3)
{A_to_B_bit_0_0_group0 A_to_B_bit_0_1_group0 A_to_B_bit_0_2_group0
A_to_B_bit_0_3_group0 A_to_B_bit_0_4_group0 A_to_B_bit_0_5_group0
A_to_B_bit_0_6_group0 A_to_B_bit_0_7_group0 A_to_B_bit_0_8_group0
A_to_B_bit_0_9_group0 A_to_B_bit_0_10_group0 A_to_B_bit_0_11_group0
A_to_B_bit_0_12_group0 A_to_B_bit_0_13_group0 A_to_B_bit_0_14_group0
A_to_B_bit_0_15_group0 A_to_B_bit_0_0_group1 A_to_B_bit_0_1_group1
A_to_B_bit_0_2_group1 A_to_B_bit_0_3_group1 A_to_B_bit_0_4_group1
A_to_B_bit_0_5_group1 A_to_B_bit_0_6_group1 A_to_B_bit_0_7_group1
A_to_B_bit_0_8_group1 A_to_B_bit_0_9_group1 A_to_B_bit_0_10_group1
A_to_B_bit_0_11_group1 A_to_B_bit_0_12_group1 A_to_B_bit_0_13_group1
A_to_B_bit_0_14_group1 A_to_B_bit_0_15_group1 A_to_B_bit_0_0_group2
A_to_B_bit_0_1_group2 A_to_B_bit_0_2_group2 A_to_B_bit_0_3_group2
A_to_B_bit_0_4_group2 A_to_B_bit_0_5_group2 A_to_B_bit_0_6_group2
A_to_B_bit_0_7_group2 A_to_B_bit_0_8_group2 A_to_B_bit_0_9_group2
A_to_B_bit_0_10_group2 A_to_B_bit_0_11_group2 A_to_B_bit_0_12_group2
A_to_B_bit_0_13_group2 A_to_B_bit_0_14_group2 A_to_B_bit_0_15_group2
A_to_B_bit_0_0_group3 A_to_B_bit_0_1_group3 A_to_B_bit_0_2_group3
A_to_B_bit_0_3_group3 A_to_B_bit_0_4_group3 A_to_B_bit_0_5_group3
A_to_B_bit_0_6_group3 A_to_B_bit_0_7_group3 A_to_B_bit_0_8_group3
A_to_B_bit_0_9_group3 A_to_B_bit_0_10_group3 A_to_B_bit_0_11_group3
A_to_B_bit_0_12_group3 A_to_B_bit_0_13_group3 A_to_B_bit_0_14_group3
A_to_B_bit_0_15_group3}

```

## Performing What-If Analysis on the Topology Plan

The `optimize_topology_plans` command inserts repeaters and sequential elements into the topology plan to meet timing on the path. The net estimation rule (NER) for the topology plan determines the spacing of the repeater elements.

To experiment with routing the topology plan on different layers, you can specify a different net estimation rule for the plan and reoptimize the plan. After reoptimization, the number of buffers and sequential elements changes based on the spacing requirements of the new net estimation rule.

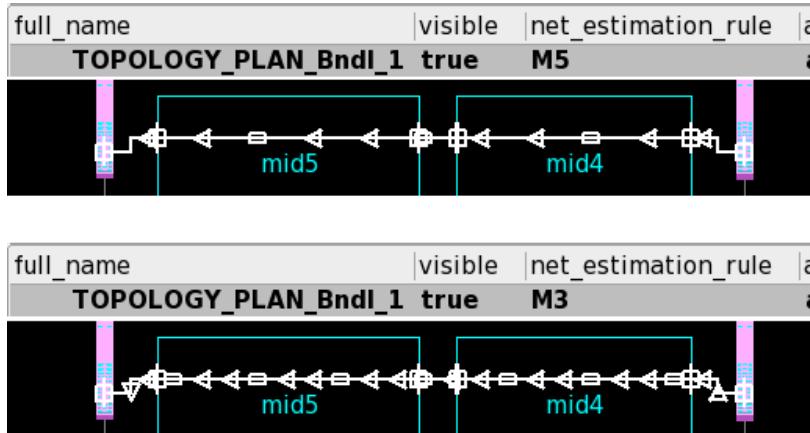
```

icc2_shell> set_attribute \
 -objects [get_topology_plans [current_topology_plan]] \
 -name net_estimation_rule -value M3
everything_shell> optimize_topology_plans [current_topology_plan]

```

After running the `optimize_topology_plans` command, the GUI updates the layout view and displays the topology plan with new buffering.

Figure 116 Optimizing Topology Plans for Different Net Estimation Rules



## Setting the Current Topology Plan

When you create a topology plan, it becomes the current topology plan. To return the current topology plan, use the `current_topology_plan` command without any arguments.

```
icc2_shell> current_topology_plan
```

You can also use the `current_topology_plan` command to set the current topology plan.

```
icc2_shell> current_topology_plan my_topology_plan
```

In the GUI, you can set the current topology plan by choosing **Set Current Topology Plan** from the context menu.

## Propagating Topology Plans

To propagate topology plans across multiple multiply instantiated block (MIB) instances, use the `propagate_topology_plans` command.

You can create the topology plan once by using the `create_topology_plan` command and then use the `propagate_topology_plans` command to copy it over to multiple MIBs that are present in the current design.

When a topology plan is propagated, this command ensures that the topology plan in the destination MIB instances is the same as the source MIB instance.

This command propagates a topology plan only if the following conditions are met:

- The topology plan is associated with a bundle that consists of supernets.
- The supernets of the bundle have the drivers and loads on the same blocks. In addition, all the driver and loads blocks are MIB cells.

The following command propagates the current topology plan across multiple MIB instances that are in the current design:

```
icc2_shell> propagate_topology_plans [current_topology_plan]
```

The following command propagates the specified topology plan, A2B, across multiple MIB instances that are in the current design:

```
icc2_shell> propagate_topology_plans [get_topology_plans A2B]
```

The following command ensures that if the current topology plan is already assigned to the destination MIB instance, then it is not overwritten. If the `-no_overwrite` option is not specified, then the topology plan in the destination MIB instance is overwritten.

```
icc2_shell> propagate_topology_plans [current_topology_plan]
-no_overwrite
```

---

## Writing Topology Plans in Tcl Format

To quickly re-create the topology plans that are in your design, elsewhere - in similar designs, you can write out the these topology plans as a script in Tcl format by using the `write_topology_plans` command. You can then run the script in designs where you need to implement these topology plans.

To write out the topology information to the console or the log file, use the `-log` option. To capture this information in a file, use the `-filename` option.

To write out topology plan information for only specific topology plans, specify a list of topology plans with the `write_topology_plans` command.

The `-pins_locations` option provides you with the flexibility to write out the topology plans by using side node constraints with start and end offsets. This enables you to re-create these topology plans in similar designs where pins are not created or placed yet.

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

The following example writes out the topology information to a file in the current directory.

```
icc2_shell> write_topology_plans -pins_locations -filename
toplogyplaninfo

icc2 S-2021.06-DEV Apr 11, 2021

design: ORCA

Mon Apr 12 11:24:03 2021

set _curr_ [current_block]
```

```

NER (per block)

current_block ORCA

remove_net_estimation_rules -reset default -quiet

Topology Groups (per block)

current_block ORCA

if {[sizeof_collection [get_topology_groups DEFAULT_TOPOLOGY_GROUP -quiet]} == 0} { create_topology_group -name DEFAULT_TOPOLOGY_GROUP }

plan name: pad[0]-pad[3]

current_block ORCA

if {[sizeof_collection [get_topology_plans pad[0]-pad[3] -quiet]} != 0} { remove_topology_plans pad[0]-pad[3] }

set _objLst_ [get_nets { pad[3] pad[4] pad[5] pad[6] pad[2] pad[1] pad[0] }]

create_topology_plan -name pad[0]-pad[3] -group DEFAULT_TOPOLOGY_GROUP -objects $_objLst_

create_topology_node -plan pad[0]-pad[3] -name TOPOLOGY_NODE0 -objects [get_cells { pad_iopad_0 }] -constraint_type side -side E -start 10.000 -end 100.000

current_block $_curr_

```

## Creating Topology Groups

Topology groups enable you to divide your topology plans into multiple smaller groups. You can then work on the smaller chunks independently, which essentially helps in dividing a large design into manageable chunks. It also enables you to group similar topology plans that allows you to take actions such as modify, remove, and so on on a set of topology plans.

You can associate a topology plan to a topology group by using the command line option or the GUI. The tool automatically assigns the topology plans that are not associated with any specific topology group to a default topology group, DEFAULT\_TOPOLOGY\_GROUP. When you migrate, all the topology plans in a design are assigned to this default topology group.

## Creating Topology Groups From the Command Line

To create topology groups for one or more topology plans from the command line for the current block, use the `create_topology_group` command.

The following example creates a topology group that includes topology plans for clock signals.

```
icc2_shell> create_topology_group -name TOPOLOGY_GROUP_CLK -plans
 sys_clk-pclk \
 -comment {For clocks}

icc2_shell> create_topology_group -name TOPOLOGY_GROUP_CLK1 -plans
 [get_topology_plans *clk*] \
 -comment {For clocks}

icc2_shell> set_attribute [get_topology_plans *clk*] group
 [get_topology_groups TOPOLOGY_GROUP_CLK1]

icc2_shell> create_topology_plan -name q -group TOPOLOGY_GROUP_CLK1 \
 create_topology_plan -name sys_clkpclk -group TOPOLOGY_GROUP_CLK1

icc2_shell> create_default_topology_plans -group TOPOLOGY_GROUP_CLK \
 [get_bundles {bundle_A2B_64 bundle_A2B_32}]
```

To edit the topology groups, use the `set_attribute` command.

```
icc2_shell> set_attribute [get_topology_plans -of_objects
 [get_topology_groups A2]] \
 -name group -value [get_topology_groups A1]
```

## Creating Topology Groups in the GUI

To create topology groups in the GUI,

1. Open the Design Assistant by choosing **View > Assistants > Design Assistant**.
2. To create topology groups for specific topology plans,
  - Create the topology plans if they do not exist previously
    - Click on the **Nets** tab and use the `get_nets` command to populate the list.
    - Select the nets from the list, right-click the selection, and select **Create Topology Plan**.
    - Enter information in the `create_topology_plan` dialog box, if required, and click **OK**.
  - Create topology group for specific topology plans
    - Click on the **TopologyPlans** tab and use the `get_topology_plans` command to populate the list.
    - Select the topology plans from the list, right-click on the selection, and select **Create Topology Group**.

One topology plan can belong to only one topology group. Topology plans cannot be shared across multiple topology groups.

- Enter information in the **create\_topology\_group** dialog box, if required, and click **OK**.

If you provide a name to the topology group and if it previously exists within the current block, an error is thrown. However, you can have topology groups with the same name within different blocks of the design.

The created topology groups are displayed in the block window.

## Setting the Current Topology Group

When you create a topology group, it becomes the current topology group. To return the current topology group, use the `current_topology_group` command without any arguments.

```
icc2_shell> current_topology_group
```

You can also use the `current_topology_group` command to set the current topology group.

```
icc2_shell> current_topology_group TOPOLOGY_GROUP_CLK
```

In the GUI, you can set the current topology group by choosing **Set Current Topology Group** from the context menu in Design Assistant.

## Removing Topology Groups

When a topology group is no longer needed, you can remove it by using the GUI or the command line option. When you remove a topology group that contains topology plans, these topology plans get automatically moved to the default topology group. You cannot remove the default topology group if it has any topology plans; however, you can use the `-remove_plans` option with the `remove_topology_groups` command to remove both the topology plans and the default topology group.

The following command removes all topology groups from the current block.

```
icc2_shell> remove_topology_groups -all
```

The following example removes topology groups from blocks A and B:

```
icc2_shell> remove_topology_groups {A/AGroupname B/BGroupname}
```

(or)

```
icc2_shell> current_block A
```

```
icc2_shell> remove_topology_groups A/AGroupname
icc2_shell> current_block B
icc2_shell> remove_topology_groups B/BGroupname
```

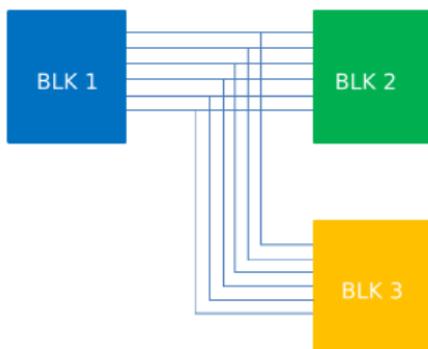
In the GUI, you can remove topology groups by choosing **Remove Topology Group** from the context menu in Design Assistant.

## Branching and Ripping Support for Bundles

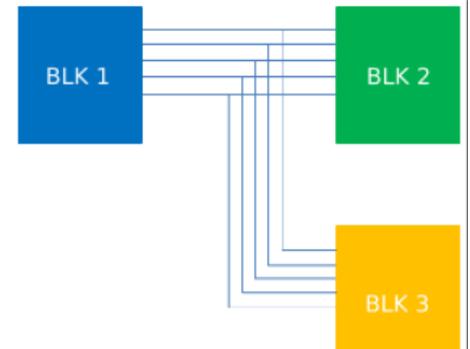
Branching allows all the nets in a bundle to have the same driver block, say BK1, and the same load blocks, say BK2 and BK3. For example, see [Figure 117](#).

Ripping allows the nets in a bundle to have the same driver block and most of the nets to have the same load blocks with a few exceptions. For example, [Figure 118](#) shows a 6-bit bundle with all nets starting from BK1, with five of the nets sharing the same load blocks (BK2 and BK3), with the last bit having only BK2 as its load block.

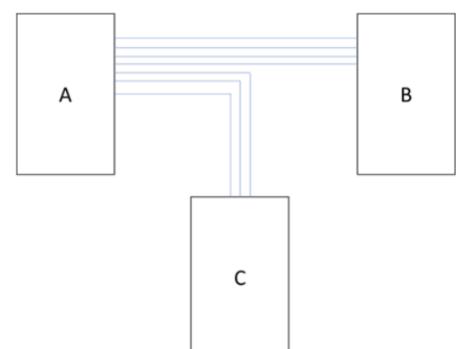
*Figure 117 Branching Bundle Example*



*Figure 118 Ripping Bundle Example 1*



*Figure 119 Ripping Bundle Example 2*



When you plan for a ripping bundle,

- Create the topology plan manually, preferably using the GUI tools to create a Manhattan plan.
- Ensure that the driver block has a topology node associated with it.
- Ensure that the load blocks, regardless whether they are connected to all the nets or a subset of the nets, have their own topology nodes.
- To have feedthrough path via specific blocks, ensure that those blocks have their corresponding topology nodes.
- Define topology edges and connect topology nodes such that all topology nodes are connected to each other directly or indirectly.

When you optimize a ripping bundle by using the `optimize_topology_plans` command,

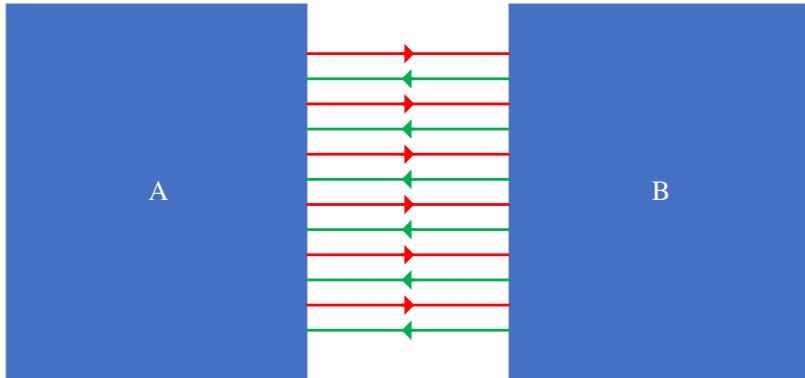
- Topology plans are updated to include new topology nodes and topology edges.
- Bundle pins are grouped correctly on each block and placed together honoring the spacing rules.
- Bundle pins on each block are associated with the correct topology node and ordered according to bit order within the bundle.
- Each topology edge have the shape-layers attribute defined.
- If NER defines the repeater distances, topology repeaters are created on the topology edges if the corresponding shape-layers attribute is defined.

---

## Bidirectional Support for Bundles

To reduce crosstalk, you can interleave nets with opposite signal directions by using a bidirectional bundle. [Figure 120](#) shows a bidirectional bundle where the red lines and green lines are the signal nets connecting block A and block B. The arrows in the middle of the lines represent the signal direction. Red lines are nets from block A to block B and green lines are nets from block B to block A.

Figure 120 Bidirectional Bundle Example



When you plan for a bidirectional bundle,

- If you want to use a bidirectional bundle within topology interconnect planning flow, ensure that all the nets in this bundle are supernets (for example, when you run the `optimize_topology_plans`, `create_default_topology_plans`, `refine_topology_plans` commands on a bundle)
- Ensure that all the nets within the bundle connect to the same two ending blocks (the leaf driving block and the leaf load block)
- If there are additional blocks between the leaf driver and leaf load block, ensure that the nets follow the same order from the leaf driver block to the leaf load block, or from the leaf load block to the leaf driver block

To create a bidirectional bundle,

1. Create a supernet by using the `set_supernet_exceptions` and `create_supernet` commands.
2. Create a bundle with supernets (from step 1) of opposite signal directions by using the `create_bundle` command.

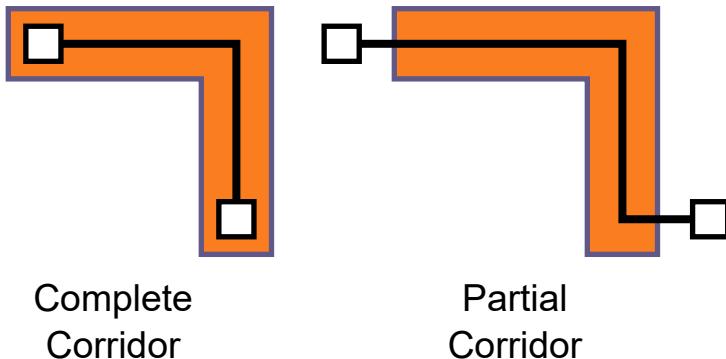
---

## Creating Routing Corridors

Routing corridors provide added control over the routing topology of individual nets or a bundle of nets. During global routing, the tool routes the associated nets through the routing corridors. Routing corridors are not exclusive; you can route other nets through the corridor, even if they are not assigned to the corridor. You can define a unique minimum and maximum layer for each corridor, or use the default minimum and maximum layers for the design. Note that during optimization, the tool does not legalize cells within a routing corridor.

The tool supports two types of routing corridors: complete and partial. A complete corridor is a corridor which encloses all the pins of the nets assigned to that corridor. A partial corridor is a corridor that has pins (of nets assigned to the corridor) that are physically placed outside of the corridor area. [Figure 121](#) shows a complete corridor and a partial corridor.

*Figure 121 Complete and Partial Routing Corridors*



To create and verify a routing corridor,

1. Use the `create_routing_corridor` to specify the routing corridor.

```
icc2_shell> create_routing_corridor -name corridor_a \
 -boundary {{1925 2170} {1980 3260}} \
 -min_layer_name METAL1 -max_layer_name METAL4 \
 -object [get_nets n1]
```

2. (Optional) Verify the routing corridor with the `report_routing_corridors` command.

```
icc2_shell> report_routing_corridors \
 [get_routing_corridors corridor_a]

Report : report_routing_corridors
Design : ORCA

CORRIDOR NAME: corridor_a
Shape Name min/max Shape

CORRIDOR_SHAPE_0 METAL1/METAL4 {1925 2170} {1980 3260}

Shapes Connected: yes
Shapes Cover Pins and Ports: no
Objects: n1

```

Alternatively, you can use the Create Routing Corridor tool in the GUI to create the corridor.

1. Select the nets to associate with the new routing corridor.
2. Select Create > Routing Corridor in the GUI.
3. Change any additional settings as needed, such as the minimum or maximum layer, the allowed drawing angles, snap settings, and so on.
4. Draw the routing corridor by clicking the mouse at the vertices of the corridor.

The IC Compiler II tool supports several commands to create, modify, and remove routing corridors. To add nets to a routing corridor, use the `add_to_routing_corridor` command. To extend the shape of the routing corridor, use the `create_routing_corridor_shape` command. To return a collection of routing corridors, use the `get_routing_corridors` command. To return a list of shapes that are contained in the routing corridor, use the `get_routing_corridor_shapes` command. To remove nets from a routing corridor, use the `remove_from_routing_corridor` command. To modify a routing corridor by removing a portion of the shape, use the `remove_routing_corridor_shapes` command.

## Creating Global Routes Within Routing Corridors

After creating routing corridors, you can use the routing corridors to guide global routing. To limit global routing to only specified nets in the design, use the `route_group` command with the `-nets` and `-global_planning true` options.

To create a routing corridor and to globally route a specific net through the corridor:

1. Use the `create_routing_corridor` command to create the corridor.

```
icc2_shell> create_routing_corridor -name corridor_a \
 -boundary {{1942 3219} {2026 2105}} \
 -object [get_nets n1]
```

2. If the routing corridor is a partial corridor, set the `route.global.connect_pins_outside_routing_corridor` application option to `true` to allow the global router to connect to pins outside the corridor. This step is not required for complete corridors which cover all pins on the route.

```
icc2_shell> set_app_options \
 -name route.global.connect_pins_outside_routing_corridor \
 -value true
route.global.connect_pins_outside_routing_corridor true
```

3. Globally route the net that is associated with the corridor by using the `route_group` command with the `-global_planning` option set to `true`.

```
icc2_shell> route_group -global_planning true -nets [get_nets \
-of_object [get_routing_corridors corridor_a]]
```

## Adding Repeater Cells

In the global planning flow, you can perform repeater insertion with the `add_buffer_on_route` command. Use this command after routing globally planned nets with the `route_group` command. Repeater insertion is done before pushing objects into blocks and creating pins on blocks.

```
icc2_shell> add_buffer_on_route \
-allow_insertion_over_cell [get_cells {i_block1 i_block2}] \
-repeater_distance 150 -lib_cell stdcell/buffer1 {net1 net2}
```

The `add_buffer_on_route` command automatically detects whether the buffer cell is an inverter or not, according to the specified library cell. The command respects placement blockages, soft macros, and hard macros by default, and allows you to add repeaters in specified soft macros or hard macros. You can use the `add_buffer_on_route` command on incomplete routes, and the command can be used during global route, track assignment and detail routing.

Options to the `add_buffer_on_route` command control how the buffers are inserted. Use the `-location`, `-user_specified_buffers`, `-repeater_distance`, or `-repeater_distance_length_ratio` options to control the positioning of buffers and the number of buffers to insert. Use the `-scaled_by_layer` and `-scaled_by_width` options to scale the distance between the buffers based on metal layer or based on the default width of the metal layer. Use the `-allow_insertion_over_cell` option to insert buffers on top of blocks. You can use the `push_down_objects` command later in the flow to push the buffers into blocks. The `add_buffer_on_route` command supports many more options; see the man page for more information.

Note that the command inserts buffers but does not legalize them. You must legalize the buffers with the `legalize_placement` command at the top and block design levels. In addition, you must use perform engineering change order (ECO) routing with the `route_eco` command to completely repair a net that has been modified by the `add_buffer_on_route` command.

## Pushing Down Repeater Cells

After inserting repeater cells in the global planning flow, you can push the repeater cells into the blocks they cover with the `push_down_objects` command. The command supports concurrent cell and routing push down.

To push down a net and create feedthroughs,

1. Set pin constraints to enable feedthroughs on the net.

```
icc2_shell> create_pin_constraint -type individual \
 -nets [get_nets n1] -allow_feedthroughs true
```

2. (Optional) If your design contains multiply instantiated blocks, check that they are correctly aligned with cell rows and wire tracks before pushing down objects.

```
icc2_shell> check_mib_alignment -cell_row -wire_tracks
----- Start Of Cell Row Misalignment Check -----
Misaligned cell_rows: 0, misaligned cell_sites: 0
----- End Of Cell Row Misalignment Check -----
----- Start Of Wire Track Misalignment Check -----
Warning: Misalignment at track TRACK_14 on layer METAL6 between
MIB instances I_ORCA_TOP/I_BLENDER_1 and
I_ORCA_TOP/I_BLENDER_6. Track offset of I_ORCA_TOP/I_BLENDER_1
is 0.205, and track offset of I_ORCA_TOP/I_BLENDER_6 is 0.41.
(MIB-203)
...
Misaligned wire_tracks: 6
----- End Of Wire Track Misalignment Check -----
```

3. Use the `push_down_objects` command to push down the net.

```
icc2_shell> push_down_objects [get_nets \
 -of_objects [get_routing_corridors corridor_a]]
```

To push down the repeater cells and the nets connected to those cells, use the `push_down_objects` command and specify the repeater cell instances.

```
icc2_shell> push_down_objects [get_cells eco_*]
```

4. Push down the repeater cells and the nets connected to those cells.

To do this, use the `push_down_objects` command and specify the repeater cell instances.

```
icc2_shell> push_down_objects [get_cells eco_*]
```

# 12

## Performing Clock Trunk Planning

---

To manage clock trunks during design planning, the tool supports clock trunk planning. A clock trunk is a coarse-grain physical distribution of the clock, which starts at the clock source and ends at the clock trunk endpoint. This capability allows you to

- Manage clock trunks earlier in the design cycle
- Provide physical guidance for clock tree implementation during clock tree synthesis
- Perform clock pin placement by using information about the clock tree
- Mitigate timing issues on the clock trunk

The result is a more robust clock topology for your design. This capability also enables you to reuse or repeat the clock topology even when there are minor modifications to the netlists.

[Figure 122](#) shows where the clock trunk planning fits in the overall design planning flow.

*Figure 122 Clock Trunk Planning in the Overall Design Planning Flow*

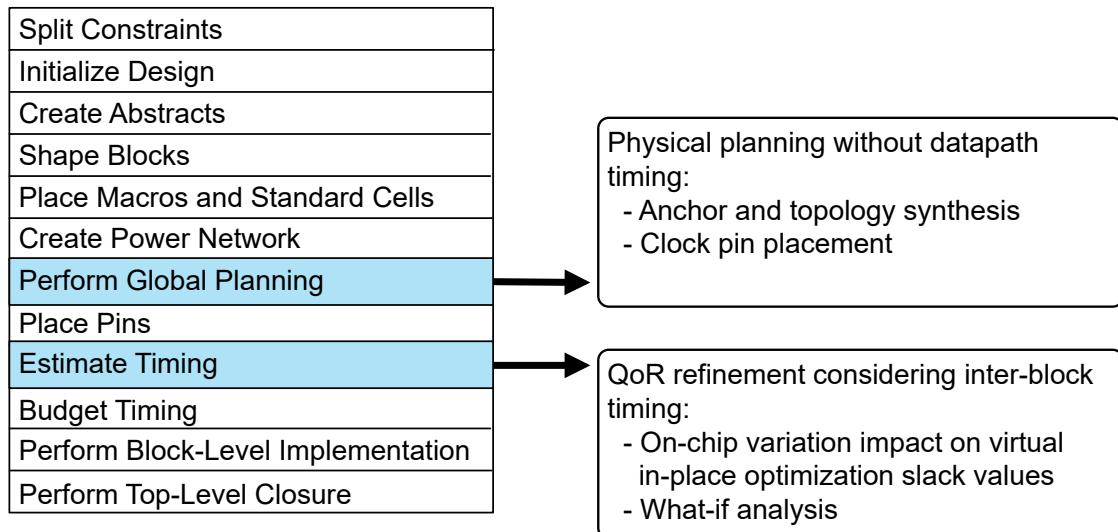


Figure 123 shows the clock trunk planning terminologies that you need to be familiar with.

Figure 123 Clock Trunk Planning Terminology

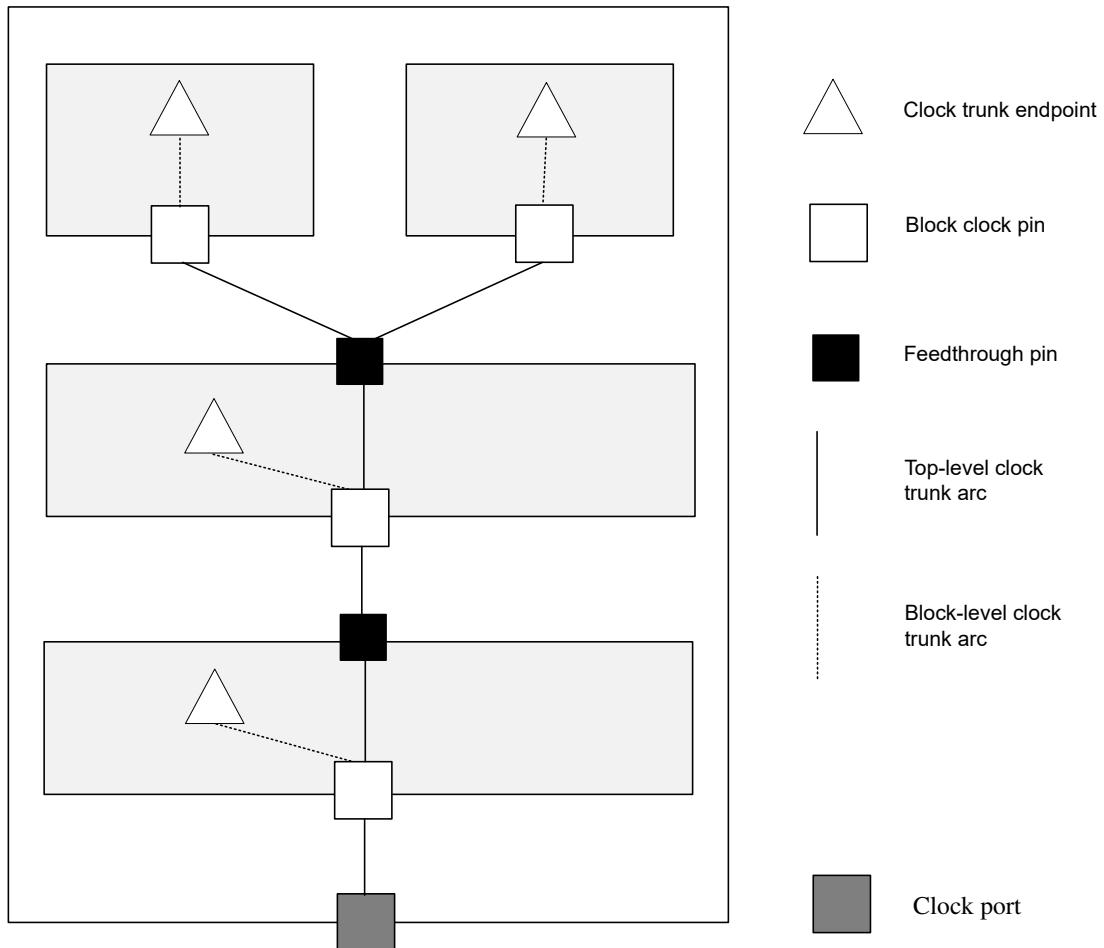
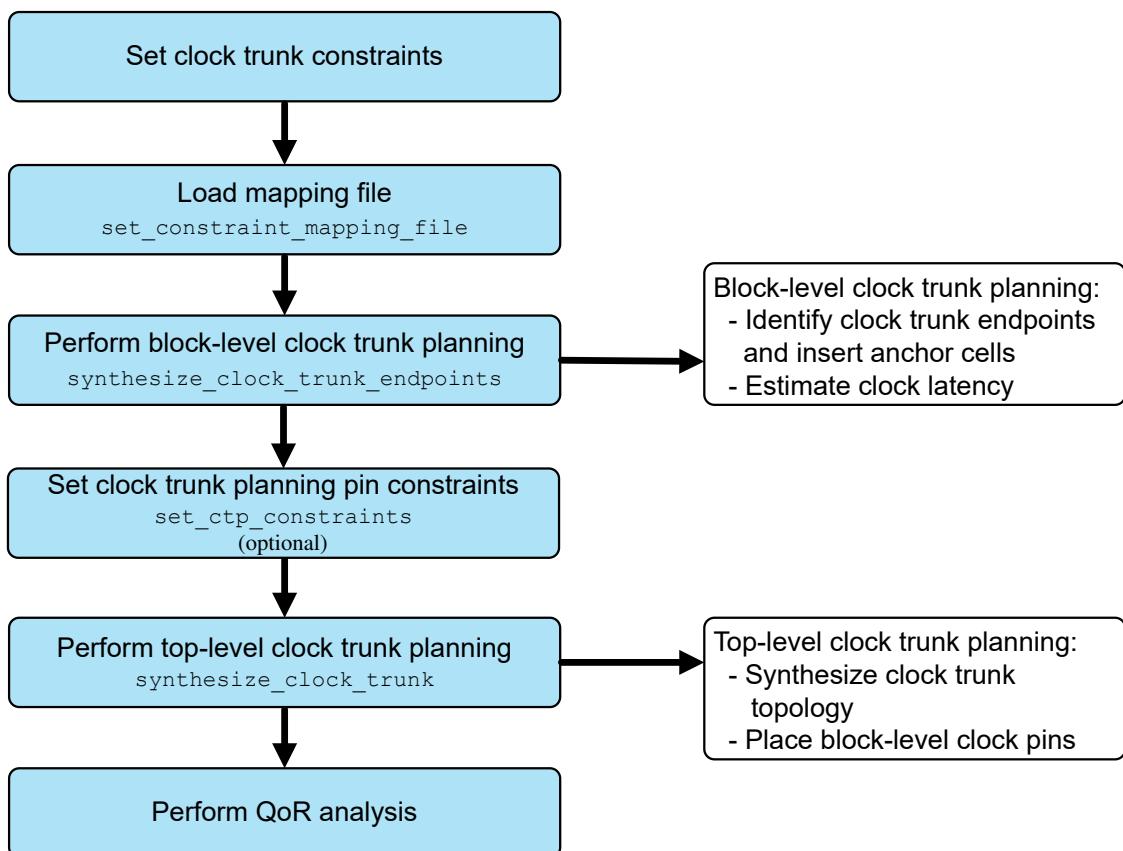


Figure 124 shows the clock trunk planning flow.

Figure 124 Clock Trunk Planning Flow



The following topics describe the steps in this flow:

- [Setting Clock Trunk Planning Constraints](#)
- [Performing Clock Trunk Planning for Multiply Instantiated Blocks](#)
- [Performing Block-Level Clock Trunk Planning](#)
- [Checking the Feasibility of Performing Clock Trunk Planning](#)
- [Performing Clock Trunk Planning for Hierarchical Designs With Abstracts](#)
- [Setting Clock Trunk Planning Pin Constraints](#)
- [Performing Top-Level Clock Trunk Planning](#)

- [Performing Top-Level Clock Trunk Planning Push-Down Flow](#)
- [Performing QoR Analysis](#)
- [Managing Timing Constraints for Clock Feedthrough Paths](#)
- [Performing Incremental Clock Trunk Planning](#)
- [Performing Clock Trunk Analysis in the GUI](#)

## Setting Clock Trunk Planning Constraints

Before you begin clock trunk planning, you must set constraints on the clock trunks, both at the block level and top level. Ensure that you set all the constraints including the clock tree synthesis constraints.

Constraints ensure that clock trunk planning produces realistic results that correlate with the actual clock tree implementation.

To set constraints,

1. Create the constraint files for block-level and top-level clock trunk planning. Constraints include the following settings:
  - Maximum capacitance set with the `set_max_capacitance` command
  - Maximum transition time set with the `set_max_transition` command
  - Routing layers set with the `set_ignored_layers` command
  - Clock tree references set with the `set_lib_cell_purpose` command
  - Nondefault routing rules
  - Driving cell set with the `set_driving_cell` command

Specify the clock tree synthesis constraints for both the block level and top level using the constraint mapping file. An example of the clock trunk mapping file, `ctp_mapfile`:

```
misc CTS_CONSTRAINT constraints/block_ctp.tcl
leon3s CTS_CONSTRAINT constraints/block_ctp.tcl
leon3s_1 CTS_CONSTRAINT constraints/block_ctp.tcl
leon3s_2 CTS_CONSTRAINT constraints/block_ctp.tcl
leon3s_3 CTS_CONSTRAINT constraints/block_ctp.tcl
leon3_mp CTS_CONSTRAINT constraints/top_ctp.tcl
```

For more information about constraints for clock tree synthesis, see the Clock Tree Synthesis topic in the *IC Compiler II Implementation User Guide*.

2. Load the constraint mapping file into the tool with the `set_constraint_mapping_file` command.

