IC Compiler
Design Planning
User Guide

Version D-2010.03-SP4, September 2010

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

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

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

Destination Control Statement

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

Disclaimer

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

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, CoMET, Design Compiler, DesignWare, Formality, Galaxy Custom Designer, HAPS, HapsTrak, HDL Analyst, HSIM, HSPICE, Identify, Leda, MAST, METeor, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

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

Service Marks (^{SM})

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

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

4. Handling Black Boxes

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

6. Performing an Initial Virtual Flat Placement

Appendix A. Using the Flip-Chip Flow

Preface

This preface includes the following sections:

What's New in This Release

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

To see the *IC Compiler Release Notes,*

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

   https://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select IC Compiler, and then select a release in the list that appears.

About This Guide

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

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

Audience

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

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

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

Related Publications

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

https://solvnet.synopsys.com/DocsOnWeb

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

- Design Compiler
- Milkyway Environment

Conventions

The following conventions are used in Synopsys documentation.

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

Customer Support

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

Accessing SolvNet

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

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

https://solvnet.synopsys.com

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

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

Contacting the Synopsys Technical Support Center

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

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

1

Introduction to Design Planning

You can use design planning and chip-level analysis capabilities to handle designs of various sizes and complexities. Design planning and chip-level analysis is intended to be used for both a fast exploration of the design to reduce die size and to implement a final, optimized, and detailed floorplan.

This chapter includes the following sections:

Overview

Design planning in IC Compiler provides you with basic floorplanning and prototyping capabilities on flat or hierarchical designs, such as:

- Dirty-netlist handling
- Automatic die size exploration
- Performing various operations with black box modules and cells
- Fast placement of macros and standard cells
- Packing macros into arrays
- Creating and shaping plan groups
- In-place optimization
- Prototype global routing analysis
- Hierarchical clock planning
- Performing pin assignment on soft macros and plan groups
- Performing timing budgeting
- Converting the hierarchy
- Refining the pin assignment

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

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

Why Plan the Design?

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

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

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

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

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

IC Compiler includes complete hierarchical design planning for both channeled and abutted layout styles. For more information, see “Supported Types of Floorplans” in Chapter 2.

Using a Hierarchical Methodology

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

Consider using a hierarchical methodology in the following scenarios:

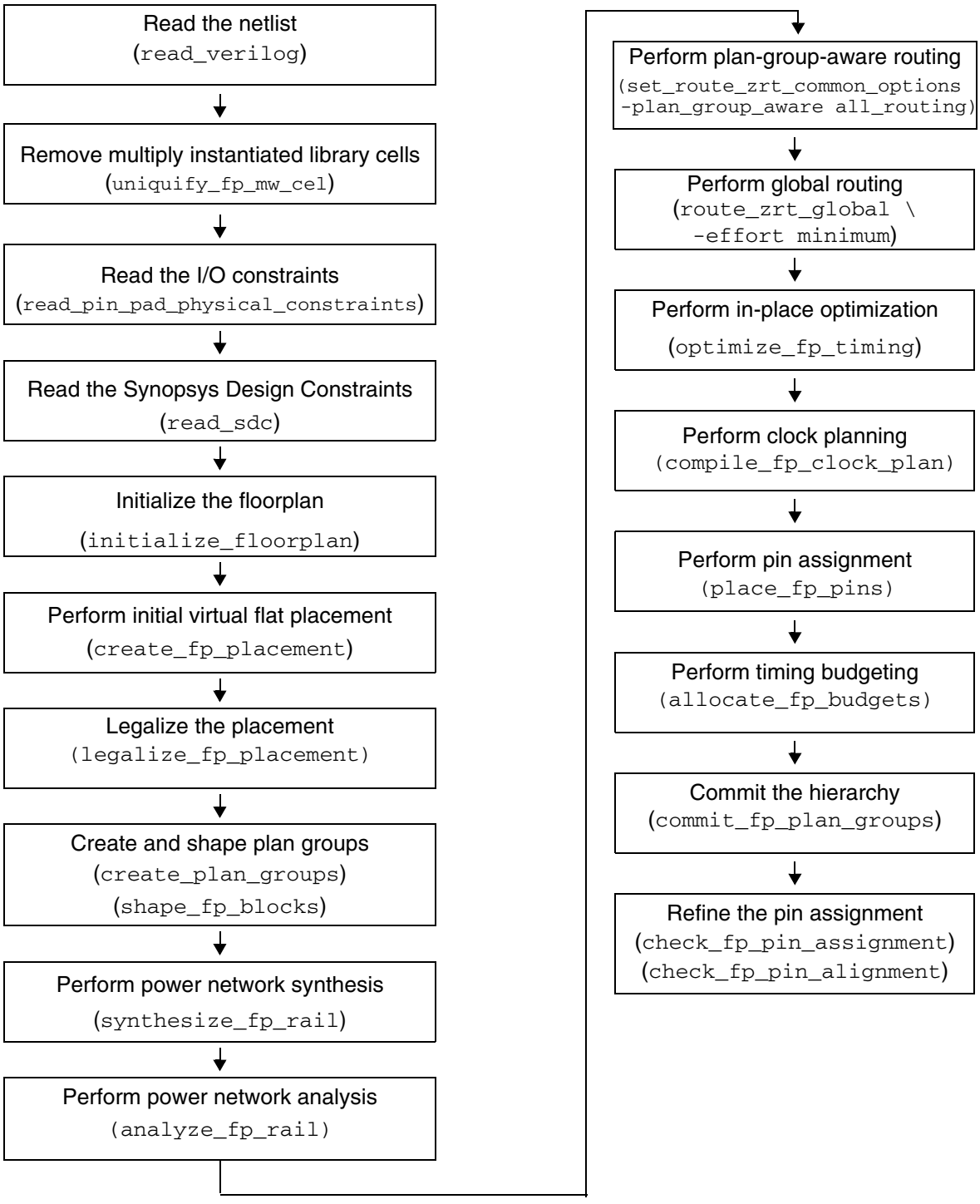
- The design is large, complex, and requires excessive computing resources in order to process the design.
- You anticipate that a number of blocks might have problems, which might cause the schedule to slip. Using a hierarchical methodology allows late design changes to be made to individual blocks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

Hierarchical Methodology for Design Planning

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

Figure 1-1 on page 1-5 shows the hierarchical flow steps for design planning.

Figure 1-1 Hierarchical Design Planning Flow



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

- Design Partitioning
- Hierarchical Floorplanning
- Hierarchical Timing Closure

Design Partitioning

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

- Size

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

- Data paths

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

- Floorplan style

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

- Connection with the hierarchical Design Compiler topographical mode

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

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

Deciding on the Physical Partitions

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

- Using the Hierarchy Browser

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

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

  - In the layout window, choose Partition > New Hierarchy Browser View
  - In the main window, choose Hierarchy > New Hierarchy Browser View

  The view window consists of two panes with an instance tree on the left and an object table on the right. The instance name of the top-level design appears at the top of the instance tree.

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

  For more information, see "Analyzing and Manipulating the Hierarchy" in Chapter 7.

- Performing Virtual Flat Placement

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

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

  For more information, see "Placing Hard Macros and Standard Cells" in Chapter 6.

- Placing and Shaping Plan Groups

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

  For more information, see "Automatically Placing and Shaping Objects in a Design Core" in Chapter 7.

Hierarchical Floorplanning

Hierarchical floorplanning includes the following steps:

- Padding the Plan Groups
- Shielding Plan Groups or Soft Macros
- Assigning Pins
- Creating Soft Macros from the Plan Groups
- Pushing Down Physical Objects

Padding the Plan Groups

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

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

For more information, see “Adding Padding to Plan Groups” in Chapter 7.

Shielding Plan Groups or Soft Macros

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

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

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

Assigning Pins

Before you can perform pin assignment, you must complete the chip-level floorplan, placement, and netlist optimization.

To perform pin assignment,

1. Identify the clock nets. Before you set the pin assignment constraints, you can ensure that there are optimal locations reserved for the pin of clock nets by using the `mark_clock_tree -clock_net` command. By doing this, pin assignment gives high priority to clock pins and avoids feedthroughs on the clock nets.
2. Set the pin assignment constraints prior to performing pin assignment by using the `set_fp_pin_constraints` command. The tool honors pin constraints during pin assignment on soft macros (traditional pin assignment) and on plan groups (pin cutting flow). For more information, see "Setting Pin Assignment Constraints" in Chapter 12.
3. You can use plan-group-aware global routing to suggest pin locations. To enable plan-group-aware global routing, use the following command:

   ```
   set_route_zrt_common_options -plan_group_aware all_routing
   ```

   When you enable plan-group-aware routing, the `route_zrt_global` command honors the hierarchical rules for plan group boundary crossings by recognizing that plan groups and the routes internal to a plan group do not cross the plan group boundaries. The routes that cross the plan groups make minimal intersections with the plan group boundaries and do not crisscross. The intersection becomes the pin location of the plan groups. You can analyze the pin assignment by checking the topology of related global routes.

   For more information, see "Performing Plan-Group-Aware Global Routing" in Chapter 12.
4. Use the `analyze_fp_routing -finalize_pins_feedthroughs` command to save the pin locations. The command calculates the pin locations for each boundary location and saves the location data to the Milkyway database as a guide to pin placement later during commit hierarchy.

   If feedthrough creation is enabled, and feedthrough nets are needed, the IC Compiler tool creates new pins for the feedthrough nets in the logical hierarchy and saves the pin locations.

   If you create feedthroughs and add feedthrough pins, you can use the `optimize_fp_timing -feedthrough_buffering_only` command to insert buffers on the feedthrough nets. This avoids `assign` statements in the Verilog output netlist and improves the timing on interblock timing paths.

   Note:

   After you finalize the routing, using the `analyze_fp_routing` command; any changes made to the global routes no longer affect pin assignment.

   For more information, see "Analyzing the Pin and Feedthrough Routing" in Chapter 12.

5. Assign top-level or block-level pins on both rectangular and rectilinear soft macros by using the `place_fp_pins` command. The pin assignment engine considers the top-level connections to plan groups, macros, and pad locations when it determines where to assign the pins.

   You can also improve pin locations based on cell placement inside the soft macros by using the `place_fp_pins -block_level` command.

   For more information, see "Performing Traditional Pin Assignment" in Chapter 12.

6. Refine the pin assignment if necessary. You can analyze and evaluate the quality of the pin assignment results by checking the placement of soft macro pins in the design and by reassigning individual pins by using the `place_fp_pins` command, or you can manually change the pin assignment in the GUI.

   Check or verify the placement of soft macro pins in a design. After pins have been assigned, added, or moved, you can quickly verify if there are any spacing errors or missing pins by using the `check_fp_pin_assignment` command.

   Check the pin alignment quality of results by using the `check_fp_pin_alignment` command.

   For more information, see "Refining the Pin Assignment" in Chapter 12.

7. After finalizing the floorplan, you can commit the physical hierarchy by using the `commit_fp_plan_groups` command to convert the plan groups into soft macros. The command also places pins physically on the soft macro. After this, you can visually confirm that the tool assigned the pins.

   For more information, see "Converting Plan Groups to Soft Macros" in Chapter 14.

Creating Soft Macros From the Plan Groups

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

For more information, see "Converting Plan Groups to Soft Macros" in Chapter 14.

Pushing Down Physical Objects

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

For more information, see “Pushing Physical Objects Down to the Soft Macro Level” in Chapter 14 and “Pushing Physical Objects Up to the Top Level” in Chapter 14.

Hierarchical Timing Closure

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

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

- Early Chip-Level Timing Feasibility Check
- Timing and Signal Integrity Budgeting
- Clock Planning

Early Chip-Level Timing Feasibility Check

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

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

For more information, see “Running In-Place Timing Optimization” in Chapter 11.

You should analyze the results, identify the root causes of the timing issues, such as the floorplan, SDC constraints, or netlist, and resolve them before proceeding to the next steps.

Timing and Signal Integrity Budgeting

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

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

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

For more information, see "Performing Timing Budgeting" in Chapter 13.

Clock Planning

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

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

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

For more information, see "Performing Clock Planning" in Chapter 10.

Using the GUI to Manage Tasks in Parallel

You can use the IC Compiler GUI to help manage the execution of a job script on a number of jobs that can be done in parallel by using the Load Sharing Facility (LSF) queuing system. Parallel job execution is typically used on a group of subblocks in a hierarchical design. For example, this can occur in a hierarchical design after the `commit_fp_plan_groups` command is run. This command creates a set of subblocks for a design for which you can define scripts and organize the runs that will implement each of these blocks in parallel during an IC Compiler virtual flat flow.

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

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

1. Choose File > Run Parallel Jobs

   The Run Parallel Jobs dialog box appears.

   Alternatively, you can use the `run_parallel_jobs` command.

2. Set the options, depending on your requirements.

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

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

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

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

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

   Environment variables – The environment variables and their settings communicate information from the parent to the child processes. The values in the list become UNIX environment variable data that can customize the behavior of the batch script. The

command passes the environment variables and settings through the queuing system to the child jobs. You can add new environment variables by selecting the “New” button and typing the name and value in the New Environment dialog box. You can also edit and remove the environment variables.

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

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

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

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

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

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

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

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

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

  Note:
  Because it can be difficult to correctly create such a script, it recommended that you use the default script.

- Job queue – Type the name of the queue from which the jobs are dispatched. The queue name has meaning only to a given installation using a specified job submission script. In the default job script, which interfaces to the Load Sharing Facility (LSF), the default LSF queue is used.

- Runtime timeout – Select this option to specify the maximum amount of elapsed time that a job can run. You can use this option to prevent stuck or runaway jobs that can prevent a flow from running to completion.

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

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

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

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

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

3. Click OK or Apply.

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

The syntax to submit jobs is

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

```
job_name_list
```

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

The `send_flow_status` command uses the following syntax:

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

2

Creating a Floorplan

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

This chapter includes the following sections:

Supported Types of Floorplans

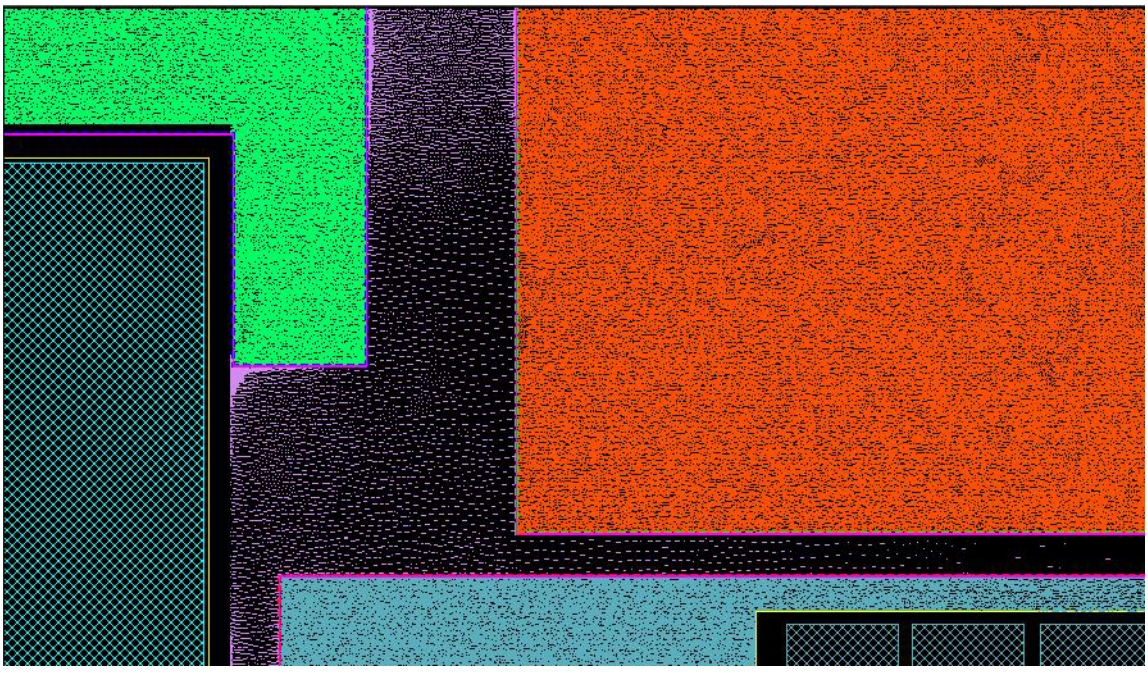
IC Compiler supports different floorplan methodologies to meet the requirements of your design. The channeled, abutted, and narrow-channel floorplan methodologies are described in the following sections.

- Channeled Floorplans
- Abutted Floorplans
- Narrow-Channel Floorplans

Channeled Floorplans

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

Figure 2-1 Channeled Floorplan

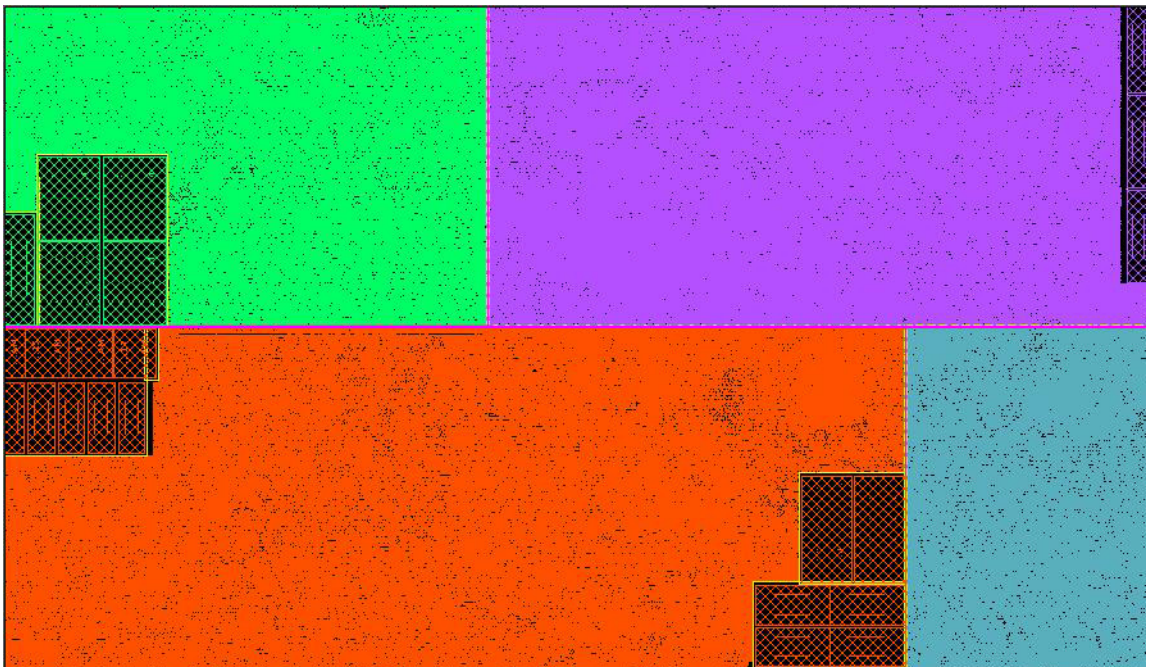


Abutted Floorplans

In the abutted floorplan methodology, blocks are touching and the tool does not allocate space for macro cell placement between blocks. Designs with abutted floorplans do not require top-level optimization, as all logic is pushed down into the blocks. However, abutted

floorplans might require over-the-block routing in order to meet design requirements. This floorplan style also requires more attention to feedthrough management. Routing congestion might also become an issue for abutted floorplan designs. An example of an abutted floorplan is shown in Figure 2-2.

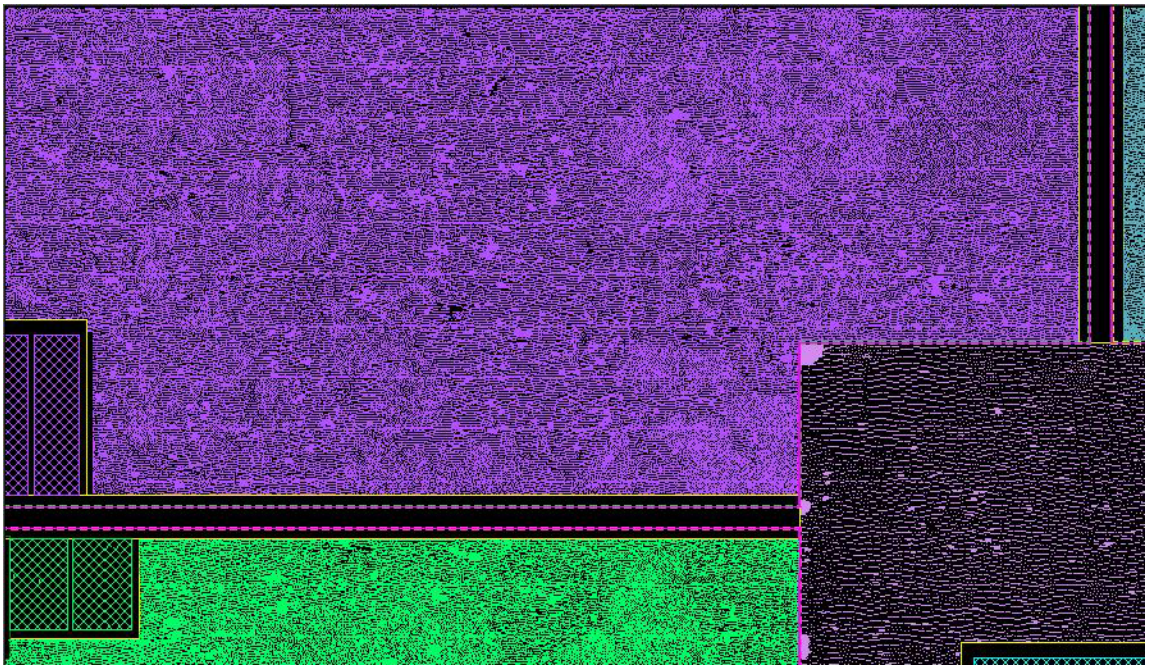
Figure 2-2 Abutted Floorplan



Narrow-Channel Floorplans

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

Figure 2-3 Narrow-Channel Floorplan



Preparing the Design

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

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

Connecting Power and Ground Ports

The macro cells and modules in your design contain power and ground pins that must be connected before initializing the floorplan. The `derive_pg_connection` command (or Preroute > Derive PG Connection in the GUI) connects power, ground, and tie-off pins to power and ground nets.