```
icc2_shell> set_constraint_mapping_file ./ctp_mapfile
1

icc2_shell> sh cat ./ctp_mapfile
misc CTS_CONSTRAINT constraints/block_ctp.tcl
leon3s CTS_CONSTRAINT constraints/block_ctp.tcl
leon3s_1 CTS_CONSTRAINT constraints/block_ctp.tcl
leon3s_2 CTS_CONSTRAINT constraints/block_ctp.tcl
leon3s_3 CTS_CONSTRAINT constraints/block_ctp.tcl
leon3_mp CTS_CONSTRAINT constraints/top_ctp.tcl
```

## Performing Clock Trunk Planning for Multiply Instantiated Blocks

In the clock trunk planning flow, multiply instantiated blocks (MIBs) are considered as read-only blocks. Therefore, if there are any MIBs in your design, you must place the clock pins of the MIBs manually before you start the clock trunk planning flow.

## Performing Block-Level Clock Trunk Planning

Block-level clock trunk planning performs clock latency estimation and inserts anchor cells. After creating the constraints for clock trunk planning, use the `synthesize_clock_trunk_endpoints` command to compute seed locations for the clock ports, create clock trunk endpoints, and add phase delay annotations to the endpoints. The command uses clock tree synthesis to measure the achievable latency for the clock and inserts anchor cells as needed based on the block size, number of clock sinks, driving cell specification, and clock tree synthesis constraints.

When you start the block-level clock trunk planning flow, if the block-level clock ports are already placed or if they are fixed, the block-level clock trunk planning flow skips finding the seed location of the block clock ports.

Block-level clock trunk planning supports multithreading.

To perform block-level clock trunk planning,

1. Check that the following conditions are met before you perform block-level clock trunk planning:
  - Blocks are shaped.
  - Top-level flip-flops are placed.
2. Run the `synthesize_clock_trunk_endpoints` command.

```
icc2_shell> synthesize_clock_trunk_endpoints \
 -blocks list_of_blocks \
 -clocks list_of_clocks \
```

**-host\_options host\_option\_name**

The following example generates clock trunk endpoints for the misc, leon3s, leon3s\_1, leon3s\_2, and leon3s\_3 blocks:

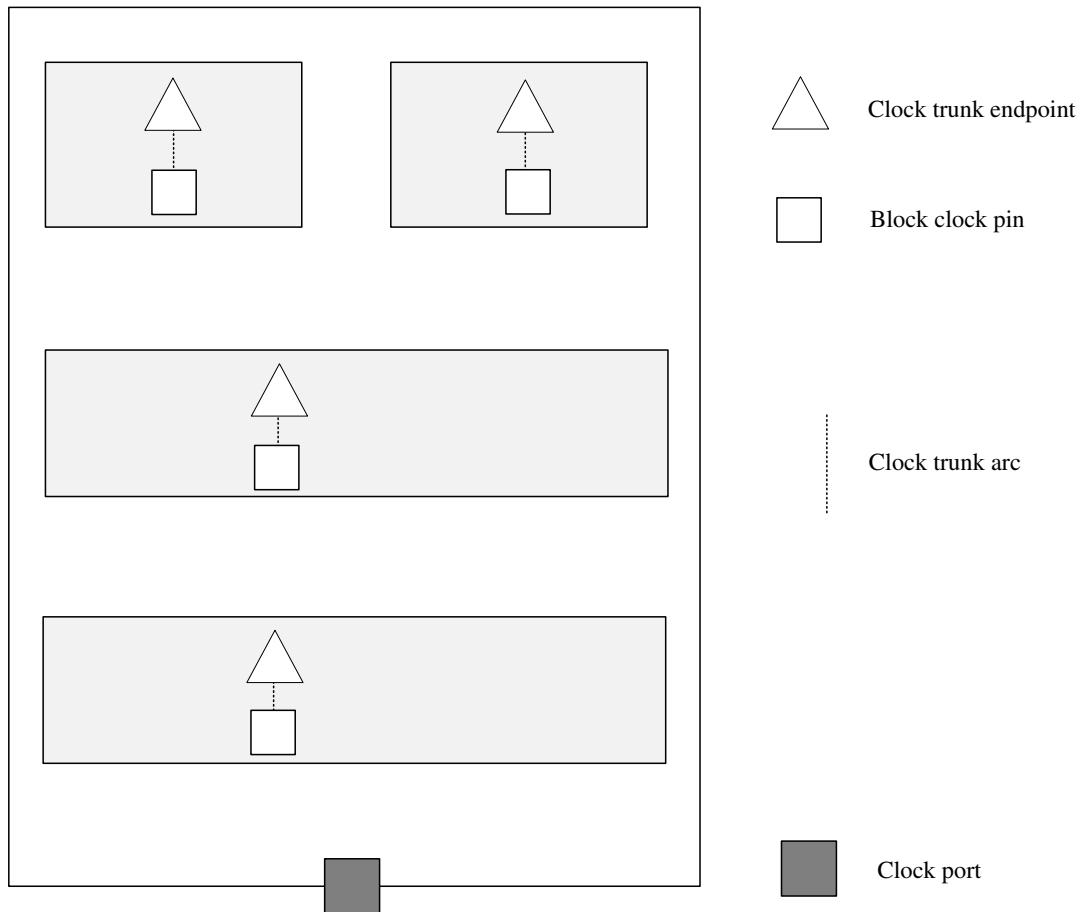
```
icc2_shell> synthesize_clock_trunk_endpoints \
 -blocks {misc leon3s leon3s_1 leon3s_2 leon3s_3} \
 -clocks list_of_clocks \
 -host_options block_script
...
Running Block misc ...
...
1
```

Alternatively, create the clock trunk endpoints manually by using the `set_clock_trunk_endpoints` command. This command is useful for creating clock trunk endpoints for black boxes.

```
icc2_shell> set_clock_trunk_endpoints u0_1/clk -delay 0.2
```

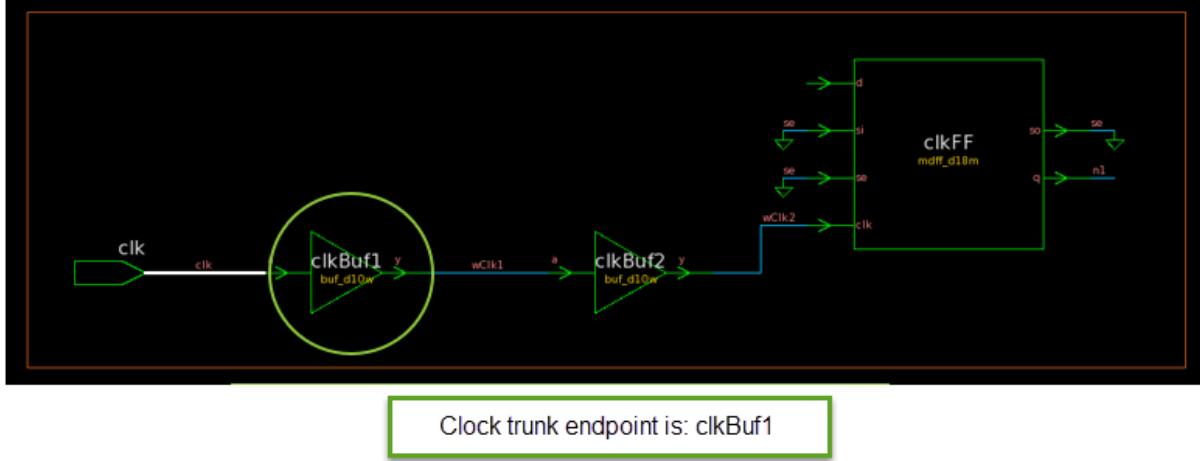
Figure 125 shows the results after performing block-level clock trunk planning.

Figure 125 Results After Performing Block-Level Clock Trunk Planning



The following figures show examples of clock trunk endpoints in various situations.

*Figure 126 Clock to Single Flip-Flop With Buffers*



*Figure 127 Clock to Multiple Flip-Flops With Buffers on Common Path*

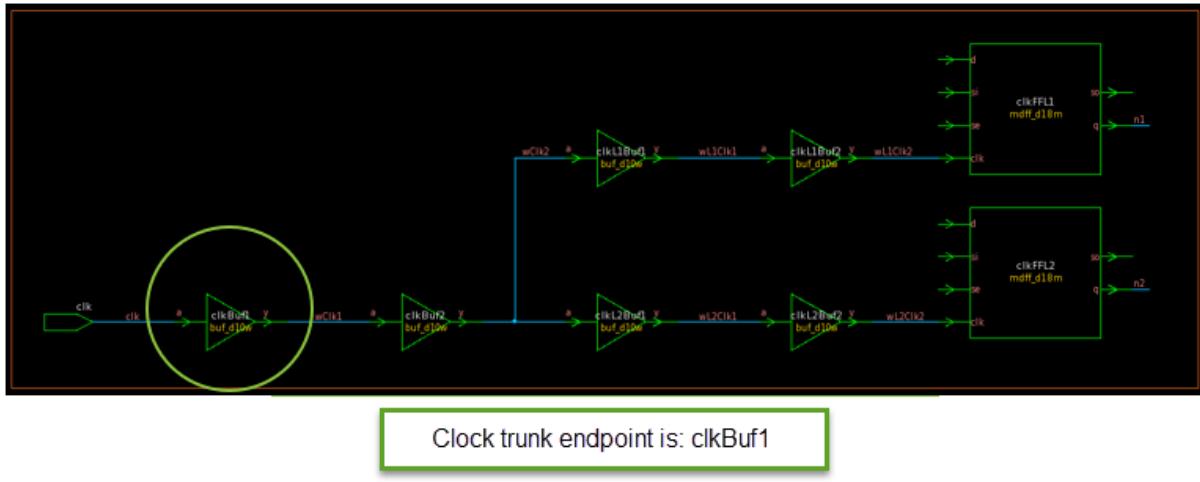


Figure 128 Clock to Multiple Flip-Flops With Buffers Only on Branch Path

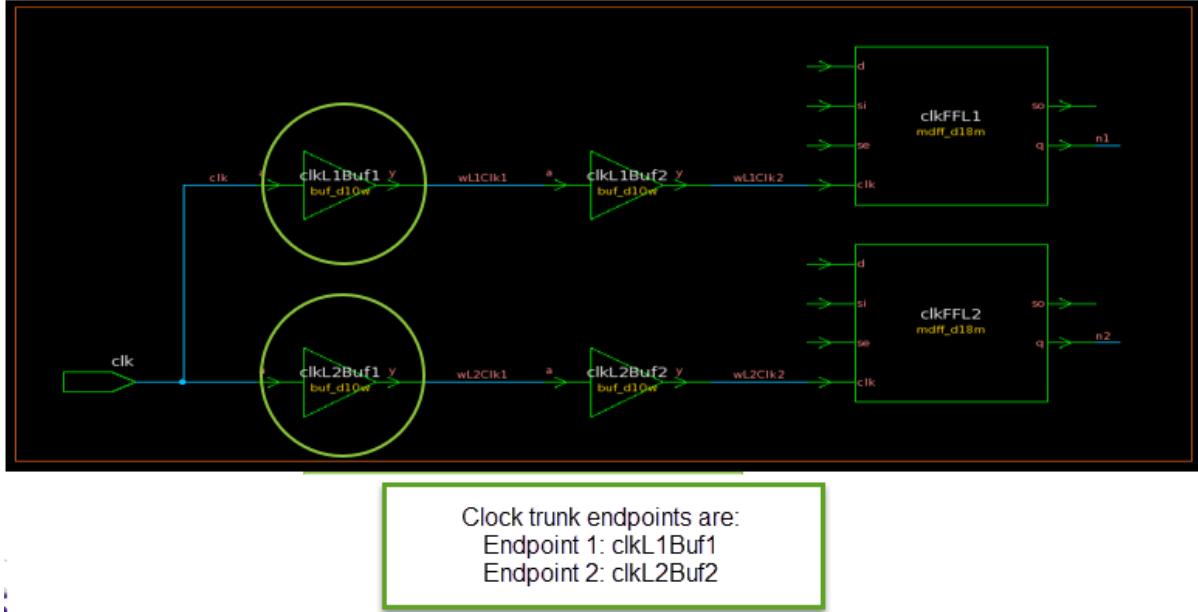


Figure 129 Clock to Flip-Flop With Buffers and Buffered Feedthrough

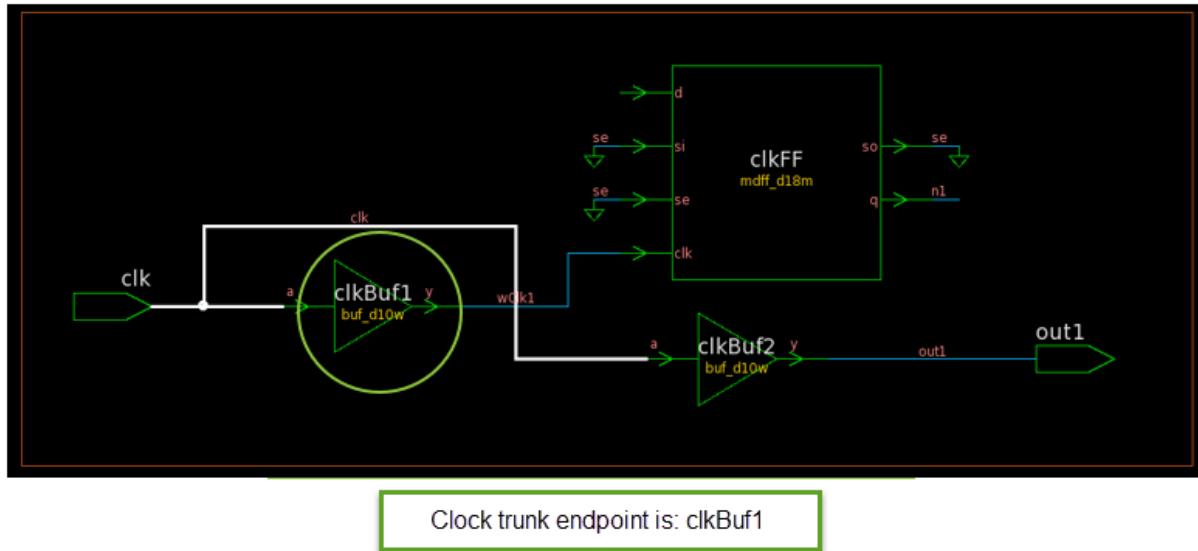


Figure 130 Clock to Flip-Flop With Buffers and Buffered Feedthrough With Balance Point

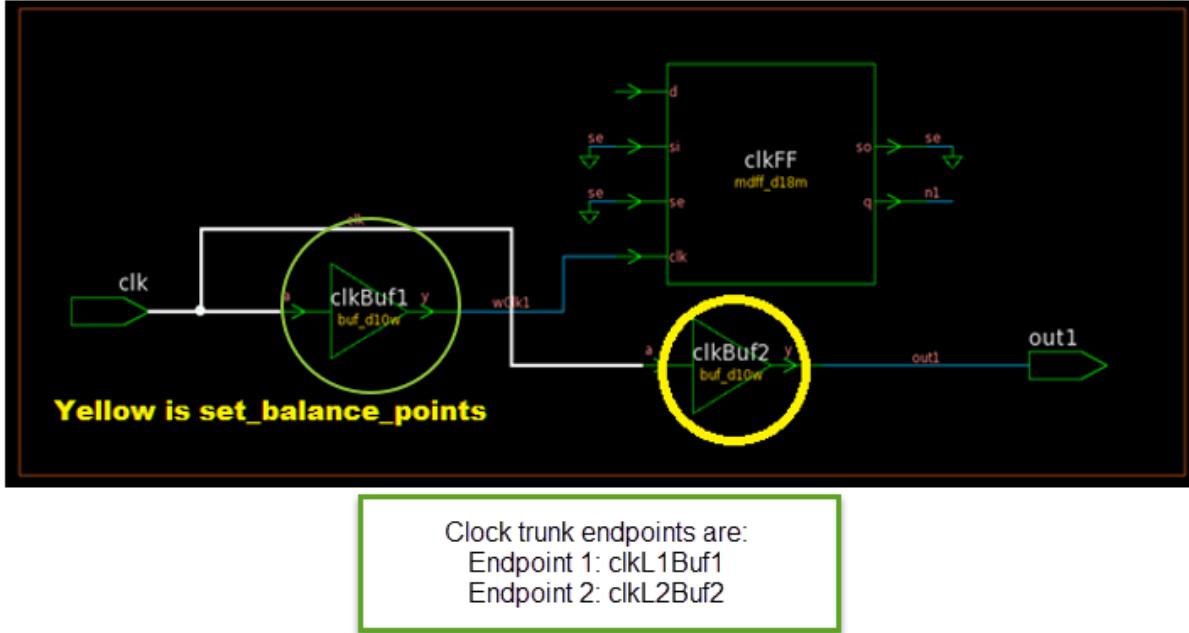


Figure 131 Buffered Feedthrough With Fixed Buffer

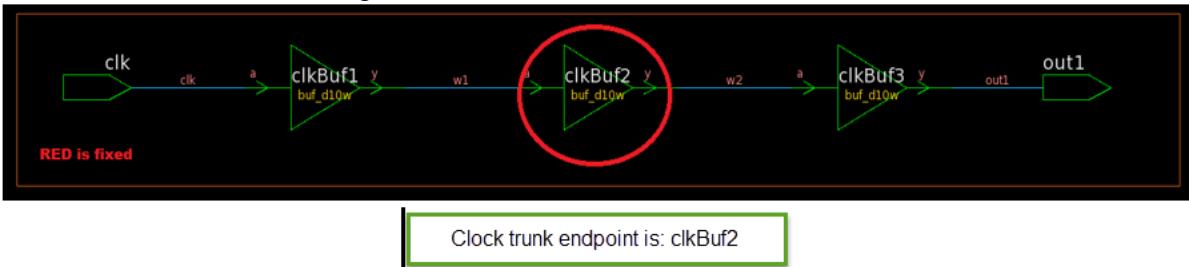
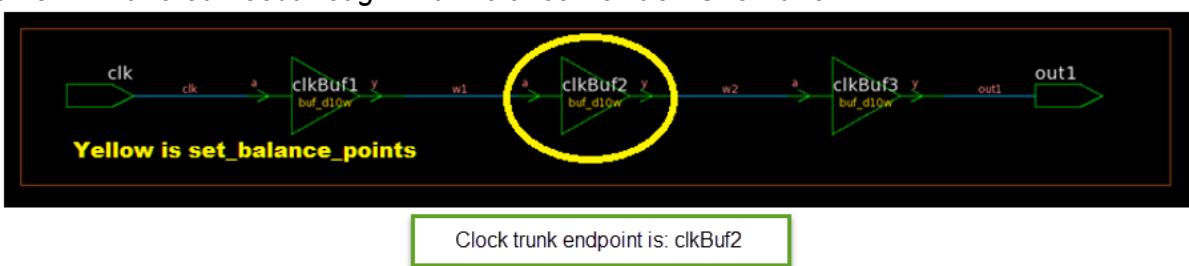


Figure 132 Buffered Feedthrough With Balance Point on One Buffer



To report clock trunk endpoints, use the `report_clock_trunk_endpoints` command. The command reports the location of the clock trunk endpoints set by the

`synthesize_clock_trunk_endpoints` and `set_clock_trunk_endpoints` commands as follows:

```
icc2_shell> report_clock_trunk_endpoints

Report : report_clock_trunk_endpoints
Design : leon3mp
Version:
Date :

Endpoint Phase Delay Dom Pt Clock Corner Block

u0_1/cts_inv_34563455/A
0.52 pci_clk RCworst u0_1
u0_1/cts_inv_34423441/A
0.30 pci_clk RCworst u0_1
U2/u0_0/cts_inv_34533452/A
0.51 pci_clk RCworst U2/u0_0
...
...
```

Alternatively, use the `report_clock_trunk_endpoints -script` command to write out the clock trunk endpoints as a series of `set_clock_trunk_endpoints` commands as follows:

```
icc2_shell> report_clock_trunk_endpoints -script
set_clock_trunk_endpoints constraints for mode func
set_clock_trunk_endpoints cts_inv_34423441/A -clock pci_clk
-corners RCworst -delay 0.296
set_clock_trunk_endpoints cts_inv_34423441/A -clock pci_clk
-corners estimated_corner -delay 0.296
...
...
```

To remove clock trunk endpoints, use the `remove_clock_trunk_endpoints` command.

```
icc2_shell> remove_clock_trunk_endpoints u0_1/clk
```

## Checking the Feasibility of Performing Clock Trunk Planning

To check the feasibility of performing clock trunk planning, use the `check_design_for_clock_trunk_planning` command. Run this command after you specify the clock trunk planning setting, but before you perform clock trunk planning by running the `synthesize_clock_trunks` command.

Based on the type of flow that you specified for the clock trunk planning, generic or push down, and the clock you specify with the `-clock` option, this command identifies the issues you would encounter during clock trunk planning. The `-summary` option provides the summary of the clock trunk planning checks that the command performs, which are shown in the following table.

**Table 17** shows the sanity checks that the command performs:

**Table 17** *Clock Trunk Planning Sanity Checks*

| Check Name                                         | Message                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SUB_BLOCK_INST_WITH_I<br>NCORRECT_TYPE_OR_VI<br>EW | Error: Block instance %s (Reference: %s) is bound to Frame view. Cannot be used for top-level optimization. (TL-112)<br><br>Error: Block instance %s (Reference: %s) has design type macro. Cannot be used for top-level optimization. (TL-134)                                                                                                                                                                                                                                                                                                                                 |
| DESIGN_VIEW_CHECK                                  | Error: Design view for the Abstract '%s' does not exist. (TL-101)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| DESIGN_MISMATCH_CHE<br>CKS                         | Error: Missing port in block %s.%s corresponding to block instance pin %s. (TL-109)<br><br>Error: Missing pin on block instance %s (Reference: %s) corresponding to port %s. (TL-110)                                                                                                                                                                                                                                                                                                                                                                                           |
| CORE_AREA_CHECK                                    | Error: Missing core area in block %s.%s. Cannot run top-level timing analysis and optimization. (TL-111)                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| LOC_SUB_BLOCK_CHECK                                | Error: Missing location of block instance %s (Reference: %s) in design %s.%s. (TL-130)<br><br>Error: Block instance '%s' (Reference: %s) boundary is outside its parent block '%s' boundary. (TL-131)                                                                                                                                                                                                                                                                                                                                                                           |
| LOC_SUB_BLOCK_PIN_CH<br>ECK                        | Error: Missing physical location for port %s of block reference %s.%s. (TL-126)<br><br>Note: Only for clock ports.                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| LOC_LEAF_INST_CHECK                                | Error: Missing location of leaf cells %s inside block reference %s.%s. (TL-128)<br><br>Error: Leaf cell %s inside block reference %s.%s is outside block boundary. (TL-129)<br><br>Note: For all Cells.                                                                                                                                                                                                                                                                                                                                                                         |
| VOLTAGE_AREA_CHECKS                                | Error: Internal Consistency Failure. Internal voltage area %s does not have a voltage area. (TL-114)<br><br>Error: Internal Consistency Failure. Voltage area %s does not have associated power domain. (TL-115)<br><br>Error: Internal Consistency Failure. Internal voltage area %s does not have associated power domain. (TL-116)<br><br>Error: Internal Consistency Failure. Internal voltage area %s does not have associated voltage area region. (TL-117)<br><br>Error: Internal Consistency Failure. Voltage area %s does not have any internal voltage area. (TL-118) |

**Table 17      Clock Trunk Planning Sanity Checks (Continued)**

| Check Name             | Message                                                                                                                                   |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| POWER_DOMAIN_CHECK_S   | Error: Power domain %s does not have associated voltage area. (TL-113)                                                                    |
| UNMAPPED_LOGIC_CHECK_K | Error: The leaf cell %s in the block reference %s is unmapped. (TL-305)<br><br>Note: For cells on clock path.                             |
| CLOCK_LOOP_CHECK       | Error: Output pin '%s' is driving input pin '%s' of the same physical hierarchy instance '%s'. (TL-140)<br><br>Note: Only for clock path. |

## Performing Clock Trunk Planning for Hierarchical Designs With Abstracts

The top-level clock trunk planning flow supports abstracts, which are light-weight representations of the blocks. The steps are as follows:

1. Create abstracts for the blocks before block-level clock trunk planning by using the following commands:

```
icc2_shell> set_editability -value true -blocks *
icc2_shell> create_abstract -blocks list_of_blocks
```

2. Prepare the hierarchical design for clock trunk planning by using the following command:

```
icc2_shell> synthesize_clock_trunk_setup_hier_context -init \
-host_options host_option_name
```

3. Synthesize the clock trunks by using the following command:

```
icc2_shell> synthesize_clock_trunks -clock \
$CTP_CLOCKS -from from_step -to to_step
```

4. Update the hierarchical design after clock trunk planning by using the following command:

```
icc2_shell> synthesize_clock_trunk_setup_hier_context -commit \
-host_options host_option_name
```

## Setting Clock Trunk Planning Pin Constraints

Setting clock tree planning pin constraints enables you to partially block and unblock the sides of design blocks. After performing block-level clock trunk planning, use the `set_ctp_constraints` command to set these clock trunk planning pin constraints. This step is optional.

The `place_pins` command honors the constraints set using the `set_ctp_constraints` command and ensures that no pins are placed on the edges that are blocked or any feedthroughs are cut on these edges.

The `set_ctp_constraints` command is additive, which means that the constraint values set by the previous `set_ctp_constraints` commands are retained unless they are explicitly overridden by subsequent calls.

The constraints set using this command takes priority over the constraints set using the `set_block_pin_constraints` or `set_individual_pin_constraints` command.

To enable or disable the constraints that were set using the `set_ctp_constraints` command on specific blocks, use the `set_editability` command. The clock trunk planning engine ignores the blocks that were disabled and does not enforce the constraints on these blocks.

To place pins on specific sides of the blocks, use the `-sides` option. To disallow placing of pins on specific sides, use the `-exclude_sides` option. Both of these options accept a positive integer that represent the sides of a block. For a block, the left-most lowest edge takes the value of 1 and as you move in a clockwise direction, increment this number by one. For example, `-sides {1 2 4}` allows pins to be placed on edges 1, 2, and 4. However, `-exclude_sides {2 3 5}` disables pins being placed on edges 2, 3, and 5. These two options are mutually exclusive. 0 is not a valid value.

To place pins on a specific portion of an edge or edges, use the `-offset` option along with the `-sides` option. To exclude a specific portion, use the `-offset` option along with the `-exclude_sides` option.

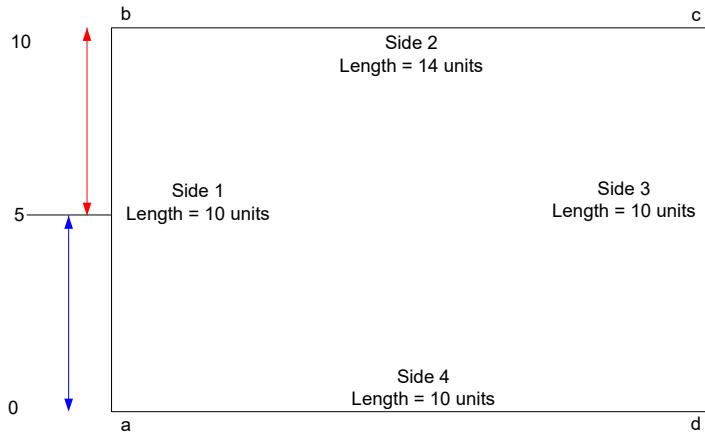
To specify the metal layers on which you want to allow pin placement, use the `-allowed_layers` option.

To allow or disallow feedthrough creation on blocks, use the `-allow_feedthroughs` option. By default, this option is set to true.

Example 1: Use the following command to allow pin placement on side 1 within the offset 0 to 5.

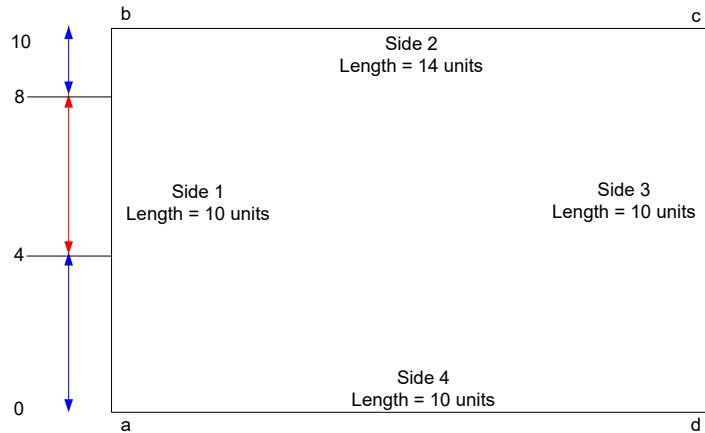
```
icc2_shell> set_ctp_constraints ABC_cell_name -sides 1 -offset {0 5}
```

Chapter 12: Performing Clock Trunk Planning  
Setting Clock Trunk Planning Pin Constraints



Example 2: Use the following command to disallow pin placement on side 1 within the offset 4 to 8.

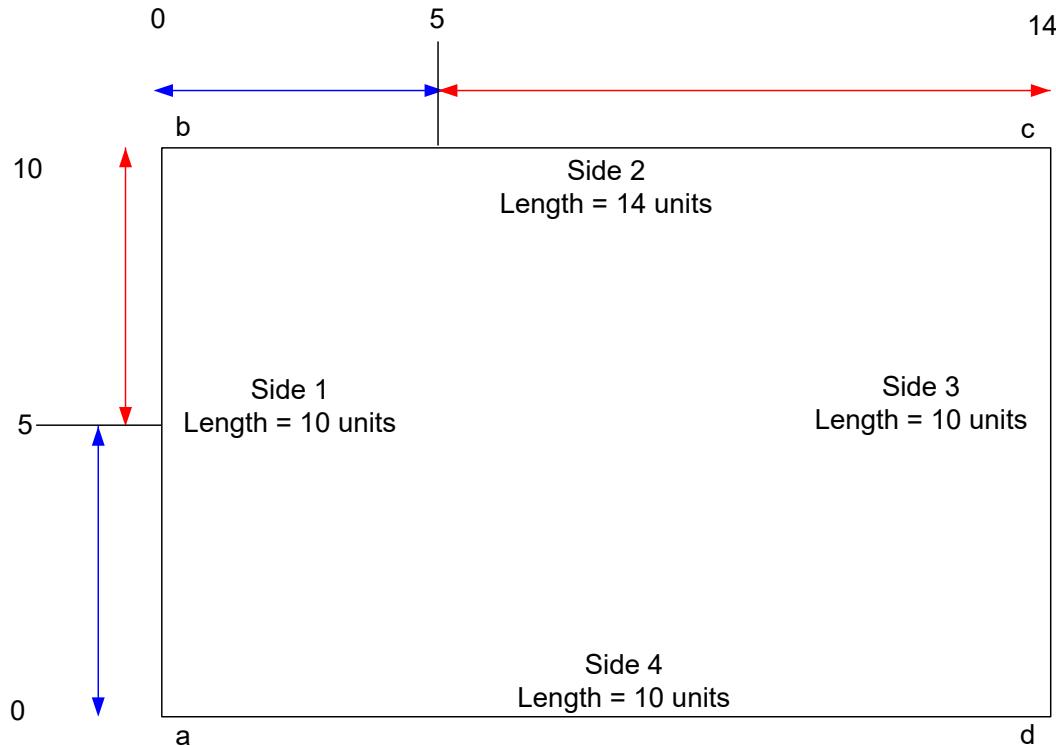
```
icc2_shell> set_ctp_constraints ABC_cell_name -exclude_sides 1
 -offset {4 8}
```



Example 3: Use the following commands to allow pin placement on sides 1 and 2 within the offset 0 to 5.