Table 2-1 describes the command options for the `derive_pg_connection` command; these options can also be set by choosing Preroute > Derive PG Connection in the GUI. For more information, see the man page.

Table 2-1 derive_pg_connection Command Options

Command option	Description
`-all` (“Perform both power and tie connections” in the GUI)	Connects both power and ground pins and tie-off pins.
`-cells` *cells* (“Cells” text box in the GUI)	Specifies the cells to connect to the PG nets.
`-create_nets` (“Create power/ground nets from UPF supply nets” in the GUI)	Creates PG nets based on the power domain definitions and supply nets.
`-create_ports none | top | all` (“Create port” radio button in the GUI)	Creates ports for the nets specified by the `-power_net` and `-ground_net` options.
`-power_net` *net_name* (“Power net” text box in the GUI)	Specifies the name of the power net to use for power and tie-high connections. Use the `-ground_net` option to specify the ground net name.
`-power_pin` *pin_name* (“Power pin” text box in the GUI)	Specifies the name of the power pin on the cells specified by using the `-cells` option. Use the `-ground_pin` option to specify the ground pin name.
`-reconnect` (“Reconnect power/ground pins” check box in the GUI)	Reconnects power and ground pins that have existing connections.
`-resolve_conflict` (“Resolve any hierarchical pg netlist conflicts with upf” check box in the GUI)	Modifies the power and ground netlist to be compliant with the UPF specification.

Table 2-1 derive_pg_connection Command Options(Continued)

Command option	Description
`-tie` ("Reconnect existing tie pins to appropriate power nets" check box in the GUI)	Connects the tie-high and tie-low pins to the PG nets.
`-verbose` ("Show detail connection information" in the GUI)	Writes out additional debugging messages regarding power, ground, and tie-off connections.

Note the following when using the `-tie` and `-reconnect` options of the `derive_pg_connection` command:

- Unless you specify the `-reconnect` option, automatic connection works only on unconnected power and ground pins
- All tie-off connections are mapped to actual PG nets in the same hierarchy
- The PG network must already exist in the design before you can run the `derive_pg_connection -tie` command to map tie-off connections to real PG nets; running the `derive_pg_connection -tie` command before the PG network is created causes unexpected results for the PG and tie-off r in the design flow

For single-voltage designs, you must apply the `derive_pg_connection` command at the following stages of implementation:

- During data setup
- Before any initial physical routing of the PG nets by using any of the `create_pad_rings`, `commit_fp_rail`, `preroute_instances` or `preroute_standard_cells` commands
- After the completion of each optimization step, including the `place_opt`, `clock_opt`, and `route_opt` commands
- Before inserting filler cells that contain metal fill polygons
- Before running the `verify_lvs` command
- Before writing out the final Verilog netlist

For more information about running the `derive_pg_connection` command during power planning, see "Creating Logical Power and Ground Connections" on page 8-59.

The following example manually connects VDD pins to the VDD power net and connects ground pins to the VSS ground net.

```
icc_shell> derive_pg_connection -power_net VDD -power_pin VDD \
 -ground_net VSS -ground_pin VSS
icc_shell> derive_pg_connection -power_net VDD -ground_net VSS \
 -cells inst_macro1 -tie
```

Creating Power and Ground Ports

Depending on the floorplan creation flow you use, you can add top-level power and ground ports at the same time that you connect the power and ground pins in your design. Use the `derive_pg_connection` command with the `-create_ports top` option to automatically create ports for the nets specified by the `-power_net` and `-ground_net` options. In the GUI, choose Preroute > Derive PG Connection. In the Derive Power Ground Connection dialog box that opens, select “Manual connection” and then select Top as the “Create port” option.

The following example connects macro cell pins named VDD to the VDD net, pins named VSS to the VSS net, and creates top-level ports for both nets. This command can be used on a design that contains a single power domain.

```
icc_shell> derive_pg_connection -power_net VDD -power_pin VDD \
           -ground_net VSS -ground_pin VSS -create_ports top
```

Adding Power, Ground, and Corner Cells

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

Table 2-2 describes the `create_cell` command options. For more information about these options, see the man page.

Table 2-2 create_cell Command Options

Command option	Description
cell_list	Adds the unique cell instances to the design.
reference_name	Identifies the design or library cell for instantiation.
-view *view_name*	Specifies the view name to use when adding the cell.
-freeze_silicon	Adds the specified cell in a transitional state. Change the state by using the `place_freeze_silicon` or `map_freeze_silicon` command.
-hierarchical	Creates a hierarchical cell.

Setting the I/O Pad Constraints

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

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

Table 2-3 describes the `set_pad_physical_constraints` command options. For more information about these options, see the man page.

Table 2-3 set_pad_physical_constraints Command Options

Command option	Description
`-pad_name` *pad_name*	Specifies the name of the pad to which to attach the constraint. Use either the `-pad_name` option or specify the object name.
object	Specifies the pad object to which to attach the constraint.
`-side` *number*	Specifies the side number on which to place the pad.
`-order` *number*	Specifies the placement order for the pad.
`-offset` *gap*	Specifies the gap between the pad and the die edge.
`-orientation` *reflection_type*	Specifies the type of reflection to apply to the pad cell.
`-min_left_iospace` *spacing*	Specifies the spacing to the next I/O pad on the left.
`-min_right_iospace` *spacing*	Specifies the spacing to the next I/O pad on the right.
`-chip_level_distance {dist_left_edge_to_pad` *gap*`}`	Specifies the minimum distance to place pads on the top and bottom die edges away from the left die edge.

Table 2-3 set_pad_physical_constraints Command Options (Continued)

Command option	Description
`-chip_level_distance {dist_right_edge_to_pad` *gap*`}`	Specifies the minimum distance to place pads on the top and bottom die edges away from the right die edge.
`-chip_level_distance {dist_top_edge_to_pad` *gap*`}`	Specifies the minimum distance to place pads on the left and right die edges away from the top die edge.
`-chip_level_distance {dist_bottom_edge_to_pad` *gap*`}`	Specifies the minimum distance to place pads on the left and right die edges away from the bottom die edge.

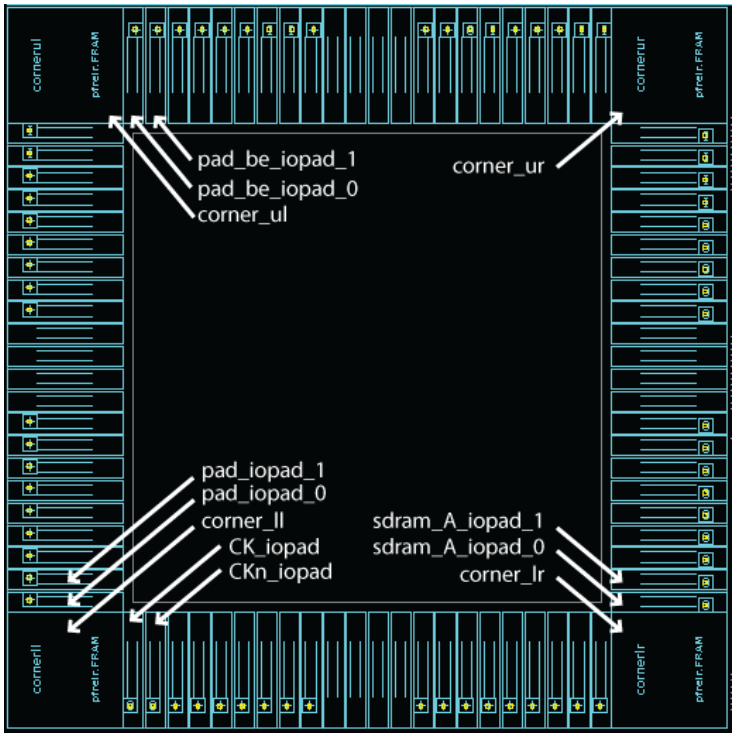
The following example script portion describes two corner pad locations and several I/O pad locations for the floorplan.

Example 2-1 Pad Constraints Script Sample

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

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

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

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

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

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

Figure 2-4 shows the floorplan created by these constraints.

Figure 2-4 Floorplan With Pad Placement



Setting the Pin Constraints

You can use the `set_pin_physical_constraints` command to set constraints on individual pins or nets. Use the `set_fp_pin_constraints` command to set global constraints for a block. If a conflict arises between the individual pin constraints and the global pin constraints, the individual pin constraints have higher priority. The constraints are stored in the Milkyway database.

Table 2-4 describes the `set_pin_physical_constraints` command options. For more information about these options, see the man page.

Table 2-4 set_pin_physical_constraints Command Options

Command option	Description
`-pin_name` *pin*	Selects the pin to which to apply the constraint. Note that the `-pin_name` option, `-nets` option and *objects* argument are mutually exclusive.
`-cell` *cell*	Specifies the name of the cell to which the pin belongs. You must also use the `-pin` option when you use the `-cell` option.
`-nets` *net_list*	Sets the constraint for the pins connected to the specified nets.
`-layers` *layers*	Defines a list of layers available for connection to the specified pin.
`-width` *pin_width*	Sets the width of the pin.
`-depth` *pin_depth*	Defines the desired depth of the pin.
`-side` *side_number*	Specifies the side number to place the pin.
`-offset` *distance*	Sets the offset distance from the starting point of the edge for pin placement.
`-order` *placement_order*	Specifies the placement order for the pin, starting from the bottommost or leftmost position.
`-off_edge` *value*	Specifies how the pin location is determined.
`-location` *xy_coordinate*	Specifies the x- and y-locations of the off-edge pins.
`-pin_spacing` *spacing*	Specifies the number of wire tracks between adjacent pins.
`-exclude_sides` *side_number*	Specifies the soft macro edges where pins must not be placed.
object	Specifies the pin to which to apply the specified constraint.

Saving the Pin and Pad Constraints

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

To create the pin and pad constraints file, use the `write_pin_pad_physical_constraints` command (or by choosing Floorplan > Write Pin/Pad Physical Constraints in the GUI). Table 2-5 describes the `write_pin_pad_physical_constraints` command options. For more information about these options, see the man page.

Table 2-5 write_pin_pad_physical_constraints Command Options

Command option	Description
`-library` *`lib_name`* (Uses the current library in the GUI)	Specifies the name of the library that contains the Milkyway cell.
`-cell` *`cell_name`* (Uses the current cell in the GUI)	Specifies the name of the Milkyway cell that contains the constraints.
`-constraint_type` *`type_of_constraint`* (“Constraint type” radio button set in the GUI)	Defines the type of location information (side only, side and order, or side and location) to output.
`-pin_only` (“Terminals only” radio button in the GUI)	Specifies that the command writes only pin constraints.
`-pad_only` (“I/O pads only” radio button in the GUI)	Specifies that the command writes only pad constraints.
`-objects` *`object_list`* (“Terminals,” “I/O pads,” and “Nets” boxes in GUI)	Specifies the list of pins, terminals, nets, and pads for constraint output.
`file_name` (“Output file name” box in the GUI)	Specifies the name of the output constraints file.

Reading an Existing Pad and Pin Constraints File

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

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

Table 2-6 describes the `read_pin_pad_physical_constraints` command options. For more information about these options, see the man page.

Table 2-6 read_pin_pad_physical_constraints Command Options

Command option	Description
`-append` ("Append" check box in the GUI)	Appends the constraints to the design.
`-cell` *`cell_name`* (Defaults to the current cell in the GUI)	Specifies the name of the Milkyway cell that receives the constraints.
`file_name` ("Input file name" in the GUI)	Specifies the name of the input constraints file.

Reporting the Pad and Pin Constraints

Use the `report_pin_pad_physical_constraints` command to display a list of `set_pin_physical_constraints` and `set_pad_physical_constraints` commands that define the pin and pad constraints for the current design. You can report only pin constraints, only pad constraints, only chip-level pad constraints, or all constraints depending on the command options you specify.

Table 2-7 describes the `report_pin_pad_physical_constraints` command options. For more information about these options, see the man page.

Table 2-7 report_pin_pad_physical_constraints Command Options

Command option	Description
`-cell` *`cell_name`*	Specifies the cell or plan group to generate the report.
`-pin_only`	Limits reporting to only pin constraints.
`-pad_only`	Limits reporting to only pad constraints.
`-chiplevel_pad_only`	Limits reporting to only chip-level pad constraints.
`objects`	Specifies the pin or pad for which to display the report.

Removing the Pad and Pin Constraints

Use the `remove_pin_pad_physical_constraints` command to remove all constraints previously set by using the `set_pin_physical_constraints` and `set_pad_physical_constraints` commands. You can remove only pin constraints, only pad constraints, only chip-level pad constraints, or all constraints based on the command options you specify. You can also remove constraints from a design other than the currently open design.

Table 2-7 describes the `remove_pin_pad_physical_constraints` command options. For more information about these options, see the man page.

Table 2-8 remove_pin_pad_physical_constraints Command Options

Command option	Description
`-cell` *`cell_name`*	Specifies the cell or plan group for which to remove constraints.
`-pin_only`	Removes only pin constraints.
`-pad_only`	Removes only pad constraints.
`-chiplevel_pad_only`	Removes only chip-level pad constraints.
`objects`	Specifies the pin or pad for which to remove constraints.

Initializing the Floorplan

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

- Initializing a Rectangular Floorplan
- Creating a Simple Rectilinear Floorplan
- Creating a Complex Rectilinear Floorplan

Initializing a Rectangular Floorplan

Using the `initialize_floorplan` command, you can define a rectangular chip boundary and periphery based on aspect ratio, specific width and height, or number of cell rows. The command also places I/O pad and corner cells based on the constraints you defined by using `set_pin_physical_constraints` commands.

To create the initial floorplan, use the `initialize_floorplan` command (or by choosing Floorplan > Initialize Floorplan in the GUI). Table 2-9 describes the `initialize_floorplan` command options. For more information about these options, see the man page.

Table 2-9 initialize_floorplan Command Options

Command option	Description
`-control_type` *`type`* (“Control type” section in the GUI)	Specifies the basis for the dimension of the floorplan: `aspect_ratio`, `width_and_height`, `row_number`, or `boundary`.
`-core_aspect_ratio` *`aspect_ratio`* (“Aspect Ratio (H/W)” box in the GUI)	Defines the target aspect ratio (height/width) for the floorplan.
`-core_utilization` *`utilization`* (“Core utilization” box in the GUI)	Specifies the target utilization (cell placement area/ total area) for the floorplan.
`-row_core_ratio` *`rows`* (“Row/core ratio” box in the GUI)	Defines the channel space between rows reserved for routing.

Table 2-9 initialize_floorplan Command Options (Continued)

Command option	Description
`-core_width` *width* (“Core width” box in the GUI)	Specifies the width of the core area when used with the `-control_type width_and_height` option.
`-core_height` *height* (“Core height” box in the GUI)	Specifies the height of the core area when used with the `-control_type width_and_height` option.
`-number_rows` *rows* (“Num rows” box in the GUI)	Specifies the number of rows in the core area when used with the `-control_type row_number` option.
`-use_vertical_row` (Uncheck to “Horizontal row” check box in the GUI)	Places rows vertically within the core area.
`-no_double_back` (Uncheck the “Double back” check box in the GUI)	Prevents placement of paired or flipped rows back-to-back.
`-start_first_row` (“Start first row” check box in the GUI)	Pairs cell rows starting at the bottom of the core for horizontal rows or the left side of the core for vertical rows. You cannot combine the `-start_first_row` option and the `-no_double_back` option.
`-flip_first_row` (“Flip first row” check box in the GUI)	Flips the first row at the bottom of the core for horizontal rows or the left side of the core for vertical rows. You cannot combine the `-flip_first_row` option and the `-no_double_back` option.
`-left_io2core` *spacing* (“Core to left” box in the GUI)	Creates a space between the left side of the core and the right side of the closest pad.
`-right_io2core` *spacing* (“Core to right” box in the GUI)	Creates a space between the right side of the core and the left side of the closest pad.
`-bottom_io2core` *spacing* (“Core to bottom” box in the GUI)	Creates a space between the bottom side of the core and the top of the closest pad.
`-top_io2core` *spacing* (“Core to top” box in the GUI)	Creates a space between the top of the core and the bottom of the closest pad.

Table 2-9 initialize_floorplan Command Options (Continued)

Command option	Description
`-keep_macro_place` ("Keep macro place" check box in the GUI)	Maintains the placement of preplaced macros.
`-keep_std_cell_place` ("Keep std cell place" check box in the GUI)	Maintains the placement of preplaced standard cells.
`-min_pad_height` ("Min pad height" check box in the GUI)	Creates the core area based on the minimum pad height.
`-pad_limit` ("Pad limit" check box in the GUI)	Places I/O pad cells with no space between them.
`-pin_snap` ("Pin snap" check box in the GUI)	Snaps the center of the terminals to the center of the closest wire track.
`-keep_io_place` ("Keep I/O placement" check box in the GUI)	Maintains the existing I/O pad placement, even if constraints set using the `set_pad_physical_constraints` command exist.

The following examples are created by using different options with the `initialize_floorplan` command. Figure 2-5 shows a floorplan created by using the `initialize_floorplan` command with no options.

```
icc_shell> initialize_floorplan
```

Figure 2-5 Floorplan With No Options Specified

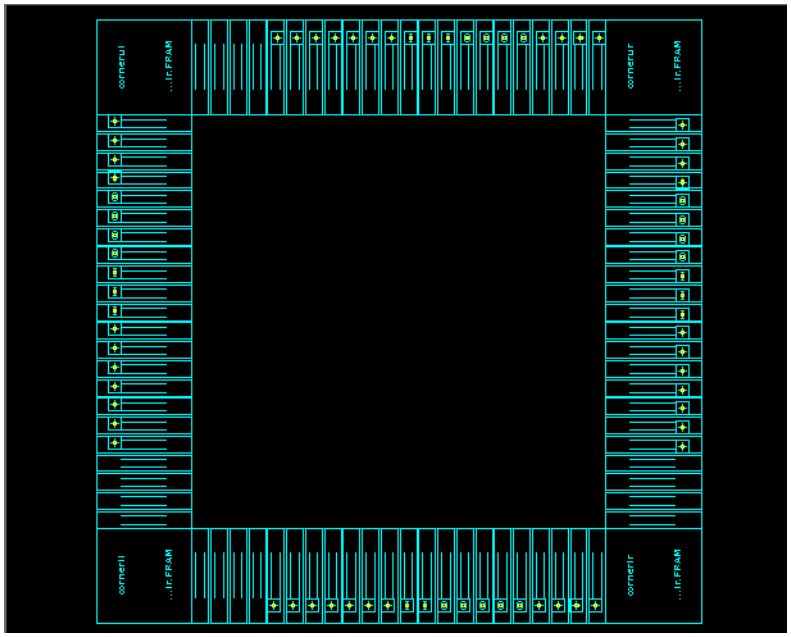


Figure 2-6 shows a floorplan created by using the `-control_type aspect_ratio` option. The following example shows the complete `initialize_floorplan` command. These two options set the aspect ratio for the design. In this example, the aspect ratio is 2.0, which creates a floorplan with a height equal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

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

Figure 2-6 Floorplan With Aspect Ratio of 2.0

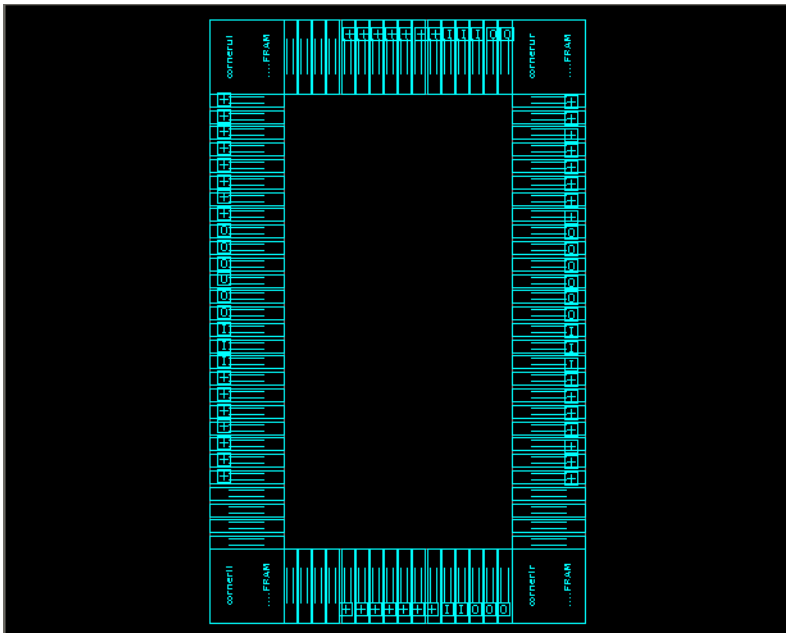


Figure 2-7 shows the first few rows of a floorplan created by using the `initialize_floorplan` command with the `-start_first_row` and `-row_core_ratio` options. The `-start_first_row` option begins row pairing at the bottom of the core area; the `-row_core_ratio` option sets the ratio between standard cell rows and routing channels.

```
icc_shell> initialize_floorplan -row_core_ratio 0.5 -start_first_row
```

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

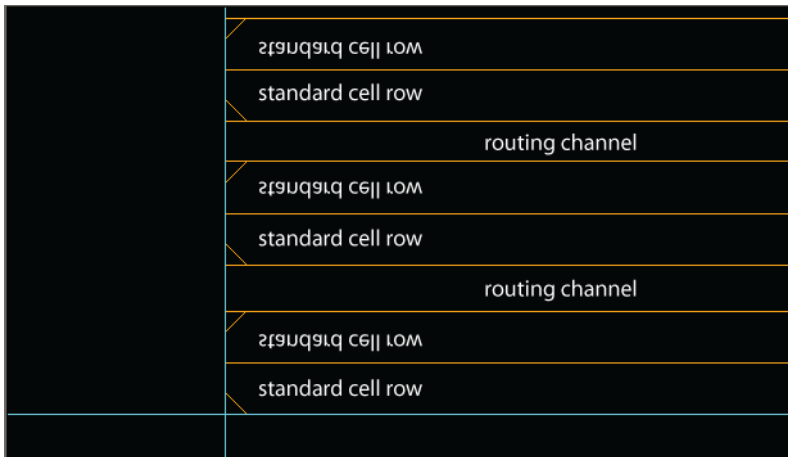


Figure 2-8 shows the first few rows of a floorplan created by using the `initialize_floorplan` command with the `-flip_first_row` and `-row_core_ratio` options. The `-flip_first_row` option flips the first row at the bottom of the floorplan; the `-row_core_ratio` option sets the ratio between standard cell rows and routing channels.

```
icc_shell> initialize_floorplan -row_core_ratio 0.5 -flip_first_row
```

Figure 2-8 Floorplan With First Row Flipped

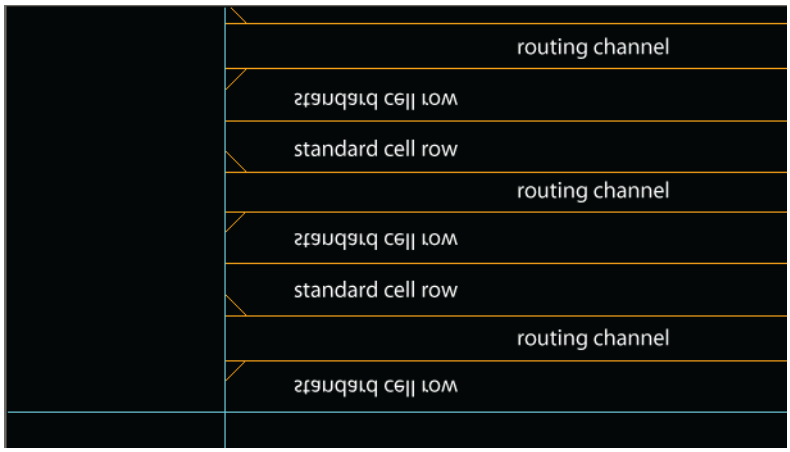


Figure 2-9 shows the first few rows of a floorplan created by using the `initialize_floorplan` command with the `-no_double_back` and `-row_core_ratio` options. The `-no_double_back` option prevents flipping or pairing of standard cell rows; the `-row_core_ratio` option sets the ratio between standard cell rows and routing channels.

```
icc_shell> initialize_floorplan -row_core_ratio 0.5 -no_double_back
```

Figure 2-9 Floorplan With No Row Flipping

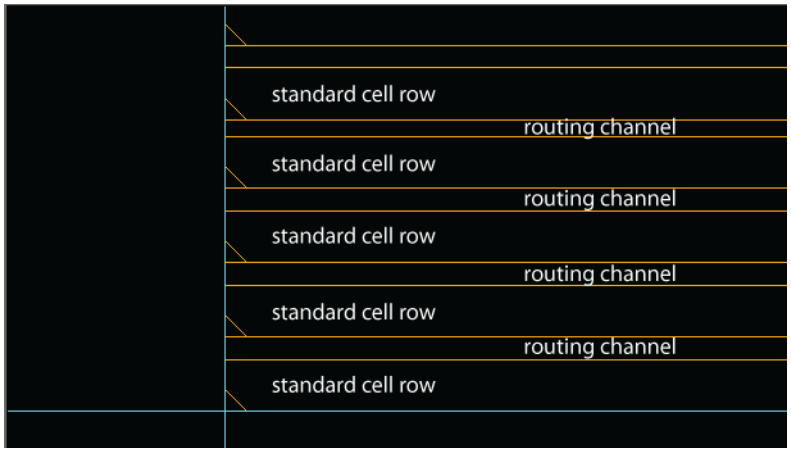


Figure 2-10 shows the first few rows of a floorplan created by using the `initialize_floorplan` command with the `-use_vertical_row` option. The `-vertical_row` option creates standard cell rows in a vertical orientation.

```
icc_shell> initialize_floorplan -use_vertical_row
```

Figure 2-10 Floorplan With Vertical Rows

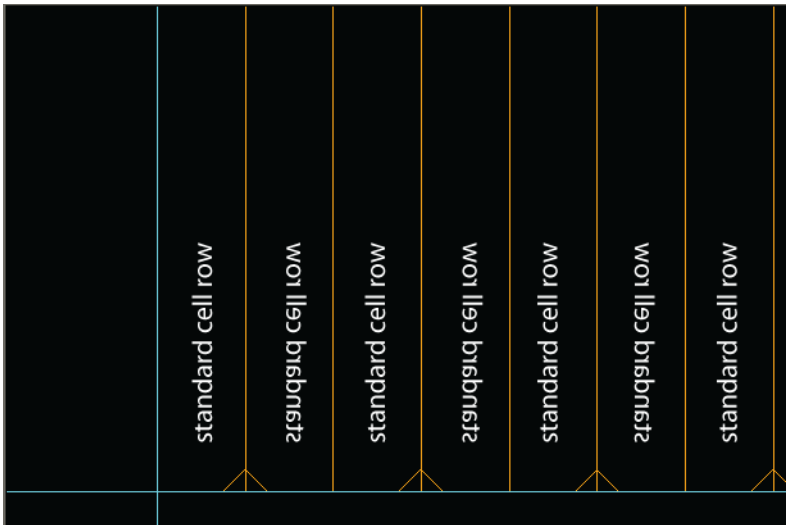
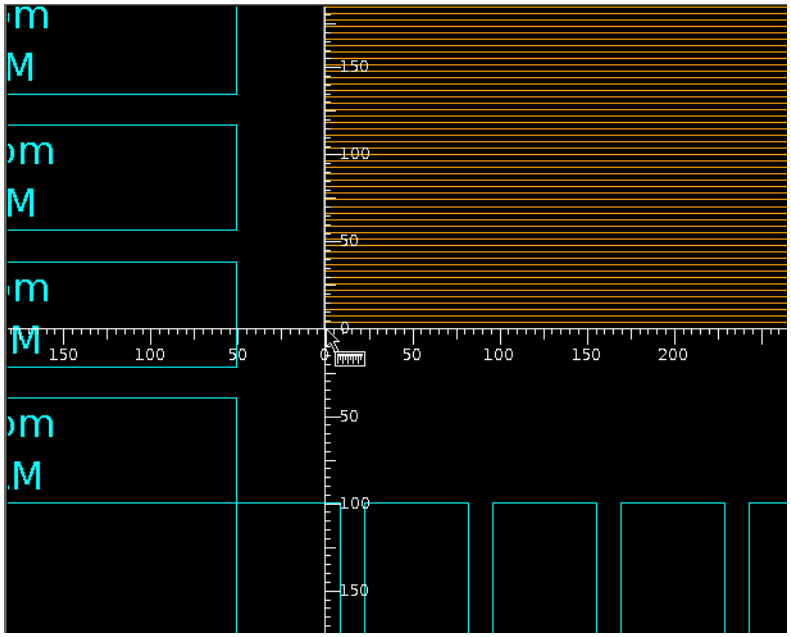


Figure 2-11 shows a zoomed-in view of a floorplan created by using the `initialize_floorplan` command with the `-bottom_io2core` and `-left_io2core` options. These options set the distance between the core and the pad cells on the bottom and left sides of the die.

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

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



Creating a Simple Rectilinear Floorplan

Use the `initialize_rectilinear_block` command to create L-shaped, T-shaped, U-shaped, or cross-shaped rectilinear blocks. You can vary the edge dimensions and rotation by specifying command options. The `initialize_rectilinear_block` command shares several options with the `initialize_floorplan` command; for more information, see “Initializing a Rectangular Floorplan” on page 2-15. For complex rectilinear floorplans, use the `create_boundary` command with the `initialize_rectilinear_block` command, as described in “Creating a Complex Rectilinear Floorplan” on page 2-29.

To create a simple rectilinear block, use the `initialize_rectilinear_block` command (or by choosing Floorplan > Initialize Rectilinear Block in the GUI). Table 2-10 describes the `initialize_rectilinear_block` command options. For more information about these options, see the man page.

Table 2-10 initialize_rectilinear_block Command Options

Command option	Description
`-shape L | T | U | X` ("Shape" in the GUI)	Specifies the shape of the block.
`-core_side_dim { len_a len_b len_c len_d len_e len_f }` ("Sides" in the GUI)	Specifies the relative or absolute edge length for each edge of the block. See the man page or GUI for the mapping between the argument position and its associated edge.
`-control_type ratio | length` ("Control Type" in the GUI)	Selects how the command applies the core side dimensions, as actual dimensions or relative ratios.
`-use_current_boundary` ("Current boundary" radio button in the GUI)	Uses the existing boundary created by the `initialize_rectilinear_block -shape` command or the `create_boundary -poly` command.
`-row_core_ratio ratio` ("Row/core ratio" box in the GUI)	Defines the channel space between rows reserved for routing.
`-core_utilization ratio` ("Core utilization" box in the GUI)	Specifies the target utilization (cell placement area/ total area) for the floorplan.
`-orientation N | W | S | E` ("Orientation" section in the GUI)	Specifies the rotation for the floorplan.
`-use_vertical_row` (Uncheck the "Horizontal rows" check box in the GUI)	Places rows vertically in the core area.
`-no_double_back` (Uncheck the "Double back" check box in the GUI)	Prevents placement of paired or flipped rows back-to-back.

Table 2-10 initialize_rectilinear_block Command Options (Continued)

Command option	Description
`-start_first_row` ("Start first row" check box in the GUI)	Pairs cell rows starting at the bottom of the core for horizontal rows or the left side of the core for vertical rows. You cannot combine the `-start_first_row` option and the `-no_double_back` option.
`-flip_first_row` ("Flip first row" check box in the GUI)	Flips the first row at the bottom of the core for horizontal rows, or the left side of the core for vertical rows. You cannot combine the `-flip_first_row` option and the `-no_double_back` option.
`-left_io2core` *distance* ("Core to left" box in the GUI)	Creates a space between the left side of the core and the right side of the closest pad.
`-right_io2core` *distance* ("Core to right" box in the GUI)	Creates a space between the right side of the core and the left side of the closest pad.
`-top_io2core` *distance* ("Core to top" box in the GUI)	Creates a space between the top of the core and the bottom of the closest pad.
`-bottom_io2core` *distance* ("Core to bottom" box in the GUI)	Creates a space between the bottom side of the core and the top of the closest pad.
`-keep_macro_place` ("Macro cells" check box in the GUI)	Maintains the placement of preplaced macros.
`-keep_std_cell_place` ("Std cells" check box in the GUI)	Maintains the placement of preplaced standard cells.
`-keep_io_place` ("I/O terminals" check box in the GUI)	Maintains the existing I/O pad placement.

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

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

Figure 2-12 T-Shaped Rectilinear Block

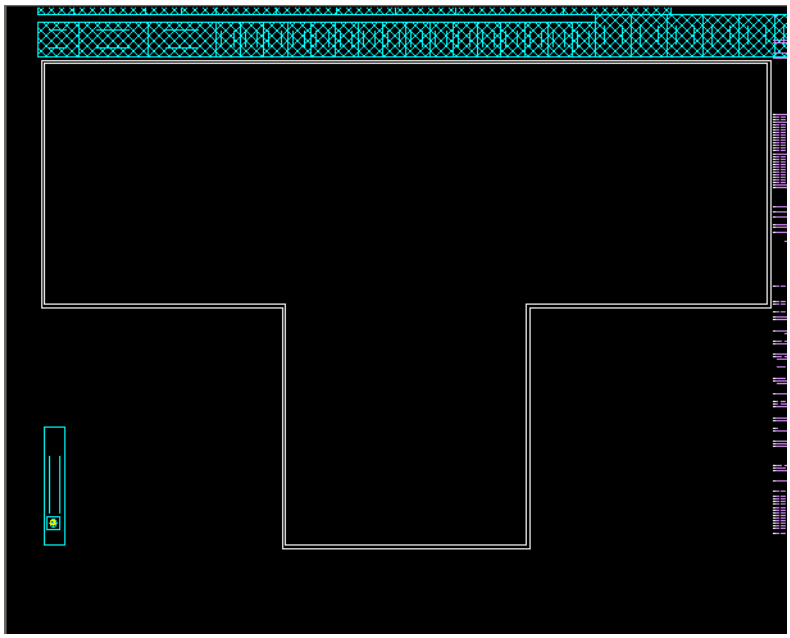


Figure 2-13 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape L` option, which creates a L-shaped rectilinear block.

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

Figure 2-13 L-Shaped Rectilinear Block

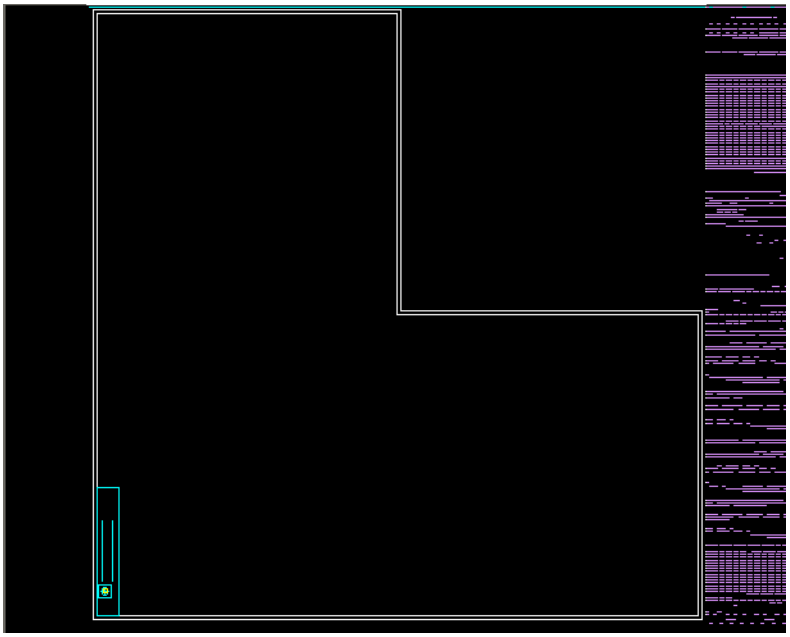


Figure 2-14 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape X` option, which creates a cross-shaped rectilinear block.

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

Figure 2-14 Cross-Shaped Rectilinear Block

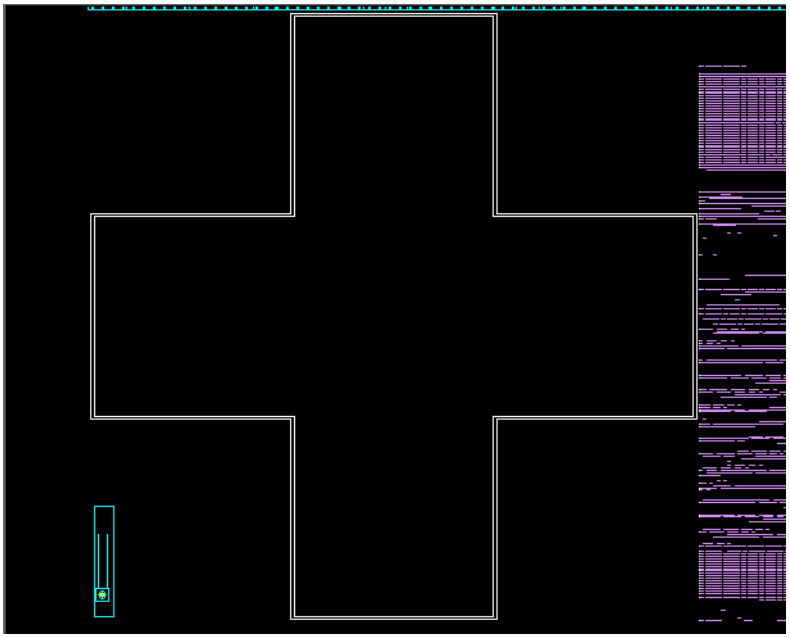


Figure 2-15 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape T` and `-orientation S` options, which creates a T-shaped rectilinear block that is rotated 180 degrees.

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

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

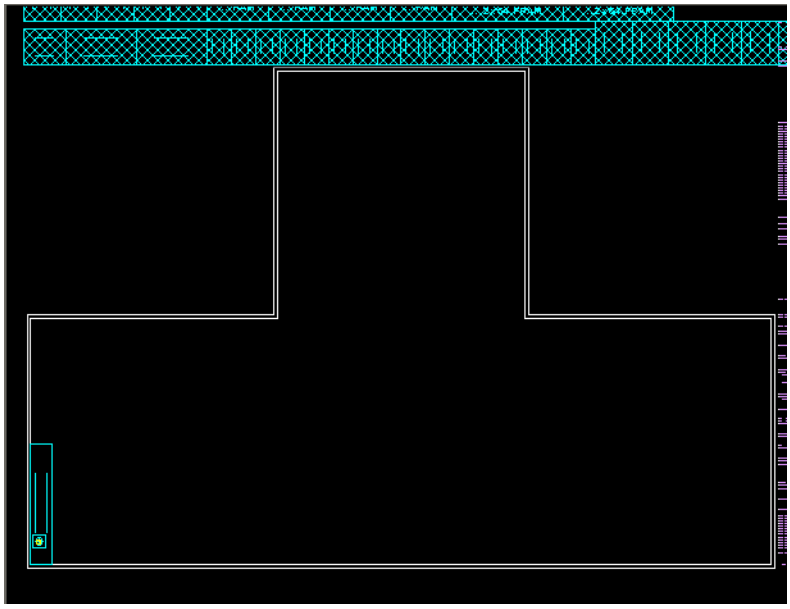
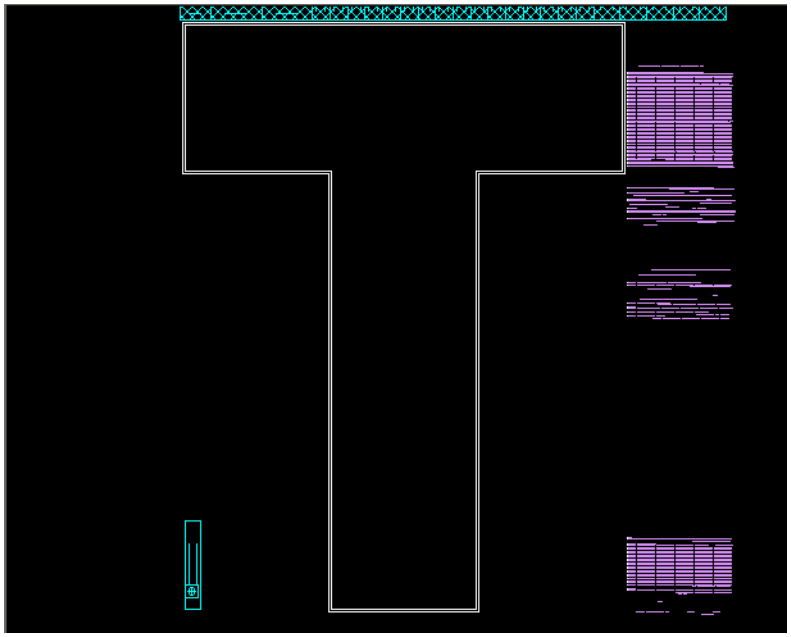


Figure 2-16 shows a floorplan created by using the `initialize_rectilinear_block` command with the `-shape`, `-control_type ratio`, and `-core_side_dim` options. The following command creates a T-shaped floorplan with a base section three times longer than the top section. For more information about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

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

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



Creating a Complex Rectilinear Floorplan

You can create complex rectilinear floorplans by using the `create_boundary` command together with the `initialize_rectilinear_block` command. The `create_boundary` command defines a floorplan based on a simple rectangle or a more complex shape based on the points of a polygon.