```
icc2_shell> set_editability
icc2_shell> set_ctp_constraints ABC_cell_name -sides {1 2} -offset {0
 5 0 5}
(or)
```

```
icc2_shell> set_ctp_constraints ABC_cell_name -sides {1 2}
 -offset {{0 5} {0 5}}
```



## Removing Clock Tree Planning Pin Constraints

To remove the clock tree planning pin constraints on specific blocks, use the `remove_ctp_constraints` command.

This command removes the constraint set by the `set_ctp_constraints` command. All other pin constraints set using block/individual pin constraint remain as is.

The `remove_ctp_constraints` command has no effect on the blocks that are disabled using the `set_editability` command.

The following command removes the clock tree planning pin constraints from all the blocks in your current design.

```
icc2_shell> remove_ctp_constraints
```

## Reporting Clock Tree Planning Pin Constraints

To report the clock tree planning pin constraints defined on specific blocks, use the `report_ctp_constraints` command. The following command reports the clock tree planning pin constraints on all the blocks in your current design.

```
icc2_shell> report_ctp_constraints
```

## Performing Top-Level Clock Trunk Planning

During top-level clock trunk planning, the tool synthesizes a physical topology for the clock trunk at the top level by using clock tree synthesis. This step compensates for the phase delays in the block-level clock trunk endpoints and tries to maximize the common clock path between blocks that communicate with each other. This flow supports,

- Multiple levels of physical hierarchy
- Multivoltage
- Abutted design
- Nonabutted design (channel design)

To perform top-level clock trunk planning,

1. Check that the following conditions are met:

- Blocks are shaped.
- Block clock ports are placed.
- Top-level flip-flops are placed.
- The clock trunk endpoints are created by running the `synthesize_clock_trunk_endpoints` or `set_clock_trunk_endpoints` command.

Verify the clock trunk endpoint by using the `report_clock_trunk_endpoints` command.

2. Control the hierarchical optimization of the lower-level blocks by specify the following application option settings:

- `icc2_shell> set_app_options -name plan.clock_trunk.flow_control \
-value optimize_subblocks`
- `icc2_shell> set_app_options -name plan.clock_trunk.enable_new_flow \
-value true`

3. For multiple levels of physical hierarchy designs, enable generation of top-level pin constraints, by using the following command:

```
icc2_shell> set_app_options -name
 plan.clock_trunk.enable_mph_constraints \
 -value true
```

4. Enable feedthrough cells on clock nets by using the following application option settings:

```
icc2_shell> set_app_options \
 -name plan.pins.exclude_clocks_from_feedthroughs -value false
```

If this application option is set to true, the tool does not create feedthroughs on clock nets.

5. Remove the following pin constraints already defined on the clock nets:

- Topological pin feedthrough constraints, which are defined with the `create_topological_constraint` command, by using the following command:

```
icc2_shell> remove_topological_constraints -all
```

- Individual pin constraints, which are defined with the `set_individual_pin_constraints` command, by using the following command:

```
icc2_shell> remove_individual_pin_constraints
```

During clock trunk planning, the tool honors the following constraints when determining the location of block pins and feedthrough pins:

- Feedthroughs allowed in physical block by using the `set_block_pin_constraints -allow_feedthroughs true -blocks` command
- Block editability setting specified with the `set_editability -value true -blocks` command
- Sides included and excluded for pin creation by using the `set_block_pin_constraints -sides -blocks` command
- Routing blockages
- Clock metal layer assignments

6. (Optional) Set the `cts.common.user_instance_name_prefix` application option to identify the buffers or inverters added by the top-level clock trunk planning.

```
icc2_shell> set_app_options \
 -name cts.common.user_instance_name_prefix -value ICC2_TOP_CTP_
```

7. Check the feasibility of performing clock trunk planning as described in [Checking the Feasibility of Performing Clock Trunk Planning](#).

8. Synthesize the top-level clock trunk by using the `synthesize_clock_trunks` command.

This command consists of the following two stages:

- a. `pin_constr_generation`, during which the tool generates a topological pin constraint file

By default, the constraint file is named `read_pin_constr_place_pins.tcl`. However, you can change this name by using the `plan.clock_trunk.topo_constr_pin_place_script_name` application option.

- b. `read_pin_constr_and_place_pins`, during which the tool reads the topological pin constraint file generated in the previous step and places all pins of all the synthesized clock nets

By default, the tool runs both stages of the `synthesize_clock_trunks` command. To run only a specific stage, use the `-from` and `-to` options. The following example runs only the `pin_constr_generation` stage:

```
icc2_shell> synthesize_clock_trunks \
-to pin_constr_generation
```

9. Analyze the QoR as described in [Performing QoR Analysis](#).

## Performing Top-Level Clock Trunk Planning Push-Down Flow

You can also use the traditional push-down flow for top-level clock trunk planning. This flow supports abstracts, abutted, and nonabutted designs; however, it does not support multiple levels of physical hierarchy or multivoltage designs. This flow treats blocks as read-only and does not take into consideration move bounds and blockages inside first-level blocks.

To use the push-down flow,

1. Set up the type of flow by using the following command:

```
icc2_shell> set_app_options -name plan.clock_trunk.flow_control \
-value pushdown
```

2. Check that the following conditions are met:

- Blocks are shaped.
- Block clock ports are placed.
- Top-level flip-flops are placed.

- The clock trunk endpoints are created by running the `synthesize_clock_trunk_endpoints` or `set_clock_trunk_endpoints` command.

Verify the clock trunk endpoint by using the `report_clock_trunk_endpoints` command.

- Check the feasibility of performing clock trunk planning as described in [Checking the Feasibility of Performing Clock Trunk Planning](#).

- Synthesize the top-level clock trunk with the `synthesize_clock_trunks` command.

```
icc2_shell> synthesize_clock_trunks
```

- Push down the top-level clock trunk into the blocks with the `push_down_clock_trunks` command.

```
icc2_shell> push_down_clock_trunks
```

This command also runs the `place_pins` command on the top-level clock nets while honoring pin constraints. The final clock pin placement is guided by the clock trunk cell placement, and pushed down cells are removed.

Analyze the QoR as described in [Performing QoR Analysis](#).

## Performing QoR Analysis

To perform QoR analysis,

- Ensure that the timing derate values are applied
- Estimate and back-annotate virtually-optimized delay information by using the `estimate_timing` command
- Analyze the timing of the clock trunk by using the `report_clock_trunk_qor` command, which analyzes inter block relationships using the estimated on-chip variation (OCV) and clock reconvergence pessimism (CRP) values, as shown in the following example report.

*Figure 133 Clock Trunk QoR Report*

| Clock Timing                             | Trunk End Point<br>(Launch)       | Trunk End Point<br>(Capture) | Divergence<br>Point      | WNS<br>(Ideal Clock) | OCV          | CRP          | WNS    | TNS    | Number<br>of Paths | Violating<br>Paths | Timing<br>Start Point |
|------------------------------------------|-----------------------------------|------------------------------|--------------------------|----------------------|--------------|--------------|--------|--------|--------------------|--------------------|-----------------------|
| <hr/>                                    |                                   |                              |                          |                      |              |              |        |        |                    |                    |                       |
| clk                                      | cc/cts_inv_34163415/A             | ne/cts_inv_27812780/A        | cc/ctbuf_net_87_PUSHDOWN | -0.089               | <b>0.033</b> | <b>0.002</b> | -0.120 | -0.517 | 16                 | 16                 |                       |
| cc/se/seo2/jbr5/reg07/CLK [CC/placement] | ne/sw/jbr1/reg07/D [NE/placement] |                              |                          |                      |              |              |        |        |                    |                    |                       |
| clk                                      | cc/cts_inv_34163415/A             | se/cts_inv_28112810/A        | cts_inv_1516115160/Y     | -0.089               | <b>0.031</b> | <b>0.003</b> | -0.117 | -0.721 | 16                 | 16                 |                       |
| cc/ne/neo1/jbr5/reg06/CLK [CC/placement] | se/mw/jbr1/reg06/D [SE/placement] |                              |                          |                      |              |              |        |        |                    |                    |                       |
| clk                                      | ne/cts_inv_27812780/A             | se/cts_inv_28112810/A        | cc/ctbuf_net_87_PUSHDOWN | -0.053               | <b>0.033</b> | <b>0.002</b> | -0.084 | -0.331 | 16                 | 16                 |                       |
| ne/se/seo2/jbr5/CLK [NE/placement]       | se/sw/jbr1/reg05/D [SE/placement] |                              |                          |                      |              |              |        |        |                    |                    |                       |

To consider clock trunk planning during budgeting and analyze the clock reconvergence pessimism (CRP) effects,

1. Enable budgeting to use the clock trunk planning results by using the `compute_budget_constraints` command.

```
icc2_shell> compute_budget_constraints -latency_targets estimated \
 -balance true
```

2. Write out the tool-derived budgeting constraints by using the `write_script` command.

```
icc2_shell> write_script -include budget -output budget_constraints
```

3. Check for the `-crp` option in the output file, as shown in the following example output:

```
set_latency_budget_constraints -default -crp 0.00205341 -from_clock
 clk:nw/clk -to_clock clk:cc/clk
set_latency_budget_constraints -corner default -crp 0.00205341
 -from_clock clk:nw/clk -to_clock clk:cc/clk
set_latency_budget_constraints -default -crp 0.00252273 -from_clock
 clk:sw/clk -to_clock clk:cc/clk
```

## Managing Timing Constraints for Clock Feedthrough Paths

Top-level clock trunk planning can create new clock feedthrough input ports for the blocks. To be able to inherit clock tree synthesis constraints, nondefault routing rules, and so on, the clock feedthrough paths need to be treated as clocks. Therefore, clocks must be defined for each newly created clock feedthrough ports. You can achieve this by running the `split_constraints` command with the following application option settings:

1. `icc2_shell> set_app_options \
 -name plan.budget.add_missing_feedthrough_clocks -value true`
2. `icc2_shell> set_app_options -name \
 plan.budget.include_feedthrough_clocks \
 -value true`
3. `icc2_shell> split_constraints`

The preceding steps add the clock information for each newly created clock input port. For more information about the `split_constraints` command, see the [Splitting Constraints](#) topic.

## Performing Incremental Clock Trunk Planning

You can perform incremental clock trunk planning multiple times on your design to achieve the best clock pin placement based on the comparison of outputs from the various incremental clock trunk planning cycles.

To run incremental clock trunk planning, use the `synthesize_clock_trunks -incremental` command. You can run incremental clock trunk planning on all clocks in the design or on a subset of clocks, on which clock trunk planning has already been performed. Both the `pin_constr_generation` and `read_pin_constr_and_place_pins` steps are supported with incremental clock trunk planning. The command uses the constraints that are already defined and generates a modified pin topology file.

### Example 1

In the following example, incremental clock trunk planning honors the constraints that are set in step 2.

1. `icc2_shell> synthesize_clock_trunks -from pin_constr_generation \ -to pin_constr_generation -clock [get_clocks {clk1 clk2}]`
2. Modify clock trunk planning block and pin constraints.
3. `icc2_shell> synthesize_clock_trunks -from pin_constr_generation \ -to pin_constr_generation -clock [get_clocks {clk1 clk2}] -incremental`

### Example 2

In the following example, incremental clock trunk planning keeps the pin constraints for clk2 same as step 1, whereas for clk1, it overwrites the pin constraints based on constraints set at step2.

1. `icc2_shell> synthesize_clock_trunks -from pin_constr_generation \ -to pin_constr_generation -clock [get_clocks {clk1 clk2}]`
2. Modify clock trunk planning block and pin constraints.
3. `icc2_shell> synthesize_clock_trunks -from pin_constr_generation \ -to pin_constr_generation -clock {clk1} -incremental`

### Example 3

In the following example, incremental clock trunk planning writes the pin constraints file and pin placement only for clk1 (that is, for the nets specified by using the `-to read_pin_constr_and_place_pins` option).

1. `icc2_shell> synthesize_clock_trunks -from pin_constr_generation \ -to pin_constr_generation -clock [get_clocks {clk1 clk2}]`

2. Modify clock trunk planning block and pin constraints.
3. `icc2_shell> synthesize_clock_trunks \ -to read_pin_constr_and_place_pins -clock {clk1} -incremental`

#### Example 4

In the following example, in step 4, the tool places the pins based on constraints from step 3 for clk1 and from step 1 for clk2.

1. `icc2_shell> synthesize_clock_trunks -from pin_constr_generation \ -to pin_constr_generation -clock [get_clocks {clk1 clk2}]`
2. Modify clock trunk planning block and pin constraints.
3. `icc2_shell> synthesize_clock_trunks -from pin_constr_generation \ -to pin_constr_generation -clock {clk1} -incremental`
4. `icc2_shell> synthesize_clock_trunks \ -from read_pin_constr_and_place_pins \ -to read_pin_constr_and_place_pins -clock [get_clocks {clk1 clk2}]`

## Performing Clock Trunk Analysis in the GUI

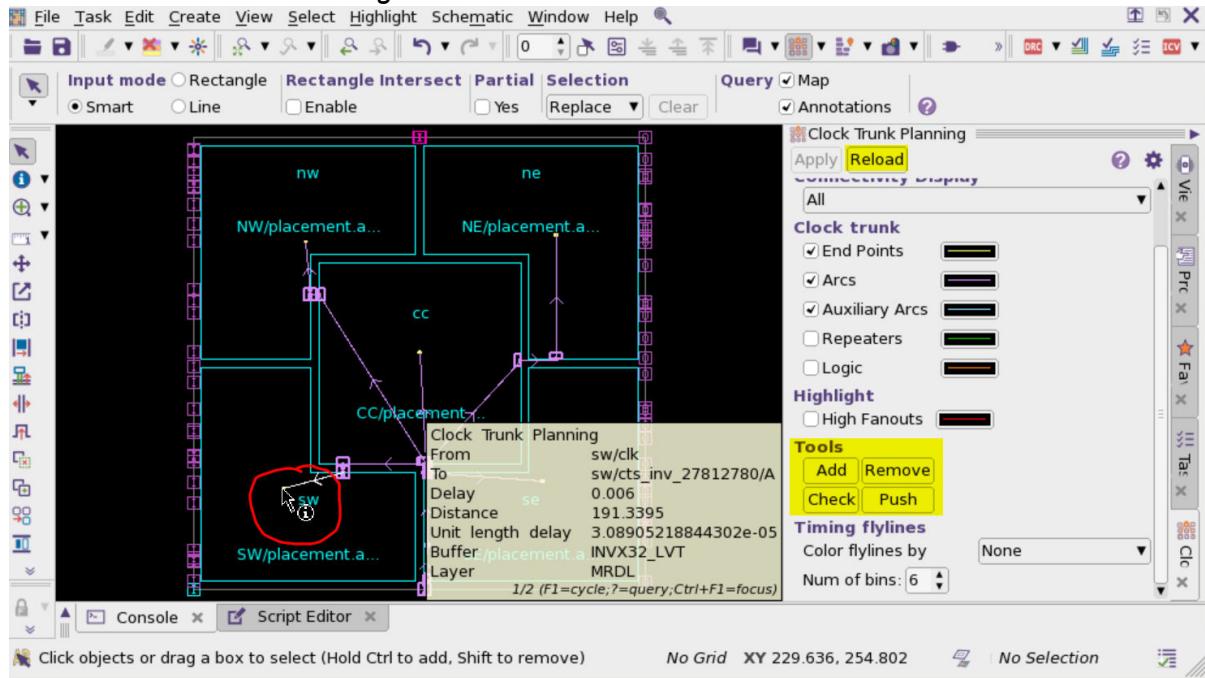
The GUI supports the Clock Trunk Planning panel to load clock trunk information from the design. Use this tool to view clock trunks, identify clock trunk endpoints, insert clock buffers, remove clock buffers, and perform other tasks.

To begin clock trunk analysis in the GUI,

1. Choose View > Flylines > Clock Trunk Planning from the menu.
2. Click the Reload button.
3. Select Automatic or Specified in the Reload dialog box and click OK to load data for all clocks or only for the clocks you specify.
4. In the Clock Trunk Planning panel, select the name of the clock to analyze in the Clock list box.
5. (Optional) Modify the clock trunk by using the Add, Remove, Check, or Push button in the Tools section of the panel.

When you select and apply the options in the Clock Trunk panel, the layout window updates to display the selected data. To obtain more information about a clock trunk, hover the pointer over the path to show the clock name, startpoint, endpoint, and other information in an InfoTip.

Figure 134 Clock Trunk Planning Panel



# 13

## Performing Pin Assignment

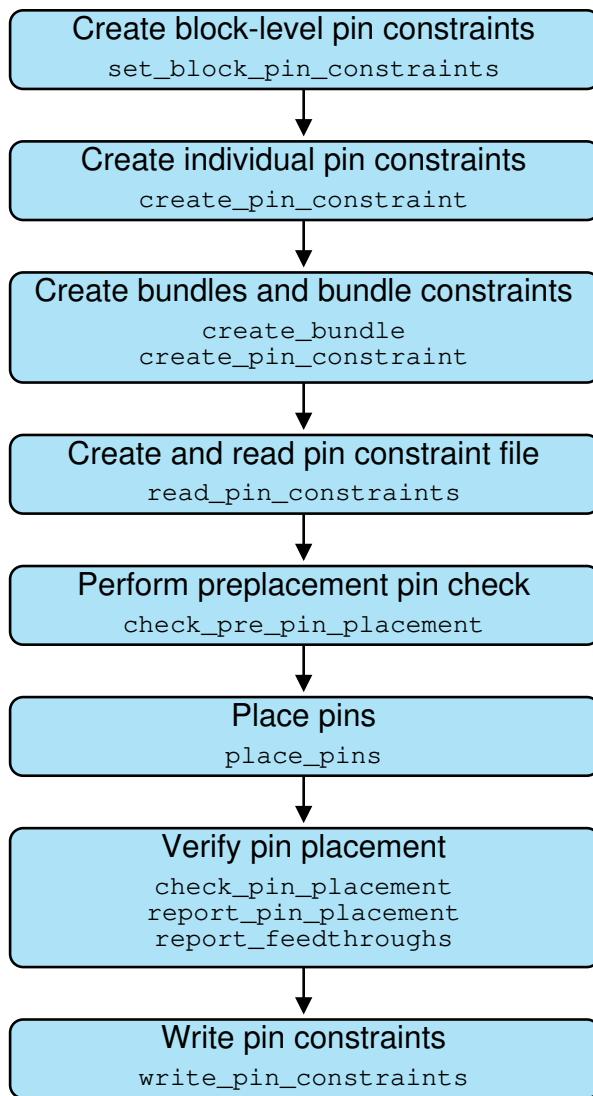
---

The IC Compiler II tool provides extensive control over pin placement and feedthrough creation during floorplanning. Pin placement is based on global routing and can be guided by user-defined constraints. You can make minimal changes to an existing pin placement by changing constraints and running pin assignment in incremental mode.

Pin placement considers all multiply instantiated block (MIB) instances during global routing and pin assignment. Comprehensive pin placement validation and reporting commands are available to qualify pin placement results. The GUI supports pin editing, moving, shaping to adjust placement, as well as pin-based net connectivity analysis and topological constraint visualization. During the ECO process, you can re-create existing pin locations and feedthroughs.

The flow to generate pin constraints, place pins, incrementally modify pin placement, and validate placement results is shown in [Figure 135](#).

*Figure 135 Pin Assignment Flow*



For more details, see the following topics:

- [Creating Pin Blockages](#)
- [Precedence Rules and Conflict Resolution for Pin Constraints](#)
- [Creating Block Pin Constraints](#)

- [Creating Individual Pin Constraints](#)
- [Creating Bundles and Bundle Constraints](#)
- [Setting Pin Constraints](#)
- [Setting Pin Placement Priority on Nets and Bundles](#)
- [Performing Global Routing for Pin Placement](#)
- [Creating a Channel Congestion Map](#)
- [Checking the Design Before Pin Placement](#)
- [Placing Pins](#)
- [Checking Pin Placement](#)
- [Reporting and Writing Pin Constraints](#)
- [Writing Out Feedthrough Information as Tcl Commands](#)
- [Creating Topological Pin Constraints in the GUI](#)
- [Reporting Estimated Wire Length](#)

## Creating Pin Blockages

You can create a region in which pins cannot be placed by using the `create_pin_blockage` command before running the `place_pins` command. The region boundary can be a rectangle, a polygon, or a collection of objects with physical geometry. The boundary can span multiple layers.

If you specify a list of physical pins, nets, or ports, the list must be homogeneous; it cannot contain a mixture of object types. However, specified pins, nets, and ports can be associated with multiple pin blockages. All specified pins, pins of specified nets, and specified ports are placed outside of the blockage. If you do not specify physical objects, all pins and ports are placed outside of the blockage during pin assignment.

You can create a feedthrough-only blockage, excluding feedthrough pins in the specified region, by specifying the `-feedthrough_only` option with the `create_pin_blockage` command. If there are conflicts during pin placement, the feedthrough-only pin blockages takes higher priority than any other pin constraints.

The following example creates a pin blockage named PB\_1 with a rectangular boundary that applies to all layers:

```
icc2_shell> create_pin_blockage -boundary {{100 100} {200 200}} -name
PB_1
```

The following example creates a pin blockage with a rectilinear boundary that applies to the M1 and M2 layers only:

```
icc2_shell> create_pin_blockage -boundary {{0 0} {2000 0} \
{2000 2000} {1000 2000} {1000 1000} {0 1000}} -layers {M1 M2}
```

The following example creates a pin blockage with a rectangular boundary that is associated with physical pins:

```
icc2_shell> set pins [get_pins -physical_context cell1/pin*]
icc2_shell> create_pin_blockage -boundary {{100 100} {200 200}} $pins
```

## Precedence Rules and Conflict Resolution for Pin Constraints

The following precedence rules are used when multiple constraints are specified for the same pin:

- Individual pin constraints take precedence over bundle pin constraints.
- Bundle pin constraints take precedence over nondefault rules.
- Nondefault rules take precedence over block pin constraints.
- More pessimistic or restrictive constraints take precedence over less restrictive constraints.

For example, a constraint that specifies an exact location takes precedence over a pin guide.

- When there are multiple constraints specified for the same pin constraints, the tool applies the last specified constraint.
- Constraints included in the topological map and feedthrough control sections of the pin constraint file override global feedthrough control settings.

The following conflict resolution rules are used when multiple constraints are specified for the same pin:

- If there are feedthrough constraint conflicts between topological map constraints and previous feedthrough constraints, feedthrough constraints from the constraint file have priority.
- If there are conflicts between feedthrough and topological map constraints, topological map constraints have priority.
- If the pin spacing control section contains conflicting constraints for the same edge, the same layer, and the same block, the last constraint takes priority and the command issues an error or warning message.

- When individual net (or pin) side, offset, layer, or order constraints conflict with bundle constraints, the `place_pins` command ignores bundle pin constraints for those nets (or pins) and places other bundle pins together.
- When individual net (or pin) width, length or spacing constraints conflict with bundle width, length, or spacing constraints, the `place_pins` command honors both individual and bundle constraints.
- If there are pins in a bus with a different spacing or width from other pins in the bus, all bus pins are placed on the same layer in the order defined in the bundle pin constraints.

Pin width is determined by the pin constraints applied to the block and by the following precedence rules:

- If the pin is placed on a wire track with nonzero track width, the pin width is the same as track width and any other width-related pin constraints are ignored. In this case, the minimum width rule defined in the technology file is ignored.
- If the pin has no width-related pin constraints, but its net has a nondefault routing rule that requires a specific width on the pin layer, the pin width is the width specified by the nondefault rule.
- If the pin does not have width-related pin constraints or a nondefault routing rule, the pin width is the same as the minimum width rule defined in the technology file.

## Creating Block Pin Constraints

Block-level pin constraints restrict the metal layers, spacing, block sides, and the creation of feedthroughs for the blocks in your design. The tool supports feedthroughs for both normal and multiply instantiated blocks (MIB). To create block-level pin constraints,

- Use the `set_block_pin_constraints` command and specify the constraint options.

```
icc2_shell> set_block_pin_constraints -pin_spacing 1 \
 -allowed_layers {M2 M3 M4 M5 M6 M7 M8 M9}
```

- (Optional) Verify the constraint settings with the `report_block_pin_constraints` command.

```
icc2_shell> report_block_pin_constraints

Report : report_block_pin_constraints
Design : alu

Pin constraints report on blocks
=====
block u_1/u0_3 :
 feedthroughs allowed: false
```

```
corner_keepout_num_tracks: 5
pin_spacing: 1
allowed_layers: M2 M3 M4 M5 M6 M7 M8 M9
hard_constraints: off
```

3. If needed, remove the block pin constraints with the `remove_block_pin_constraints` command and reapply the constraints with the `set_block_pin_constraints` command.

Alternatively, use the Task Assistant to define the block pin constraints.

1. Select Task > Task Assistant in the GUI.
2. Select Pin Assignment > Block-based Constraints.
3. Click the Block-based Constraint Table.
4. Enter the constraints in the table and click Apply.

Use options with the `set_block_pin_constraints` command to perform the following actions:

- Restrict pin placement to a specific set of layers

```
icc2_shell> set_block_pin_constraints -allowed_layers {M2 M3}
```

- Create additional offset from the corner of a block to the closest pin by specifying a number of tracks or a distance in microns

```
icc2_shell> set_block_pin_constraints -corner_keepout_num_tracks 5
icc2_shell> set_block_pin_constraints -corner_keepout_distance 10
```

- Set the spacing between adjacent pins by specifying a number of tracks or a distance in microns

```
icc2_shell> set_block_pin_constraints -pin_spacing 5
icc2_shell> set_block_pin_constraints -pin_spacing_distance 10
```

- Specify the allowed block sides where pins can be placed or the sides where pins cannot be placed

```
icc2_shell> set_block_pin_constraints -sides {1 2}
icc2_shell> set_block_pin_constraints -exclude_sides {3 4}
```

- Enable feedthroughs for a block; disable feedthroughs by setting the option to `false`

```
icc2_shell> set_block_pin_constraints -allow_feedthroughs true
```

- Set constraints on a specific list or collection of blocks

```
icc2_shell> set_block_pin_constraints -cells {CNTL REG}
```

- Specify which constraints must be met, even if the tool creates a DRC violation

```
icc2_shell> set_block_pin_constraints \
 -hard_constraints {spacing location layer length}
```

- Specify the length and width of the block pins

```
icc2_shell> set_block_pin_constraints \
 -length {{METAL3 0.2} {METAL4 0.3}}
icc2_shell> set_block_pin_constraints \
 -width {{METAL3 0.2} {METAL4 0.3}}
```

## Creating Individual Pin Constraints

After setting block pin constraints, define individual pin constraints to control pin placement details on a pin-by-pin basis. Individual pin constraints take precedence over block pin constraints. To create individual pin constraints,

- (Optional) Remove existing individual pin constraints with the `remove_pin_constraints` command.

```
icc2_shell> remove_pin_constraints [get_pin_constraints \
 -filter {pin_constraint_type=="individual"}]
```

- Use the `create_pin_constraint` command and specify the constraint options.

```
icc2_shell> create_pin_constraint -type individual \
 -pins {u0_1/rstn} \
 -sides 4 -width {{M3 3} {M5 5}} -length {{M3 3} {M5 5}}
```

- Verify the constraint settings with the `report_pin_constraints` command:

```
icc2_shell> report_pin_constraints [get_pin_constraints \
 -filter {pin_constraint_type=="individual"}]
```

Alternatively, use the Task Assistant to define the individual pin constraint.

- Select Task > Task Assistant in the GUI.
- Select Pin Assignment > Pin-based Constraints.
- Click the Pin-based Constraint Table.
- Enter the constraints in the table and click Apply.

Note that if you specify multiple conflicting pin constraints for the same pin, the last constraint takes precedence.

## Setting Individual Pin Constraint Options

Use options with the `create_pin_constraint` command to enable the following actions.

**Note:**

You must specify the `-type individual` option and at least one of the following options: `-pins`, `-ports`, or `-nets`.

- Restrict the pin placement to the specified set of pins, nets, ports, cells, and layers

```
icc2_shell> create_pin_constraint -type individual \
 -pins pins -nets nets \
 -ports ports -cells cells -layers layers
```

- Set the number of wire tracks between adjacent pins

```
icc2_shell> create_pin_constraint -type individual \
 -pin_spacing_tracks 5
```

- Set the distance in microns between adjacent pins

```
icc2_shell> create_pin_constraint -type individual \
 -pin_spacing_distance 2
```

- Specify the sides of the block that can be used to place the pin

```
icc2_shell> create_pin_constraint -type individual \
 -sides {1 2 3}
```

- Specify the sides of the block that cannot be used to place the pin

```
icc2_shell> create_pin_constraint -type individual \
 -exclude_sides {4}
```

- Specify the width or length of the pin as layer-width or layer-length pairs

```
icc2_shell> create_pin_constraint -type individual \
 -width {{M3 1} {M5 2}} -length {{M3 2} {M5 4}}
```

- Create a specified distance between the starting point for the block edge and the first pin on that edge

```
icc2_shell> create_pin_constraint -type individual \
 -sides 1 -offset 10
```

Alternatively, specify a range of locations in which the pins can be placed by specifying a minimum and maximum offset as follows:

```
icc2_shell> create_pin_constraint -type individual \
 -sides 1 -offset {5 15}
```

- Set the x- and y-location in the top-level design to place the pin

```
icc2_shell> create_pin_constraint -type individual \
-pins {u0_1/rstn} -location {1430 1914}
```

## Creating Bundles and Bundle Constraints

Net bundles are used to manage bus signals and other related signal nets. You can use bundles to group signals together, create a bundle object, and trace their flylines in the layout. Any net can be placed into a bundle. Signals within the bundle can be rearranged and sorted to create custom pin ordering. Bundles can be created by based on standard bus naming patterns. Bundles can contain other bundles.

During pin placement, the `place_pins` command routes bundle nets together through the block edge and honors bundle constraints such as pin spacing, pin width, and so on.

To create an explicit bundle by specifying individual net names,

1. Use the `create_bundle` command and specify the bundle name and the signals in the bundle.

```
icc2_shell> create_bundle -name Bundle1 \
[get_nets {sd[0] sd[1] sd[2] sd[3]}]
{Bundle1}
```

You can also create bundles of supernets with the `create_bundle` command.

```
icc2_shell> create_bundle -name supernet_bundle1 \
[get_supernets {supernet_0 supernet_1}]
```

2. (Optional) Verify the net contents of the bundle with the `report_bundles` command.

```
icc2_shell> report_bundles [get_bundles Bundle1]
Begin bundle report ..
 Bundle NAME: `Bundle1'
 Number of objects in bundle 4
 Bundle Type 'net'
 Objects in bundle
 Position = 0, net sd[0]
 ...
 ...
```

To automatically create bus bundles based on nets with similar prefixes,

1. Use the `create_bundles_from_patterns` command and specify the prefix of the net group, or omit the `-net_name_prefix` option to automatically create multiple net bundles for all net groups.

```
icc2_shell> create_bundles_from_patterns -net_name_prefix ahbmi
Information: There are 1 new bundles defined. (LED-102)
{B_ahbmi}
```

2. (Optional) Verify the contents of the bundle with the `report_bundles` command.

```
icc2_shell> report_bundles [get_bundles {B_ahbmi}]
Begin bundle report ..
 Bundle NAME: `B_ahbmi'
 Number of objects in bundle 24
 ...

```

The `create_bundles_from_patterns` command supports several options to control how the bundles are created. Use the `-net_name_prefix prefix` option to specify a set of net names for a bundle. To specify the ordering of the net bundles, use the `-net_order ascending | descending` option. To constrain the minimum and maximum number of nets in a bundle, use the `-minimum_nets min` and `-maximum_nets max` options.

If you omit the `-net_name_prefix` option, the `create_bundles_from_patterns` command creates a bundle by identifying signals that contain a bus index pattern. The supported bus index patterns are `[i]`, `(i)`, `<i>`, `_i`, `:i` and `{i}`, where "i" is the bus index. To limit the set of recognized bus patterns, specify the `-no_braces`, `-no_brackets`, `-no_angle_brackets`, `-no_underscores`, `-no_colons`, or `-no_parentheses` options.

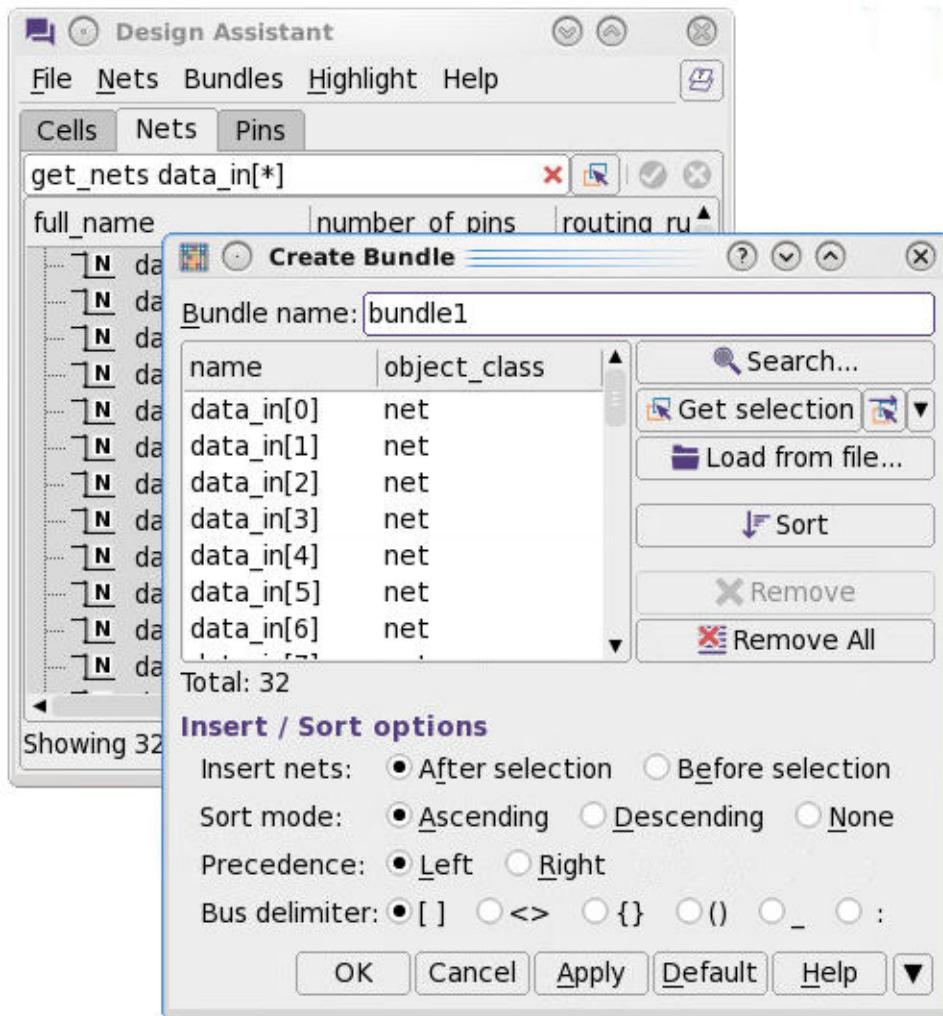
To modify an existing bundle by adding or removing nets, use the `add_to_bundle` and `remove_from_bundle` commands.

You can also use the Design Assistant dialog box in the GUI to create or modify bundles of nets as follows:

1. Select View > Assistants > Design Assistant in the GUI.
2. Enter the net selection command in the command box.
3. Select the nets to add to the bundle from the list.
4. Choose Bundles > Create Bundle from the menu.
5. Enter the bundle name.
6. Sort the list by selecting the sort options and clicking the Sort button, or by dragging and dropping the signals to reorder the list.
7. Click OK to create the bundle.

Figure 136 shows the Design Assistant and Create Bundle dialog boxes.

*Figure 136 Design Assistant and Create Bundle Dialog Boxes*



To return a collection of bundles, use the `get_bundles` command. To remove a bundle from the design, use the `remove_bundles` command.

Bundle pin constraints restrict the placement of net bundles on a physical design block. You can use the constraints to keep bundle bits together, specify pin ordering, specify the allowed layers for bundles, and so on. To create a bundle constraint,

1. Use the `create_pin_constraint` command and specify the constraint options.

```
icc2_shell> create_pin_constraint -type bundle \
 keep_pins_together true
1
```

2. (Optional) Verify the constraint settings with the `report_pin_constraints` command.

```
icc2_shell> report_pin_constraints [get_pin_constraints \
 -filter {pin_constraint_type=="bundle"}]

Report : report_pin_constraints
Design : ORCA

```

| Pin Constraints  | Description                                                |
|------------------|------------------------------------------------------------|
| -----            |                                                            |
| Pin_Constraint_0 | Type: Bundle<br>Obj Type: Net<br>Feedthrough allowed: true |
| 1                |                                                            |

The `create_pin_constraint` command supports several options to constrain bundles:

- To create the constraint on specific bundles, use the `-bundles` option.
- To apply the constraints to specific cells, use the `-cells` option.
- To apply the constraints only at the top level, use the `-self` option.
- To specify whether bundle pins are kept together, use the `-keep_pins_together` option.
- To specify a region on the block where the bundle pins should be placed, use the `-sides` and `-range` options.
- To control the ordering of the pin placement for signals in the bundle, use the `-bundle_order` option.
- To allow or disallow feedthroughs for a bundle, use the `-allow_feedthroughs` option.
- To control which layers can be used for the bundle pins, use the `-layers` option.
- To specify the number of wire tracks between adjacent pins, use the `-pin_spacing_tracks` option.
- To specify the minimum distance between adjacent pins in microns, use the `-pin_spacing_distance` option.

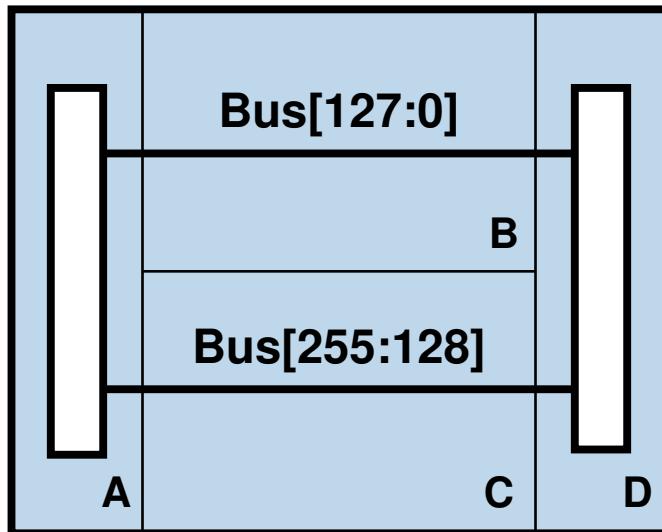
## Bundle Constraint Example

The following example uses bundle constraints to control the global routing of a bus. A 256-bit bus is split into two individual 128-bit buses. Bits 0 through 127 are routed from block A to block D through block B, and bits 128 through 255 are routed from block A to block D through block C. Each bundle is constrained independently and is routed

separately. The commands to create this routing are as follows and the simplified layout result is shown in [Figure 137](#).

```
icc2_shell> create_bundle -name bundle1 [get_nets \
 {Bus_0 Bus_1 ... Bus_127}]
icc2_shell> create_pin_constraint ... -bundles bundle1 \
 -layers {M3 M5} -cells B
icc2_shell> create_bundle -name bundle2 [get_nets \
 {Bus_128 Bus_129 ... Bus_255}]
icc2_shell> create_pin_constraint ... -bundles bundle2 \
 -layers {M3 M5} -cells C
```

*Figure 137 Splitting a Large Bus into Two Bundles*



## Setting Pin Constraints

Pin constraints describe the preferred connection path, feedthrough control, pin spacing, layer and other pin-related information for the blocks in the design. Pin constraints are defined with the `create_pin_constraint` and `set_block_pin_constraints` commands, or by reading a pin constraints file with the `read_pin_constraints` command.

To read the pin constraints file, use the `read_pin_constraints` command and specify the constraints file name as follows:

```
icc2_shell> read_pin_constraints -file_name constraints.txt
```

After reading the constraints, use the `report_pin_constraints` command to view the constraints.

The pin constraints file is divided into sections, and each section of the file begins with a section header and ends with a footer. Within each section, constraints are surrounded by curly brackets and each record description ends with a semicolon (;). Empty lines are ignored, and comments begin with a pound sign (#).

## Topological Map

The topological map section specifies how a net or group of nets should be routed through blocks during pin placement. Pin constraints specified in the topological map section take precedence over block-level pin constraints.

The topological map section of the constraints file is specified as follows:

- Each record begins with the `Nets` or `Bundles` keyword, contains the names of one or more nets or bundles, and is followed by pairs of objects on the net or bundle.
- Each cell, pin, or port specification should be grouped together by a pair of curly braces.
- Each keyword and its value should be enclosed by a pair of curly braces.
- For clarity, the description of the whole net or bundle constraint should be enclosed by curly braces; however, this is not strictly required and is not shown in the following example.
- Use an asterisk (\*) as a wildcard character to specify nets with similar names. An asterisk is not supported as a wildcard character for bundle names.
- The net or bundle constraint is terminated with a semicolon (;).

```
start topological map;
{Nets net_names}
 {{Port port_name_1} {Cell cell_name_1}}
 {{Cell cell_name_1} {Cell cell_name_2}}
 {{Cell cell_name_2} {Pins pin_name_1}};
{Bundles bundle_names}
 {{Cell cell_name_1} {Cell cell_name_2}};
end topological map;
```

For block objects specified by the `Cell` keyword, you can include side, offset, and layer constraints for the block pins by using the `side`, `offset` and `layer` keywords. The side constraint specifies the side number for the block where the pins are inserted. The side number is a positive integer that starts from 1. Given any block with a rectangular or rectilinear shape, the lowest leftmost edge is side number 1. The sides are numbered consecutively as you proceeded clockwise around the shape.

The offset specifies the distance in microns from the starting point of a given edge to the routing cross-point on that edge in clockwise direction. The starting point for edge number 1 is the lower vertex, the starting point for edge number 2 is the left vertex, and so on. You must specify the side number when you specify an offset. The layer constraint specifies the metal layers to use for the pins.

The general syntax for a block object constraint is as follows:

```
{ {Cell cell_name_1} {sides side_number}
 {offset offset_1} {layers {layer_1 layer_2 ...}} }
```

For examples of topological pin constraints, see [Topological Constraint Examples](#).

## Topological Constraint Examples

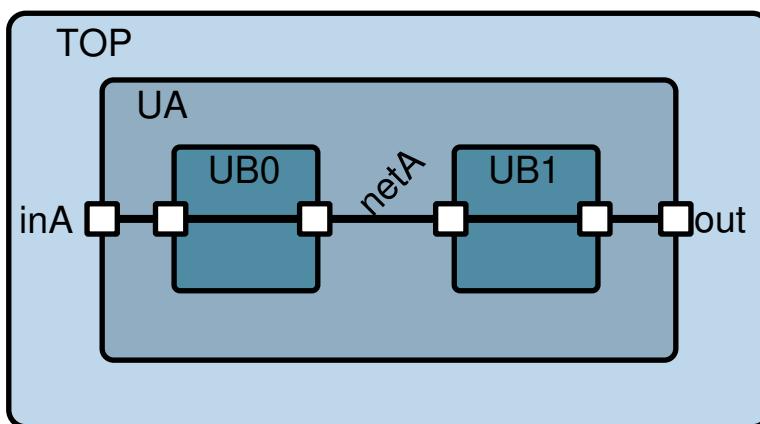
The following topics show brief usage examples for topological constraints.

### Specifying Routing Topology Using Side Constraints

The following example connects hierarchical nets and pins in the UA, UA/UB0, and UA/UB1 blocks as shown in the following figure.

```
Example 1
start topological map;
{nets UA/netA}
{{port UA/inA}}
{{cell UA/UB0} {sides 1}}}
{{cell UA/UB0} {sides 3}}
{{cell UA/UB1} {sides 1}}}
{{cell UA/UB1}{sides 3}}
{{port UA/out}}};
end topological map;
```

*Figure 138 Topological Constraints Example 1*

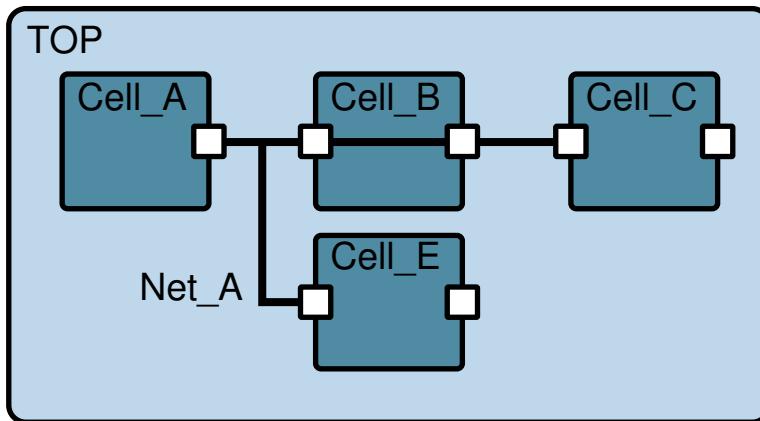


## Forcing a Connection Through a Block

The following example connects Net\_A to blocks Cell\_A, Cell\_C, and Cell\_E; the tool forces the routing between blocks Cell\_A and Cell\_C to pass through block Cell\_B.

```
Example 2
start topological map;
{Nets Net_A}
 {{Cell Cell_A} {Cell Cell_B}}
 {{Cell Cell_B} {Cell Cell_C}}
 {{Cell Cell_A} {Cell Cell_E}};
end topological map;
```

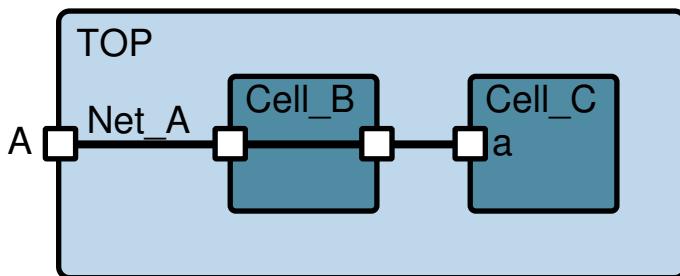
*Figure 139 Topological Constraints Example 2*



The following example connects top-level port A to pin Cell\_C/a. If Net\_A does not connect to Cell\_B, the tool forces the connection to pass through Cell\_B.

```
Example 3
start topological map;
{Nets Net_A}
 {{Port A} {Cell Cell_B}}
 {{Cell Cell_B} {Pins Cell_C/a}};
end topological map;
```

*Figure 140 Topological Constraints Example 3*

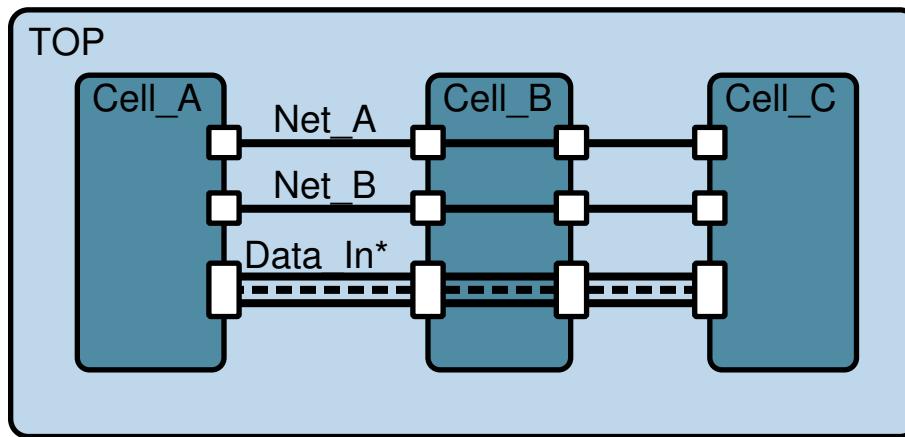


## Using Net Grouping in Topological Constraints

The following example uses curly brackets and a wildcard for net grouping. The constraint creates a similar routing pattern from Cell\_A to Cell\_B to Cell\_C for nets Net\_A, Net\_B, and all nets with net names that begin with DataIn\*.

```
Example 4
start topological map;
{Nets Net_A Net_B DataIn*}
 {{Cell Cell_A} {Cell Cell_B}}
 {{Cell Cell_B} {Cell Cell_C}};
end topological map;
```

Figure 141 Topological Constraints Example 4



The following example routes the Bundle1 and Bundle2 net bundles through the Cell\_A and Cell\_B blocks.

```
Example 5
start topological map;
{Bundles Bundle1 Bundle2}
 {{Cell Cell_A} {Cell Cell_B}};
end topological map;
```

## Specifying Side, Offset, and Layer Topological Constraints

The following example sets side, offset, and layer topological constraints for pins that connect to Net\_A. The routing connects Cell\_C through side 1 at offset 20 on layer M2 to Cell\_B through side 3 at offset 40 on layer M4. The routing also connects Cell\_B through side 1 at offset 10 on layer M4 to Cell\_A through side 3 at offset 30 on layer M2.

```
Example 6
start topological map;
{Nets Net_A}
 {{{{Cell Cell_C} {sides 1}
 {offset 20} {layers M2}}}}
```

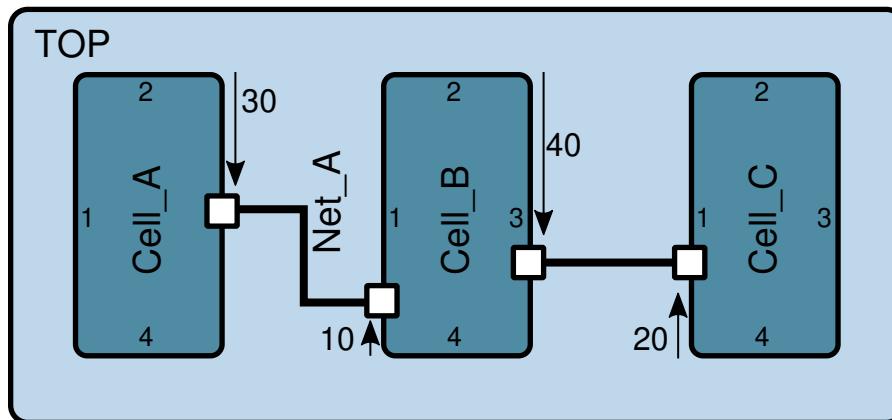
```

{{Cell Cell_B} {sides 3}
 {offset 40} {layers M4}}}

{{{{Cell Cell_B} {sides 1}
 {offset 10} {layers M4}}
 {{Cell Cell_A} {sides 3}
 {offset 30} {layers M2}}}};
end topological map;

```

Figure 142 Topological Constraints Example 6



## Specifying a Connection Without Feedthroughs

The following example contains three blocks: A, B, and C. The specification requires that net Net1 connect blocks A and C without creating a feedthrough in block B. To implement this design, create the following constraints.txt pin constraints file and specify the `read_pin_constraints -file_name constraints.txt` and `set_block_pin_constraints -allow_feedthroughs true` commands before running the `place_pins` command.

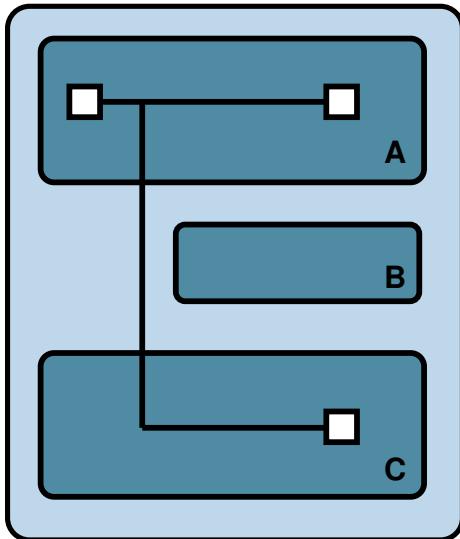
```

start topological map;
{Nets Net1}
 {{Cell A} {Cell C}};
end topological map;

```

The layout result is shown in [Figure 143](#).

*Figure 143 Direct Connection Between Blocks*



Note that you can create the same topology for a group of nets by specifying net names separated by white space within the curly brackets. You can also use the asterisk character (\*) as a wildcard.

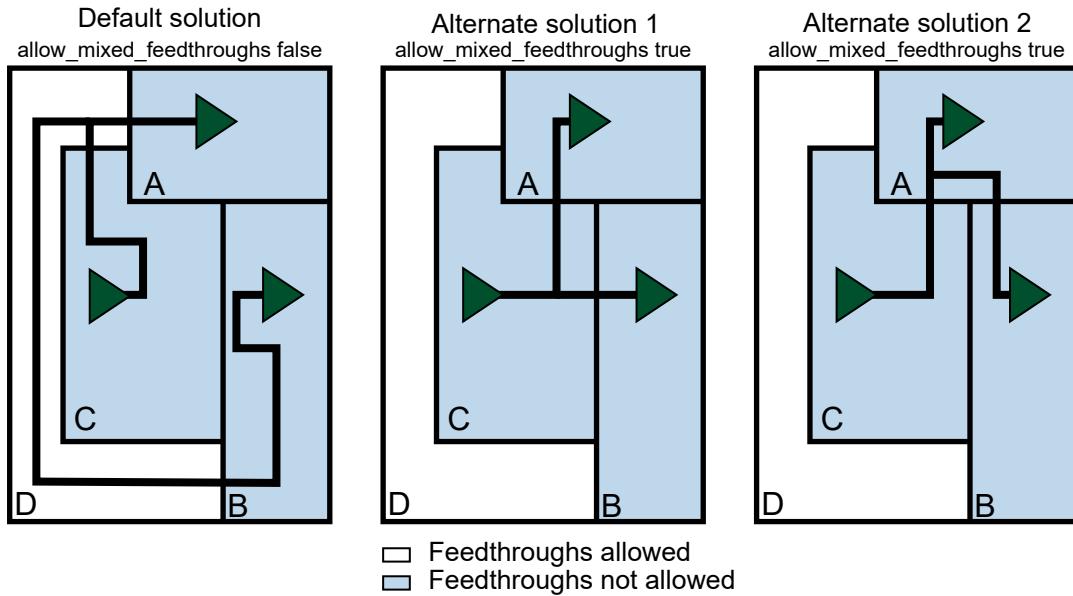
## Allowing Mixed Feedthroughs on Abutted and Narrow Channel Blocks

The `place_pins` command honors the topological pin feedthroughs constraints set with the `-allow_feedthroughs` option of the `set_block_pin_constraints`, `set_bundle_pin_constraints`, and `set_individual_pin_constraints` commands, in addition to the topological pin constraints file and the GUI. For designs with abutted blocks and tight feedthrough constraints, the global router might create long routing detours to avoid creating feedthroughs.

To improve routing, you can relax the feedthrough constraints and allow the `place_pins` command to create mixed feedthroughs on blocks where feedthroughs are disallowed. Mixed feedthroughs are feedthroughs that have an existing connection to a pin on the block. To relax the feedthrough constraints, set the `plan.pins.allow_mixed_feedthroughs` application option to `true`.

The following figure shows three layout results. The figure on the left is the default behavior when feedthroughs are disallowed on abutted blocks. The middle and right figures show possible layout results when the `plan.pins.allow_mixed_feedthroughs` application option is `true`.

Figure 144 Mixed Feedthroughs in Disallowed Blocks



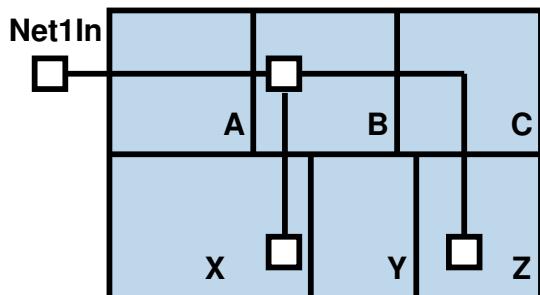
## Specifying a Partial Path

The following example contains six blocks: A, B, C, X, Y, and Z. The specification requires that port Net1In connect to block A, and block A connect to block B. No feedthroughs are allowed on blocks X and Y. To implement this feedthrough routing, create the following constraints.txt pin constraints file.

```
start topological map;
{Nets Net1}
{{Port Net1In} {Cell A}}
{{Cell A} {Cell B}};
end topological map;
start feedthrough control;
{Nets Net1}
{NoFeedthrough X Y};
end feedthrough control;
```

The layout result is shown in [Figure 145](#).

*Figure 145 Partial Path Specification*



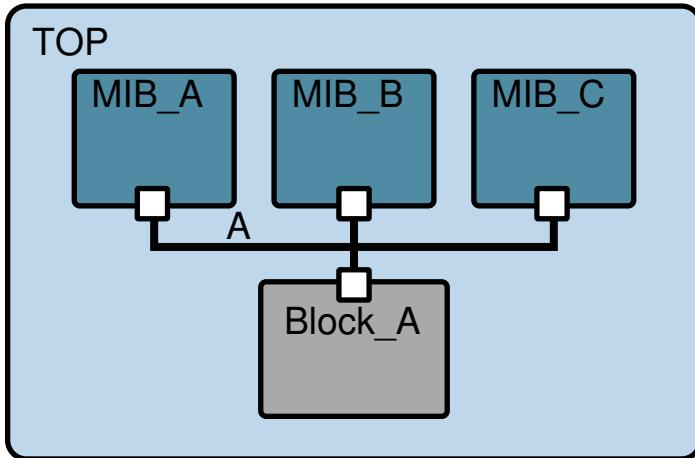
Note that Net1 has pins in blocks B, X, and Z, and the global router routes the net by creating a feedthrough in block B. Any feedthroughs created by the tool are labeled with the “\_FEEDTHRU\_n” suffix. The NoFeedthrough statement in the constraints file prevents the tool from creating a feedthrough in blocks X or Y, and the global router creates a feedthrough in block C instead.

## Specifying Topological Constraints for Multiply Instantiated Blocks

The following example contains multiply instantiated blocks MIB\_A, MIB\_B, and MIB\_C. The one-to-n topological constraint specifies that each MIB instance has the same size and offset constraint. After global routing and pin assignment, feedthroughs are created as shown in [Figure 146](#). No extra feedthroughs are created for net A. Note that the same routing topology is specified for blocks MIB\_A, MIB\_B, and MIB\_C; this is a requirement for multiply instantiated blocks.

```
start topological map;
{net A}
{{{{cell Block_A} {side 2}}}
 {{cell MIB_A} {side 4} {offset 10}}}
 {{{cell Block_A} {side 2}}}
 {{cell MIB_B} {side 4} {offset 10}}}
 {{{cell Block_A} {side 2}}}
 {{cell MIB_C} {side 4} {offset 10}}}};
end topological map;
```

Figure 146 Topological Constraints for Multiply Instantiated Blocks



## Feedthrough Control

The feedthrough control section specifies the blocks that allow or disallow feedthroughs. Pin constraints specified in the feedthrough control section take precedence over block-level pin constraints. Each record contains one or more net or bundle names within a pair of curly brackets, followed by a `Feedthrough` or `NoFeedthrough` keyword, followed by block cell names within curly brackets.

Any feedthrough nets created by the tool contain the `_FEEDTHRU_n` suffix. To modify this suffix, set the `plan.pins.new_port_name_tag` application option. Block feedthrough pins created by the tool have the `is_feedthrough_port` attribute set to `true`. Inside the block, all feedthrough nets, ports, and corresponding routing objects created by the tool have the `is_shadow` attribute set to `true`.

Note that you can route signals over blocks without creating feedthroughs. Disable feedthroughs for the block and net by specifying the `NoFeedthrough` keyword in the feedthrough control section, and create a routing corridor with the `create_routing_corridor` command. When you create pins with the `place_pins` command, the signal is routed over the block and feedthroughs are not created for the specified signal.

The general syntax is as follows. Note that this is only a syntax example; you typically do not specify both the `Feedthrough` and `NoFeedthrough` keywords for the same net.