To create a complex rectilinear floorplan, define the perimeter by using the `create_boundary` command. Next, use the `initialize_rectilinear_block -use_current_boundary` command to initialize the floorplan using your boundary specification. Table 2-11 describes the `create_boundary` command options. For more information about these options, see the man page.

Table 2-11 create_boundary Command Options

Command option	Description
`-coordinate {{`*`x1 y1`*`} {`*`x2 y2`*`}}`	Defines a rectangular boundary using two points.
`-poly {`*`point1 point2 ... pointn`*`}`	Defines a polygon-based boundary described by the vertices of the polygon. The first and last points in the list must be the same.

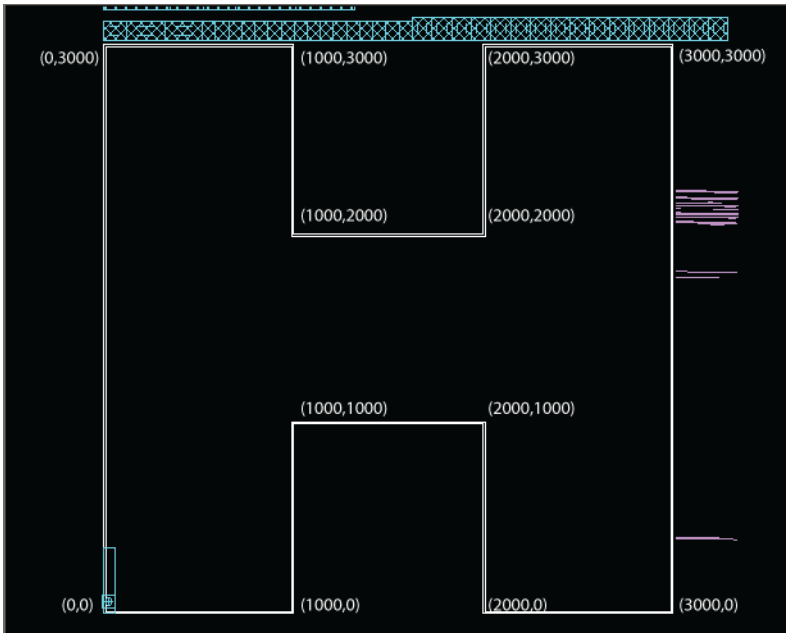
Table 2-11 create_boundary Command Options (Continued)

Command option	Description
`-by_terminal`	Creates a boundary based on a box that contains all terminals in the design.
`-core`	Defines the core area using the bounding box as defined by the `-coordinate` option.
`-left_offset` *l_offset*	Adjusts the left side of the boundary.
`-right_offset` *r_offset*	Adjusts the right side of the boundary.
`-top_offset` *t_offset*	Adjusts the top side of the boundary.
`-bottom_offset` *b_offset*	Adjusts the bottom side of the boundary.
`-lib_cell_type` *type*	Identifies the library cell for which to set the boundary.
design	Identifies the design for which to set the boundary.

Figure 2-17 shows a floorplan created by using the `create_boundary` command and the `initialize_rectilinear_block -use_current_boundary` command.

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

Figure 2-17 H-Shaped Floorplan



Refining the Floorplan

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

- Adjusting the Floorplan
- Manually Modifying the Floorplan

Adjusting the Floorplan

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

Table 2-12 describes the `adjust_fp_floorplan` command options. There is no GUI for the `adjust_fp_floorplan` command. For more information about these options, see the man page.

Table 2-12 adjust_fp_floorplan Command Options

Command option	Description
`-bottom_io2core` *spacing*	Creates a space between the bottom side of the core and the top of the closest pad.
`-core_aspect_ratio` *aspect_ratio*	Defines the target aspect ratio (height/width) for the floorplan.
`-core_height` *height*	Specifies the height of the core area when used with the `-control_type width_and_height` option.
`-core_utilization` *utilization*	Specifies the target utilization (cell placement area/ total area) for the floorplan.
`-core_width` *width*	Specifies the width of the core area when used with the `-control_type width_and_height` option.
`-die_height` *height*	Specifies the die height.
`-die_origin {`*x y*`}`	Specifies the die origin.
`-die_width` *width*	Specifies the die width.
`-fc_in_core` *string*	Specifies the number of flip-chip pads or drivers inside the core.
`-fc_periphery` *string*	Specifies the number of flip-chip pads or drivers within the periphery.
`-flip_first_row true | false`	Flips the first row at the bottom of the core for horizontal rows or the left side of the core for vertical rows. You cannot combine the `-flip_first_row true` option and the `-no_double_back true` option.
`-left_io2core` *spacing*	Creates a space between the left side of the core and the right side of the closest pad.
`-maintain_placement`	Keeps the current placement of standard cells and macros.
`-min_pad_height true | false`	Creates the core area based on the minimum pad height.

Table 2-12 adjust_fp_floorplan Command Options (Continued)

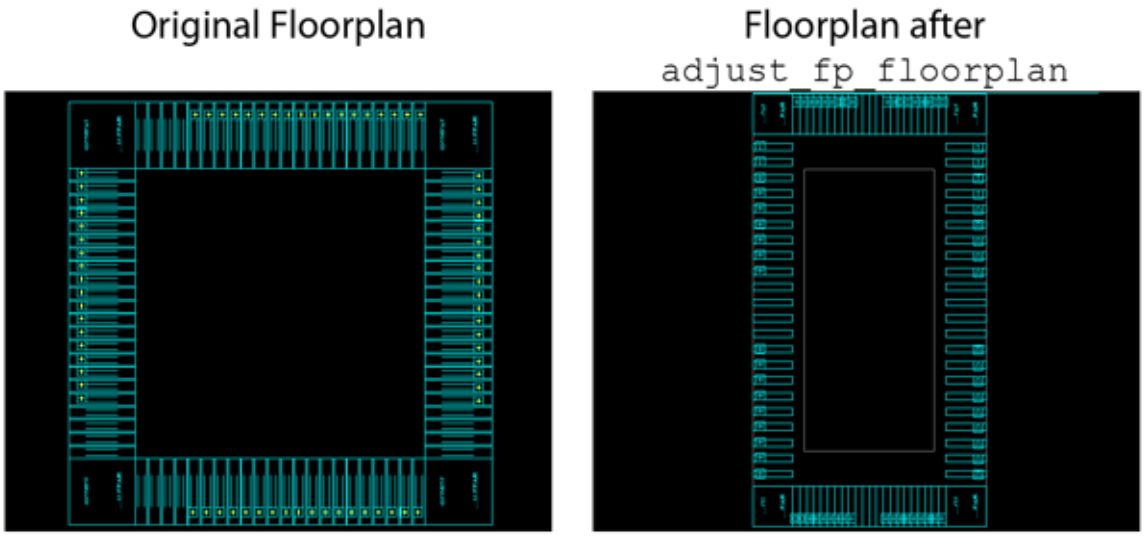
Command option	Description
`-no_double_back true | false`	Prevents placement of paired or flipped rows back-to-back.
`-number_rows` *rows*	Specifies the number of rows in the core area when used with the `-control_type row_number` option.
`-remove_filler_io`	Removes any filler I/O pad cells.
`-right_io2core` *spacing*	Creates a space between the right side of the core and the left side of the closest pad.
`-row_core_ratio` *rows*	Defines the channel space between rows reserved for routing.
`-sm_utilization` *string_list*	Specifies a utilization factor for individual soft macros.
`-start_first_row true | false`	Pairs cell rows starting at the bottom of the core for horizontal rows or the left side of the core for vertical rows. You cannot combine the `-start_first_row true` option and the `-no_double_back true` option.
`-top_io2core` *spacing*	Creates a space between the top of the core and the bottom of the closest pad.
`-use_vertical_row true | false`	Places rows vertically within the core area.

Figure 2-18 shows a floorplan that was initialized by using the `initialize_floorplan` command, and modified by using the `adjust_fp_floorplan` command. The `adjust_fp_floorplan` command makes the following changes:

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

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

Figure 2-18 Adjusted Floorplan



Manually Modifying the Floorplan

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

- Defining the Die Area
- Removing Cell Rows
- Adding Cell Rows
- Defining the Wire Tracks

Defining the Die Area

You can redefine the die area by using the `set_die_area` command. This command accepts two coordinates that define the lower-left corner and upper-right corner of the die. Table 2-13 describes the `set_die_area` command option. For more information about this option, see the man page.

Table 2-13 set_die_area Command Options

Command option	Description
`-coordinate {x1 y1 x2 y2}`	Specifies the lower-left corner and upper-right corner of the die.

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

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

Removing Cell Rows

You can remove cell rows from the floorplan by using the `cut_row` command (or by choosing Floorplan > Cut Site Row in the GUI). This command removes all cell rows in the design or only the rows in the specified area. Table 2-14 describes the `cut_row` command options. For more information about these options, see the man page.

Table 2-14 cut_row Command Options

Command option	Description
`-all` (“All rows in design” check box in the GUI)	Removes all rows from the current design.
`-area {{x1 y1} {x2 y2}}` (“Coordinates” box in the GUI)	Specifies the lower-left corner and upper-right corner of the row removal area.

For example, the following command removes the cell rows in the die area bounded by the coordinates `{0 0}` and `{1000 1000}`.

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

Adding Cell Rows

You can add cell rows to the floorplan by using the `add_row` command (or by choosing Floorplan > Add Site Row in the GUI). The command inserts cell rows based on the configuration options you provide and implements the rows based on the current floorplan. If you specify a design area that already contains rows, the `add_row` command does not create an overlap condition with the existing rows. Table 2-15 describes the `add_row` command options. For more information about these options, see the man page.

Table 2-15 add_row Command Options

Command option	Description
`-area {{x1 y1} {x2 y2}}` (Coordinates box in the GUI)	Specifies the area in which to insert the rows.
`-direction horizontal | vertical` (“Direction” radio buttons in the GUI)	Specifies the row placement orientation.
`-dont_snap_to_existing_row` (“Snap to existing row” check box in the GUI)	Prevents new rows from aligning to existing rows.
`-flip_first_row` (“Flip first row” check box in the GUI)	Flips the first row at the bottom of the core for horizontal rows or the left side of the core for vertical rows. You cannot combine the `-flip_first_row` option and the `-no_double_back` option.
`-in base_array_name` (“Tile base” in the GUI)	Specifies the name of the base array in which to create the rows.
`-minimal_channel_height height` (“Minimum channel height” box in the GUI)	Specifies the minimum channel height for routing between rows.
`-no_double_back` (Uncheck the “Double back” check box in the GUI)	Prevents placement of paired or flipped rows back-to-back. You cannot combine the `-flip_first_row` option and the `-no_double_back` option.
`-no_snap_to_wire_track` (No GUI available for this option)	Prevents the bottom of horizontal rows and the left side of vertical rows from snapping to the wire track.
`-no_start_from_first_row` (Uncheck “Start from first row” check box)	Prevents pairing cell rows starting at the bottom of the core for horizontal rows or the left side of the core for vertical rows.

Table 2-15 add_row Command Options (Continued)

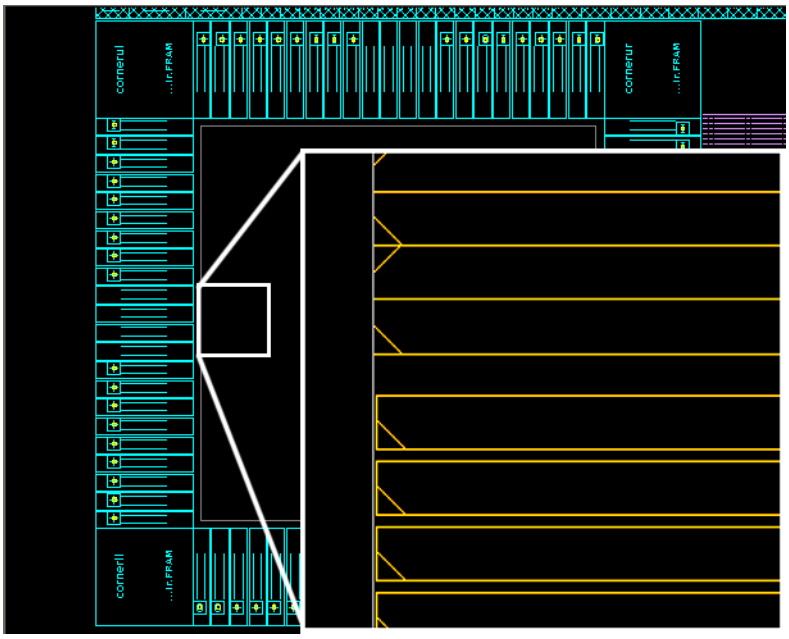
Command option	Description
`-tile_name` *name* (“Tile name” box in the GUI)	Identifies the unit tile to use for creating rows.
`-tile_pattern` *name* (“Tile name filter” box in the GUI)	Specifies the name of the tile pattern to use for creating rows.

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

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

Figure 2-19 shows a zoomed-in view of the section of floorplan modified by using the `add_row` command. Note that the rows in the bottom half of the cutout are inserted by using the `add_row -no_double_back` command, while the rows in the top half of the cutout were originally inserted without using the `-no_double_back` option.

Figure 2-19 Floorplan After Reinserting Rows



Defining the Wire Tracks

You can create routing tracks in the floorplan by using the `create_track` command (or by choosing Floorplan > Create Track in the GUI). Table 2-16 describes the `create_track` command options. For more information about these options, see the man page.

Table 2-16 create_track Command Options

Command option	Description
`-bounding_box {{x1 y1} {x2 y2}}` (“Coordinates” box in the GUI)	Defines the bounding box that encloses the routing tracks.
`-coord num` (“First track location” box in the GUI)	Specifies the location of the first track.
`-count number_of_tracks` (“Number of tracks” box in the GUI)	Specifies the number of tracks to create.
`-dir X | Y` (“Direction” radio buttons in the GUI)	Specifies the track placement direction.

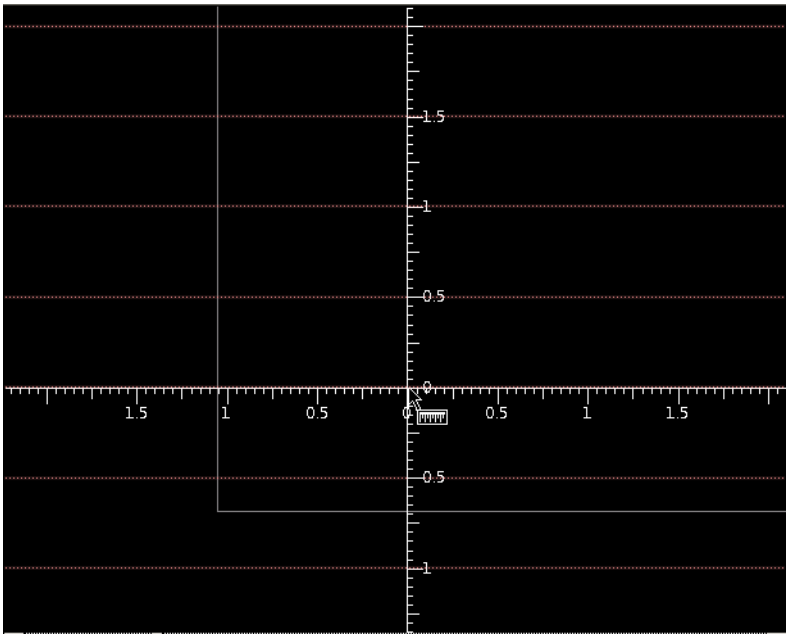
Table 2-16 create_track Command Options (Continued)

Command option	Description
`-layer layer_name` ("Layer" selection area in the GUI)	Assigns the layer to which to add the routing tracks.
`-space track_pitch` ("Pitch" box in the GUI)	Specifies the pitch between each routing track.

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

```
icc_shell> remove_track -all
1
icc_shell> create_track -layer METAL3 -space 0.5
1
icc_shell> report_track
****************************************
Report track
Design : TEST
Version: D-2010.03-ICC-SP1
Date   : Wed Apr 28 14:56:34 2010
****************************************
Layer          Direction     Start         Tracks    Pitch          Attr
--------------------------------------------------------------------------------
Attributes :
         usr : User defined
         def : DEF defined
METAL3             Y         0.250          4307      0.500          usr
1
```

Figure 2-20 Floorplan After Inserting Wire Tracks



Adjusting the I/O Placement

You can refine the spacing and relative locations of I/O pads by using the `adjust_fp_io_placement` command (or by choosing Floorplan > Adjust I/O Placement in the GUI). The `adjust_fp_io_placement` command operates on one side of the chip at a time and supports an undo function that reverts the design to its previous state. Table 2-17 describes the `adjust_fp_io_placement` command options. For more information about these options, see the man page.

Table 2-17 adjust_fp_io_placement Command Options

Command option	Description
`-offset` *`offset_value`* (“Offset from” radio button and box in the GUI)	Specifies the distance between the bottommost or leftmost I/O pad and the chip edge.
`-pitch` *`io_pitch`* (“Pad pitch” radio button in the GUI)	Specifies the pitch between I/O pads.

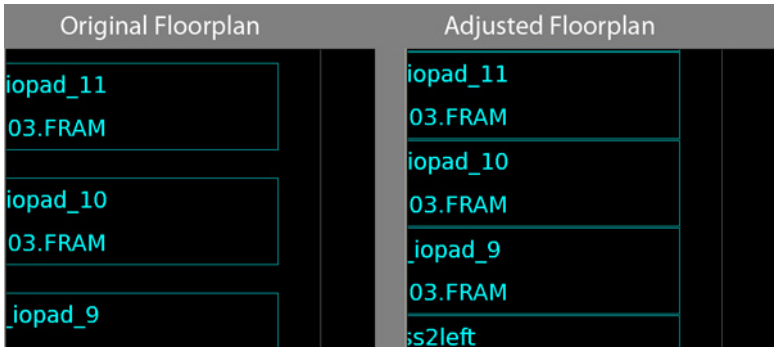
Table 2-17 adjust_fp_io_placement Command Options (Continued)

Command option	Description
`-side l | r | t | b` ("Side" radio button and Left, Right, Bottom, Top check boxes in the GUI)	Specifies the side of the chip to adjust.
`-spacing` *`io_spacing`* ("Spacing" radio button and box in the GUI)	Specifies the spacing between I/O pads.
`-undo` ("Undo" button in the GUI)	Reverts the design to its previous state prior to the `adjust_fp_io_placement` command.
list_of_io_cells	Specifies the I/O cells to adjust.

Figure 2-21 shows a floorplan adjusted by using the `adjust_fp_io_placement` command. In this example, the original pad spacing is 30 microns as specified by using the `set_pad_physical_constraints -min_left_iospace` command. The `adjust_fp_io_placement -side l -spacing 1` command changes the spacing between the I/O pads to 1 micron for pads on the left side of the chip.

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

Figure 2-21 Floorplan With Adjusted I/O Pads



Saving the Floorplan Information

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

- Writing the Floorplan File
- Writing the Floorplan to a DEF File
- Writing Floorplan Physical Constraints for DC Ultra Topographical Mode

Writing the Floorplan File

You can write the current floorplan to a Tcl file by using the `write_floorplan` command (or by choosing Floorplan > Write Floorplan in the GUI). The `write_floorplan` command creates a Tcl file that contains objects, placement, and attribute information for I/Os, terminals, standard cells, hard macros, and routing tracks in the floorplan. The Tcl file can be used to rebuild the floorplan by using the `read_floorplan` command. You can control the information written to the Tcl file by using command options. The command does not write relative placement group and relative placement keepout information. Table 2-18 describes the `write_floorplan` command options. For more information about these options, see the man page.

Table 2-18 write_floorplan Command Options

Command option	Description
`-all` (“Write all” radio button in the “Top level” section of the GUI)	Writes all floorplan information to the Tcl script.
`-cell` *`design_name`* (No GUI, defaults to current design)	Specifies the input design for which to write the floorplan.
`-create_bound` (“Create Bounds” check box in the GUI)	Writes `create_bounds` commands instead of `update_bounds` commands for bound objects.
`-create_terminal` (“Create terminals” check box in the GUI)	Writes `remove_terminal` and `create_terminal` commands to the Tcl script when used with the `-placement {terminal}` option.