```
start feedthrough control;
{Nets net_name}
Specify either Feedthrough or NoFeedthrough
but not both for a given net
{Feedthrough cell_name_1 cell_name_2 cell_name_3}
{NoFeedthrough cell_name_4 cell_name_5 cell_name_6};
{Bundles bundle_name}
```

```
{Feedthrough cell_name_7 cell_name_8 cell_name_9}
{NoFeedthrough cell_name_10 cell_name_11 cell_name_12};
end feedthrough control;
```

The following rules apply to feedthrough control:

- For a given net, constraint statements that appear later in the file override earlier statements.
- The `Feedthrough` and `NoFeedthrough` keywords specify whether feedthroughs are allowed or disallowed for either specified block cells or for the entire design.

If block cells are specified, all other blocks cells receive the opposite feedthrough constraint value. Therefore, there is no need for more than one `Feedthrough` or `NoFeedthrough` constraint line for a given net.

- Topological map constraints can take priority over feedthrough control statements. For example, A->B and B->C forces a feedthrough on B, regardless of whether feedthroughs are allowed on B.

A single `Feedthrough` or `NoFeedthrough` statement completely specifies the allowed feedthrough modules for the net. Feedthrough statements are not cumulative. If multiple feedthrough statements exist for the same net, the last feedthrough statement overrides the earlier statements and a warning message is issued.

`Feedthrough` and `NoFeedthrough` statements override the default feedthrough permission set by the `set_block_pin_constraints -allow_feedthroughs true | false` command for the specified nets.

The statement `{ Feedthrough A B }` does not force the creation of feedthroughs on A and B. Instead, the statement allows feedthroughs on blocks A and B, while restricting feedthroughs on all other blocks. To force a specific feedthrough path, you must use topological constraints.

If there are feedthrough constraint conflicts between topological map constraints and previous feedthrough constraints, the feedthrough constraints from the constraint file take priority. If there are conflicts between feedthrough constraints and topological map constraints, topological map constraints take priority.

The following example enables feedthroughs for net `Xecutng_Instrn[0]` on blocks `I_ALU`, `I_REG_FILE`, and `I_STACK_TOP`.

```
start feedthrough control;
{Nets Xecutng_Instrn[0]}
 {Feedthrough {I_ALU I_REG_FILE I_STACK_TOP}};
end feedthrough control;
```

The following example prevents feedthroughs for net `Xecutng_Instrn[0]` on blocks `I_ALU`, `I_REG_FILE`, and `I_STACK_TOP`.

```
start feedthrough control;
{Nets Xecutng_Instrn[0]}
 {NoFeedthrough I_ALU I_REG_FILE I_STACK_TOP};
end feedthrough control;
```

## Physical Pin Constraints

The physical pin constraints section specifies location and layer constraints for individual nets and pins. The general syntax is as follows:

```
start physical pin constraints;
{Net net_name} {cell cell_name}
 {layers {layer1 layer2}} {sides {side1 side2}}
 {offset offset1} {order {order_spec}}
 {start start_offset} {end end_offset}
 {width layer1 width1 layer2 width2}
 {length layer1 length1 layer2 length2};
{Pins pin_name} {reference reference_cell}
 {off_edge} {location {x y}};
end physical pin constraints;
```

**Supported keywords are:** net, pins, reference, sides, layers, offset, order, start, end, off\_edge, location, width, and length. The side constraint specifies on which side number the pin should be inserted for the specified block cell. The side number is a positive integer that starts from 1. Given any rectangular or rectilinear shape, the lower leftmost edge is side number 1. The sides are numbered consecutively as you proceeded clockwise around the shape. The shape refers to the reference boundary shape for the block.

The offset constraint specifies the distance in microns from the starting point of a specific edge to the routing cross-point on that edge in the clockwise direction. The starting point for side 1 is the lowest vertex, the starting point for side 2 is the left vertex, and so on. Since the offset is specific for each edge, you must specify the side information with the offset.

Each record in the physical pin constraints section is contained within curly brackets. The first element in the record can be a top-level net or a port for a reference block. If the record is a port object, the next record should be the reference block name associated with the port.

In the following example, ports C and D on reference block REFA are constrained. This provides flexibility to use the constraint both inside the block design and at the top level.

```
start physical pin constraints;
{Net E} {cell A} {layers M2} {sides 2};
{Net E} {cell B} {layers M4} {sides 4};
{Pins C} {reference REFA} {layers M2} {sides 2};
{Pins D} {reference REFA} {layers M2} {sides 2}
 {width 0.8} {length 0.8};
```

```
{Pins F} {reference REFA} {off_edge} {location {10 10}};
end physical pin constraints;
```

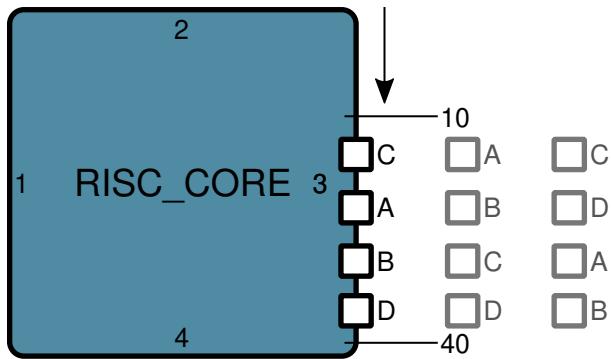
If the top-level block cells are instances of multiply instantiated blocks (MIBs), and there is more than one record for each port on the MIB's reference block, the last constraint record takes priority over previous constraint records.

## Physical Pin Constraint Example

In this example, pins A, B, C, and D must be placed within the 10 to 40 micron offset on side 3 of the RISC\_CORE reference block. The constraint specifies the pin order for pins A and B, however, pins C and D are not included in the order constraint. As a result, pins C and D can be anywhere within the 10 to 40 micron offset on side 3. [Figure 147](#) shows three possible placements for pins A, B, C, and D based on these constraints.

```
start physical pin constraints;
{reference RISC_CORE}
{sides 3}
{start 10}
{end 40}
{ {order {A B}}
{pins {C D}}};
end physical pin constraints;
```

*Figure 147 Physical Pin Constraint Example*



## Pin Spacing Control

The pin spacing control section specifies pin spacing constraints on individual edges for block cells. Each record specifies the pin spacing constraints for one or more edges on one or more layers for a particular block cell. The constraint contains the `reference`, `side`, and `layer` keywords. The general syntax is as follows:

```
start pin spacing control;
{reference ref_cell_name} {sides {side_a side_b}}
```

```
{layer_a spacing_a} {layer_b spacing_b};
end pin spacing control;
```

The `reference` keyword specifies the block reference that this constraint applies to. For a block cell, this constraint refers to the reference cell name, not the cell instance name, if it constrains the top level.

The `sides` keyword specifies one or more sides of the block. If there is more than one side, specify the side number within curly brackets separated by white space. Use an asterisk (\*) as a wildcard character to specify that all sides are allowed.

The `layer` and `spacing` constraints are specified as pairs. You can specify multiple layer-spacing pairs. The layer value is the metal layer name and the spacing value is the minimum number of wire tracks between adjacent pins for the specified layer. Only the specified layers are used for pin placement; unspecified layers are not used by the pin placer. In addition, only layers that are allowed in the corresponding reference blocks are allowed for pin placement.

Keywords are case-insensitive. The reference name and the layer name are case-sensitive. If a line contains a syntax error, the line is ignored. The command issues warnings or errors for any lines that are ignored.

If the pin spacing control section contains conflicting constraints for the same edge, same layer and the same block, the last constraint takes priority and the command issues an error or warning message.

In the following example, the CPU block cell is constrained to a minimum pin spacing of 2 wire tracks on layer M2, a minimum pin spacing of 3 wire tracks on layer M4, and a minimum pin spacing of 3 wire tracks on layer M6 for sides 2 and 4 of the block. If the pins are placed on sides 2 or 4, the pins must be placed on layer M2, M4, or M6. Note that when `reference`, `sides`, and `layer-spacing` pairs are given, unspecified layers are strictly not used by the tool for pin placement on the specified block sides. If the pins are placed on sides other than 2 or 4, the pins must honor the block pin constraints applied to the CPU block. Note that individual or bundle pin constraints can override these constraints.

```
start pin spacing control;
{reference CPU} {sides {2 4}} {M2 2} {M4 3} {M6 3};
end pin spacing control;
```

## Block Pin Constraints

The block pin constraints section specifies pin constraints that apply to all the pins of the specified block cell. You can specify block-related pin constraints for feedthroughs, layers, spacing, corner keepouts, hard pin constraints, sides, and exclude sides. The same constraint can be set with the `set_block_pin_constraints` Tcl command. Supported keywords are: `reference`, `sides`, `exclude_sides`, `layers`, `feedthrough`, `spacing`,

`spacing_distance`, `corner_keepout_distance`, `corner_keepout_num_tracks`, and `hard_constraints`.

The general syntax is as follows:

```
start block pin constraints;
{reference reference_name}
 {layers {layer_name_1 layer_name_2 ...}}
 {feedthrough true | false}
 {corner_keepout_num_tracks number_of_tracks}
 {spacing number_of_tracks}
 {hard_constraints {spacing location layer}}
 {exclude_sides {side_1 side_2}};
end block pin constraints;
```

The following example sets pin constraints on references `DATA_PATH` and `ALU`:

```
start block pin constraints;
{reference DATA_PATH}
 {layers {METAL3 METAL4 METAL5}}
 {feedthrough true}
 {corner_keepout_num_tracks 1}
 {spacing 5}
 {exclude_sides {2 3}};

{reference ALU}
 {layers {METAL3 METAL4 METAL5}}
 {feedthrough true}
 {hard_constraints {spacing location layer}}
 {sides {2 3}};
end block pin constraints;
```

## Setting Pin Placement Priority on Nets and Bundles

To specify a higher placement priority for certain pins that connect to nets in a bundle, set the `pin_placement_priority_id` attribute on the bundle as follows:

```
icc2_shell> set_attribute [get_bundles b1] pin_placement_priority_id 5
{b1}
```

Pins of bundles are placed as follows when the `pin_placement_priority_id` attribute is set:

- If bundle pin placement is enabled, bundle nets have higher placement priority than nonbundle nets.
- If two bundles have a `pin_placement_priority_id` setting, pins on the bundle with the smaller value have a higher placement priority.

## Performing Global Routing for Pin Placement

If your design contains multiply instantiated blocks or existing feedthroughs, you can aid pin placement by performing global routing on the top level and on child blocks before placing block pins. In the top-level design view, the blocks can be either design or abstract views. You can place pins and create feedthroughs for designs with top-level terminal or cell objects that are placed inside the block boundary.

To perform global routing on all the blocks in your design,

1. (Optional) Set host options for distributed processing.

```
icc2_shell> set_host_options -name distributed ...
```

2. Use the `route_global` command with the `-virtual_flat all_routing` option.

```
icc2_shell> route_global -floorplan true -host_options distributed \
-virtual_flat all_routing
```

Use the `-virtual_flat route_type` option to the `route_global` command to specify the hierarchy level to global route.

- Specify the `-virtual_flat all_routing` option to route all nets in both the top-level and child-level blocks.
- Specify `-virtual_flat top_and_interface_routing_only` option to route only nets in top-level design and nets that connect to child-level blocks, including interface and feedthrough nets that cross block boundaries.

After virtual flat global routing, you can place pins based on the global routing results with the `place_pins -use_existing_routing` command. Note that the `place_pins -use_existing_routing` command supports designs with or without multiply instantiated blocks.

To limit global routing and reduce runtime when iterating on different pin placements for a small collection of pins, use the `-nets_to_exclude_from_routing` option with the `place_pins` command. The command does not perform global routing for the specified nets.

To check for unused feedthroughs, use the `check_feedthroughs -unused_feedthroughs` command. You can use the `-congestion_map_only` option to create congestion maps without writing the global routes, after running the `route_global` command.

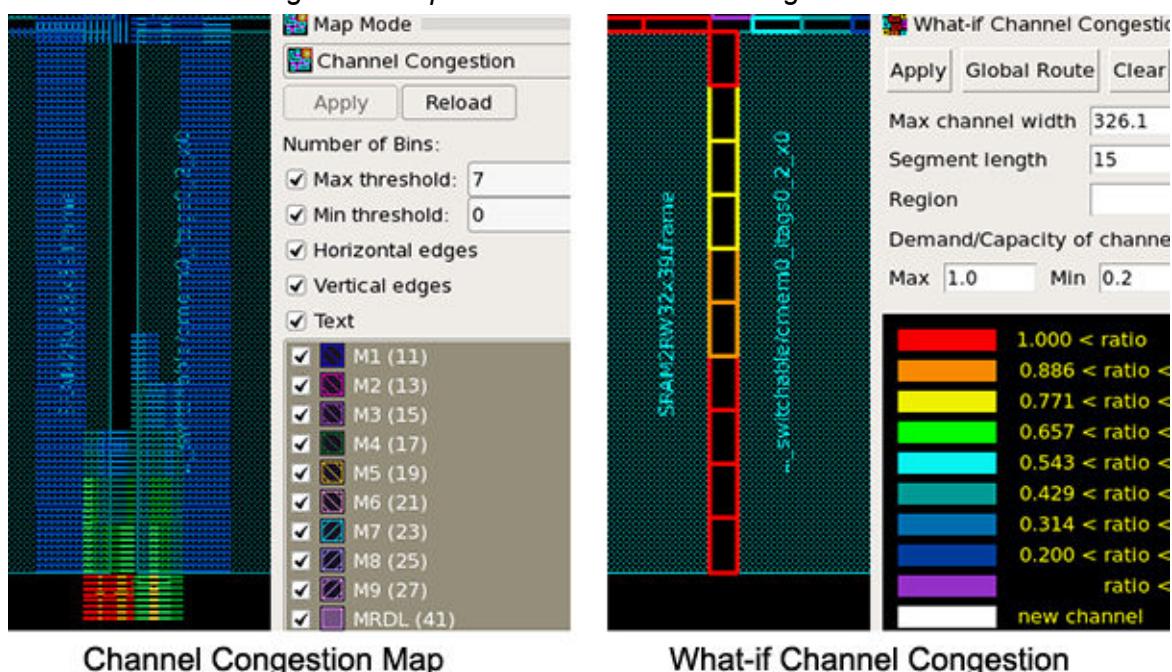
## Creating a Channel Congestion Map

Before performing pin placement, you can identify potentially congested areas between macros or block boundaries by creating a channel congestion map. The channel congestion map is based on the global route. To generate the channel congestion map, use the `create_channel_congestion_map` command as follows. The `-channel_width_threshold` option specifies the maximum channel width when generating the congestion map; channels larger than the specified width are not considered. The command first performs a global route on the design by using the `route_global` command.

```
icc2_shell> create_channel_congestion_map -channel_width_threshold 10
```

To view the congestion map, select View > Map > Channel Congestion in the GUI. The IC Compiler II tool also provides the What-if Channel Congestion panel for interactive analysis of congested channels. Figure 148 shows a channel congestion map and the What-if Channel Congestion panel.

Figure 148 Channel Congestion Map and What-if Channel Congestion



## Checking the Design Before Pin Placement

Before performing pin placement, you should check your design for the following design issues or pin constraint specification errors:

- Placement issues for cells and multiply instantiated blocks

**Use the `report_placement`, `check_mib_alignment`, and `check_mib_for_pin_placement` commands for these checks.**

- Pin constraint errors

**Use the `report_individual_pin_constraints`, `report_bundle_pin_constraints`, `report_block_pin_constraints`, and `report_topological_constraints` commands for these checks.**

- Other design issues that might affect pin placement

**Use the `check_design -checks {dp_pre_block_shaping dp_pre_macro_placement dp_pre_pin_placement}` command for this check.**

The `check_pre_pin_placement` or `check_design -checks {dp_pre_pin_placement}` command checks for the following conditions before pin placement. Some of these checks can be performed on top-level ports by specifying the `-self` option with the `check_pre_pin_placement` command.

- The routing direction of a given layer is not consistent with the block edge constraints

In this example, the routing direction for layer M3 is horizontal, but the pin access is set to the top and bottom sides of the block.

```
icc2_shell> set_attribute \
 -quiet [get_layers M3] routing_direction horizontal
icc2_shell> set_block_pin_constraints -cells B2 \
 -allowed_layers [get_layers M3] -sides {2 4}
icc2_shell> check_pre_pin_placement
...
----- Start Of Constraint Conflict Check -----
Warning: Conflicting block pin constraints
detected. Routing direction of allowed layers M3
incompatible with allowed sides 2, 4 of block
block2. (DPPA-441)
----- End Of Constraint Conflict Check -----
```

- Inconsistent edge directions in topological constraint pairs

In this example, the preferred routing direction for the METAL1 layer is horizontal; however, the left and right sides of blocks are excluded from pin placement.

```
icc2_shell> set_block_pin_constraints -exclude_sides {1 3} \
 -allowed_layers {METAL1}
```

Chapter 13: Performing Pin Assignment  
Checking the Design Before Pin Placement

```
icc2_shell> check_pre_pin_placement
...
----- Start Of Bad Constraint Check -----
Warning: Conflicting block pin constraints detected. Routing direction
of allowed layers METALL incompatible with excluded sides 1, 3 of
block block2. (DPPA-441)
```

- Invalid topological constraint that specifies a connection that cannot be created
- Fixed, off-edge pins that are part of a constraint

```
icc2_shell> check_pre_pin_placement
...
----- Start Of Off Edge Pin Check -----
Warning: The fixed pin U_BLK1/in on net in1 is off-edge, outside
boundary of U_BLK1. The fixed pin will be ignored in global route and
pin placement. (DPPA-458)
----- End Of Off Edge Pin Check -----
```

- Infeasible connection between cells that are separated by a block that has feedthroughs disabled. Feedthroughs must be enabled on blocks that separate two cells to allow a net connection through the block.
- Possible overlaps with existing internal block objects (route blockage, pin blockage, fixed pin, preroute, and so on) during pin creation

```
icc2_shell> check_pre_pin_placement
...
----- Start Of Pin Blockage Check -----
Warning: Fixed pin iu/clk_cts_3_10 might cause newly-placed abutted
pin in cell dcram_shell to overlap with blockage SNPS_PB_0 that is
close to the abutted edge 7 of cell iu. (DPPA-442)
----- End Of Pin Blockage Check -----
```

- Poorly specified topological constraints that create a loop

```
icc2_shell> sh cat constraint.txt
start topological map;
{net n2}
{{cell BLK3 {sides 3}} {cell BLK4 {sides 1}}}
{{cell BLK4 {sides 3}} {cell BLK2 {sides 3}}}
{{cell BLK2 {sides 2}} {cell BLK3 {sides 2}}};
end topological map;
```

```
icc2_shell> check_pre_pin_placement
----- Start Of Bad Constraint Check -----
Error: Loop back topological constraints start from and end at cell
BLK3 detected without indicating pin names. (DPPA-453)
----- End Of Bad Constraint Check -----
```

- Inconsistent routing or pin blockages across multiply instantiated blocks

```
icc2_shell> check_pre_pin_placement
...
----- Start Of Pin Blockage Check -----
Warning: Detected inconsistent blockage across MIBs. Blockage RB_0 for
cell mib1 does not have a counterpart for cell mib2. (DPPA-464)
...
```

- Fixed block pins or top terminals in pin blockages or routing blockages
- Fixed pins or top terminals outside associated pin guide or routing corridor
- Fixed pins shorted to each other
- Fixed pins that have inconsistent alignment
- A range, offset, or location overlap between net, bundle, or pin constraints
- Overlap between locations of existing fixed pins and the offset, location, or range values of net, bundle, or pin constraints

## Placing Pins

After defining pin placement constraints, use the `place_pins` command to perform global routing and place pins on the blocks in the current block.

You can limit pin placement to specific layers, cells, nets, pins, and ports, and perform incremental pin placement. The following command places pins on all blocks within the current block.

```
icc2_shell> place_pins
```

To place pins in a block,

1. (Optional) Set application options as needed.

```
icc2_shell> set_app_options -name plan.pins.incremental \
 -value true
plan.pins.incremental true
```

2. (Optional) Specify the number of processor cores to use for multithreading.

```
icc2_shell> set_host_options -max_cores 4
```

3. (Optional) Remove existing tool-created feedthroughs.

```
icc2_shell> remove_feedthroughs
```

4. Use the `place_pins` command to place pins on all blocks within the current block.

```
icc2_shell> place_pins
```

Alternatively, use the Task Assistant to place pins.

1. Select Task > Task Assistant in the GUI.
2. Select Pin Assignment > Place Pin/Global Route/Congestion and select the Placement tab.
3. Choose the Incremental or “Place only top-level ports” options if required.

If you choose the Incremental option, you must first remove all existing feedthroughs with the `remove_feedthroughs` command before proceeding.

4. Click Apply to place the pins.

To perform incremental pin placement after modifying the block or changing constraints,

1. Remove any existing feedthroughs with the `remove_feedthroughs` command.

When the feedthroughs are removed, the tool has more freedom to move pins along the block edge to create a layout with better quality of results (QoR). The tool can also change the layer used for the pin.

2. (Optional) Set the `plan.pins.incremental` application option to `true` to specify that pins should stay in their existing locations.

The tool moves the pin within the distance and layer ranges defined by the `plan.pins.pin_range` and `plan.pins.layer_range` application options, if the wire length, congestion, and pin alignment do not degrade.

3. Run the `place_pins` command.

Use options with the `place_pins` command to control how the pins are placed.

- Place pins only on specific cells, nets, pins, or ports by using the `-cells`, `-nets`, `-pins`, `-ports` option.

```
icc2_shell> place_pins -cells {I_ALU}
icc2_shell> place_pins -nets [get_nets Oprnd_B*]
icc2_shell> place_pins -pins [get_pins I_DATA_PATH/Oprnd_A*]
icc2_shell> place_pins -ports [get_ports]
```

For designs that contain multiply instantiated blocks (MIBs), use the `-cells` option to specify the instance to use when placing pins. Pin placement honors all user-specified blockages on all MIB instances and ignores congestion around MIB instances not specified with the `-cells` option.

- Place pins only on the current block by using the `-self` option.

```
icc2_shell> place_pins -self
```

- Avoid placing pins that connect to specific nets by using the `-exclude_nets` option.

```
icc2_shell> place_pins -exclude_nets [get_nets Oprnd_B*]
```

- Avoid global routing for some nets by using the `-nets_to_exclude_from_routing` option. This option might be useful to reduce runtime when iterating on different pin placements for a small collection of pins.

```
icc2_shell> place_pins \
-nets_to_exclude_from_routing [get_nets RegPort_B*]
```

The tool determines pin locations for excluded nets based on net connectivity and user-specified constraints. This option honors all pin constraints but does not honor feedthrough constraints or create feedthroughs; feedthrough creation requires route information from the global router. You should use the global router for pin placement whenever possible.

- Legalize pin placement by using the `-legalize` option.

```
icc2_shell> place_pins -legalize
```

The `-legalize` option honors the `plan.pins.pin_range` and `plan.pins.layer_range` application options and enables the following behavior for the `place_pins` command:

- Removes pin overlaps with other pins, vias, preroutes, PG straps, and so on
- Snaps pins to the nearest track

For multiply instantiated block (MIB) instances, pin placement considers all MIB instances during global routing and pin assignment. Pin placement for MIBs considers blockages, routing congestion, and wire length. You can perform pin placement on selected MIB instances by specifying their instance names on the `place_pins` command line. Pin placement honors all user-specified blockages for all MIB instances, and ignores the congestion around unselected MIB instances.

## Application Options for Pin Placement

The `place_pins` command supports the following application options to control pin placement.

*Table 18 Pin Placement Application Options*

| To do this                                                                                                                                                                                                                                                                                                                                                                       | Use this application option                                         |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|
| Specify whether feedthroughs are created at the top level of the block for channel-only nets.                                                                                                                                                                                                                                                                                    | <code>plan.pins.allow_feedthroughs_for_internal_channel_nets</code> |
| Specify whether the global router is permitted to route through narrow channels between blocks, between hard macros, and between blocks and hard macros. This option also controls whether the pin placer is allowed to place pins in narrow channel areas.                                                                                                                      | <code>plan.pins.allow_pins_in_narrow_macro_channels</code>          |
| Specify whether the tool treats the block as a black box.                                                                                                                                                                                                                                                                                                                        | <code>plan.pins.black_box_design_style</code>                       |
| Specify whether the tool should honor color-based span spacing rules loaded from the technology file. When this application option is <code>true</code> , the <code>place_pins</code> and <code>check_pin_placement</code> commands use these span spacing rules during pin creation and pin checking.                                                                           | <code>plan.pins.enable_color_span_rules</code>                      |
| Specify whether the tool should honor span spacing rules loaded from the technology file. When this application option is <code>true</code> , the <code>place_pins</code> and <code>check_pin_placement</code> commands use these span spacing rules during pin creation and pin checking.                                                                                       | <code>plan.pins.enable_span_rules</code>                            |
| Specify whether feedthroughs are allowed on clock pins.                                                                                                                                                                                                                                                                                                                          | <code>plan.pins.exclude_clocks_from_feedthroughs</code>             |
| Specify whether feedthrough ports are created on nets that connect to multiple driver pins or nets without driver pins, or nets that connect to inout pins.                                                                                                                                                                                                                      | <code>plan.pins.exclude_inout_nets_from_feedthroughs</code>         |
| When this application option is <code>true</code> , the <code>place_pins</code> command ignores the minimum end-to-end spacing rule.                                                                                                                                                                                                                                             | <code>plan.pins.ignore_pin_spacing_constraint_to_pg</code>          |
| When this application option is <code>true</code> , pins should stay at or near their existing locations. The tool moves the pin within the distance defined by the <code>plan.pins.pin_range</code> application option and the layers specified by the <code>plan.pins.layer_range</code> application option, if the wire length, congestion, and pin alignment do not degrade. | <code>plan.pins.incremental</code>                                  |
| Specify the maximum allowable layer displacement from the original pin layer.                                                                                                                                                                                                                                                                                                    | <code>plan.pins.layer_range</code>                                  |
| Specify whether the tool should group together pins on the block boundary that have the same block-to-block connections.                                                                                                                                                                                                                                                         | <code>plan.pins.maximize_black_box_pin_grouping</code>              |

*Table 18 Pin Placement Application Options (Continued)*

| To do this                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Use this application option                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------|
| Minimizes the number of feedthroughs at the possible cost of additional routing detours.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <code>plan.pins.minimize_feedthroughs</code>          |
| Specify the naming suffix for newly created feedthrough ports and feedthrough nets; this setting overrides the default of <code>_FEEDTHRU_#</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <code>plan.pins.new_port_name_tag</code>              |
| Specify whether the port name tag defined by the <code>plan.pins.new_port_name_tag</code> application option is inserted at the beginning or end of the net or port name.                                                                                                                                                                                                                                                                                                                                                                                                                                            | <code>plan.pins.new_port_name_tag_style</code>        |
| Specify whether the driver and receiver block names are inserted into the feedthrough net name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | <code>plan.pins.new_split_nets_use_block_names</code> |
| Specify whether intermediate combinational cells are ignored and the locations of the driver and load registers takes higher priority when placing pins. Note that this application option also affects the <code>route_global -virtual_flat</code> command and <code>route_group -global_planning</code> command.<br><br>This application option setting does not affect the pin detour or path detour checks that the <code>check_pin_placement</code> command performs. For these checks, use the <code>-pin_detour</code> and <code>-path_detour</code> options of the <code>check_pin_placement</code> command. | <code>plan.pins.path_based_pin_placement</code>       |
| Specify the maximum allowable pin location displacement from the original pin location when the <code>plan.pins.incremental</code> application option is set to true.                                                                                                                                                                                                                                                                                                                                                                                                                                                | <code>plan.pins.pin_range</code>                      |
| Specify the maximum channel size for routing, where a channel is the space between a block or module on which pins can be placed and any adjacent module, hard macro, or I/O pad. For narrow-channel blocks, specify this option to limit routing in the channel and allow feedthroughs by specifying <code>set_block_pin_constraints -allow_feedthroughs true</code> . Specify a larger value for <code>plan.pins.reserved_channel_threshold</code> to increase the number of feedthroughs created; specify a smaller value to decrease the number of feedthroughs created.                                         | <code>plan.pins.reserved_channel_threshold</code>     |
| Specify how new feedthrough nets for buses are named if they are not marked as bus ports or nets in the database.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | <code>plan.pins.retain_bus_name</code>                |

*Table 18 Pin Placement Application Options (Continued)*

| To do this                                                                                                                                                                                                                                                                                                               | Use this application option                    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| Specify whether pins are placed such that all connected pins are aligned. When this option is <code>true</code> , pin placement ignores routing tracks and places pins to strictly align with other placed and connected pins on the same layer.                                                                         | <code>plan.pins.strict_alignment</code>        |
| Specify whether new pins are created on neighboring abutted blocks and are connected to the original single pins. To add a suffix or prefix to the new pin names, use the <code>plan.pins.synthesize_abutted_pins_name_tag</code> and <code>plan.pins.synthesize_abutted_pins_name_tag_style</code> application options. | <code>plan.pins.synthesize_abutted_pins</code> |

*Table 18 Pin Placement Application Options (Continued)*

| To do this                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Use this application option               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| <p>Specify whether the tool should consider a narrow channel design as an abutted design for pin placement when the channel widths are less than or equal to the threshold value specified for this application option. This application option is honored during pin placement by the <code>place_pins</code> and <code>push_down_objects</code> commands.</p> <p>When a threshold value is specified in X- and Y-directions by using this application option, the tool analyzes the design for any single pins in narrow channel design blocks and places corresponding pins in the blocks that are without any pins. For example, as shown in the following diagram, the tool considers Blocks 1 and 2 to be abutted and creates a new pin in Block 1 to match the pin in Block 2.</p> <p>If this application option is not set, which is the default, the tool does not consider narrow channel designs as abutted designs. Therefore, the tool does not place any corresponding pins for any single pins that may be present in narrow channel designs.</p> <p>This application behaves as expected only when <code>plan.pins.synthesize_abutted_pins</code> is set to <code>true</code>.</p> <p><b>Pin placement when the application option is not set</b></p> <p><b>Pin placement when the application option is set</b></p> | <code>plan.pins.abutment_threshold</code> |

## Removing Pins and Terminals

Normally you do not need to explicitly remove pins and terminals after running the `place_pins` command; the command removes all nonfixed terminals each time it runs. To force removal of the terminals, use the `remove_terminals` command and specify the block pins as arguments to the command. The following example removes the terminals from the `I_ORCA_TOP/I_CONTEXT_MEM` block.

```
icc2_shell> set pins [get_pins -of_objects {I_ORCA_TOP/I_CONTEXT_MEM}]
icc2_shell> remove_terminals [get_terminals -of_objects $pins]
```

Avoid using the `remove_pins` command unless absolutely necessary, because this command alters the netlist by disconnecting nets from the pin.

## Performing Incremental Bundle Pin Placement

The tool supports an incremental bundle pin placement flow to improve the placement quality of pins for bundle nets. In this flow, you begin by placing pins for the most difficult bundles pins, such as bundles with wide width, complex connectivity, long feedthrough paths, and so on. Next, fix the location for the placed pins and repeat with a different set of bundles until all bundle pins are placed. Using this methodology, you can focus on a small set of bundle nets to improve bundle pin alignment and bundle feedthrough sharing. Bundle pins placed earlier in the flow have more flexibility in selecting edges, layers, and offsets for better placement.

The following example performs two iterations of pin placement: the first iteration for difficult-to-place bundle B1, and the second for bundle B2.

```
Allow feedthroughs on all blocks
set_block_pin_constraints -allow_feedthroughs true

Create two bundles
create_bundle -name {B1} {Data[0] Data[1] Data[2] Data[3]}
create_bundle -name {B2} {Data[4] Data[5] Data[6] Data[7]}

Create bundle pin constraints
set_bundle_pin_constraints -keep_pins_together true

Place pins for the first bundle
place_pins -nets [get_nets -of_objects [get_bundles {B1}]]

Fix the pin locations for bundle B1
set_fixed_objects [get_pins -filter "is_hierarchical==true" \
 -of_objects [get_nets -segments [get_nets \
 -of_objects [get_bundles {B1}]]]]]

Place pins for the second bundle
place_pins -nets [get_nets -of_objects [get_bundles {B2}]]
```

```
Fix the pin locations for bundle B2
set_fixed_objects [get_pins -filter "is_hierarchical==true" \
 -of_objects [get_nets -segments [get_nets \
 -of_objects [get_bundles {B2}]]]]
```

## Controlling Feedthrough Sharing in Multiply Instantiated Blocks

When feedthrough creation is enabled on a block, the `place_pins` command creates feedthroughs as needed to perform routing through blocks. To reduce the number of feedthroughs created for multiply instantiated blocks, set the `feedthrough_sharing_matching_id` attribute on feedthrough nets or bundles. The tool attempts to share feedthroughs only on nets with the same `feedthrough_sharing_matching_id` value. By default, the `feedthrough_sharing_matching_id` attribute is assigned to 0 for all nets. Avoid setting this attribute on both bundles and nets.

The following example sets the `feedthrough_sharing_matching_id` attribute on nets b[0] and b[1]. As a result, feedthroughs are not shared between nets b[0] and b[1].

```
icc2_shell> set_attribute [get_nets b[0]] \
 feedthrough_sharing_matching_id 10
icc2_shell> set_attribute [get_nets b[1]] \
 feedthrough_sharing_matching_id 20
```

## Checking Pin Placement

After placing pins, you should verify the placement and investigate any issues with pin alignment, pin spacing, feedthroughs, and other pin placement problems. To verify pin placement,

1. Use the `check_pin_placement` command and specify the verification options.

```
icc2_shell> check_pin_placement -pin_spacing true -layers true
----- Start Of Pin Layer Check -----
No violation has been found
----- End Of Pin Layer Check -----
----- Start Of Missing Pin Check -----
Port ahbi_HGRANT_0 on cell u1/u0_3 has no pin
Port ahbi_HGRANT_1 on cell u1/u0_3 has no pin
----- End Of Missing Pin Check -----
----- Start Of Pin Spacing Check -----
No violation has been found
----- End Of Pin Spacing Check -----
----- Start Of Pin Short Check -----
No violation has been found
----- End Of Pin Short Check -----
```

2. Use the `check_feedthroughs` command to check for unused feedthroughs, reused feedthroughs, and feedthrough constraint violations.

```
icc2_shell> check_feedthroughs -unused_feedthroughs \
 -reused_feedthroughs -net_constraints -topo_constraints

Un-used feedthrough ports:

Block Name | Instance Name | Pin Name
=====
block1 | U2/u0_0 | U2/u0_0/ahbso_FEEDTHRU_1[0]

Un-used feedthrough summary:

Block Name | Instance Name | # of unused | # of total
 | | feedthroughs | feedthroughs
=====
block1 | U2/u0_0 | 1 | 2

```

Re-used feedthrough ports:

```

Block Name | Instance Name |Pin Name

```

Re-used feedthrough summary:

```

Block Name | Instance Name | # of reused | # of total
 | | feedthroughs | feedthroughs

```

Checking Feedthrough Constraint Violations

Error: The net ahbso[0] violates feed-through constraints on cell u0\_1 with feed-through ports : u0\_1/...

Error: The net ahbso[0] violates feed-through constraints on cell U2/u0\_0 with feed-through ports : U2/... violating feedthrough ports:

```

Instance Name | Pin Name
=====
u0_1 | u0_1/ahbso_FEEDTHRU_1[0]
=====
U2/u0_0 | U2/u0_0/ahbso_FEEDTHRU_1[0]

```

--- Start Of Check Feedthrough Net Constraints Summary ---

Number of Feedthrough ports checked:18

Number of Violating Feedthrough ports:2

| Inst Name | Number of Ports | Number of Violating Ports |
|-----------|-----------------|---------------------------|
| u0_1      | 6345            | 1                         |

## Chapter 13: Performing Pin Assignment

### Checking Pin Placement

```
=====
U2/u0_0 | 6345 | 1
=====
U1/u0_2 | 6344 | 0
=====
U1/u0_3 | 6348 | 0
--- End Of Check Feedthrough Net Constraints Summary ---
```

End of feed-through constraint violations

Checking Topological Constraint Violations

```
Error: The net ahbs0[0] violates topological constraints
 by not having a direct connection between cells u0_1 and ...
Error: The net ahbs0[0] violates topological constraints
 by not having a direct connection between cells U1/u0_3 ...
```

Checked a total of 1 nets with topological constraints,
found 1 nets with violations

End of topological constraint violations

3. If your design contains multiply instantiated block instances, verify pin placement with the `check_mib_for_pin_placement` command.

```
icc2_shell> check_mib_for_pin_placement -swapped_connections \
 -top_level_terminal_locations
```

4. Use the `report_pin_placement` command to save the block name, layer, side, and offset for the current pin placement for each block.

```
icc2_shell> report_pin_placement > pin_placement.rpt
```

5. Use the `report_feedthroughs` command to generate a list of feedthrough nets and their associated block pins.

```
icc2_shell> report_feedthroughs -reporting_style net_based
```

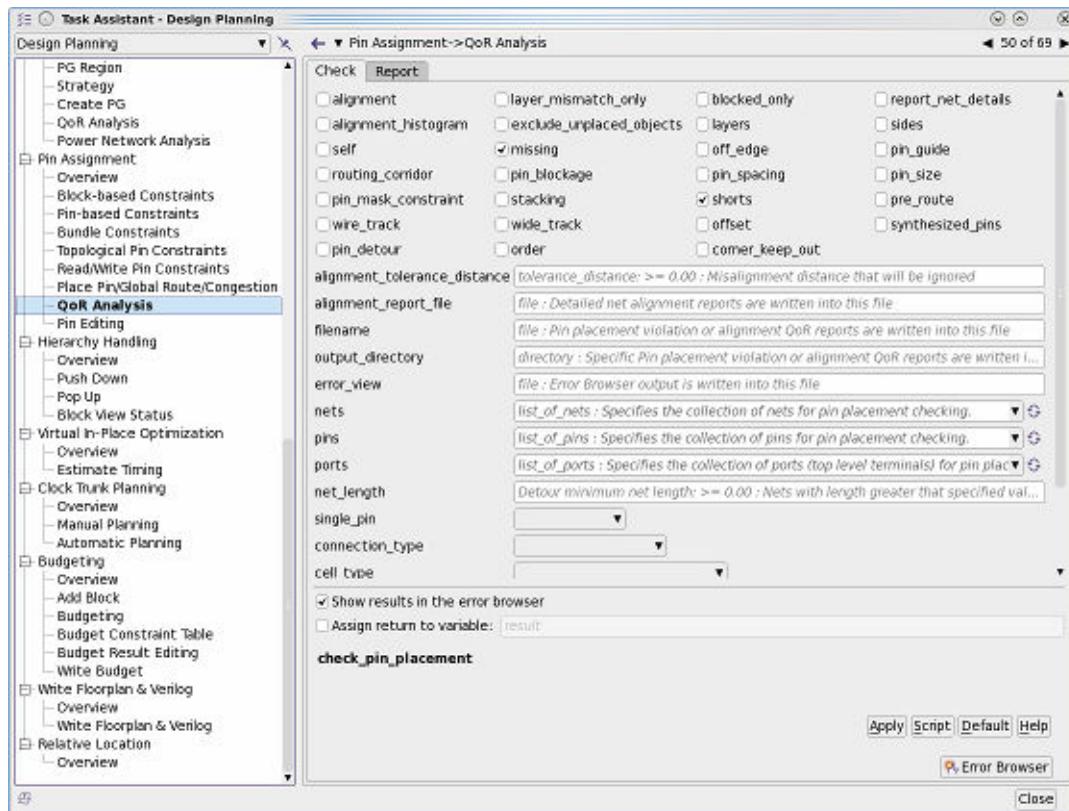
Alternatively, use the Task Assistant to verify the pin placement and create the pin and feedthrough reports.

1. Select Task > Task Assistant in the GUI.
2. Select Pin Assignment > QoR Analysis and click the Check tab.
3. Select the checking options as needed to limit the pin verification.
4. Click Apply to check pin placement.
5. Select Pin Assignment > QoR Analysis and click the Report tab.
6. Select reporting options as needed to limit the report to specific nets or blocks.

7. Click Apply to generate the pin placement report.

Note that the `check_feedthroughs` and `report_feedthroughs` commands are not available in the Task Assistant.

**Figure 149 Check Pin QoR Task Assistant Dialog Boxes**



For net bundles created with the `create_bundle` or `create_bundles_from_patterns` command, or by using the GUI, the `check_pin_placement` command checks that the bundle meets the pin constraints. Bundle pin constraints are created with the `set_bundle_pin_constraints` command, and the `check_pin_placement` command checks spacing, layer, and offset constraints for the bundle. Note that individual pin constraints take precedence over bundle pin constraints, and bundle pin constraints take precedence over block pin constraints.

Use the `report_pin_placement` command to generate a list of pins on blocks in the design. The report contains the block name, layer, block side, and offset for each pin. To limit the report to a specified set of pins, nets, ports, or cells, use the `-pins pins`, `-nets nets`, `-ports ports`, or `-cells cells` option. Use the `-self` option to report pin placement for only the top-level ports. Use the `-format {layer side offset cell}` option to limit the report content to one or more of layer, side, offset or block name.

Use the `report_feedthroughs` command to generate a report that contains feedthrough nets and their associated block pin names for each feedthrough. To limit the report to a specified set of feedthrough nets, use the `-nets` option. Use the `-include_buffered` option to report feedthroughs which contain buffer cells. Use the `-include_original_feedthroughs` option to report feedthroughs inserted by the `place_pins` command and feedthroughs that previously existed in the original netlist. Use the `-reporting_style net_based` option to generate a single report containing feedthrough net names and associated block pins for each feedthrough in the design. Use the `-reporting_style block_based` option to generate multiple feedthrough reports, one for each design block. The report generated with the `-reporting_style block_based` option contains a list of feedthrough net names. Use the `-self` option to limit feedthrough reporting to only the current block; by default, the command checks all planning levels.

## Checking Pin Alignment

Use options with the `check_pin_placement` command to control pin alignment checking as follows (the `-alignment true` option also enables a check for block edges to ensure that they are aligned to the litho grid).

- Limit alignment checking to only two-pin nets only of the specified connection types: `BLOCK_TO_BLOCK`, `BLOCK_TO_MACRO`, `BLOCK_TO_IO`, or `ALL`. The default is `ALL`.

```
icc2_shell> check_pin_placement -alignment true \
 -connection_type BLOCK_TO_BLOCK
```

- Check pin alignment within a specified tolerance, or identify aligned pins that have a physical block or hard macro between the pins; the default alignment tolerance is zero.

```
icc2_shell> check_pin_placement -alignment true \
 -alignment_tolerance_distance 5
```

- Report nets that are aligned, but the pins are placed on different layers

```
icc2_shell> check_pin_placement -alignment true \
 -layer_mismatch_only true
```

- Print out additional information about the nets connected to the violating pins and report detailed net information about pins with alignment violations; include the `-alignment_report_file filename` option to write the output to a file

```
icc2_shell> check_pin_placement -alignment true \
 -report_net_details true
...
***** total unaligned nets *****
...
***** unaligned alignable unblocked nets *****
```

## Chapter 13: Performing Pin Assignment

### Checking Pin Placement

```
Two-Pin net q1_35 with pins u1/q35 and u2/d35
 of type BLOCK->BLOCK is unaligned.
 The net is alignable and unblocked.

...
***** unaligned unalignable nets *****
Two-Pin net q0_0 with pins u0/q0 and u1/d0
 of type BLOCK->BLOCK is unaligned.
 The net is unalignable (Due to Side Orientation) and blocked.

...
```

- Skip checking for nets that have unplaced connections when checking pin alignment; by default, the `-alignment true` option includes unplaced pins when checking alignment

```
icc2_shell> check_pin_placement -alignment true \
 -exclude_unplaced_objects true
```

- Limit the checking to the specified connection type: ALL, BLOCK\_TO\_BLOCK, BLOCK\_TO\_MACRO, or BLOCK\_TO\_IO when checking alignment; by default, ALL is used

```
icc2_shell> check_pin_placement -alignment true \
 -connection_type BLOCK_TO_BLOCK
```

- Limit the alignment check to two-pin nets that are aligned but have a physical block or hard macro placed between the two pins

```
icc2_shell> check_pin_placement -alignment true \
 -blocked_only true
```

- Write a table of misalignment offsets and the number of pins in each offset range

```
icc2_shell> check_pin_placement -alignment true \
 -alignment_histogram true
...
Total of 17 alignment violations
-----Start Of Alignment Histogram-----
Total Two-Pin Nets: 65
Total Misaligned Nets: 17
Average Misalignment Distance: 24.3668um
Standard Deviation: 48.3um
Maximum: 151.048um

Low Violation (um) High Violation (um) Number

0 50 14
50 100 1
100 150 1
150 200 1
-----End Of Alignment Histogram-----
```

## Saving Output From `check_pin_placement`

Use options with the `check_pin_placement` command to specify how the information from the pin check is saved:

- Specify the name of the error view for pin placement checking

```
icc2_shell> check_pin_placement -error_view pincheck.err
```

- Specify the name of a report file in which to write the console output from the `check_pin_placement` command; by default, the tool writes output only to the console

```
icc2_shell> check_pin_placement -filename pinchecks.txt
```

## Enabling Various Pin Placement Checks

Use options with the `check_pin_placement` command to limit pin checking to the specified checks or change the default pin check behavior. By default, the command skips these checks unless otherwise specified.

- Check for mismatches in pin directions for nets with only two pins.

```
icc2_shell> check_pin_placement -pin_direction true
```

- Check the pins only of the specified cell type: BLOCK, HARD\_MACRO, EXTRACTED\_TIMING\_MODEL, or ALL. The default is BLOCK.

```
icc2_shell> check_pin_placement -cell_type HARD_MACRO
```

- Check the pins only of the specified type: PG\_PINS, SIGNAL\_PINS, or ALL. The default is SIGNAL\_PINS.

```
icc2_shell> check_pin_placement -pin_type ALL
```

- Check for corner keepout violations on the edges of physical blocks

```
icc2_shell> check_pin_placement -corner_keep_out true
```

- Check if pins are placed only on allowed layers

```
icc2_shell> check_pin_placement -layers true
```

- Skip checking for block pins without a pin shape; by default, this check is performed

```
icc2_shell> check_pin_placement -missing false
```

- Restrict the check to the specified nets; by default, the tool checks all block pins

```
icc2_shell> check_pin_placement -nets {clk}
```

- Check for off-edge pins
 

```
icc2_shell> check_pin_placement -off_edge true
```
- Check if any pins violate individual pin offset constraints
 

```
icc2_shell> check_pin_placement -offset true
```
- Check for pin-order constraint violations
 

```
icc2_shell> check_pin_placement -order true
```
- Check if the center of the pin is inside an associated pin blockage and if the pin shape is on the layer associated with the blockage
 

```
icc2_shell> check_pin_placement -pin_blockage true
```
- Create an ordered list of pins that might have nonoptimal placement and create a longer routing path than necessary

The `-pin_detour` check reports nets that have a bounding-box greater than the bounding box of the standard-cell pins and the top-level terminals or ports that it connects.

When the `-detour_consider_pin_constraints` option is set to `true`, the tool takes the individual and block pin constraints into account while checking for any pin detours.

```
icc2_shell> check_pin_placement -pin_detour true \
 -detour_tolerance 1.5 -detour_consider_pin_constraints true
```

- Create an ordered list of pins that might have nonoptimal placement and create a longer routing path than necessary

The `-path_detour` check reports nets that have a bounding-box greater than the bounding box of the driver and load registers and the top-level terminals or ports that it connects.

When the `-detour_consider_pin_constraints` option is set to `true`, the tool takes the individual and block pin constraints into account while checking for any path detours.

```
icc2_shell> check_pin_placement -path_detour true \
 -detour_tolerance 1.5 -detour_consider_pin_constraints true
```

- Check if the center of the pin is within its associated pin guide and whether the pin is placed on a layer allowed by the pin guide

```
icc2_shell> check_pin_placement -pin_guide true
```

- Check whether pins have adequate spacing between them

```
icc2_shell> check_pin_placement -pin_spacing true
```

To modify the spacing check and treat PG pins as a stop or shield, set the `plan.pins.treat_pg_as_shield` application option to `true`.

```
icc2_shell> set_app_options -name plan.pins.treat_pg_as_shield \
 -value true
```

- Check for pins with coloring conflicts

```
icc2_shell> check_pin_placement -pin_mask_constraint true
```

- Check for pins that fail to honor their length constraints, width constraints, or double patterning pin length requirements

```
icc2_shell> check_pin_placement -pin_size true
```

- Check for pins shorted with PG or signal preroutes; the `-shorts` check does not check for these shorts

```
icc2_shell> check_pin_placement -pre_route true
```

- Check for pins that are outside the routing corridor; this option also check for pins that are on a different layer than specified by the routing corridor

```
icc2_shell> check_pin_placement -routing_corridor true
```

- Check only top-level terminals and ignore block pins; by default, the command checks all block pins

```
icc2_shell> check_pin_placement -self
```

- Disable checking for shorted or overlapping pins; by default, the shorts check is enabled

```
icc2_shell> check_pin_placement -shorts false
```

- Check for pin side placement violations

```
icc2_shell> check_pin_placement -sides true
```

- Check for single, unrouteable pins under different conditions; valid values are connected, unconnected, and all

```
icc2_shell> check_pin_placement -single_pin all
```

- Check for pin stacking violations

```
icc2_shell> check_pin_placement -stacking true
```

- Report all the pins that were inserted by the tool to avoid single pins in an abutted design

```
icc2_shell> check_pin_placement -synthesized_pins true
```

- Check that pins are centered on a wire track

```
icc2_shell> check_pin_placement -wire_track true
```

- Check that pins satisfy track width constraints

```
icc2_shell> check_pin_placement -wide_track true
```

- Check that pins satisfy location constraints

```
icc2_shell> check_pin_placement -location true
```

- Disable checking for violations of technology-related minimum layer spacing and fat metal spacing between signal pins, signal and PG pins, and signal and preroute pins

```
icc2_shell> check_pin_placement -technology_spacing_rules false
```

## Checking Pin Placement on Multiply Instantiated Blocks

Use the `check_mib_for_pin_placement` command to check multiply instantiated blocks (MIBs) for pin placement issues.

- Check for pins across MIB instances where the MIB block pin does not connect to the same number or pins or same type of pins.

```
icc2_shell> check_mib_for_pin_placement -asymmetric_connections
----- Start Of Logical Asymmetry Check -----
Error: Port pin_b on MIB Reference Module CPU
 has asymmetric connections. (DPPA-205)
```

- Check MIB instances for two-pin nets that do not connect to the same pin across all instances of the reference MIB block. Swapped connections are a subset of asymmetric connections.

```
icc2_shell> check_mib_for_pin_placement -swapped_connections
----- Start Of Swapped Connection Check -----
Error: Port pin_a on MIB Reference Module CPU
 is a potentially swapped port. (DPPA-203)
```

- Check for top-level terminals that connect to the same MIB pin, but at different locations on the MIB instances.

```
icc2_shell> check_mib_for_pin_placement -top_level_terminal_locations
----- Start Of Top Level Terminal Check -----
Error: Port A[0] on MIB Reference Module dlm_test2 connects to
```

top level terminals which are not all at the same location. (DPPA-204)

- Check only pins on specific cells, or pins only connected to specific nets.

```
icc2_shell> check_mib_for_pin_placement -asymmetric_connections \
 -cells {U_MIB2_2/U_DLM_TEST2}
icc2_shell> check_mib_for_pin_placement -asymmetric_connections \
 -nets {U_MIB1_2/A}
```

## Checking Feedthroughs

Use options with the `check_feedthroughs` command to specify how feedthroughs are checked and what information is reported:

- Specify a collection of blocks for feedthrough analysis

```
icc2_shell> check_feedthroughs -cells {u0_1}
```

- Write out a list of feedthroughs that are reused across MIB instances

```
icc2_shell> check_feedthroughs -reused_feedthroughs
```

- Check for feedthrough violations on each feedthrough net on each block and for net bundles

```
icc2_shell> check_feedthroughs -net_constraints
```

- Check for redundant feedthroughs

```
icc2_shell> check_feedthroughs -redundant
```

- Check feedthrough nets that connect top-level ports in the current block

```
icc2_shell> check_feedthroughs -self
```

- Check topological constraints for each feedthrough net

```
icc2_shell> check_feedthroughs -topo_constraints
```

- Check for unused feedthroughs

```
icc2_shell> check_feedthroughs -unused_feedthroughs
```

- Include repeater and buffer cells in feedthrough tracing when timing libraries are loaded

```
icc2_shell> check_feedthroughs -include_buffered
```

- Check mixed feedthroughs

```
icc2_shell> check_feedthroughs -mixed
```

The tool supports two type of mixed feedthroughs:

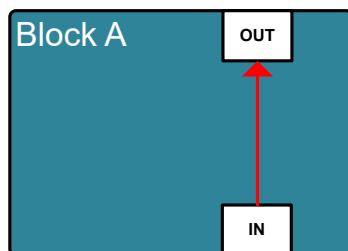
- Feedthroughs with connections to combinational cells (other than buffers or inverters)
- Feedthrough nets that own both the original feedthrough port and the newly created feedthrough port
- Check pure feedthroughs

```
icc2_shell> check_feedthroughs -pure
```

Pure feedthroughs are feedthroughs that have no physical pins or connections other than buffers or inverters within the block that the net passes through.

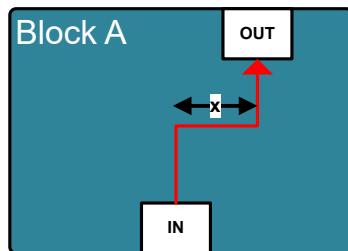
- Check intra-block feedthrough alignment for two-pin nets

```
icc2_shell> check_feedthroughs -alignment \
-alignment_tolerance_distance 1
```



`check_feedthroughs -alignment`

Without the `-alignment_tolerance_distance` option, the two pins are considered aligned if their pin bounding boxes are aligned, as shown in the image to the left.



`check_feedthroughs -alignment \
-alignment_tolerance_distance x`

The two pins are considered unaligned when the distance between them is greater than the value specified by the `-alignment_tolerance_distance` option. In this example, if the distance between the pins is greater than "x," the tool considers it as an alignment violation.

## Reporting and Writing Pin Constraints

Use the `report_pin_constraints` command to display the individual pin constraints, block pin constraints, bundle pin constraints, and topological constraints for a specific nets, ports, or pins. The command also displays the constraint information for related nets, pins, and blocks.

Use the `write_pin_constraints` command to write out the current set of pin constraints for a specific pins, ports, nets, or bundles. Use options to the command to specify the type of information written for topological and physical pin constraints.

## Reporting Pin Constraints

Use the `report_pin_constraints` command to display the individual pin constraints, block pin constraints, bundle pin constraints, and topological constraints for a specific net, port, or pin. The command also displays the constraint information for related pins and blocks.

The following example reports pin constraints for net `rst_1`:

```
icc2_shell> report_pin_constraints -nets rst_1

Report : report_pin_constraints
Design : top

Pin constraints report on nets
=====

Pin constraints report on connected nets
=====

--Topological Pin Constraints On Net rst_1--
{Nets rst_1}
{{port rst_1}} {{cell U_H1_0} {sides 1}
{offset {231 266}} {layers {M3}}}}
{{cell U_H1_0} {sides 1} {offset {231 266}} {layers {M3}}}
{{cell U_H1_0/U_H2_0} {sides 1} {offset {30 65}} {layers {M3}}}}
...
{{cell U_H2_3} {sides 4} {offset {38 88}} {layers {M4}}}
{{cell U_H2_2} {sides 2} {offset {36 87}} {layers {M4}}}}
```

## Writing Pin Constraints

Use the `write_pin_constraints` command to write out a constraints file that can be used as input for incremental pin placement and feedthrough generation. To write the pin constraints for the current design, use the `write_pin_constraints` command with the appropriate options.

```
icc2_shell> write_pin_constraints -from_existing_pins \
-topological_map {side offset layer} \
-file_name pin_constraints.map
```

Use the `write_pin_constraints` command to write out topological and physical pin constraints for the current block. To limit the output to a specified set of pins, nets, ports, cells, or bundles, use the `-pins pins`, `-nets nets`, `-ports ports`, `-cells cells`, or

`-bundles bundles` option. Use the `-exclude_nets` option to exclude specified nets from the list of pins constraints written by this command. Use the `-self` option to write out pin constraints for the top-level ports. Specify the output file name with the `-file_name` option.

Use the `-topological_map {side layer offset offset_range offset_range layer_range layer_range}` option to write out only the specified fields to the topological map section of the constraint file. If you specify the `offset_range` argument, the command adds the `offset_range` value to, and subtracts the `offset_range` value from, the current offset position. This provides additional flexibility for the pin placer during incremental pin placement. For example, if the current offset is 20 and you specify `-topological_map {offset offset_range 5}`, the command writes the offset portion of the constraint as `{offset {15 25}}`. If you specify the `layer_range` argument, the command adds layers above and below the current layer to the output constraint. For example, if the current layer is M4 and you specify `-topological_map {layer layer_range 2}`, the command adds two layers above and two layers below the current pin layer and writes the layers portion of the constraint as `{layers {M2 M3 M4 M5 M6}}`.

In the following example, the command writes out the topological map section of the constraints file, including net name, pin name, cell name, side, layer range, and offset range for each constraint. The constraints are based on the existing pin placement. The `pin_constraints.map` generated by this command is also shown.

```
icc2_shell> write_pin_constraints -from_existing_pins \
 -file_name pin_constraints.map \
 -topological_map {side offset offset_range 5 layer layer_range 1}
1
icc2_shell> sh head pin_constraints.map
START TOPOLOGICAL MAP;
{n832}
 {{pin u_m/U1192/Y}} {{cell u0_0}}
 {layers {M5 M6 M7}} {sides 4}
 {offset {756.280 766.280}}};
{n1319}
 {{pin U45/Y}} {{cell u0_0}}
 {layers {M5 M6 M7}} {sides 4}
 {offset {1013.464 1023.464}}};
...

```

Use the `-physical_pin_constraint {side layer offset offset_range offset_range layer_range layer_range}` option to write out only the specified fields to the physical pin constraints section of the constraint file. The `offset_range` and `layer_range` arguments have the same meaning as in the topological map section.

In the following example, the command writes out the physical pin constraints section of the constraints file, including pin name, reference name, side, layer, and offset range for each constraint. The constraints are based on the existing pin placement. The `physical_constraints.map` generated by this command is also shown.

```
icc2_shell> write_pin_constraints \
 -file_name physical_constraints.map -from_existing_pins \
 -physical_pin_constraint {side offset offset_range 5 layer}
1
icc2_shell> sh cat physical_constraints.map
START PHYSICAL PIN CONSTRAINTS;
{pins clk} {reference block1}
 {layers M5} {sides 3} {offset {694.720 704.720}};
{pins rstn} {reference block1}
 {layers M6} {sides 4} {offset {756.280 766.280}};
{pins ahbi[0]} {reference block1}
 {layers M2} {sides 4} {offset {171.536 181.536}};
{pins ahbi[1]} {reference block1}
 {layers M4} {sides 4} {offset {171.384 181.384}};
...
...
```

## Writing Out Feedthrough Information as Tcl Commands

The `place_pins` command places pins and creates feedthroughs in blocks, and the `push_down_objects` command creates feedthroughs and moves objects into blocks from a higher level of hierarchy. When these commands add a new object to a block, the command sets the `is_shadow` attribute on these objects to `true`. To write out a script that contains `create_net`, `create_pin`, and `connect_net` commands to re-create the feedthrough topology, use the `write_shadow_eco` command as shown in the following example. The example also shows the first few lines of the script written by the command; you can source this script in another session of the IC Compiler II tool to re-create the feedthroughs. Use the `-command_style dc` option to write out a script that is compatible with the Design Compiler tool.

```
icc2_shell> write_shadow_eco -command_style icc2 \
 -output feedthroughs.tcl
1

icc2_shell> sh cat feedthroughs.tcl
original net: si[31]
create_net U1/si_FEEDTHRU_0[31]
disconnect_net -net U1/si[31] [get_pins U1/u0_3/si[118]]
connect_net -net U1/si_FEEDTHRU_0[31] [get_pins U1/u0_3/si[118]]
create_pin -direction out U1/u0_2/si_FEEDTHRU_1[31]
connect_net -net U1/si_FEEDTHRU_0[31] \
 [get_pins U1/u0_2/si_FEEDTHRU_1[31]]
connect_net -net U1/u0_2/si[118] \
 [get_pins U1/u0_2/si_FEEDTHRU_1[31]]
...
...
```

When multiple `place_pins` or `push_down_objects` commands are used in the same tool session, the `write_shadow_eco` command writes out commands to re-create feedthroughs for the entire session. You can also write out shadow ECO commands from specific `place_pins` or `push_down_objects` commands in the session. First,

set the `plan.pins.enable_feedthrough_timestamps` application option to `true` at the beginning of the session. After running the `place_pins` or `push_down_objects` commands, specify the `-latest_rounds {rounds}` option with the `write_shadow_eco` command. The `rounds` argument specifies the `place_pins` or `push_down_objects` command most recent occurrence count, where “1” is the most recent `place_pins` or `push_down_objects` command, “2” is the second most recent command, and so on.

For example, to write out Tcl commands to re-create the feedthroughs from the two most `place_pins` commands and ignore earlier `place_pins` commands, specify the `-latest_rounds {1 2}` option as follows. Note that the feedthroughs created by the first `place_pins` command are not included in the `feedthroughs.tcl` file.