Table 2-18 write_floorplan Command Options (Continued)

Command option	Description
`-no_bound` (Uncheck the “Bounds” check box in the “Top level” section of the GUI)	Prevents the writing of bound information to the Tcl script.
`-no_create_boundary` (Uncheck the “Boundary” check box in the GUI)	Prevents the writing of boundary information to the Tcl script.
`-no_placement_blockage` (Uncheck the “Placement blockages” check box in the “Top level” section of the GUI)	Prevents the writing of placement blockage information to the Tcl script.
`-no_plan_group` (Uncheck the “Plan groups” check box in the “Top level” section of the GUI)	Prevents the writing of plan group information to the Tcl script.
`-no_route_guide` (Uncheck the “Route guides” check box in the “Top level” section of the GUI)	Prevents the writing of route guide information to the Tcl script.
`-no_voltage_area` (Uncheck the “Voltage areas” check box in the “Top level” section of the GUI)	Prevents the writing of voltage area information to the Tcl script.
`-objects` *`object_collection`* (“Objects” box in the GUI)	Specifies the objects to write to the Tcl script.
`-pin_guide` (“Pin guides” check box in the GUI)	Writes pin guide information to the Tcl script.
`-placement {`*`placement_info_types`*`}` (“Placement” section check boxes in the “Top level” section of the GUI)	Writes the specified placement information to the Tcl script. Valid placement values are: `io`, `std_cell`, `hard_macro`, `soft_macro`, and `terminal`. You can specify multiple values.

Table 2-18 write_floorplan Command Options (Continued)

Command option	Description
`-preroute` ("Preroutes" check box in the "Top level" section of the GUI)	Writes preroute information to the Tcl script.
`-row` ("Rows" check box in the GUI)	Writes standard cell row information to the Tcl script.
`-sm_all` ("Write all" radio button in the "Hierarchical" section of the GUI)	Writes floorplan information for all soft macros.
`-sm_bound` ("Bounds" check box in the "Hierarchical" section of the GUI)	Writes floorplan bound information for soft macros.
`-sm_cell_row` ("Cell rows" check box in the "Hierarchical" section of the GUI)	Writes cell row information for soft macros.
`-sm_placement {`*placement_info_types*`}` ("Placement" check boxes in the "Hierarchical" section of the GUI)	Writes the specified placement information to the Tcl script for soft macros. Valid placement values are: `io`, `std_cell`, `hard_macro`, `soft_macro`, and `terminal`.
`-sm_placement_blockage` ("Placement blockages" check box in the "Hierarchical" section of the GUI)	Writes placement blockage information for soft macros.
`-sm_plan_group` ("Plan groups" check box in the "Hierarchical" section of the GUI)	Writes plan group information for soft macros.
`-sm_preroute` ("Preroutes" check box in the "Hierarchical" section of the GUI)	Writes preroute information for soft macros.

Table 2-18 write_floorplan Command Options (Continued)

Command option	Description
`-sm_route_guide` ("Route guides" check box in the "Hierarchical" section of the GUI)	Writes route guide information for soft macros.
`-sm_track` ("Tracks" check box in the "Hierarchical" section of the GUI)	Writes routing track information for each soft macro.
`-sm_voltage_area` ("Voltage areas" check box in the "Hierarchical" section of the GUI)	Writes voltage area information for soft macros.
`-track` ("Track" check box in the "Top level" section of the GUI)	Writes routing track information for soft macros.

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

```
icc_shell> write_floorplan floorplan_1.tcl
1
icc_shell> exec cat floorplan_1.tcl
############################################################
# Created by write_floorplan on Thu Apr 29 11:10:35 2010
############################################################
undo_config -disable

remove_die_area

create_die_area  \
  -poly { {0.000 0.000} {2154.560 0.000} {2154.560 2153.330} \
          {0.000 2153.330} {0.000 0.000}}
set oldSnapState [set_object_snap_type -enabled false]
set_object_snap_type -enabled $oldSnapState
undo_config -enable
```

Writing the Floorplan to a DEF File

You can save the current floorplan to a DEF file by using the `write_def` command (or by choosing File > Export > Write DEF in the GUI). The `write_def` command writes the floorplan, including design netlist, layout, and constraint information. Table 2-19 describes the `write_def` command options. For more information about these options, see the man page.

Table 2-19 write_def Command Options

Command option	Description
`-output` *`filename.def`* ("Output file name" box in the GUI)	Specifies the name of the output DEF file.
`-all_vias` ("Include all vias" check box in the GUI)	Writes all vias in the design and the technology file to the DEF output file.
`-blockages` ("Blockages" check box in the GUI)	Writes the BLOCKAGES section to the DEF output file.
`-components` ("Components" check box in the GUI)	Writes the COMPONENTS section to the DEF output file.
`-compressed` ("Output compressed" check box in the GUI)	Compresses the DEF output file using gzip compression.
`-diode_pins` ("Output diode pins" in the GUI)	Includes diode pins in the NETS section of the DEF output file.
`-fills` (Not supported in the GUI)	Writes the FILLS section to the DEF output file.
`-fixed` ("Fixed cells only" in the GUI)	Writes only cells with the `is_fixed` variable set to true to the COMPONENTS section of the DEF output file.
`-floating_metal_fill` ("Floating metal fill" check box in the GUI)	Includes floating metal fill in the FILLS section of the DEF output file.

Table 2-19 write_def Command Options (Continued)

Command option	Description
`-lef` *`file_name`* (“Export incremental LEF for rotated vias and NDR” check box in the GUI)	Writes rotated vias and design-specific nondefault rules to the LEF file.
`-macro` (“Nonstandard cells” check box in the “Component Options” section in the GUI)	Writes macro cells to the COMPONENTS section of the DEF output file and does not write standard cells.
`-nets` (“Nets” check box in the GUI)	Writes the NETS section to the DEF output file.
`-no_legalize` (Uncheck the “Enable legalization capability” check box in the GUI)	Prevents name legalization; the command does not escape special characters in the DEF output file.
`-nondefault_rule` (“Output all design nondefault rules” in the GUI)	Writes the NONDEFAULTRULES section to the DEF output file.
`-notch_gap` (“Notch/Gap” check box in the “Special nets” section of the GUI)	Writes notch and gap information to the SPECIALNETS section of the DEF output file.
`-pg_metal_fill` (“P/G metal fill” check box in the “Special Nets” section of the GUI)	Writes power and ground metal fill information to the SPECIALNETS section of the DEF output file.
`-pins` (“Pins” check box in the GUI)	Writes the PINS section to the DEF output file.
`-placed` (“Placed cells only” check box in the “Components” section of the GUI)	Writes only placed cells to the COMPONENTS section of the DEF output file.

Table 2-19 write_def Command Options (Continued)

Command option	Description
`-regions_groups` ("Regions/Groups" check box in the GUI)	Writes the REGIONS and GROUPS sections to the DEF output file.
`-routed_nets` ("Routed nets only" check box in the GUI)	Writes only routed nets to the SPECIALNETS and NETS sections of the DEF output file.
`-rows_tracks_gcells` ("Rows/Tracks/GCells" check box in the GUI)	Writes the ROW, TRACK, and GCELLGRID sections to the DEF output file.
`-scanchain` ("Scan chain" check box in the GUI)	Writes the SCANCHAINS section to the DEF output file.
`-specialnets` ("Special nets" check box in the GUI)	Writes the SPECIALNETS section to the DEF output file.
`-unit 100 | 200 | 1000 | 2000 | 10000 | 20000` ("Generate in units of" box in the GUI)	Specifies the value appended to the UNITS DISTANCE MICRONS statement in the DEF output file.
`-verbose` ("Verbose message" check box in the GUI)	Writes additional informational messages to the screen during command processing.
`-version` *def_version* ("Version" box in the GUI)	Specifies the DEF version string to write to the DEF output file.
`-vias` ("Vias" check box in the GUI)	Writes the VIAS section to the DEF output file.

This example uses the `write_def` command to write the floorplan to a DEF file.

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

Writing Floorplan Physical Constraints for DC Ultra Topographical Mode

You can save floorplan information from IC Compiler for use in Design Compiler. The `write_physical_constraints` command writes floorplan information to a Tcl file for use in Design Compiler topographical mode. Table 2-20 describes the `write_physical_constraints` command options. For more information about these options, see the man page.

Table 2-20 write_physical_constraints Command Options

Command option	Description
`-output` *`tcl_file`*	Specifies the name of the Tcl output file.
`-no_site_row`	Prevents the writing of site row commands.
`-pre_route`	Writes preroute information as `create_net_shape` commands.
`-port_side`	Writes side constraints for ports.

The following command writes the physical constraints to the constraints.tcl file:

```
icc_shell> write_physical_constraints -output constraints.tcl
Info: output rectangular core area.
Info: output location of 72 ports.
Info: output location of 72 IO pad cells.
Info: output location of 41 macro instances.
Info: output 0 placement blockages.
Output 0 voltage areas.
1
```

Reading In an Existing Floorplan

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

- Reading DEF Files
- Reading Floorplan Files

Reading DEF Files

To read a DEF file, use the `read_def` command (or by choosing File > Import > Read DEF in the GUI). Table 2-21 describes the `read_def` command options. For more information about these options, see the man page.

Table 2-21 read_def Command Options

Command option	Description
`-check_only` (“Check only mode” check box in the GUI)	Analyzes the DEF file without modifying the design.
`-enforce_scaling` (“Enforce scaling” check box in the GUI)	Ignores roundoff errors during unit conversion.
`-inexactly_matched_via_to_inst` (“Translate inexactly matched vias to Cell instances” radio button in the GUI)	Creates via cell instances from vias that have nonmatching enclosure dimensions.
`-lef { file1 file 2}` (“Input lef files” box in the GUI)	Imports rotated via information from the specified LEF files.
`-no_incremental` (Uncheck the “Incremental” check box)	Removes existing physical annotations.
`-preserve_wire_ends` (“Preserve wire ends” check box in the GUI)	Creates power and ground nets as paths without extensions, creates special fill wiring as rectangles, and creates all other nets as wires with half-width extensions.
`-snet_no_shape_as_detail_route` (“Mark no-shape wire segments as Detailed route” radio button in the GUI)	Marks wire segments without the “+ SHAPE” statement with the `signal_route_detail` route type.

Table 2-21 read_def Command Options (Continued)

Command option	Description
`-snet_no_shape_as_user_enter` ("Mark no-shape wire segments as User entered" radio button in the GUI)	Marks wire segments without the "+ SHAPE" statement with the `user_enter` route type.
`-turn_via_to_inst` ("Translate turn vias to Cell instances" radio button)	Creates vias as cell instances, not as path objects.
`-verbose` ("Verbose" check box in the GUI)	Prints additional debugging messages.

The following command reads the chip.def DEF file with no options.

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

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

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

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

Reading Floorplan Files

You can use the `read_floorplan` command to import a previously saved floorplan file. A floorplan file is Tcl script file created by using the `write_floorplan` command (or by choosing Floorplan > Write Floorplan in the GUI). For more information about creating a

floorplan file, see “Writing the Floorplan File” on page 2-42. Table 2-22 describes the `read_floorplan` command option. For more information about this option, see the man page.

Table 2-22 read_floorplan Command Option

Command Option	Description
`-echo`	Displays each command in the floorplan script as it is executed.

Note:

If the power and ground nets are not explicitly defined in your netlist, you must create the power and ground connections by using the `derive_pg_connection` command before reading the floorplan file. For information about how to do this, see “Creating Logical Power and Ground Connections” on page 8-59.

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

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

Copying a Floorplan From Another Design

To copy physical data from the layout (CEL) view of one design in the current Milkyway design library to another, use the `copy_floorplan` command (or by choosing Floorplan > Copy Floorplan). Table 2-23 describes the `copy_floorplan` command options. For more information about these options, see the man page.

Table 2-23 copy_floorplan Command Options

Command option	Description
`-filler` (“Copy Filler cells placement” check box in the GUI)	Copies placement information for filler cells.
`-from` *`design_name`* (“From cell” box in the GUI)	Specifies the name of the source design.
`-incremental` (“Incremental (non-overwriting)” check box in the GUI)	Specifies that the command does not update existing named objects.

Table 2-23 copy_floorplan Command Options (Continued)

Command option	Description
`-library` *library_name* (“Library name” box in the GUI)	Specifies the name of the Milkyway design library for both the source and destination designs.
`-macro` (“Copy macro placement” check box in the GUI)	Copies placement information for macros.
`-pad` (“Copy IO pad placement” check box in the GUI)	Copies placement information for I/O, corner, and flip-chip pads.
`-power_plan` (“Connect power net and copy power planning data” check box in the GUI)	Copies power net connection and routing information.
`-verbose` (“Verbose” check box in the GUI)	Prints additional debugging messages.

Note:

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

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

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

This command copies the following physical data:

- General floorplan data

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

- Obstructions

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

- Cell instances

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

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

- Preroutes

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

- Power net connections
- Plan groups
- Macro cell placement (optional)

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

- I/O pad placement (optional)

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

- Filler cell placement (optional)

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

- Power plan (optional)

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

The command does not copy the following data:

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

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

Note:

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

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

3

Automating Die Size Exploration

This chapter describes how to use MinChip technology in IC Compiler. Given a design that is floorplanned and placed, MinChip technology automates the processes of exploring and identifying the valid die areas to determine the smallest routable die size for your design while maintaining the relative placement of hard macros and I/O cells. Optionally, it uses power network synthesis for power routing that meets voltage drop requirements and routing estimation technology to access routing feasibility. The technology is integrated into the Design Planning tool through the `estimate_fp_area` command. The input is a physically flat Milkyway CEL view.

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

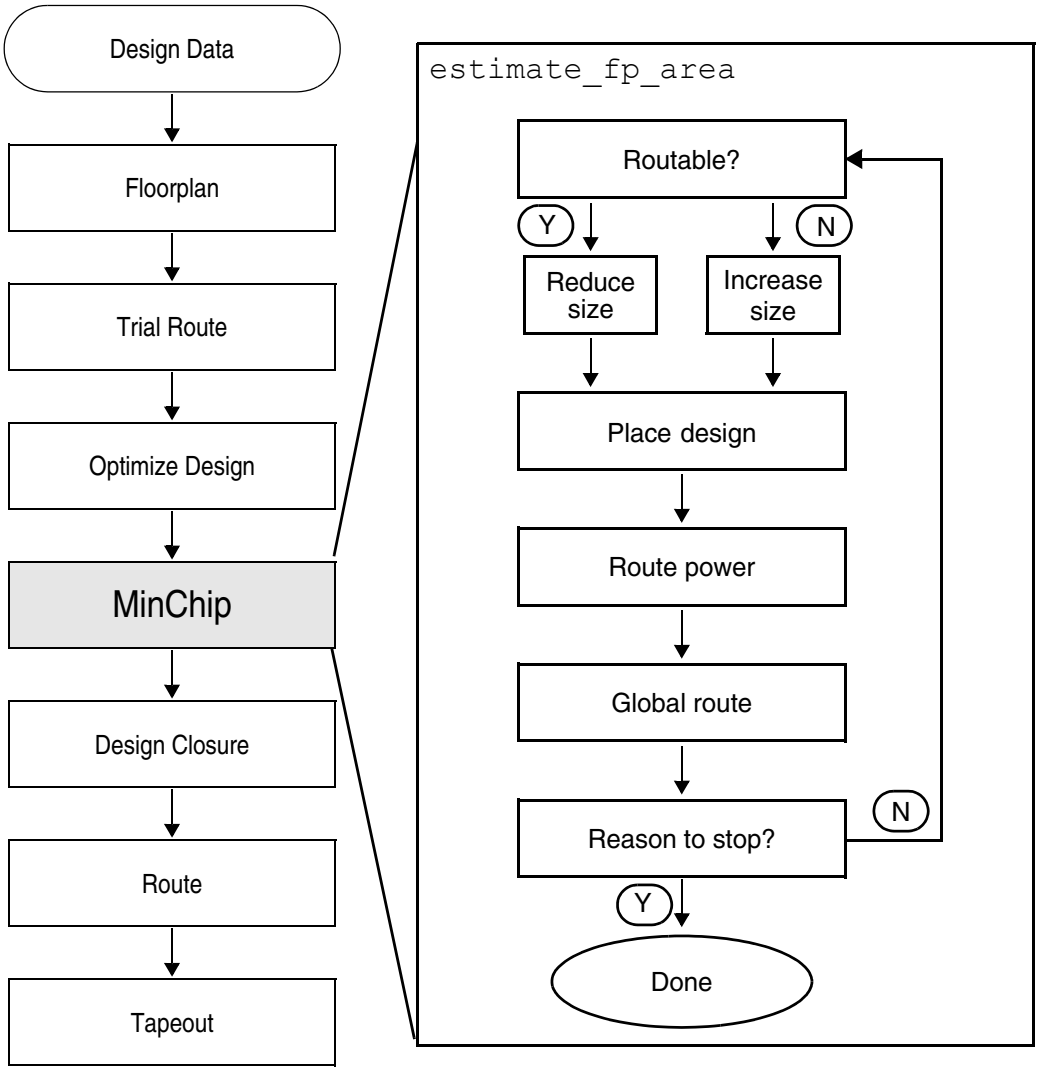
By using the MinChip technology, floorplan sizing iterations can be completed in hours, returning a result that represents the minimum area in which the design can be implemented and remain routable while retaining the characteristics of the original floorplan.

This chapter includes the following sections:

- Using the estimate_fp_area Command and Options
- Using Multivoltage Designs in MinChip Technology

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

Figure 3-1 MinChip Technology Flow and Steps



The `estimate_fp_area` command performs the following steps inside MinChip technology:

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

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

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

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

2. Places all the design cells.

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

3. Routes the design.

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

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

4. Analyzes the routing.

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

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

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

Preparing the Design for MinChip Technology in IC Compiler

Because MinChip technology does exactly what you specify based on the options you select, it is important to understand the nuances of your design, so that MinChip technology can achieve the best reduction in die size in the fastest time possible. The quality of the results depends on the input CEL view and the `estimate_fp_area` command options you select.

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

Note:

Applying MinChip technology to netlists and floorplans that are not implementation proven is not recommended as the results can be misleading. If the final netlist is larger than the netlist input to MinChip technology, it might not fit.

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

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

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

  To create an accurate power mesh and rings, use the `-run_pns_script` option.
  - To estimate the power network, use the power network synthesis script file that you developed during the detailed implementation of the floorplan.

  - If a custom power structure is required, create a power network synthesis script by using the `synthesize_fp_rail` command to emulate the power network. It models the custom power structure by specifying the width, pitch, and offset of the core rings, block rings, and power straps. The objective is to accurately consume the routing resources that are required by the power network.

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

- Preroute standard cell scripts

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

  To run the preroute script within the `estimate_fp_area` command, specify the script name using the `-run_preroute_script` option. Do this so that metal1 routing resources as well as the stacked vias are taken into account during die size reduction.

- Relative hard macro placement

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

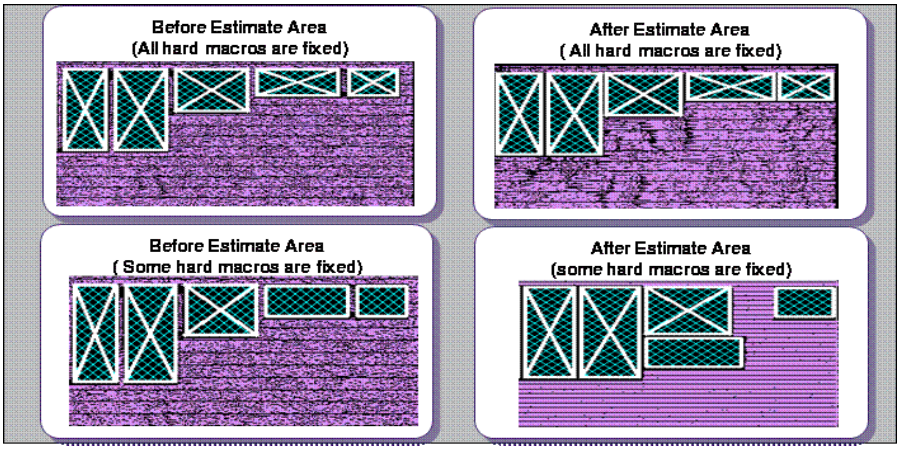
  The `estimate_fp_area` command attempts to maintain the relative placement of unfixed macros, but that relative placement cannot be guaranteed. Macros that are not fixed can move to nearby locations.

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

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

  Figure 3-2 on page 3-6 shows before and after examples of maintaining relative hard macro placement.

Figure 3-2 Maintaining Relative Hard Macro Placement



- Hard macro padding

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

  - By default, IC Compiler calculates the padding for hard macros. You can set specific values for padding by instance or master name. It is also possible, but not recommended, to set the padding to zero.
  - Although macro padding resembles a halo blockage around a macro, you can specify different values for different edges.
  - If the padding of adjacent macros abuts or overlaps before the `estimate_fp_area` command is run, the macros can be packed closer together.
  - If the padding of adjacent macros does not overlap before you run the `estimate_fp_area` command, then it will not overlap after running the `estimate_fp_area` command.
  - If the macro padding overlaps with other padding before you run the `estimate_fp_area` command, it will still overlap other padding after the `estimate_fp_area` command is run, but it will not overlap another macro.