```
icc2_shell> set_app_options \
 -name plan.pins.enable_feedthrough_timestamps -value true
icc2_shell> place_pins -nets A # latest round 3
icc2_shell> place_pins -nets B # latest round 2
icc2_shell> place_pins -nets C # latest round 1
icc2_shell> write_shadow_eco -output feedthroughs.tcl \
 -latest_rounds {1 2}
```

## Creating Topological Pin Constraints in the GUI

You create topological pin constraints by using Tcl commands and constraint files, or by drawing them in the GUI. Use the GUI to view and examine the current topological constraints set with the `create_topological_constraint` command or with the Topological Pin Constraints tool in the GUI.

To create a topological pin constraint with a Tcl command, specify the `create_topological_constraint` command with the net name, starting and ending block, starting and ending sides, and so on as shown in the following example:

```
icc2_shell> create_topological_constraint -start_object I_DATA_PATH \
 -start_sides 3 -end_object I_ALU \
 -end_sides 4 [get_nets Op_Result[0]]
```

To create the same constraint in the GUI by drawing it on the layout,

1. Start the Topological Pin Constraints tool by choosing Create > Topological Pin Constraint from the menu.
2. Enter the net name in the net name box in the Topological Pin Constraints tool. In this example, the net name is `Op_Result[0]`.

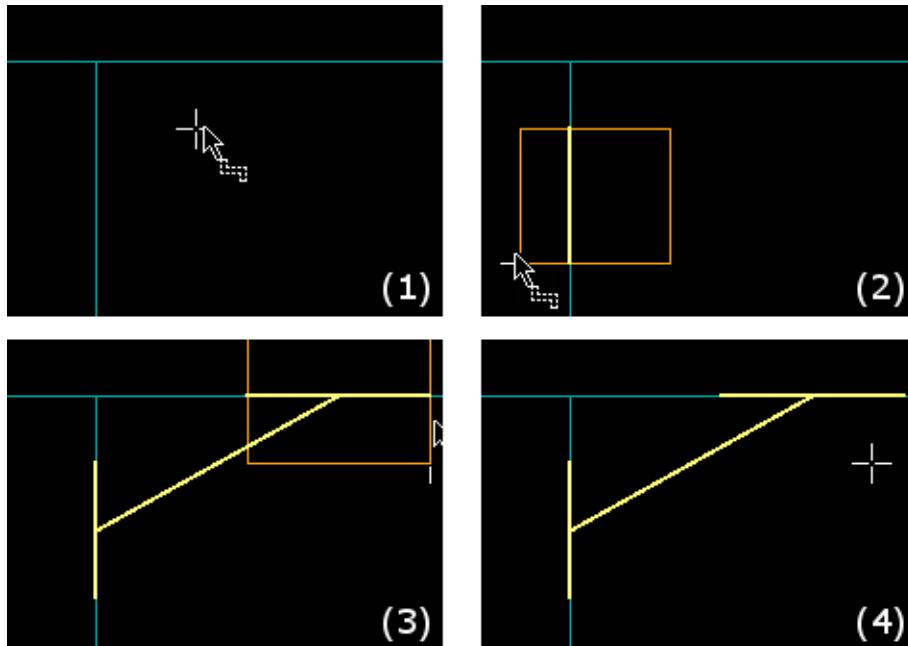
You can also select the net before starting the tool to automatically populate the net name box.

3. (Optional) Specify the allowed layers for the net by selecting them in the Topological Pin Constraints toolbar.

4. Draw a rectangle to define the topological constraint for the start block as shown in step (2) in the following figure.
5. Draw a rectangle to define the topological constraint for the end block as shown in step (3).

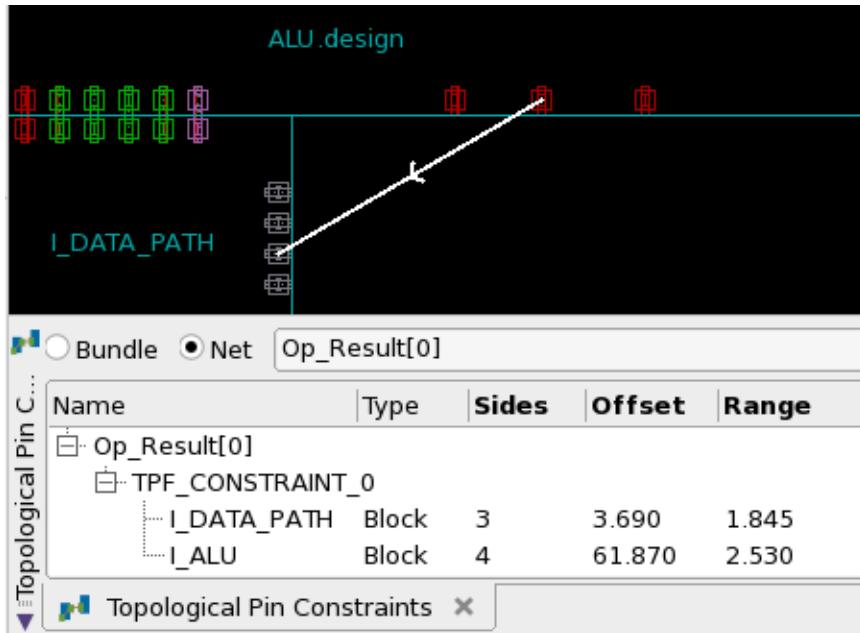
This completes the constraint as shown in step (4) in the following figure.

*Figure 150 Creating a Topological Constraint*



After defining the constraint, use the `place_pins` command to place the pins. The following figure shows the layout after running the `place_pins` command with the topological pin constraint.

Figure 151 Pin Placement Result After Defining a Topological Constraint



After defining a topological constraint, you can perform the following operations:

- Retrieve the currently defined topological constraints with the `get_topological_constraints` command.
- Write a report that describes the currently defined constraints with the `report_topological_constraints` command.

```
icc2_shell> report_topological_constraints -all
TPF Constraint Description

TPF_CONSTRAINT_0 Owner Type blk_net
 Owner Name Op_Result[0]
 Start Type blk_inst
 Start Name I_DATA_PATH
 End Type blk_inst
 End Name I_ALU
...

```

- Edit the allowed sides, offset, range, and layer specification for the constraint by clicking on the cell in the constraint table and modifying the contents of the cell.
- Remove a constraint by clicking the right mouse key over the constraint in the table and choosing Delete from the context menu.
- Remove a constraint by specifying the `remove_topological_constraints` command with the constraint name.

---

## Reporting Estimated Wire Length

After placing the pins with the `place_pins` command, report the wire length for individual nets or for the entire design with the `get_estimated_wirelength` command. The command produces a wire length estimate by creating a virtual route for the net across all design hierarchies. Use the `set_editability` command to limit the design hierarchies considered by the `get_estimated_wirelength` command. The `get_estimated_wirelength` command should be used after placing block pins with the `place_pins` command.

The following example reports the estimated wire length for the entire design and for only the `din[0]` and `dout[1]` nets.

```
icc2_shell> get_estimated_wirelength
Total wirelength of net is: 4117.640
1

icc2_shell> get_estimated_wirelength -nets [get_nets {din[0] dout[0]}]
The length of net din[0] is: 138.478
The length of net dout[0] is: 137.485
1
```

# 14

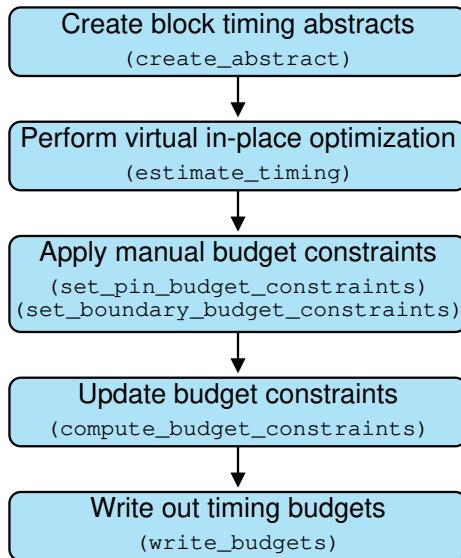
## Performing Timing Budgeting

---

To support a hierarchical design flow, the IC Compiler II tool provides timing budgeting to allocate timing among the blocks in the design. The budgeting process begins by partitioning the chip-level timing constraints into top-level and block-level constraints. To speed the budget creation process, the tool creates lightweight timing abstract representations for the blocks in the design. Each block is virtually optimized to negate the effects of high-fanout nets and provide more accurate timing estimations. The timing budgeter uses the optimized timing for each block to derive new budgets. In the final step, top-level and block-level timing budgets are written out in preparation for further block-level optimization. The timing budgeter is fully multimode and multicorner aware, and handles designs with multiply instantiated blocks.

The flow to perform timing budgeting is shown in [Figure 152](#).

*Figure 152 Timing Budget Flow*



[Figure 152](#)

To generate timing budgets, chip-level Synopsys Design Constraints (SDC) and UPF files must be partitioned into top-level and block-level files. The IC Compiler II tool uses the top-

level constraints to constrain the top-level logic when abstract representations of blocks are used. See [Split Constraints Flow](#) for more information about partitioning chip-level constraint and UPF files.

You typically perform timing budgeting on a design with module pins that are already placed. Note that timing budgeting step requires separate constraints for the top-level and block-level logic. See [Split Constraints Flow](#) for more information about splitting chip-level constraints into top-level and block-level constraints.

To efficiently run the same task on several blocks in your design, you can use the `run_block_script` command to enable distributed processing and perform the tasks in parallel. The `run_block_script` command accepts a Tcl script and applies the commands in the script to the blocks you specify. For more information about running and monitoring parallel tasks, see [Running Tasks in Parallel](#) and [Monitoring Distributed Tasks](#).

For more information, see the following topics on budgeting:

- [Creating Block Timing Abstracts](#)
- [Performing Virtual In-Place Optimization](#)
- [Performing Incremental Virtual In-Place Optimization](#)
- [Applying Manual Budget Constraints](#)
- [Constraining Layers, Delays, and Partial Routes for Timing Estimation](#)
- [Updating Budget Information](#)
- [Writing Out Budget Information](#)

## Creating Block Timing Abstracts

To update timing budgets quickly and efficiently in the IC Compiler II tool, use timing abstract representations for the blocks in your design. Timing abstracts are smaller, more efficient block representations that are useful for generating timing budgets. The timing abstracts are created from blocks with pins already placed. There is no need to re-create cell placement within the blocks before creating the timing abstracts. To create block timing abstracts in parallel by using distributed computing, see [Running Tasks in Parallel](#).

To create a block timing abstract,

1. Open the design representation for the block with the `open_block` command.

```
icc2_shell> open_block lib:ORCA.design
```

2. Merge any changes that were made to the abstract view into the full design view with the `merge_abstract` command.

```
icc2_shell> merge_abstract
```

If you placed the pins on your block by using an abstract representation, you must use the `merge_abstract` command to copy the pin information into the full design view.

3. Source the constraints for the block.

```
icc2_shell> source split/ORCA/top.tcl
```

Note that this step applies internal constraints for the block and is run only one time. You must apply either manually generated constraints or constraints generated by the `split_constraints` command before applying timing budgets. If the constraints were generated by using the `split_constraints` command, the `top.tcl` constraints file is stored in the `split/block_name` directory.

4. Insert feedthrough buffers with the `add_feedthrough_buffers` command.

```
icc2_shell> add_feedthrough_buffers
```

5. Create the block timing abstract with the `create_abstract` command and include the `-estimate_timing` option.

```
icc2_shell> create_abstract -estimate_timing
```

**Note:**

If your design contains multiple levels of physical hierarchy, the `create_abstract` command converts only blocks at the lowest level of hierarchy into abstract views; intermediate levels of hierarchy are kept as design views.

6. Save the changes to the design.

```
icc2_shell> save_lib
```

The `create_abstract -estimate_timing` command runs virtual timing optimization on the interface logic and annotates the abstract view with the result. You can omit the `-estimate_timing` option if you performed virtual timing optimization earlier in the flow.

Create separate abstracts for the black box blocks in your design by using the following steps:

1. Open the design representation for the block with the `open_block` command.

```
open_block lib:cpu.design
```

2. Define the blocks that do not use black boxes and create a list of block references.

```
set non_bb_blocks $DP_BLOCK_REFS
foreach bb $DP_BB_BLOCK_REFS {
 set idx [lsearch -exact $non_bb_blocks $bb]
```

```
 set non_bb_blocks [lreplace $non_bb_blocks $idx $idx]
 }
```

**3. Create a list of instances that do not use black boxes.**

```
set non_bb_insts ""
foreach ref $non_bb_blocks {
 set non_bb_insts "$non_bb_insts [get_object_name \
[get_cells -hierarchical -filter ref_name==$ref]]"
}
```

**4. Identify all black box instances at the lowest level of hierarchy.**

```
set non_bb_for_abs [get_attribute \
-objects [filter_collection [get_cells $non_bb_insts] \
!has_child_physical_hierarchy] -name ref_name]
```

**5. Create abstracts for all non black boxes at the lowest hierarchy levels.**

```
create_abstract -estimate_timing -blocks [list $non_bb_for_abs]
```

**6. Load constraints and create abstracts for black box blocks.**

```
load_block_constraints -blocks [list $DP_BB_BLOCK_REFS] -type SDC
create_abstract -blocks [list $DP_BB_BLOCK_REFS]
```

You can use the `report_abstracts` command to report information about the abstract representations, including leaf cell count, compression percentage, and available modes and corners as shown in the following example.

```
icc2_shell> report_abstracts

Report : report_abstracts
Design : ORCA

Block Instances
I_ORCA_TOP/I_BLENDER_2: ref=BLENDER_6:BLENDER_6/dp.abstract,
 level=1, design_view_only=false, ok_to_abstract=true
...

Total 7 instances of 7 abstract view designs found in current design.

Abstract Design BLENDER_6/dp
Library name: .../BLENDER_6
Design Type: module
Abstract Type: timing
Version: 6.0
Editable: true
Number of leaf cells: 258
Number of additional constrained cells kept: 0
Number of additional constrained hierarchical pins kept: 0
Number of leaf cells in design view: 5146
Compression percentage: 94.99
```

```

Number of ports: 166
Number of PG ports: 0

Available modes and corners

Total 1 modes:
 Mode s1.mode has 2 corners:
 corner estimated_corner
 corner s1.corner
Total 2 corners:
 Corner s1.corner has 1 modes:
 mode s1.mode
 Corner estimated_corner has 1 modes:
 mode s1.mode
...
Top Level
Total 7 out of 7 blocks are abstracts
Number of leaf cells in all abstract blocks: 1814
Number of leaf cells at top level: 11793
Number of leaf cells in current netlist: 13607
Number of leaf cells in full netlist: 42970
Compression percentage: 68.33

Summary
Abstract_Design Design_Type Abstract_Type ...
BLENDER_6:BLENDER_6/dp module timing,estimated_timing ...
...

```

## Performing Virtual In-Place Optimization

Virtual in-place optimization optimizes the top-level and block interface logic in the design without changing the design netlist. The optimization begins by examining the timing between top-level I/O and block interfaces. To improve timing, the tool identifies nets with a large number of fanouts, then resizes drivers and inserts buffers as needed to improve timing. The changes are not permanent and the tool only annotates the design with the new timing information for later processing. Using this technique, the tool can create more accurate timing budgets earlier in the design process without requiring a full design optimization.

The `estimate_timing` command optimizes top-level logic, whether or not block-level designs are editable. Whether a block can be edited or not is determined by the `set_editability` setting for the block. The command optimizes editable blocks and all block boundary nets that cross editable blocks or read-only blocks. The `estimate_timing` command avoids optimizing cells or nets that are completely inside a noneditable block.

The `estimate_timing` command also checks whether you have defined any unified topology constraints on a net. If they are defined, the command uses this information to determine the delay of the net. Check the delay value and if it does not meet your timing requirements, redefine the unified topology constraints and rerun the `estimate_timing`

command. Repeat this process until you get the required results. For more information about unified topology constraints, see [Topology Interconnect Planning](#). Defining unified topology constraints helps you in routing top-level signals in a complex SoC design where there are space constraints. If unified topology constraints are not defined, the tool uses the normal buffering technique to estimate the delay. To determine whether unified topology constraints have been used by the `estimate_timing` command, query the `estimated_analysis` attribute of a pin as follows:

```
icc2_shell> get_attribute -name estimated_analysis \
 -objects [get_pins PIN2]
```

To perform virtual in-place optimization on the top-level design and interface logic,

1. Open the top-level design.

```
icc2_shell> open_block orca:ORCA
```

2. Set the constraint mapping file.

```
icc2_shell> set_constraint_mapping_file split/mapfile
```

The constraint mapping file specifies a list of blocks and their associated constraint files as follows:

```
icc2_shell> sh cat split/mapfile
BLENDER_0 CLKNET BLENDER_0/clocknets.tcl
BLENDER_0 SDC BLENDER_0/top.tcl
BLENDER_1 CLKNET BLENDER_1/clocknets.tcl
BLENDER_1 SDC BLENDER_1/top.tcl
BLENDER_2 BTM BLENDER_2/blender.btm
...
ORCA CLKNET ORCA/clocknets.tcl
ORCA SDC ORCA/top.tcl
```

You can create the constraint mapping file in a text editor or by running the `split_constraints` command. See [Split Constraints Output Files](#) for more information about the constraint files generated by the `split_constraints` command.

3. Source the top-level constraints created by the `split_constraints` command.

```
icc2_shell> source ./split/ORCA/top.tcl
```

The top-level constraints, such as clock definitions and exceptions, are necessary to constrain the virtual in-place optimization. Note that the top-level constraints were previously saved to the `split/design_name/top.tcl` file by the `split_constraints` command. If you are using the full netlist flow, source the full-chip SDC file instead of the top-level SDC file.

4. Run virtual in-place optimization to optimize the top-level and interface logic.

```
icc2_shell> estimate_timing
```

The `estimate_timing` command creates a new corner, `estimated_corner`, and optimizes the top-level interface logic by using the block-level timing abstracts. The `estimated_corner` is an artificial corner which aggregates the worst delays from multiple corners. The timing in these scenarios is stored as timing annotations. If corner A is slow for some paths and corner B is slow for other paths, the `mode::estimated_corner` is annotated with information from both corner A and corner B.

If you have defined unified topology constraints on a net, this command uses this information to estimate the timing delay and adds a message to the Analysis column of the timing report (which is generated when you run the `report_timing` command).

In the following example, `topo_1` is the unified topology constraint defined on the net and 0.1384 is the delay estimated based on unified topology constraint.

| Point | Incr     | Path     | Analysis                             |
|-------|----------|----------|--------------------------------------|
| Pin1  | 0417 e   | 0.0417 r | Size: ANDX                           |
| Pin2  | 0.1384 e | 0.1800 r | Buff: Using topology topo_1 is added |

In the following example, the `estimate_timing` command takes into account the delay in all the buffers for the topology interconnect planning-defined net, N1.

| Topology interconnect planning- defined net with existing cells in the design | Timing path delay information                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |                                                    |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
|-------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|----------------------------------------------------|------|----------|------------|------|------|--|----------|--------|------|-----------|--------|--------|------|---------------------------------|--------|--------|------|--|--------|--------|------|---------------------------------|--------|--------|------|--|--------|--------|------|---------------------------------|--------|--------|------|--|----------|---------|-------|----------------------------------------------------|
|                                                                               | <table border="1"> <thead> <tr> <th>Point</th><th>Incr</th><th>Path</th><th>Analysis</th></tr> </thead> <tbody> <tr> <td>A/FF_A/clk</td><td>0.00</td><td>0.00</td><td></td></tr> <tr> <td>A/FF_A/Q</td><td>0.02 e</td><td>0.02</td><td>Size: FF1</td></tr> <tr> <td>Buf1/A</td><td>0.00 e</td><td>0.02</td><td>Buff: Using topology plan added</td></tr> <tr> <td>Buf1/Y</td><td>0.00 e</td><td>0.02</td><td></td></tr> <tr> <td>Buf2/A</td><td>0.00 e</td><td>0.02</td><td>Buff: Using topology plan added</td></tr> <tr> <td>Buf2/Y</td><td>0.00 e</td><td>0.02</td><td></td></tr> <tr> <td>Buf3/A</td><td>0.00 e</td><td>0.02</td><td>Buff: Using topology plan added</td></tr> <tr> <td>Buf3/Y</td><td>0.00 e</td><td>0.02</td><td></td></tr> <tr> <td>B/FF_B/D</td><td>13.71 e</td><td>13.73</td><td>Buff: Using topology plan TOPOLOGY_PLAN_* is added</td></tr> </tbody> </table> | Point | Incr                                               | Path | Analysis | A/FF_A/clk | 0.00 | 0.00 |  | A/FF_A/Q | 0.02 e | 0.02 | Size: FF1 | Buf1/A | 0.00 e | 0.02 | Buff: Using topology plan added | Buf1/Y | 0.00 e | 0.02 |  | Buf2/A | 0.00 e | 0.02 | Buff: Using topology plan added | Buf2/Y | 0.00 e | 0.02 |  | Buf3/A | 0.00 e | 0.02 | Buff: Using topology plan added | Buf3/Y | 0.00 e | 0.02 |  | B/FF_B/D | 13.71 e | 13.73 | Buff: Using topology plan TOPOLOGY_PLAN_* is added |
| Point                                                                         | Incr                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Path  | Analysis                                           |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| A/FF_A/clk                                                                    | 0.00                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 0.00  |                                                    |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| A/FF_A/Q                                                                      | 0.02 e                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 0.02  | Size: FF1                                          |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| Buf1/A                                                                        | 0.00 e                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 0.02  | Buff: Using topology plan added                    |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| Buf1/Y                                                                        | 0.00 e                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 0.02  |                                                    |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| Buf2/A                                                                        | 0.00 e                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 0.02  | Buff: Using topology plan added                    |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| Buf2/Y                                                                        | 0.00 e                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 0.02  |                                                    |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| Buf3/A                                                                        | 0.00 e                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 0.02  | Buff: Using topology plan added                    |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| Buf3/Y                                                                        | 0.00 e                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 0.02  |                                                    |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |
| B/FF_B/D                                                                      | 13.71 e                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 13.73 | Buff: Using topology plan TOPOLOGY_PLAN_* is added |      |          |            |      |      |  |          |        |      |           |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |        |        |      |                                 |        |        |      |  |          |         |       |                                                    |

In the following example, the timing path information for multiple fanout nets is shown.

| Topology interconnect planning- defined net with existing cells in the design                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Timing path delay information |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------|--|-------|-------|------|----------|------------|------------|------|------|----------|----------|--------|-----------|-----------|--------|--------|---------------------------------|---------------------------------|--------|--------|------|--------|--------|--------|---------------------------------|---------------------------------|--------|--------|------|--------|--------|--------|---------------------------------|---------------------------------|--------|--------|------|----------|----------|---------|----------------------------------------------------|----------------------------------------------------|
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                               | <b>The timing path from A to G:</b> <table> <thead> <tr> <th>Point</th> <th>Incr</th> <th>Path</th> <th>Analysis</th> </tr> </thead> <tbody> <tr> <td>A/FF_A/clk</td> <td>0.00</td> <td>0.00</td> <td></td> </tr> <tr> <td>A/FF_A/Q</td> <td>0.02 e</td> <td>0.02</td> <td>Size: FF1</td> </tr> <tr> <td>Buf1/A</td> <td>0.00 e</td> <td>0.02</td> <td>Buff: Using topology plan added</td> </tr> <tr> <td>Buf1/Y</td> <td>0.00 e</td> <td>0.02</td> <td></td> </tr> <tr> <td>Buf2/A</td> <td>0.00 e</td> <td>0.02</td> <td>Buff: Using topology plan added</td> </tr> <tr> <td>Buf2/Y</td> <td>0.00 e</td> <td>0.02</td> <td></td> </tr> <tr> <td>Buf3/A</td> <td>0.00 e</td> <td>0.02</td> <td>Buff: Using topology plan added</td> </tr> <tr> <td>Buf3/Y</td> <td>0.00 e</td> <td>0.02</td> <td></td> </tr> <tr> <td>G/FF_G/D</td> <td>10.02 e</td> <td>10.04</td> <td>Buff: Using topology plan TOPOLOGY_PLAN_* is added</td> </tr> </tbody> </table> |                                                    |  |       | Point | Incr | Path     | Analysis   | A/FF_A/clk | 0.00 | 0.00 |          | A/FF_A/Q | 0.02 e | 0.02      | Size: FF1 | Buf1/A | 0.00 e | 0.02                            | Buff: Using topology plan added | Buf1/Y | 0.00 e | 0.02 |        | Buf2/A | 0.00 e | 0.02                            | Buff: Using topology plan added | Buf2/Y | 0.00 e | 0.02 |        | Buf3/A | 0.00 e | 0.02                            | Buff: Using topology plan added | Buf3/Y | 0.00 e | 0.02 |          | G/FF_G/D | 10.02 e | 10.04                                              | Buff: Using topology plan TOPOLOGY_PLAN_* is added |
| Point                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Incr                          | Path                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Analysis                                           |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| A/FF_A/clk                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 0.00                          | 0.00                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| A/FF_A/Q                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 0.02 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Size: FF1                                          |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf1/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Buff: Using topology plan added                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf1/Y                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf2/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Buff: Using topology plan added                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf2/Y                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf3/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Buff: Using topology plan added                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf3/Y                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| G/FF_G/D                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 10.02 e                       | 10.04                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Buff: Using topology plan TOPOLOGY_PLAN_* is added |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| <b>The timing path from A to H:</b> <table> <thead> <tr> <th>Point</th> <th>Incr</th> <th>Path</th> <th>Analysis</th> </tr> </thead> <tbody> <tr> <td>A/FF_A/clk</td> <td>0.00</td> <td>0.00</td> <td></td> </tr> <tr> <td>A/FF_A/Q</td> <td>0.02 e</td> <td>0.02</td> <td>Size: FF1</td> </tr> <tr> <td>Buf1/A</td> <td>0.00 e</td> <td>0.02</td> <td>Buff: Using topology plan added</td> </tr> <tr> <td>Buf1/Y</td> <td>0.00 e</td> <td>0.02</td> <td></td> </tr> <tr> <td>Buf2/A</td> <td>0.00 e</td> <td>0.02</td> <td>Buff: Using topology plan added</td> </tr> <tr> <td>Buf2/Y</td> <td>0.00 e</td> <td>0.02</td> <td></td> </tr> <tr> <td>Buf3/A</td> <td>0.00 e</td> <td>0.02</td> <td>Buff: Using topology plan added</td> </tr> <tr> <td>Buf3/Y</td> <td>0.00 e</td> <td>0.02</td> <td></td> </tr> <tr> <td>H/FF_H/D</td> <td>8.15 e</td> <td>8.17</td> <td>Buff: Using topology plan TOPOLOGY_PLAN_* is added</td> </tr> </tbody> </table> |                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                    |  | Point | Incr  | Path | Analysis | A/FF_A/clk | 0.00       | 0.00 |      | A/FF_A/Q | 0.02 e   | 0.02   | Size: FF1 | Buf1/A    | 0.00 e | 0.02   | Buff: Using topology plan added | Buf1/Y                          | 0.00 e | 0.02   |      | Buf2/A | 0.00 e | 0.02   | Buff: Using topology plan added | Buf2/Y                          | 0.00 e | 0.02   |      | Buf3/A | 0.00 e | 0.02   | Buff: Using topology plan added | Buf3/Y                          | 0.00 e | 0.02   |      | H/FF_H/D | 8.15 e   | 8.17    | Buff: Using topology plan TOPOLOGY_PLAN_* is added |                                                    |
| Point                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Incr                          | Path                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Analysis                                           |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| A/FF_A/clk                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 0.00                          | 0.00                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| A/FF_A/Q                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 0.02 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Size: FF1                                          |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf1/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Buff: Using topology plan added                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf1/Y                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf2/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Buff: Using topology plan added                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf2/Y                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf3/A                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Buff: Using topology plan added                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| Buf3/Y                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0.00 e                        | 0.02                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                    |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |
| H/FF_H/D                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 8.15 e                        | 8.17                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Buff: Using topology plan TOPOLOGY_PLAN_* is added |  |       |       |      |          |            |            |      |      |          |          |        |           |           |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |        |        |        |                                 |                                 |        |        |      |          |          |         |                                                    |                                                    |

The `estimate_timing` command should be run before the budgeting step, as the timing annotations in these new scenarios are used for budgeting. For a multicorner-multimode design, budgets are generated for all corners, modes, and scenarios (both corner A and B in the previous example).

To run the virtual in-place optimization task more efficiently on large designs, you can use the `-host_options option_name` option to enable distributed processing. To achieve good quality-of-results, the timing constraints must be reasonable.

5. Perform a timing check to identify any large top-to-block or block-to-top timing violations.

```
icc2_shell> report_timing -modes [all_modes] -corners estimated_corner
```

After running the `estimate_timing` and `report_timing` commands, the timing report now contains timing information for the new corner named `estimated_corner`.

The paths reported for `estimated_corner` are annotated with additional information that indicate how the tool calculated the timing values:

- "E" or "e" indicate that the timing value is generated based on the `estimate_timing` command.
- "A" indicates that the timing value is generated based on the `create_abstract -estimate_timing` command.
- "a" indicates that the timing value is generated based on static timing analysis from the `create_abstract -estimate_timing` command.
- "S" indicates that the timing value originates from an SPEF or SDF file.

The “Delta Incr” column shows the difference in delay for each output on the path between the current corner and the `estimated_corner` corner. The “Analysis” column reports any changes made to the net during estimated timing optimization. The following examples are possible changes made by the `estimate_timing` command.

- Size: None no changes were made
- Size: BUF1 The buffer was resized; the new buffer is BUF1
- Buff: Short net The net is too short and buffering cannot improve timing
- Buff: 3 BUF1 added Three BUF1 cells were added to the net

The following is a short example of a timing report, with annotations for `estimated_corner`:

```
icc2_shell> report_timing -modes [all_modes] -corners estimated_corner

Report : timing -path_type full -delay_type max
 -max_paths 1 -report_by design
Options: --
Design : ORCA
Version:
Date : Wed May 25 15:44:30 2016

Startpoint: I_DATA_PATH/I_0_ (...)
Endpoint: I_ALU/Lachd_Result_reg_15_ (...)
Mode: s1.mode
Corner: estimated_corner
```

```

Scenario: s1.mode::estimated_corner
Path Group: CLK
Path Type: max

Point Incr Path Delta Analysis
 Incr

clock CLK (rise edge) 0.00 0.00
clock network delay (ideal) 0.00 0.00

I_DATA_PATH/I_0/_CP (senrq1) 0.00 0.00 r 0.00
I_DATA_PATH/I_0/_Q (senrq1) 0.36 0.36 f 0.02 Size: None
I_ALU/U56/Z (bufbd4) 0.14 e 0.50 f -0.01 Size: bufbd7
I_ALU/U34/ZN (inv0d4) 0.05 e 0.56 r -0.00 Size: invbdf
I_ALU/U36/ZN (invbd2) 0.06 e 0.62 f -0.03 Size: invbda

```

6. (Optional) Override the estimated delay on a specific timing arc with the `set_annotationed_delay` command. Use this feature to fine-tune the timing values from virtual in-place optimization.

```
icc2_shell> set_annotationed_delay -from b3/I -to b3/Z 4.321 -cell \
 -corners estimated_corner
```

The `-cell` option of the `set_annotationed_delay` command is used to override cell timing arcs; the `-net` option is used to override net timing arcs.

7. (Optional) View the updated timing result with the `report_timing` command.

```
icc2_shell> report_timing -corners estimated_corner
```

For delay values annotated with the `set_annotationed_delay` command, the tool labels the values for the `estimated_corner` by replacing the "e" annotation with an asterisk (\*). The following example shows the original estimated timing value in the timing report for the b3 inverter and the result after running the `set_annotationed_delay` and `report_timing` commands.

```

before
b3/I (BUFFHVT0) 0.0000 e 0.1584 r 0.0000 Buff: Short net
b3/Z (BUFFHVT0) 0.0327 e 0.1911 r ~ -0.0064 Size: CKBXHVTD3
b4/I (BUFFHVT0) 0.0000 e 0.1911 r 0.0000 Buff: Short net

after
b3/I (BUFFHVT0) 0.0000 e 0.1584 r 0.0000 Buff: Short net
b3/Z (BUFFHVT0) 4.3210 * 4.4794 r ~ 4.2819 Size: CKBXHVTD3
b4/I (BUFFHVT0) 0.0000 4.4794 r 0.0000 Buff: Short net

```

8. (Optional) Rerun timing estimation with the `estimate_timing` command and report the updated timing with the `report_timing` command

If the design contains user-specified delays set with the `set_annotation_delay` command, the `estimate_timing` command uses them and the `report_timing` command reports the affected delay values with an asterisk (\*) as shown in the previous step.

9. (Optional) Remove user-annotated delays with the `remove_annotation_delay` command.

```
icc2_shell> remove_annotation_delay -from b3/I -to b3/z \
 -corners estimated_corner
```

## Performing Incremental Virtual In-Place Optimization

By default, virtual in-place optimization operates on the current design and all subblocks. Each time you run virtual in-place optimization, all annotated timing data for the previous run is removed and the tool recalculates new timing information.

To run virtual in-place optimization on only a subset of the design, use the `-nets` or the `-pins` option with the `estimate_timing` command to improve runtime and avoid recalculating the timing for the entire design. When either the `-nets` or `-pins` option is specified, virtual in-place optimization updates just the delay and capacitance annotations for the specified nets or pins. Other annotations generated by the previous `estimate_timing` run are retained.

To run incremental virtual in-place optimization,

1. (Optional) If the design contains abstracts, create a timing abstract for each block design with the `create_abstract` command.

```
icc2_shell> create_abstract -estimate_timing -all_blocks
```

2. Run virtual in-place optimization on the entire design with the `estimate_timing` command.

```
icc2_shell> estimate_timing
...
Virtually optimized 903 out of 13637 cells
Virtually buffered 903 cell outputs
...
```

3. Analyze timing with the `report_timing` command.

4. Adjust the placement of cells on the critical paths, either at the top-level or in the subblocks.

5. (Optional) If the design contains abstracts, re-create the block-level abstracts for the modified design with the `create_abstract` command.

```
icc2_shell> create_abstract -estimate_timing -force_recreate \
-blocks U1
```

6. Perform incremental virtual in-place optimization by specifying a pin or net on the adjusted path.

```
icc2_shell> estimate_timing \
-pins [get_pins I_RESET_BLOCK/sdram_rst_n_buf_reg/Q]
...
Virtually optimized 36 out of 2093 cells
Virtually buffered 36 cell outputs
...
```

Note the difference in the number optimized cells between the full-chip virtual in-place optimization run and the incremental run.

## Applying Manual Budget Constraints

In the timing budgeting flow, you can create the block-level budgets automatically with the tool or manually by writing individual constraints. Using the automatic approach, the `compute_budget_constraints` command derives the constraints based on the gate delays in the design. In the manual approach, you enter the budget constraints and source them before creating the block budgets. You can combine these approaches and augment the automatic constraints with your own manual constraints. By using manual budget constraints, you can create block budgets that are independent of actual circuit timing. This feature provides you with full control when creating the block budgets.

Pin budget constraints are defined with the `set_pin_budget_constraints` command and specify how the timing budgeter allocates the available path delay through a pin.

Pin budget constraints are created automatically for every hierarchical pin in the current design with the `compute_budget_constraints` command. For example, the following command constrains the timing budgeter to assign 40% of the available path delay through pin u0\_0/rstn to the inside of the block.

```
icc2_shell> set_pin_budget_constraints [get_pins u0_0/rstn] \
-internal_percent 40
```

Boundary budget constraints are defined with the `set_boundary_budget_constraints` command and specify boundary conditions for block pins in terms of driving cells and loads. You can associate boundary budget constraints with specific pins by using the `set_pin_budget_constraints` command as shown in the last line of the following example. Boundary budget constraints are also created automatically for every hierarchical pin in the current design with the `compute_budget_constraints` command.

```
icc2_shell> set_boundary_budget_constraints -name OutputLoad \
 -default -load_capacitance 0.15
icc2_shell> set_boundary_budget_constraints -name OutputLoad \
 -corner RCworst -load_capacitance 0.15
icc2_shell> set_boundary_budget_constraints \
 -name driving_cell_buf -driving_cell NBUFFX2_RVT
icc2_shell> set_pin_budget_constraints \
 -late_boundary driving_cell_buf -inputs u0_0/*
```

Latency budget constraints are defined with the `set_latency_budget_constraints` command and define expected or target values for top-level clock latencies. Latency budget constraints are created automatically for each nonvirtual clock in the current circuit when you use the `compute_budget_constraints -latency_targets actual` command. The following example constrains the budgeter to use a latency of 5ns for signal `clk` when performing budgeting calculations.

```
icc2_shell> set_latency_budget_constraints -corner RCworst \
 -early_latency 5 -late_latency 5 -clock clk
```

## Constraining Layers, Delays, and Partial Routes for Timing Estimation

To further control the estimated timing values produced by the `estimate_timing` command, use application options and Tcl commands to do the following:

- Assign layers for RC estimation
- Control layer promotion
- Set a maximum estimated delay for nets and for cells
- Specify whether to use partial routes

### Assign Layers for RC Estimation

To set additional constraints for timing estimation, use the `set_net_estimation_rule` command to specify parameters for the layers to use for RC estimation and other information for the `estimate_timing` command. After creating the net estimation rule, use the `set_attribute` command to associate the new rule with a specific collection of nets.

The following example

1. Creates a new net estimation rule
2. Displays the attributes affected by the `set_net_estimation_rule` command
3. Associates the rule with the `netA` and `netB` nets

```
icc2_shell> set_net_estimation_rule -parameter layer \
 -horizontal_value M7 -vertical_value M8 rule1
icc2_shell> get_attribute [get_net_estimation_rules rule1] h_layer
M7
icc2_shell> get_attribute [get_net_estimation_rules rule1] v_layer
M8
icc2_shell> set_attribute -objects [current_design] \
 -name estimate_timing_net_rule -value default
icc2_shell> set_attribute -objects [get_nets {netA netB}] \
 -name estimate_timing_net_rule -value rule1
{netA netB}
icc2_shell> get_attribute -objects [get_nets netA] \
 -name estimate_timing_net_rule
rule1
icc2_shell> estimate_timing
...
Detected 1 estimate_timing rules on 2 nets.
...
```

### Controlling Layer Promotion During Timing Estimation

The `estimate_timing` command can apply layer promotion to critical nets during virtual in-place optimization. Layer promotion helps to reduce the resistance on critical nets and reduce congestion by routing the net on a higher layer of metal.

To enable layer promotion, set the `plan.estimate_timing.optimize_layers` application option to true. This application option also allows layer promotion when running the `create_abstract` command with the `-estimate_timing` option. When layer promotion is enabled, the global router runs in floorplan mode during timing estimation to improve runtime.

### Specifying Maximum Delays for Timing Estimation

To specify a maximum cell delay and maximum net delay for the `estimate_timing` command, use the `plan.estimate_timing.maximum_net_delay` application option. During timing estimation, calculated cell delays that are larger than the specified maximum delay are replaced by the maximum cell delay. Similarly, calculated net delays that are larger than the specified maximum delay are replaced by the maximum net delay.

The following example sets a maximum net delay of 1.0 and maximum cell delay of 1.0.

```
icc2_shell> set_app_options \
 -name plan.estimate_timing.maximum_net_delay -value 1.0
plan.estimate_timing.maximum_net_delay 1
icc2_shell> set_app_options \
 -name plan.estimate_timing.maximum_cell_delay -value 1.0
plan.estimate_timing.maximum_cell_delay 1
```

After running the `estimate_timing` command, determine which cells and nets were affected by these maximum settings by querying the `is_et_delay_clipped` attribute on

the pins and ports of the design. The attribute is `true` for any pins where the estimated delay for the net or cell was replaced by the application option value. The following example returns the pins of the design where the `is_et_delay_clipped` attribute is `true`.

```
icc2_shell> get_pins -of_objects [get_cells *] \
 -filter is_et_delay_clipped==true
{U0/Z U1/Z U2/Z U3/Z U4/Z ...}
```

### Using Partial Routes During Timing Estimation

The `estimate_timing` command can use partial routes during timing estimation. When using partial routes for nets, the tool estimates only the missing or incomplete net segments and creates a virtual connection. Route segments that are globally routed or detail routed are considered during timing estimation, and existing partial routes are used to guide buffer insertion. To enable this behavior, set the `plan.estimate_timing.honor_routes` application option to `true`. By default, the `estimate_timing` command ignores partial routes.

## Updating Budget Information

After performing virtual in-place optimization on the top-level logic and block-level abstracts, you can generate updated timing budgets for the blocks in your design. Block timing budgets typically contain the following constraints:

- Data path I/O constraints created with the `set_input_delay` and `set_output_delay` commands
- Virtual clocks referenced in the data path I/O constraints
- Virtual clock latencies for source and network
- Real clock latencies for source only
- I/O electrical constraints created with `set_driving_cell`, `set_load`, and other commands

To generate updated timing budgets,

1. Specify the blocks to budget.

```
icc2_shell> set_budget_options -add_blocks {u0_0 u0_1 u0_2 u0_3}
```

2. Update the timing budgets with the `compute_budget_constraints` command.

```
icc2_shell> compute_budget_constraints -estimate_timing
```

The `-estimate_timing` option sets the `plan.budget.estimate_timing_mode` application option to `true` and specifies that budgeting is performed on the `estimated_corner` corner. Based on this application option setting, the `report_budget`

command reports timing information for the estimated\_corner corner. The `compute_budget_constraints` command creates pin budget constraints for every hierarchical pin in the current circuit implementation.

The `compute_budget_constraints` command supports options to fine-tune the budgeting task and update budget values only for specified pins in the design. Using these options, you can iteratively refine the timing budget for your design. To update budgets only for specified pins, use the `-pins pin_list` option. To update budgets only for inputs to blocks, use the `-inputs` option. To update budgets only for block outputs, use the `-outputs` option. To update budgets for pins which have no current budgets, use the `-unspecified` option. To update budgets for feedthrough pins, use the `-feedthrough` option. To update budgets for pins with a slack value less than a specified value, use the `-slack slack_value` option.

Note that you should only use the `-estimate_timing` option when you want to use optimized delay values produced by the `estimate_timing` command. If your design timing is already optimized, run the `compute_budget_constraints` command without the `-estimate_timing` option.

3. Report the updated budgets with the `report_budget` command.

```
icc2_shell> report_budget -html budget_report.html
```

The budget report contains a comprehensive summary of timing budgets for each block specified by the `set_budget_options -add_blocks` command. The HTML-format report contains a list of timing path segments and summary sections for block timing, clock timing, timing to block boundary pins, and path segments. [Figure 153](#) shows the first few lines of the HTML-format budget report generated by the `report_budget` command.

*Figure 153 Budget Report*

**Budget report for design ORCA, mode s1.mode**

Clock summaries  
Path type summaries

```
Segment summary for block I_ORCA_TOP/I_BLENDER_1
Segment summary for block I_ORCA_TOP/I_BLENDER_2
Segment summary for block I_ORCA_TOP/I_BLENDER_3
Segment summary for block I_ORCA_TOP/I_BLENDER_4
Segment summary for block I_ORCA_TOP/I_BLENDER_5
Segment summary for block I_ORCA_TOP/I_BLENDER_6
Segment summary for block I_ORCA_TOP/I_BLENDER_7
```

Segment summary for block I\_ORCA\_TOP/I\_BLENDER\_1

```
* WNS= 0.000 TNS= 0.000 Count=20 NVIO=0 To input pin of block From start point
* WNS= 0.000 TNS= 0.000 Count=32 NVIO=0 From output pin of block To end point
* WNS= 0.000 TNS= 0.000 Count=32 NVIO=0 To output pin of block From start point
* WNS= 0.000 TNS= 0.000 Count=69 NVIO=0 From input pin of block To end point
```

To further investigate timing budgets for individual paths, you can use the `-through pin_name` and `-warning_pins` options with the `report_budget` command. The `report_budget -through pin_name` command reports the budget of the worst-slack path through the specified budget pin and shows the proportion of the delay consumed by each segment.

The following example reports the worst-slack path through the I\_ORCA\_TOP/I\_BLENDER\_1/result[16] pin.

```
icc2_shell> report_budget -through I_ORCA_TOP/I_BLENDER_1/result[16]
Information: Budget for this design will be based
on estimated_corner delays. (ABS-240)

Report : report_budget -through I_ORCA_TOP/I_BLENDER_1/result[16]
Module : ORCA
Mode : s1.mode

Path: (S= 6.387 / D= 1.413(0.54) / B= 7.800)
SYS_CLK.1_r->SYS_CLK.top_r
From startpoint inside of block
Segment (S= 2.747 / D= 0.317(0.00) / B= 3.064) 39%
Through output pin I_ORCA_TOP/I_BLENDER_1/result[16]
Segment (S= 3.640 / D= 1.096(0.54) / B= 4.736) 61%
To endpoint outside of block
```

The `report_budget -warning_pins` command reports additional details about potential problems in the design or timing budget. The report lists pins with seemingly

impossible constraints, pins with missing budget constraints, and pins with no timing paths. The following example uses the `-warning_pins` option to create a report and saves the report to the `warning_pins.txt` file in the local directory.

```
icc2_shell> report_budget -warning_pins > warning_pins.txt
icc2_shell> sh head warning_pins.txt

Report : Budgeted pins with no timing paths
Module : cpu
Mode : func

u0_0/ahbi[31]
u0_0/ahbi[30]
u0_0/ahbi[29]
...
```

To view the reason why there is no timing path for a pin, use the `-verbose` option. Ensure to use this option with the `-warning_pins` option in the `report_budget` command.

Here is an example report built using the `-warning_pins -verbose` options that includes reasons as to why some of the pins do not have timing paths. For information about other possible reasons that cause missing timing paths, see the `report_budget` man page.

| Reason Column           | Description                                                                                                   |
|-------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>clock_pin</code>  | Indicates that the specific pin is part of a clock network                                                    |
| <code>false_path</code> | Indicates that all paths through the specific block pin are covered by <code>set_false_path exceptions</code> |

In the report, the `Pin` column indicates the pin that is causing the lack of timing path through a specific block pin. You can use the information in this column to identify the next steps to fix the issue. For example,

| Reason Column                         | Pin Column | Description                                                                                                                                                                                                                              |
|---------------------------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>unclocked_endpoint</code>       | ABC        | It indicates that pin ABC is not connected to a clock.                                                                                                                                                                                   |
| <code>unconstrained_input_port</code> | XYZ        | <p>It indicates that the XYZ input port is not constrained.</p> <p>If you were expecting constrained timing paths through this port and there are none, then you need to look at the XYZ input port to identify and solve the issue.</p> |

**Note:**

If a pin is bidirectional, it is expected to have at least one input and one output delay. If one of them is missing and there are no timing paths going through the pin in the corresponding direction, then that pin appears in the report.

```
icc2_shell> report_budget -warning_pins -verbose

Report : Pins with no timing paths
Design : TOP_DG
Mode : m2
Version: T-2022.03-BETA
Date : Thu Jan 13 13:26:14 2022


```

| Block       | Pin | Dir.     | Reason     |
|-------------|-----|----------|------------|
| Reason      | Pin |          |            |
| M/clk1_in   | NA  | (input)  | clock_pin  |
| M/clk2_in   | NA  | (input)  | clock_pin  |
| B/clk1_in   | NA  | (input)  | clock_pin  |
| B/clk2_in   | NA  | (input)  | clock_pin  |
| M/A/Pin_0   | NA  | (input)  | false_path |
| M/A/Pout_0  | NA  | (output) | false_path |
| M/A/clk1_in | NA  | (input)  | clock_pin  |
| M/B/clk1_in | NA  | (input)  | clock_pin  |
| M/B/clk2_in | NA  | (input)  | clock_pin  |

...

To generate a report of the feedthroughs that are in the design, use the `-feedthroughs` option with the `report_budget` command. The report includes information about the feedthrough pins and their location as well as the timing and budget for this segment.

The `FROM_FROZEN` and `TO_FROZEN` columns in the feedthrough report indicate whether the budget for the feedthrough is automatically generated by the tool or is defined by a user constraint, `set_pin_budget_constraints -frozen <pin>`. The `IS_FIXED`

column indicates whether the budget for a segment is calculated by the tool or is marked fixed by the user.

In the following example, the feedthrough report is generated for the specified pins.

```
icc2_shell> report_budget -feedthroughs -pins B/Pin_0

Report : report_budget -feedthroughs
Design : TOP_DG
Version: T-2022.03-BETA
Date : Thu Jan 13 19:04:26 2022

FROM_PIN FROM_PIN_LAYER FROM_PIN_LOCATION LAUNCH_CLOCK FROM_FROZEN
 TO_PIN TO_PIN_LAYER TO_PIN_LOCATION CAPTURE_CLOCK TO_FROZEN
IS_FIXED MANHATTAN_DISTANCE ESTIMATION_RULE BUDGET DELAY SLACK
INTENT BUDGET_MINUS_INTENT

B/Pin_0 M2 22.9400 43.3120 clk1 False
B/Pout_0

 M2 22.9400 43.1840 clk1 False
False 0.1280 0.006 0.000 0.006
 0.000 0.006

B/Pin_0 M2 22.9400 43.3120 clk1 False
B/Pout_0

 M2 22.9400 43.1840 clk2 False
False 0.1280 0.006 0.000 0.006
 0.000 0.006

B/Pin_0 M2 22.9400 43.3120 clk2 False
B/Pout_0

 M2 22.9400 43.1840 clk1 False
False 0.1280 0.006 0.000 0.006
 0.000 0.006

B/Pin_0 M2 22.9400 43.3120 clk2 False
B/Pout_0

 M2 22.9400 43.1840 clk2 False
False 0.1280 0.018 0.000 0.017
 0.000 0.017

...

```

To capture the feedthrough report information in a CSV format, use the `-csv` option with the `report_budget -feedthroughs` command. Ensure that you add the `.csv` extension in the command as shown in the following example, where the feedthrough information is saved in the `feedthrough_list.csv` file.

```
icc2_shell> report_budget -feedthroughs -csv feedthrough_list.csv
```

| FROM_PIN                                           | FROM_PIN | FROM_PIN_LOCATION   | LAUNCH_CLOCK | FROM_FROZEN | TO_PIN                                             | TO_PIN_LAYER | TO_PIN_LOCATION     | CAPTURE_CLOCK | TO_FROZEN |
|----------------------------------------------------|----------|---------------------|--------------|-------------|----------------------------------------------------|--------------|---------------------|---------------|-----------|
| _ORCA_TOP /_RISC_CORE/n99_FEEDTHRU_0               | METAL2   | 1223.5500 681.3400  | SDRAM_CLK    | FALSE       | _ORCA_TOP /_RISC_CORE/n99_FEEDTHRU_1               | METAL3       | 1635.7000 616.6900  | SD_DDR_CLK    | FALSE     |
| _ORCA_TOP /_RISC_CORE/n101_FEEDTHRU_0              | METAL3   | 1201.8500 637.2900  | SDRAM_CLK    | FALSE       | _ORCA_TOP /_RISC_CORE/n101_FEEDTHRU_1              | METAL3       | 1635.7000 607.6500  | SD_DDR_CLK    | TRUE      |
| _ORCA_TOP /_CONTEXT_MEM/net_pserr_n_out_FEEDTHRU_0 | METAL3   | 1199.4000 1611.6700 | PCI_CLK      | FALSE       | _ORCA_TOP /_CONTEXT_MEM/net_pserr_n_out_FEEDTHRU_1 | METAL2       | 1545.3800 1631.8100 | PCI_CLK       | FALSE     |
| _ORCA_TOP /_CONTEXT_MEM/net_sdram_BWS_0_FEEDTHRU_0 | METAL2   | 1449.8700 820.2100  | SDRAM_CLK    | TRUE        | _ORCA_TOP /_CONTEXT_MEM/net_sdram_BWS_0_FEEDTHRU_1 | METAL3       | 1635.7000 1168.7700 | SD_DDR_CLK    | FALSE     |
| _ORCA_TOP /_PCI_CORE/n52_FEEDTHRU_0                | METAL2   | 490.6000 997.3300   | PCI_CLK      | FALSE       | _ORCA_TOP /_PCI_CORE/n52_FEEDTHRU_1                | METAL3       | 375.5650 1257.3500  | PCI_CLK       | FALSE     |
| _ORCA_TOP /_RISC_CORE/net_sdram_A_6_FEEDTHRU_0     | METAL2   | 1243.2300 661.3400  | SDRAM_CLK    | FALSE       | _ORCA_TOP /_RISC_CORE/net_sdram_A_6_FEEDTHRU_1     | METAL3       | 1635.7000 669.9500  | SD_DDR_CLK    | FALSE     |
| _ORCA_TOP /_PCI_CORE/n130_FEEDTHRU_0               | METAL3   | 7765.7000 1485.6150 | PCI_CLK      | FALSE       | _ORCA_TOP /_PCI_CORE/n130_FEEDTHRU_1               | METAL2       | 675.3800 1631.8100  | PCI_CLK       | FALSE     |
| _ORCA_TOP /_CONTEXT_MEM/net_sdram_BWS_1_FEEDTHRU_0 | METAL2   | 1448.2300 820.2100  | SDRAM_CLK    | TRUE        | _ORCA_TOP /_CONTEXT_MEM/net_sdram_BWS_1_FEEDTHRU_1 | METAL3       | 1635.7000 1276.4050 | SD_DDR_CLK    | TRUE      |

## Writing Out Budget Information

After updating the timing budgets with the `compute_budget_constraints` command, use the `write_budgets` command to write out the updated timing constraints.

```
icc2_shell> write_budgets -blocks {u0_0 u0_1 u0_2 u0_3} -force
```

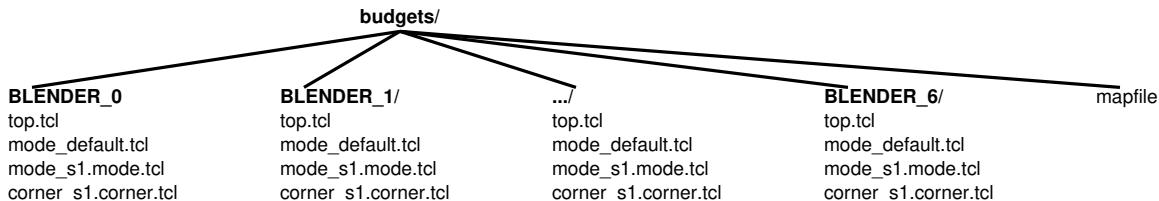
The `write_budgets` command writes out the constraints for the block references specified by the `set_budget_options -add_blocks` command. The `write_budgets` command creates a directory for each block reference name, writes a constraint file for each corner and mode, and writes a top.tcl file that sources the other constraint files for the block. The corner constraint files are named `corner_cornername.tcl` and the mode constraint files are named `mode_modename.tcl`.

Budgets apply only to the boundary paths of the blocks, not to the internal segments. Whenever you generate a new budget with the `write_budgets` command, you must source the top.tcl budget file in the block. You can apply budget files multiple times, whenever the budgets are changed or updated. When you source a new budget file, the tool overwrites any existing boundary constraints from any previously sourced budget file.

To write I/O constraints for all blocks, use the `-all_blocks` option of the `write_budgets` command. To write constraints for only a specified set of blocks, use the `-blocks {block_list}` option. To write the output constraint files to a specific directory, use the `-output_directory_name` option. To overwrite an existing output directory, use the `-force` option.

Figure 154 shows the contents of the budgets directory created by the `write_budgets` command.

*Figure 154 budgets Directory*



During block development later in the flow, apply the constraints written by the `write_budgets` command by sourcing the `top.tcl` file in the `budgets` directory.

Note that the constraints generated by the `write_budgets` command should be sourced after you source the constraints created by the `split_constraints` command. For example, to apply constraints to the ORCA block, open the block and use the following steps.

1. Read in the constraint mapping file for the SDC and UPF files.

```

icc2_shell> sh cat split/mapfile
BLENDER_0 SDC BLENDER_0/top.tcl
BLENDER_0 UPF BLENDER_0/top.upf
...
icc2_shell> set_constraint_mapping_file split/mapfile
1

```

2. (Optional) Remove specific constraint types or specific blocks from the constraint mapping file with the `update_constraint_mapping_file` command.

```
icc2_shell> update_constraint_mapping_file -remove_blocks {BLENDER_6}
```

3. Source the constraints for the block generated by the `split_constraints` command.

```
icc2_shell> source split/BLENDER_0/top.tcl
```

4. Read in the mapping file for the budgets.

```

icc2_shell> sh cat budgets/mapfile
BLENDER_0 BUDGET BLENDER_0/top.tcl
BLENDER_1 BUDGET BLENDER_1/top.tcl
BLENDER_2 BUDGET BLENDER_2/top.tcl

icc2_shell> set_constraint_mapping_file budgets/mapfile
1

```

5. Source the I/O budget constraint.

```
icc2_shell> source budgets/BLENDER_0/top.tcl
```

## Application Options for Budgeting

The `write_budgets` command supports the following application options:

`plan.budget.all_design_subblocks`: Retains the internal constraints of subblocks in the budget output by using the design representation instead of the abstract representation.

`plan.budget.all_full_budgets`: Controls whether the `write_budgets` command writes out only incremental boundary constraints or writes out full constraints for a budgeted block.

`plan.budget.allow_top_only_exceptions`: Allows exceptions created by the `set_false_path` and `set_multicycle_path` commands at the top level, without regenerating or modifying block-level constraints.

`plan.budget.hold_buffer_margin`: Constrains one segment of the path so that it meets the path hold constraint with the specified delay margin.

`plan.budget.pessimistic_driving_cells`: When calculating setup budgets at block inputs, adjusts delays so that the receiving block is charged only with the driving cell delay that is caused by an excess capacitive load.

`plan.budget.write_hold_budgets`: Includes `set_input_delay -min` and `set_output_delay -min` constraints in the output budget to constrain hold optimization at the block boundaries.

## Budget Shells

A budget shell is a macro-like representation of the I/O timing and physical interface of a block. You can use budget shells for top-level design exploration before lower-level blocks are implemented.

In a design hierarchy, a budget shell can replace a lower-level block. The boundary ports of a budget shell are budgeted: paths that originally went from the parent hierarchy into a child hierarchy are replaced with new budgeted paths from the parent hierarchy to the ports of the budget shell. The timing of the new path is adjusted to match the budget constraints for the replaced block.

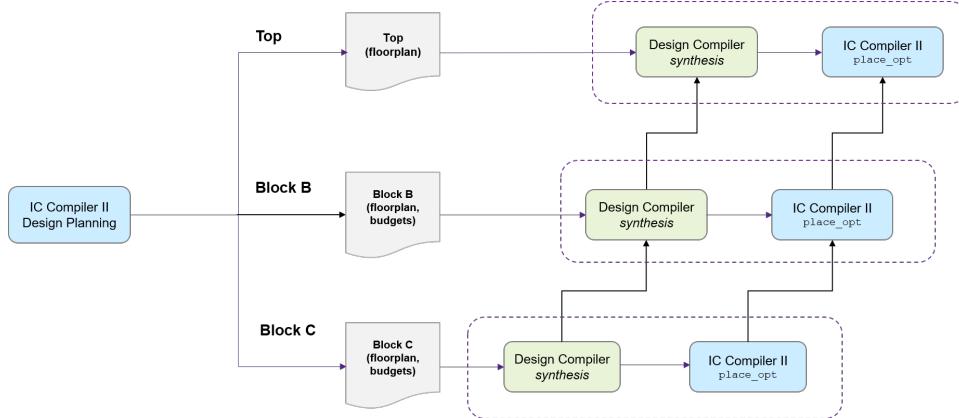
To create a budget shell, use the `write_budgets` command with the `-shell_subblocks` option. The command saves scripts in the directory specified with the `-output` option.

### Note:

For final chip production to generate a tapeout database, do not use the budget shell flow. Use the traditional hierarchical flow based on implemented blocks, starting from the RTL.

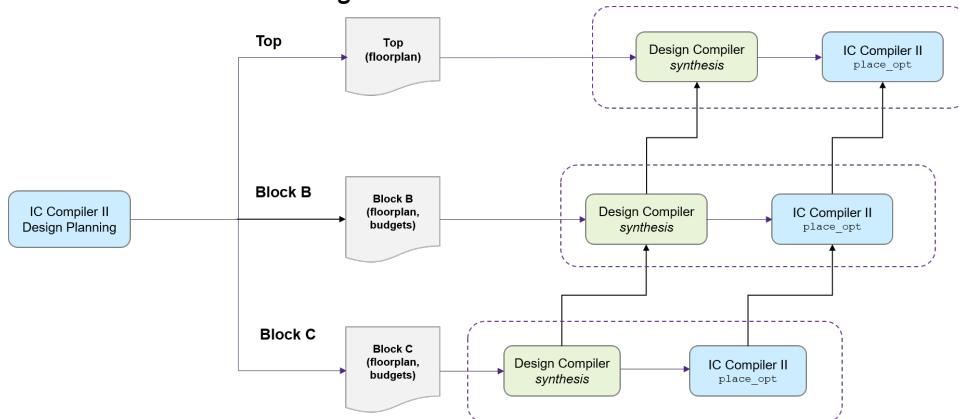
For example, consider a top-level design (Top) that contains block B, which in turn contains block C. As shown in [Figure 155](#), a traditional bottom-up design flow requires that the synthesis and implementation steps for block C be completed to serve as input for block B, which must itself be completed to provide input to the top level.

*Figure 155 Traditional Design Flow*



Alternatively, you can use budget shells to represent blocks to allow top-level design exploration until the implemented versions of block B and block C become available, as shown in [Figure 156](#). This method can provide early information about the quality of the top-level floorplan and improve overall turnaround time.

*Figure 156 Hierarchical Design Flow*



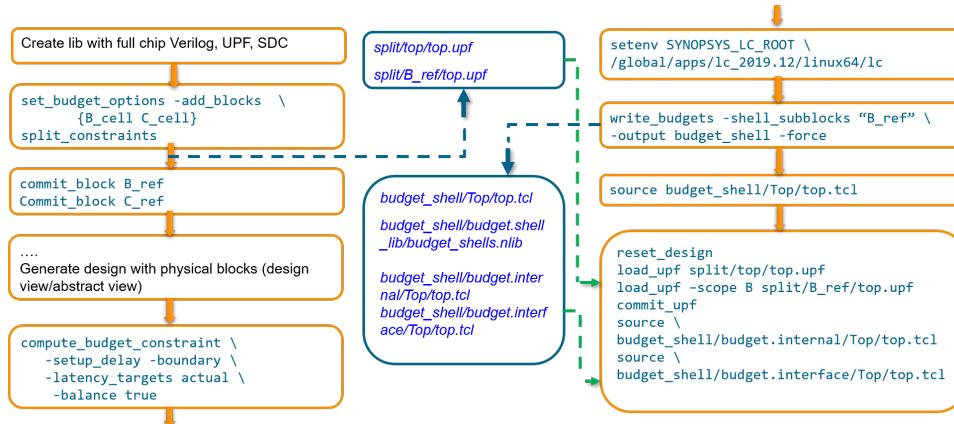
By default, the budget shell contains information about PG pins, related power or ground pins, and other multivoltage Liberty attributes. Including this information improves the top-level implementation for multivoltage designs. You can disable saving the multivoltage information by setting the `mv.hierarchical.include_pg_in_budget_shell` application option to `false`.

The multivoltage attributes saved in the budget shell are as follows:

- voltage\_map
- pg\_pin group
- related\_power\_pin
- related\_ground\_pin
- is\_isolated (**isolation** attribute)
- input\_signal\_level (**level shifter** attribute)
- input\_voltage\_range (**level\_shifter** attribute)
- is\_unconnected (**unconnected pin** attribute)
- short (**feedthrough pin** attribute)
- is\_analog (**analog pin** attribute)
- is\_macro\_cell (set automatically when the shell is created)

When you use budget shells for a multivoltage design, you must reload the UPF files for the budget shells, as shown in [Figure 157](#).

*Figure 157 Budget Shell Flow for Multivoltage Designs*



The general procedure is as follows:

1. Use the split constraints flow to separate the top-level and block-level constraints files.
2. Generate a design with physical blocks.
3. Compute the budget constraints.
4. Use the `write_budgets -shell_subblocks` command to create budget shells.

5. Source the budget shell for the top-level design.
6. Reset the design.
7. Reload the UPF files by using the `load_upf` command separately for the top-level design and the budget shells.
8. Commit the UPF with the `commit_upf` command.
9. Source the timing constraints files that were generated by the `write_budgets -shell_subblocks` command.

The following script is a generic budget constraints flow:

```
#Include the required hierarchical cells
set_budget_options -add_blocks [get_cells "B_cell" "C_cell"]

#Compute the budgets based on estimated corner
compute_budget_constraints -setup_delay -boundary -latency_targets actual
 -balance true

#Specify the path to the Library Compiler root directory
setenv SYNOPSYS_LC_ROOT <lc_path>/lc

#Write the budget shell
write_budgets -shell_subblocks "B_reference" "C_reference" -output
 budget_shell
```

For a fixed percentage budget constraints flow (40 per cent in this example), add the following command after the `compute_budget_constraints` command:

```
set_pin_budget_constraints -internal_percent 40 -all
```