- Sliver size

  The -`sliver_size` option defines a “critical” distance for the `estimate_fp_area` command. If the distance between two objects (core edges, blockages, or macros) is less than or equal to the `sliver_size`, before running the `estimate_fp_area` command, then this distance is not reduced by the `estimate_fp_area` command.

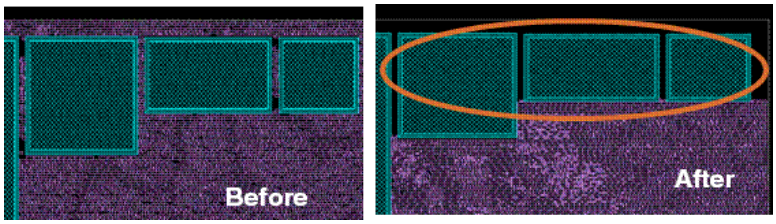
  The default value for the `sliver_size` option is 0.

  Analyze your floorplanned design to determine an appropriate value for the `sliver_size` and, before running the `estimate_fp_area` command, set the following parameter.

`set_fp_placement_strategy -sliver_size` *`distance`*

If the distance between two hard macros is less than or equal to the sliver size, the distance is honored by the `set_fp_placement_strategy -sliver_size` *`distance`* parameter, as shown in Figure 3-3.

Figure 3-3 Sliver Size Packing Limit



- Placement blockages

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

- Edge blockages

  A blockage that covers any side of a core area is called an “edge blockage.” Edge blockages must extend over the complete edge dimension. In some cases, edge blockages are used to account for implementation requirements. For example, consider a design that requires an M1/M2 core ring as part of the power mesh. Within the `estimate_fp_area` command, the power structure is formed after placement. If the core rings are likely to cover the placement area, a problem exists. Standard cells might have been placed in these areas. Edge blockages can reserve the placement area for the rings. During the `estimate_fp_area` iterations, the relative placement of edge blockages is maintained in relation to the core area.

- Blockages over fixed hard macros

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

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

- Filler cells

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

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

Using the Estimate Area GUI

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

By default, area estimation uses the Zroute router. To use the classic router instead, run the following command before running the `estimate_fp_area` command.

```
icc_shell> set_route_mode_options -zroute false
```

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

1. Choose Floorplan > Estimate Area.

   The Estimate Area dialog box appears.

   Alternatively, you can use the `estimate_fp_area` command.

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

   - Floorplan Control
   - Search Control
   - Power Network Control

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

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

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

Variable aspect ratio – Allows the command to vary the aspect ratio. This is the default behavior.

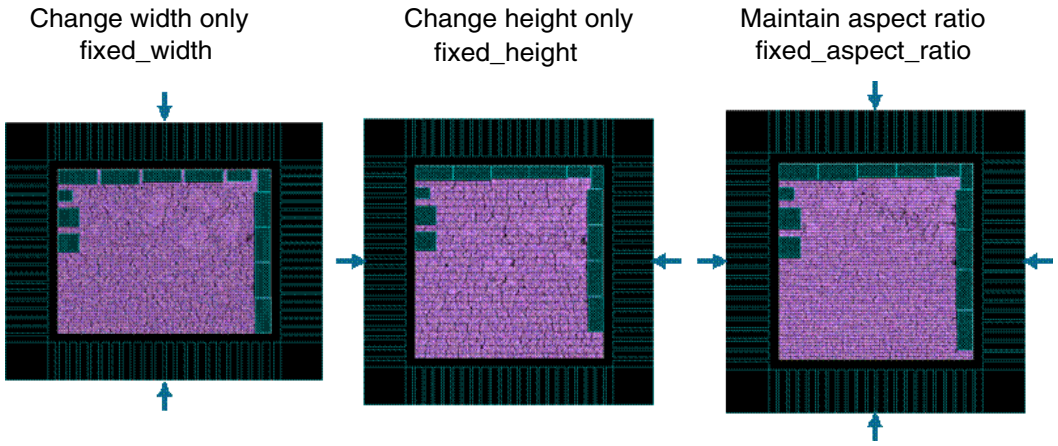
Fixed aspect ratio – Honors the aspect ratio of the original design size and stops shrinking the width or height of the die when the width or height becomes pad or macro limited.

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

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

Figure 3-4 shows an example of controlling the chip sizing.

Figure 3-4 Controlling the Chip Size



- Specify the core area boundaries.

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

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

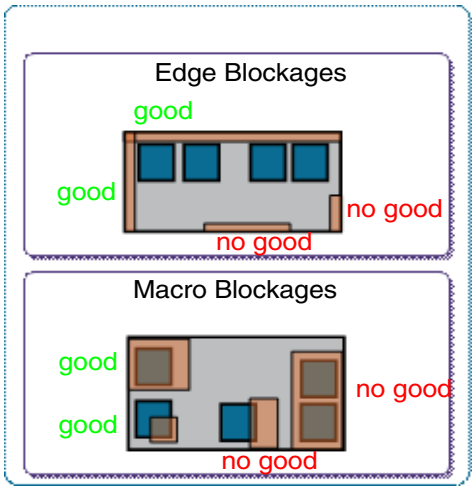
  Maximum width – Enter the upper bound of the core width in microns to stop the die search process. By default, the command does not consider die sizes that result in utilizations less than 0.30, or 30%.

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

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

  Some blockages are recognized as macro blockages or edge blockages, as shown in Figure 3-5.

Figure 3-5 Blockage Types



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

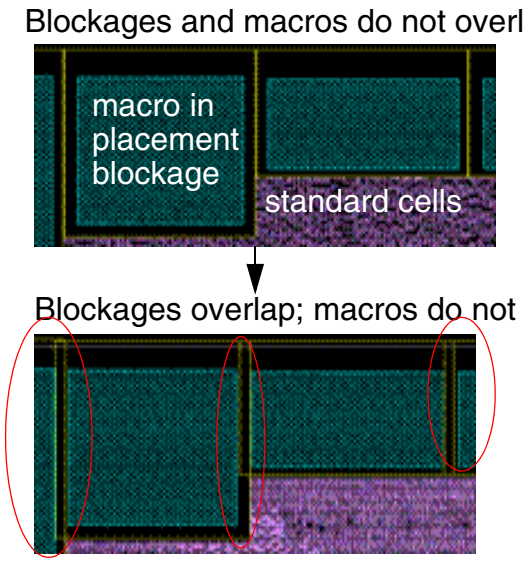
Macro blockages are placement blockages assigned to macros. If a placement blockage overlaps a macro such that it covers 70 percent or more of the macro area, the placement blockage is assumed to be associated with the macro. The purpose of a macro blockage is to keep standard cells from being placed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

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

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

During die size reduction, macro blockages move along with the macros, which might result in blockage overlaps, as shown in Figure 3-6 on page 3-11. Blockage overlaps are allowed. Macro overlaps, however, are not allowed.

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



Note:

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

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

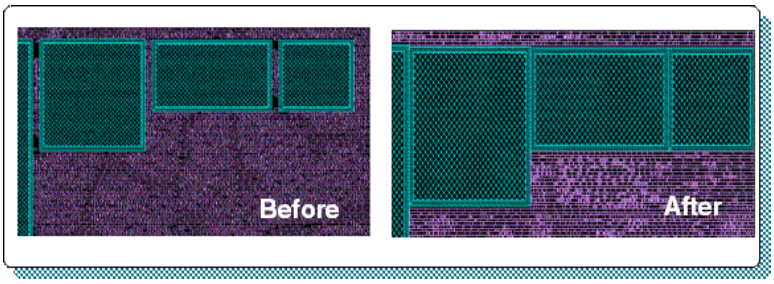
If there are no overlaps between the macro padding, and the distance between the macros is less than or equal to the `sliver_size`, the `estimate_fp_area` command does not reduce the space between the hard macros.

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

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

If there is a hard macro keepout, macros can be packed until the keepout region abuts, as shown in Figure 3-7 on page 3-12.

Figure 3-7 Set Keepout Region Packing Limit



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

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

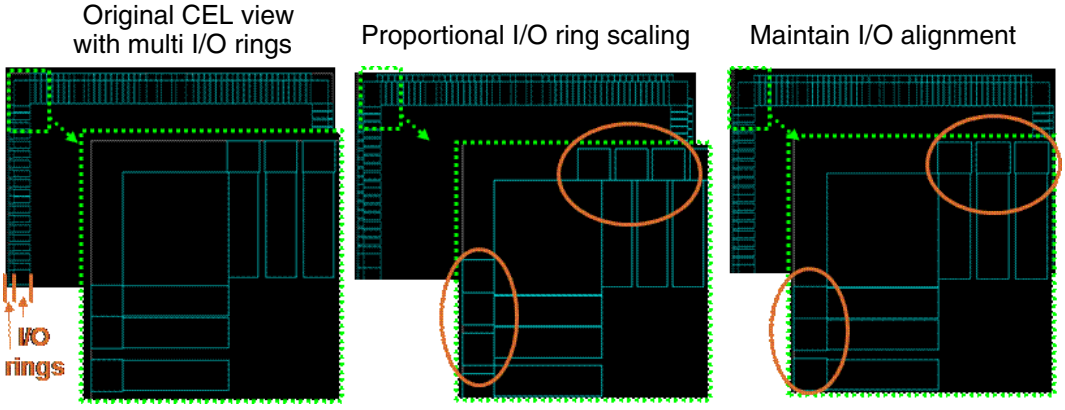
  Figure 3-8 shows an example of maintaining I/O pad alignment.

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

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

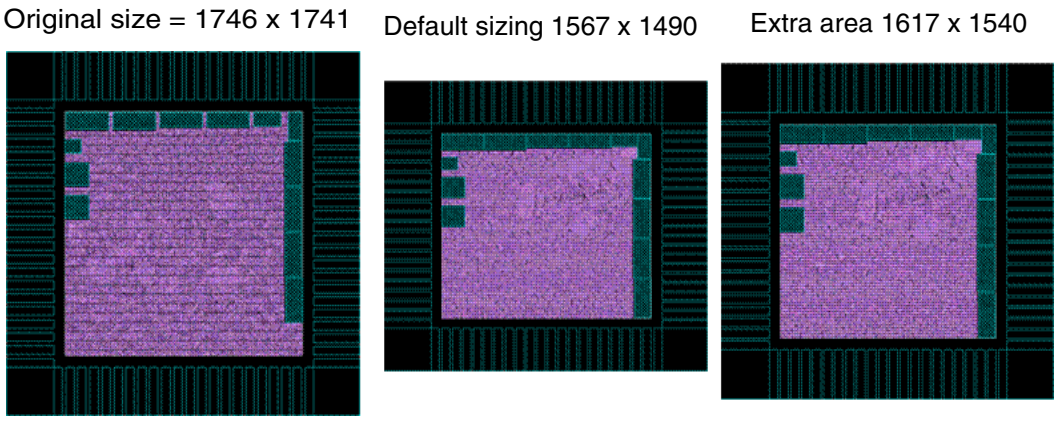
  When you run the `estimate_fp_area` command and select the “Maintain I/O pad alignment” option, the shrinkage occurs by eliminating the unnecessary space in the I/O ring and by maintaining the I/O alignment between the I/O rings, as shown in the image on the right.

Figure 3-8 Maintaining I/O Alignment



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

Figure 3-9 Increasing Space in the Core Area



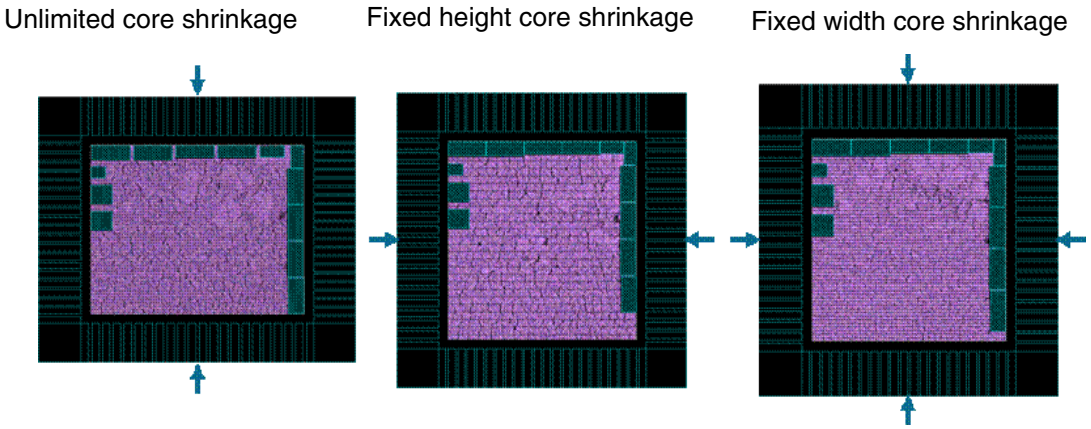
- Core sizing only – Core sizing applies only to full-chip designs. In designs that have complex custom I/O structures, the tool does not modify the I/O structure appropriately. You can shrink the size of the core area only and ignore the I/O pad ring in the input floorplan. Note that if the core size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

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

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

  Figure 3-10 on page 3-14 shows an example of core sizing within the I/O ring.

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



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

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

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

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

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

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

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

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

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

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

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

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

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

  Figure 3-11 shows an example of extracting rectangular and rectilinear core rings, and straps.

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

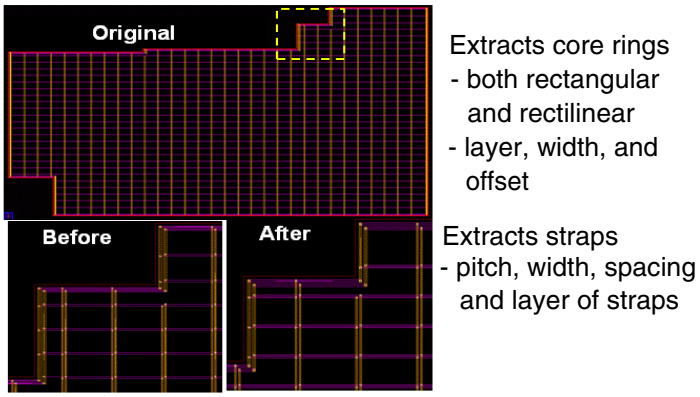
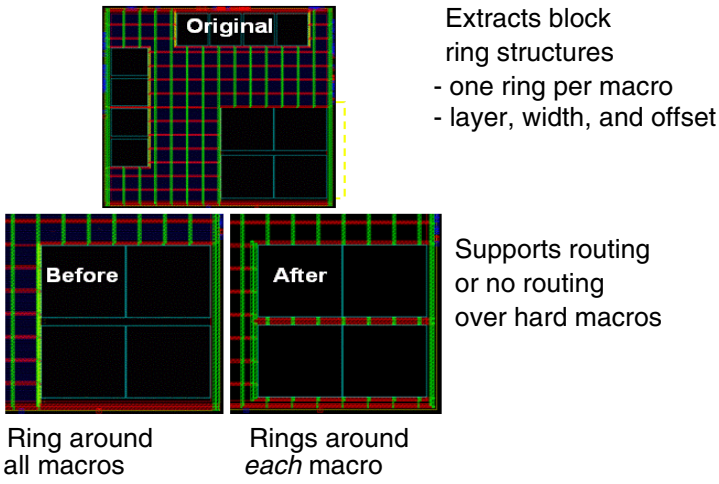


Figure 3-12 on page 3-16 shows an example of extracting block ring structures. Routing or no routing over hard macros is also supported.

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



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

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

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

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

  ```
  VDD, VSS
  ```

  By default, no power net synthesis is run.

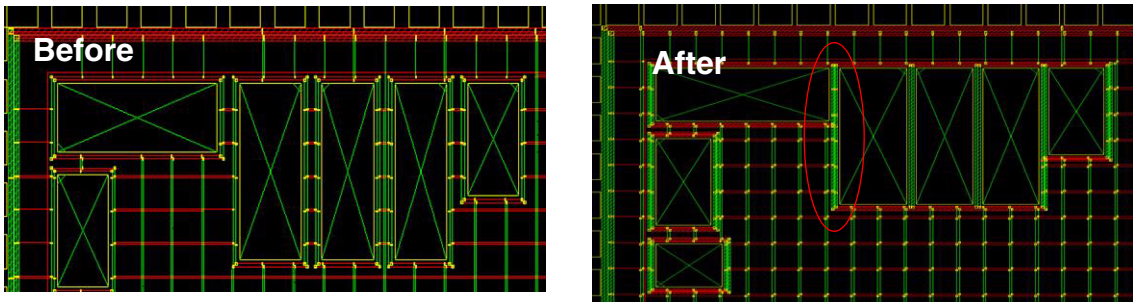
- Custom PNS script – Runs a custom power network synthesis script instead of the default settings used to run power network synthesis.

  By default, no power network synthesis is run.

  It is important that the script not contain any absolute coordinates as they do not apply to the various design sizes that the `estimate_fp_area` command iterates through.

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

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



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

  It is important that the script not contain any absolute coordinates as they do not apply to the various design sizes that the `estimate_fp_area` command iterates through.

  By default, no preroutes are created.

5. Save the result of the die size exploration.
   - Save as – Explicitly name the saved CEL view. By default, the tool saves the last routable result as a CEL view named *design_name* EstimateArea.
6. Select OK or Apply.

   When you run the `estimate_fp_area` command, the command does the following:

   - Saves the starting CEL view with the original cell name and starts working on this CEL view.
   - Deletes both signal and power routing.
   - Maintains the orientation of the fixed macros; the command does not rotate the macros.
   - Tests the routability of the input CEL view by using the current utilization of the CEL core to set the die size. The `estimate_fp_area` command then attempts to make the die size as small as possible.

- Stops if the design is pad limited. If the design becomes pad limited during estimate area iterations, the command stops and informs you that sizing stopped because the design had become pad limited.

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

Using the estimate_fp_area Command and Options

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

- Search Control
- Floorplan Control
- Power Network Control

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

The `estimate_fp_area` command uses the following syntax:

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

By default, the `estimate_fp_area` command uses the Zroute router. To use the classic router instead, run the following command before running the `estimate_fp_area` command.

```
icc_shell> set_route_mode_options -zroute false
```

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

Table 3-1 MinChip Technology Search Control Options

Search control options	Description
`-min_height` *min_height*	Specifies the lower bound of the core height, in microns, required to stop the die search process. It stops making a routable core smaller if the total utilization for the core is less than the target value you specify. By default, no die sizes that have utilizations greater than 95 percent are considered.
`-max_height` *max_height*	Specifies the upper bound of the core height, in microns, required to stop the die search process. It stops making a routable core larger if the total utilization for the core is greater than the target value you specify. By default, no die sizes that have utilizations greater than 30 percent are considered.
`-min_width` *min_width*	Specifies the lower bound of the core width, in microns, required to stop the die search process. It stops making a routable core smaller if the total utilization for the core is less than the target value you specify. By default, no die sizes that have utilizations greater than 95 percent are considered.
`-max_width` *max_width*	Specifies the upper bound of the core width, in microns, required to stop the die search process. It stops making a routable core larger if the total utilization for the core is greater than the target value you specify. By default, no die sizes that have utilizations greater than 30 percent are considered.

Table 3-1 MinChip Technology Search Control Options (Continued)

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

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

Table 3-2 MinChip Technology Floorplan Control Options

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

Table 3-2 MinChip Technology Floorplan Control Options (Continued)

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

Table 3-2 MinChip Technology Floorplan Control Options (Continued)

Floorplan control options	Description
`-core_sizing_only`	Applies only to full-chip designs. It modifies the size of the core area and ignores the I/O pad ring in the input floorplan. In designs with complex custom I/O structures, the `estimate_fp_area` command might not modify the I/O structure appropriately. This option can be used to ignore the I/O structure and modify the size of the core only. Note that if the core size is increased, it will overlap the I/O structure. By default, the block or chip shrinks with the core and the I/O pad or pin placement is scaled.
`-estimate_optimization`	Estimates the number of buffers added during optimization and reserves the area needed for buffer insertion. Therefore, the final die size after running the `estimate_fp_area` command will include the area needed for the current netlist plus some additional area reserved for optimization. MinChip technology will output the amount of area reserved for buffer insertion.
`-shrink_only`	By default, when the input floorplan is unroutable, the `estimate_fp_area` command expands the die to find the minimal routable size. If you specify the `-shrink_only` option, only the solution space of smaller dies sizes is considered. Therefore, the command exits if it determines that the floorplan is unroutable.

Table 3-3 describes the MinChip Technology power network control options.

Table 3-3 MinChip Technology Power Network Control Options

Power network control options	Description
`-power_net_names` *`list_of_names`*	Performs power network synthesis during each new die size exploration in `estimate_fp_area`. The power plan routes provide a more realistic routing overflow estimate because the routing resources used by the power preroutes are taken into account during the global route cell overflow calculations. You must specify a power and ground net name pair. A comma must be used to separate the names of the power net and ground net. For example, `-power_net_names VDD, VSS`
`-run_pns_script` *`script_name`*	Causes the `estimate_fp_area` command to run a user- defined power network synthesis script instead of the default settings to run power network synthesis. By default, power network synthesis is run unless you use either the `-run_pns_script` or the `-power_nets` option. The script should not contain any absolute coordinates as these absolute coordinates do not apply to the various design sizes that the `estimate_fp_area` command iterates through.
`-run_preroute_script` *`preroute_script_name`*	Causes the `estimate_fp_area` command to run a user-defined power routing script to form preroutes to connect the power and ground pins of standard cells. During this process, stacked vias are formed to connect the preroutes to the upper layer power routes, if applicable. The script should not contain any absolute coordinates as these absolute coordinates do not apply to the various design sizes that the `estimate_fp_area` command iterates through.

Table 3-3 MinChip Technology Power Network Control Options (Continued)

Power network control options	Description
`-maintain_power_structure`	Creates a power network synthesis script file that is automatically sourced within MinChip technology to replicate the power structures in the input floorplan for a new die size. The resultant die should have a power plan that is very similar to that of the input floorplan but scaled to the new die size.
`-save_as` *`cell_name`*	Use this option to name the saved CEL explicitly. By default, the `estimate_fp_area` command saves the last routable result as a CEL named *design_name* EstimateArea.

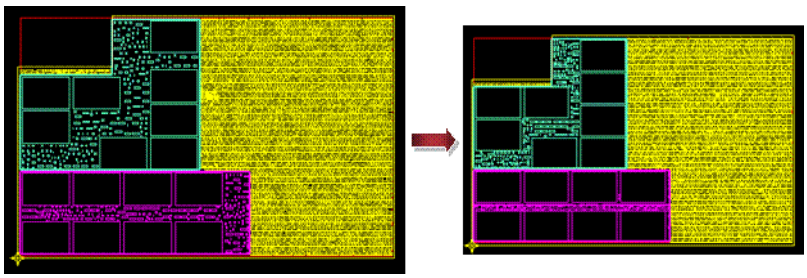
Using Multivoltage Designs in MinChip Technology

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

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

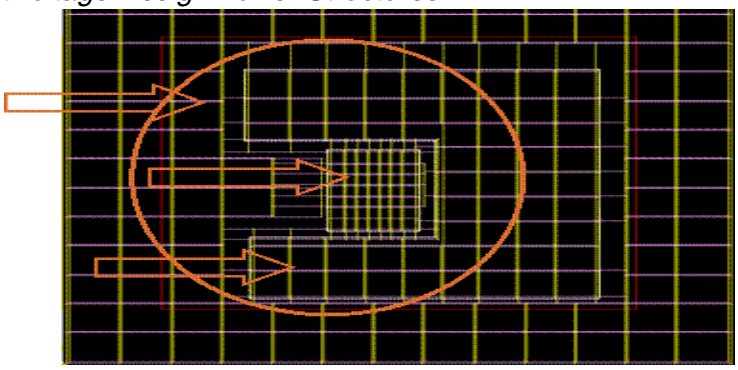
If a voltage area cannot be placed correctly because the utilization bounds are too high, the `estimate_fp_area` command stops shrinking the design. It also ensures that the aspect ratio of the voltage area changes in accordance with any changes to the aspect ratio of the top-level block or cell.

Figure 3-14 Voltage Area Sizing



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

Figure 3-15 Multivoltage Design Power Structures



4

Handling Black Boxes

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

Milkyway defines a black box as any instance in the netlist that does not have either a corresponding leaf cell in a reference library or a netlist module defined in terms of a leaf cell. Therefore, a black box can have an empty module represented in the netlist where its ports and their directions are defined. If the black box instance in the netlist does not have a corresponding module, the ports of the black box are defined in the Milkyway database with inout directions.

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

This chapter includes the following sections:

Reading Netlists With Black Boxes

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

```
icc_shell> read_verilog verilog_file
```

When IC Compiler reads the Verilog netlist, it detects the following types of logical block box modules:

- Missing

  A missing module is instantiated, but not defined, in the Verilog netlist. This module type might be an intended black box or it might indicate a missing reference library link.

  Note:
  By default, missing modules are not loaded by the `read_verilog` command. To load missing modules, you must use the `-dirty_netlist` option when you run the `read_verilog` command.

  If one or more reference libraries are not linked to the top design library, thousands of these missing black box modules might be detected. If you were not expecting to see any missing modules or were expecting to see only few, check the missing module names. If you discover that these missing modules are really leaf cell references, add the missing reference library link before continuing.

- Empty

  An empty module is defined and instantiated in the Verilog netlist. The module definition usually, but not always, has a port list and designates the input and output ports. This is the module type that you are most likely to want to keep as a black box.

  Here are two examples of an empty module:

```
module tuner (a,b,z);
 input a,b;
 output z;
endmodule

module tuner ();
endmodule
```

- Tie-off

  A tie-off module is defined and instantiated in the Verilog netlist. The module definition has assign statements with only output ports and those output ports are connected to either 1’b0 or 1’b1. Tie-off black boxes might be unexpected black boxes that were created during synthesis.

Here is an example of a tie-off module:

```
module ip_bus_tieoff (Z);
 output[7:0]Z;

 assign Z=8'b0;
endmodule
```

- Feedthrough

  A feedthrough module is defined and instantiated in the Verilog netlist. The module definition has assign statements in which all the ports of the module are found on either the left or right side of an assign statement in the module. Feedthrough black boxes might be unexpected black boxes that were created during synthesis.

  Here is an example of a feedthrough module:

```
module bus_function (In,Out);
  input [7:0]In;
  output [7:0]Out;

  assign Out=In;
endmodule
```

- Data flow

  A data flow module is defined and instantiated in the Verilog netlist. The module definition has assign statements for some, but not all, ports of the module. Data flow black boxes might be unexpected black boxes that were created during synthesis.

  Here is an example of a data flow module:

```
module ip_bus_tieoff(Z);
 output [7:0] Z;
 output X;

 assign Z = 8'b0;
endmodule
```

When the `read_verilog` command detects a logical black box module, it sets the `is_logical_black_box` attribute to true and sets the `black_box_type` attribute to one of the following values:

- `Missing` for missing modules
- `Empty` for empty modules
- `Feedthru` for feedthrough modules
- `Tie-Off` for tie-off modules
- `DF` for data flow modules

You can use these attributes with the `get_cells` command to identify the logical black boxes in your design. For example, to get all of the missing modules in your design, enter the following command:

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

Preparing Black Boxes for Design Planning

When you read the Verilog netlist, IC Compiler identifies the logical black box modules. To use black box modules as physical cells in the design planning flow, you must first complete the following tasks:

- Convert the black box from a logical black box to a physical black box
- Estimate the size of the black box
- Set routing controls for the black box

You should complete these tasks only for those black box cells that you expect in the design. If you see unexpected black box cells, determine the cause of these unexpected black box cells before continuing.

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

Converting Logical Black Boxes to Physical Black Boxes

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

Note:

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

To convert the black boxes, use the `import_fp_black_boxes` command (or choose Floorplan > Blackboxes > Import Blackboxes in the GUI). You must specify the list of black boxes to convert. On the command line, you can use the `get_cells` command with the `is_logical_black_box` and `black_box_type` to identify the black boxes. In the GUI, you

select the black box instances from the “Importable black boxes” list in the dialog box. To create a CEL view for every black box listed in the “Importable black boxes” list, click Select All.

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

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

The `import_fp_black_boxes` command sets the following attributes on the converted black box cells:

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

By default, a maximum of 100 black boxes can be imported at one time. If you try to import more than the maximum number of black boxes, the import is canceled. This is to prevent you from accidentally selecting all black boxes in a script when you might have forgotten to specify one or more reference libraries, which, in turn, can result in importing hundreds of black boxes. If your design contains more than 100 black boxes that should be imported, you can increase the maximum value by using the `-max_count` option (or the “Maximum number of black boxes to import” field in the GUI).

The `import_fp_black_boxes` command also estimates the size of each black box. By default, the tool estimates that each black box has a square boundary of 300 microns by 300 microns. You can control the size estimate of the black boxes by using one of the following methods:

- Specifying the side length

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

- Specifying the equivalent gate count and utilization

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

  Note:

  By default, IC Compiler uses the standard gate area of a 2-input-nand gate as the gate equivalent size. You can define the gate equivalent size by using the `set_fp_base_gate` command. For information about the `set_fp_base_gate` command, see “Determining the Gate Equivalent Area” on page 4-7.

Estimating the Size of Black Boxes

When you convert a logical black box to a physical black box, the tool estimates a size for the black box. You can refine the size of the black box cells to get a more accurate boundary to use for design planning by using the `estimate_fp_black_boxes` command (or by choosing Floorplan > Blackboxes > Estimate Blackboxes in the GUI). You can either explicitly specify the size or you can have IC Compiler estimate the size based on the cell's content. You can also specify that the shape is fixed, and have IC Compiler treat the black box as a hard macro rather than a soft macro.

A black box that is estimated as a soft macro will be a CEL view in the layout window and will have a thick boundary outline, just like any other soft macro. A black box that is estimated as a hard macro will be a FRAM view in the layout window. It will have a thin boundary outline, just like any other hard macro.

Specifying the Black Box Size

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

Specifying a Rectangular Black Box Boundary

To specify a rectangular black box boundary on the command line, specify the width and height of the rectangle in microns by using the `-sm_size {`*`width height`*`}` option. In the GUI, select "Soft macro size" and enter the values in the Width and Height fields.

By default, IC Compiler uses a utilization of 1.0 when you specify a rectangular black box boundary. You can change the utilization by using the `-sm_util` option.

Specifying a Rectilinear Black Box Boundary

To specify a rectilinear black box boundary on the command line, specify the coordinates of the polygon by using the `-polygon` option. In the GUI, select "Soft macro polygon" and enter the coordinates in the associated field. Whether you are using the command line or the GUI, you specify the polygon in the following format:

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

The points are interpreted as absolute coordinates in microns.

Sizing Black Boxes by Content

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

Determining the Hard Macro Area

To specify the hard macros contained in the black box cell, use the `-hard_macros` option or select “Estimate by enclosed hard macros” in the GUI. IC Compiler uses the area of the specified hard macros, but not the shape, when estimating the black box size. Whether you are using the command line or the GUI, you specify the hard macros in a comma-separated list, where each hard macro is specified in the following format:

name(*count*)

If there is only one instance of a given hard macro in the black box, you do not need to specify the count.

For example, to specify that the black box contains one instance of the ALU hard macro and 3 instances of the MUX hard macro, enter

```
-hard_macros "ALU, MUX(3)"
```

Determining the Gate Equivalent Area

IC Compiler computes the gate equivalent area by multiplying the gate equivalent size by the number of gate equivalents in the black box, and then dividing this value by the utilization. You specify these components in the following manner:

- Gate equivalent size

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

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

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

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

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

- Gate equivalent count

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

- Utilization

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

Fixing the Black Box Shape

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

Setting Routing Controls for Black Boxes

By default, no routing is allowed over black boxes to reserve the routing resource for implementation of the block at a later time. You can modify this behavior by setting the `min_layer` and `max_layer` attributes on the black box cell.

For example, to allow routing over a black box on all layers above metal6, enter the following commands:

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

Note:

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

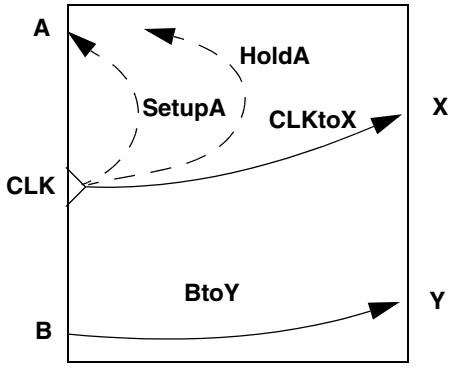
Preparing Black Boxes for Timing

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

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

Figure 4-1 shows a quick timing model representation of a black box. The constraint arcs appear as dashed lines and the delay arcs appear as solid lines. Port CLK is a clock port, ports A and B are input ports, and ports X and Y are output ports.

Figure 4-1 Quick Timing Model Representation of a Black Box



The arcs in this model are

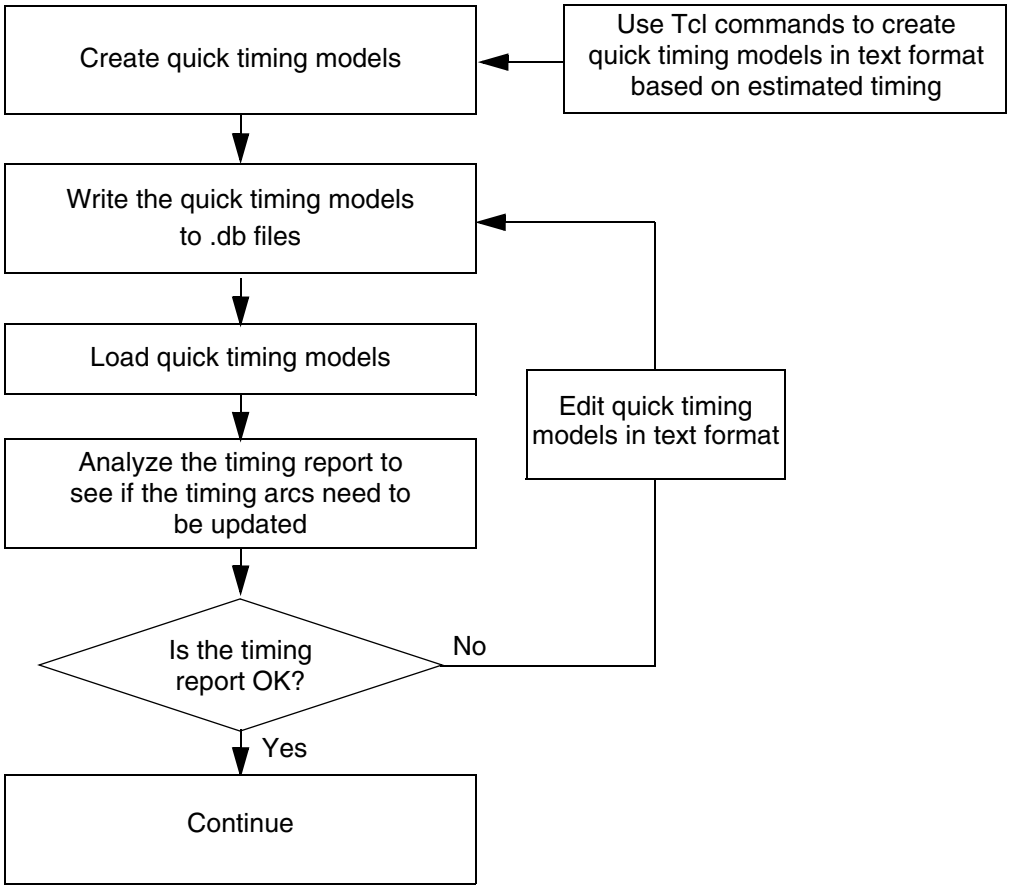
- SetupA, which constrains port A. The constraining port is clock CLK.
- HoldA, which constrains port A. The constraining port is clock CLK.
- CLKtoX, which is a sequential delay arc from CLK to X.
- BtoY, which is a combinational delay arc from B to Y.

Note:

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

Figure 4-2 shows the flow used to create a quick timing model for a black box. You create the quick timing model by specifying the model ports, the setup and hold constraints on the inputs, the clock-to-output path delays, and the input-to-output path delays. You can also specify the loads on input ports and the drive strength of output ports. After creating the quick timing model, you test it and refine it until the timing report meets your requirements. You can continue to update the black box timing arcs by performing “what-if timing budgeting” until the full top-level timing analysis results are clean.

Figure 4-2 Quick Timing Model Flow



Creating a Quick Timing Model

To create a quick timing model for a black box,

1. Specify that a new model is being created.

```
icc_shell> create_qtm_model qtm_bb
```

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

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

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

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

4. Define the ports.

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

5. Define the timing information for each port.

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

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

```
icc_shell> report_qtm_model
```

7. Save the quick timing model.

```
icc_shell> save_qtm_model
```

For more information about creating quick timing models, see SolvNet article 022081.

Writing a Quick Timing Model to a .db File

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

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

Loading a Quick Timing Model

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

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

Managing the Attributes for a Black Box

Only the `estimate_fp_black_boxes` command automatically changes the `unestimated` attribute for a black box. If a black box is user-sized by manual editing, the `estimated` attribute does not change. You can manage this attribute to monitor which black boxes have been sized by using the following two commands:

- The `set_fp_black_boxes_estimated` command specifies that the black box object types have been estimated and are therefore valid for floorplanning.
- The `set_fp_black_boxes_unestimated` command sets the specified black box object types as unestimated, so that they are flagged and sized correctly before floorplanning.

If you run the following command on the current black boxes, the value returned will be “unestimated” because all the current black boxes have not been user-estimated yet.

```
icc_shell> get_attribute BB_instance_name sm_estimation_mode
```

Converting Physical Black Boxes to Logical Black Boxes

When you create a CEL view for a black box module, the layout window contains an associated instantiation of that physical black box. If you later decide that the black box instantiation is not needed in the layout, use the `flatten_fp_black_boxes` command to remove the physical box instance from the top-level design in the layout window and restore it to the hierarchy preservation (logical view). The CEL view for the black box is retained in the library, but it does not have any effect on the top-level design where it was flattened. After running the `flatten_fp_black_boxes` command, the logical black box continues to exist as a module in the Verilog netlist output.

5

Using the On-Demand-Loading Flow for Large Designs

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

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

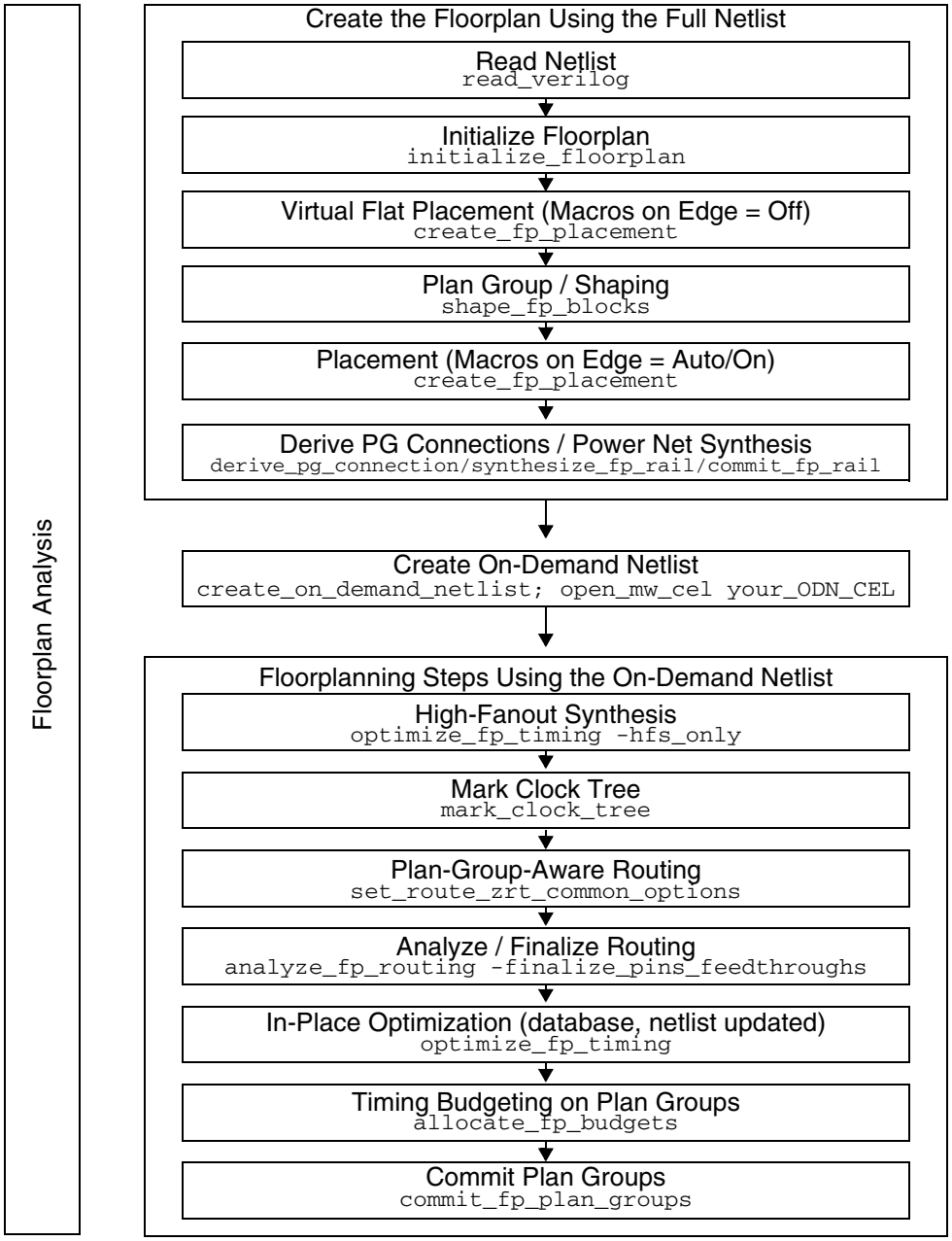
Overview of On-Demand-Loading Design Planning Flow

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

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

Figure 5-1 shows a typical on-demand-loading flow, using the design planning virtual flat commands.

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



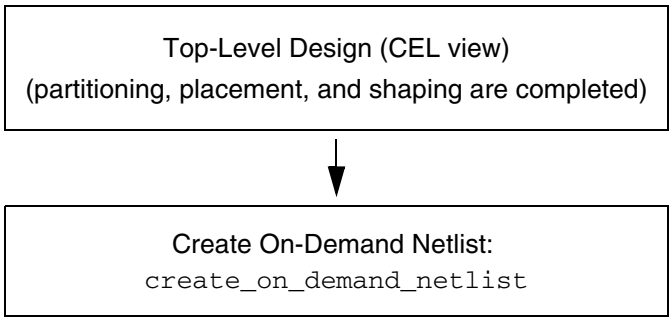
Creating On-Demand Netlists

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

- Cells, pins, and nets on the timing paths from the input ports to the output ports. These are the combinational input-to-output paths.
- Cells, pins, and nets on the timing paths that lead from the input ports to the edge-triggered registers. These are the interface logic paths.
- Cells, pins, and nets on the timing paths that lead from the edge-triggered registers to the output ports. These are the interface logic paths.
- All macro cells residing in the plan groups.

Use the `create_on_demand_netlist` command to generate the on-demand netlist. Figure 5-2 shows the basic steps for creating an on-demand netlist in IC Compiler.

Figure 5-2 Basic Steps for Creating an On-Demand Netlist in IC Compiler

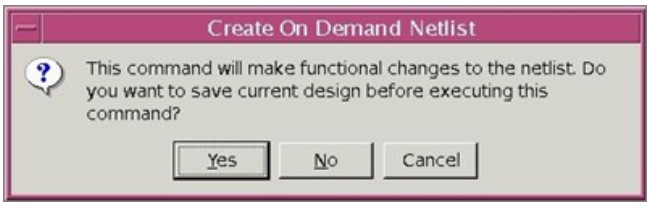


To create an on-demand netlist, follow these steps:

1. Read in the top-level design from the Milkyway design library (CEL view). The design should have already gone through partitioning, placement, and shaping. The netlist should have the plan groups placed inside the core area along with the macros.

2. Create the on-demand netlist by using the `create_on_demand_netlist` command with the appropriate options. Alternately, in the GUI, choose Partition > Create On Demand Loading Netlist. The following pop-up dialog box appears asking if you want to save the current design before creating the on-demand netlist.

Create On Demand Netlist Pop-Up Dialog Box



When creating an on-demand netlist, IC Compiler:

- Converts the current Milkyway design to a new Milkyway design that contains a reduced netlist. In this step, the tool replaces the full netlists of the plan groups with the corresponding interface netlists. The new Milkway design contains the complete top level, plan groups interface netlists, and all the macros.
- Saves all the floorplan information, such as the plan group shapes, locations, and placement information, in the internal database.
- Saves all the constraints residing in the core of the plan groups in the internal database.
- Marks all the reduced plan groups with the `is_on_demand_netlist` attribute.
- Closes the design after the on-demand netlist is created. Before you can use this on-demand netlist for the rest of the floorplanning flow, you need to open the CEL view using the `open_mw_cel your_ODN_CEL` command.

When you open an on-demand-loading design in the GUI, you can find the `is_on_demand_netlist` attribute setting on the plan groups from the InfoTip. The GUI display also tells you that the current design uses an on-demand netlist.

Summary of the On-Demand Netlist Creation Command Options

Table 5-1 summarizes the command options supported by the `create_on_demand_netlist` command.

Table 5-1 On-Demand Netlist Creation Command Options

To do this	Use this
Specify the plan groups to be reduced to their corresponding interface netlists. This is a required option.	`-plan_groups {plan_group_collection}`

Table 5-1 On-Demand Netlist Creation Command Options (Continued)

To do this	Use this
Specify the name of the on-demand netlist to be created.	`-on_demand_cell cell_name`
Specify the name of the top-level SDC file. This file should contain constraints for the entire design.	`-full_sdc_file sdc_file_name`
Save a copy of the full netlist of the current design before creating the on-demand netlist.	`-save_current_design`
Specify the name of the directory to store the temporary files during the process of creating the on-demand netlist. The default directory name is `ODL_work`.	`-work_dir dir_name`
Specify the host name to enable parallel runs with this command. Use the `set_host_options` command to control the parallel runs. To use this option, you must have the LSF configuration set up.	`-host_options host_name`
Create the interface netlist for the plan groups only. This option is mutually exclusive with the `-save_block_netlist_only` and `-save_internal_constraints_only` options. You must also specify the `-on_demand_cell` option.	`-netlist_reduction_only`
Save the full netlist of the plan groups only. This option is mutually exclusive with the `-netlist_reduction_only` and the `-save_internal_constraints_only` options.	`-save_block_netlist_only`
Save the internal constraints of the full netlists of the plan groups. This option is mutually exclusive with the `-netlist_reduction_only`, `-save_block_netlist_only`, and `-on_demand_cell` options.	`-save_internal_ constraints_only`

Improving the Efficiency of the On-Demand-Loading Flow

The on-demand-loading flow reduces the runtime of the design planning flow. You can further improve the efficiency of this flow using the parallel creation capability provided by the `create_on_demand_netlist -host_options` command. This command speeds up the creation of the reduced plan groups by creating them in parallel. By default, the reduced plan groups are created sequentially.

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

- Load Sharing Facility (LSF)
- Sun Grid Engine (SGE)
- rsh

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

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

The next example uses the `set_host_options` command to enable eight distributed processes using LSF, and then uses the `create_on_demand_netlist` command to create the on-demand netlist.

```
icc_shell> set_host_options -name lsf8 -pool lsf -num_processes 8
icc_shell> create_on_demand_netlist -on_demand_cell ORCA_ODL \
 -plan_groups [get_plan_groups *] \
 -work_dir ODL_Work -host_options lsf8
```

Note:

Distributed processing requires one IC Compiler license for every four parallel tasks. For example, to run eight parallel tasks, you must have two IC Compiler licenses. In the previous LSF example, if only one IC Compiler license is available and there are more than four plan groups, the tool begins processing on four plan groups by using parallel processing. When processing completes for one plan group, the tool submits the fifth plan group to the LSF queue. The process continues until the tool processes all plan groups.

In addition, you can speed up the creation of the reduced plan groups if you do not want to commit the design and perform block implementation at this point in the flow. This is useful early in the design planning process if you do not plan to perform block implementation. In this case, you can specify the `-netlist_reduction_only` option with the `create_on_demand_netlist` command as shown in the following example.

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

Note that you can create the block netlist with internal constraints at any time after performing the preceding step. This allows you to commit the design and proceed with block implementation. To do this, use the following `create_on_demand_netlist` commands.

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

Features Not Supported by the create_on_demand_netlist Command

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

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

Removing On-Demand Netlists

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

Reporting and Querying Information About On-Demand Netlists

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

- List the plan groups in the current design and report which plan groups have the interface netlists, created by the `create_on_demand_netlist` command.
- Report which plan groups have the netlists saved during the on-demand netlist creation.
- Report which plan groups have the internal SDC constraints saved during the on-demand netlist creation.
- Report the standard cell and macro cell count for the original plan group
- Report the standard cell and macro cell count for the reduced, On-Demand plan group
- Report the compression for the reduced plan group
- Report the area utilization for the original plan group
- Report other statistics such as top-level cell count, top-level plus On-Demand plan group cell count, top-level plus original plan groups cell count, and overall compression using the On-Demand Netlist

Example 5-1 shows a sample of the information reported by the `report_on_demand_netlist` command.

Example 5-1 Information Reported by report_on_demand_netlist

```
****************************************
Report     : report_on_demand_netlist
Design     : DesignA_ODL
Version    : D-2010.03-ICC-SP4
Date       : Fri Aug 20 04:41:37 2010
****************************************
Design DesignA_ODL is an On-Demand Netlist.

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


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

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

Timing Budgeting Using On-Demand Netlists

After you create the on-demand netlist, you can continue with the virtual flat design flow using the reduced netlist as shown in Figure 5-1. During the budgeting step, the `allocate_fp_budgets` command automatically checks if the plan groups in the current design have the `is_on_demand_netlist` attributes. If a plan group is identified as a reduced netlist with only the interface logic, the budgeter first searches for its corresponding full netlist and the internal constraints saved by the `create_on_demand_netlist` command. It then combines the constraints from the interface netlist and the internal constraints from the full netlist to create the budget for the plan group. This complete set of budgeting constraints is then passed to the next step for committing plan groups.

Committing Plan Groups Within the On-Demand-Loading Flow

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

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

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

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

6

Performing an Initial Virtual Flat Placement

The initial virtual flat placement is very fast and is optimized for wire length, congestion, and timing.

This chapter includes the following sections:

Evaluating Initial Hard Macro Placement

No straightforward criteria exist for evaluating the initial hard macro placement. Measuring the quality of results (QoR) of the hard macro placement can be very subjective and often depends on practical design experience. You should observe some basic rules when measuring QoR, such as pushing hard macros to the core boundary and aligning similar hard macros. However, the critical measurements that are used to evaluate the placement results are timing and routability.

QoR can be measured by the following criteria:

- Routability

  You can measure the routability of your design by analyzing the routing congestion produced by the global router. The data used to analyze congestion consists of a textual report and visual heat maps that show where the routing congestion hot spots exist in the design. Highly congested designs are generally not routable. Hard macros tend to create congestion around their edges and corners. You should analyze hard macro placements for routability early in the design cycle.

- Timing

  Timing calculations can use the placement information to better estimate interconnect loading. The interconnect timing calculations should take into account detours in routes caused by the presence of hard macros. Good placement will minimize timing violations.

- Wire length

  Smaller values of total wire length are a good indicator of placement quality. The total wire length is output in the placement log file. You should record and monitor this number when assessing multiple placement solutions.

  Another aspect of wire length is localized to hard macros. By looking at the signal (flyline) connections of a hard macro, you can quickly determine whether the hard macro is placed in an optimal location. A hard macro placed in the lower-left corner of the core area that is connected to logic in the upper-right corner of the die indicates a poor location for the hard macro.

- Data flow

  If you know the logic and intended operation of the design, you can assess the data flow by using the hierarchical browser at any stage in the design flow to observe where logical modules and hard macros are physically placed. Using this technique can help you make sensible placement decisions.

- Standard cell placement areas

  You can visually assess the placement of macros and standard cells. Small areas surrounded by hard macros usually cause congestion hot spots. Unless the connections to and from standard cells in these areas are completely localized, it is difficult to complete the connections from within these areas to the objects outside these areas. Generally, a contiguous standard cell placement area without bottlenecks is desirable.

After passing these QoR measurements, the placement result qualifies as a good starting point for further manual tuning.

Specifying Hard Macro Placement Constraints

The following sections describe the different methods you can use to control the preplacement of hard macros and improve the QoR of the hard macro placement.

- Creating a User-Defined Array of Hard Macros
- Setting Floorplan Placement Constraints On Macro Cells
- Placing a Macro Cell Relative to an Anchor Object
- Using a Virtual Flat Placement Strategy
- Creating Macro Blockages for Hard Macros
- Padding the Hard Macros

Creating a User-Defined Array of Hard Macros

You can create a user-defined array of hard macro cells that constrain hard macros to form a one-dimensional or two-dimensional array. The macro cells are placed in the array according to the constraints specified for the array.

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

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

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

Figure 6-1 shows one-dimensional and two-dimensional macro arrays.

Figure 6-1 One-Dimensional and Two-Dimensional Macro Arrays

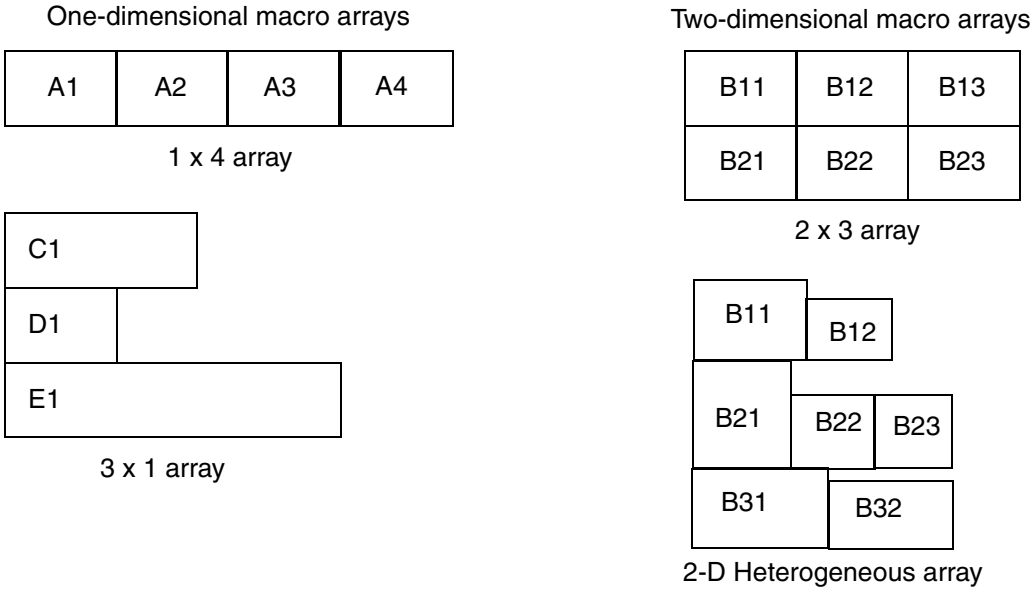
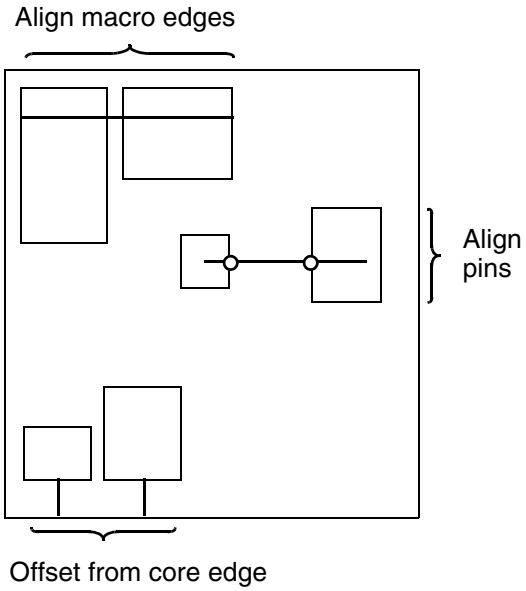


Figure 6-2 shows macro alignment to other macros, pins, and edges.

Figure 6-2 Alignment to Other Macros, Pins, and Edges



To create user-defined array of hard macro cells,

1. Choose Placement > Macro Constraints.

   The Macro Constraints dialog box appears.

   Alternatively, you can use the `set_fp_macro_array` command.

   Each constraint is displayed with a unique name, type, and list of macros or macro arrays.

   You can sort the list by name or type by clicking on the column header. Selected macro constraints are annotated graphically in the layout window. The original macro cells are also highlighted in the layout window.

2. To create a new macro array constraint, make sure the Arrays tab is selected, then click the Add button.

   The Add Macro Array dialog box appears.

3. Set the options, depending on your requirements.

   - Macro array name – Specify the macro array name. This name is required.
   - Macro cells – You can set the order for the macro cells that will compose the array by using lists and sublists. The macro array template shows how selected macro cells are arranged into an array.
   - Rectilinear and Vertical – Select the Rectilinear option to create a two-dimensional array with a rectilinear outline. You can also use the Vertical option with the Rectilinear option to create a vertical row structure for the two-dimensional array. By default, a rectilinear array has a horizontal row structure.

     Figure 6-3 shows an example of a rectilinear horizontal 2-D heterogeneous array.

Figure 6-3 Rectilinear Horizontal 2-D Heterogeneous Array

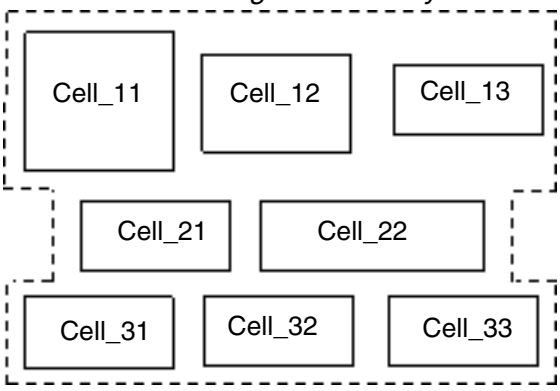
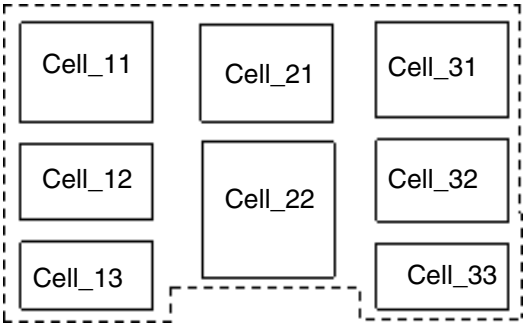


Figure 6-4 shows an example of a rectilinear vertical 2-D heterogeneous array.

Figure 6-4 Rectilinear Vertical 2-D Heterogeneous Array



- Use Keepout Margin – When arraying two adjacent macros, you can apply the Use Keepout Margin option to leave the smallest possible spacing between the rows or columns of the two macro arrays. The spacing is greater than or equal to the keepout margins on the two macros.

  This option is mutually exclusive with the X offset and Y offset options.

- Specified offset – You can specify the distance between two adjacent cells in a macro array in the x-direction, y-direction, or both directions by using the X offset and Y offset options.

  Figure 6-5 shows the result of applying an x-offset.

Figure 6-5 Result of Applying an X-Offset

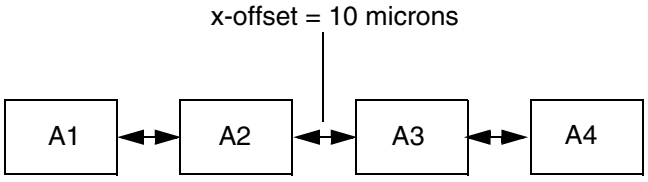
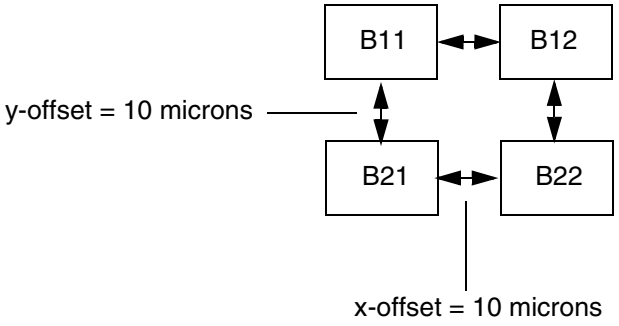


Figure 6-6 shows the result of applying both x- and y-offsets.

Figure 6-6 Result of Applying X- and Y-Offsets



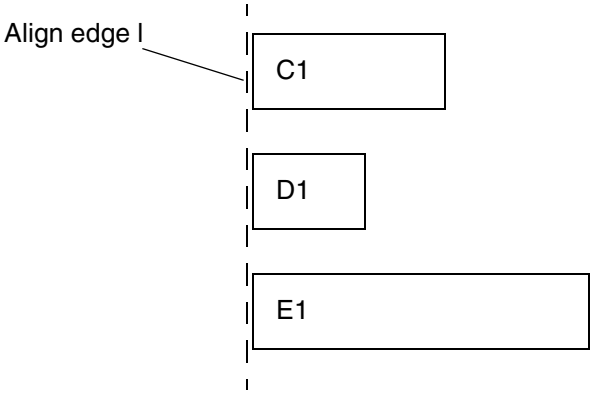
- Align edges – You can align one row of the macro cells of a heterogeneous one-dimensional macro array along an edge that you specify by selecting this option. The edge can be top, bottom, left, right, or center. The default is bottom.

  If you use this option for an array type other than a heterogeneous one-dimensional array, a warning appears and the option is ignored.

  If you specify a one-dimensional heterogeneous array and do not use this option, the default is to align the centers.

  Figure 6-7 shows the result of applying left alignment.

Figure 6-7 Result of Applying Left Alignment



- Align two macro cells with pins – You can select this option when you want to create two-dimensional macro arrays and align numerous small macro cells that have uneven shapes.

Figure 6-8 shows the result of applying the constraint to align a two-dimensional rectilinear heterogeneous macro array.

Figure 6-8 Aligning a Rectilinear 2-D Heterogeneous Array

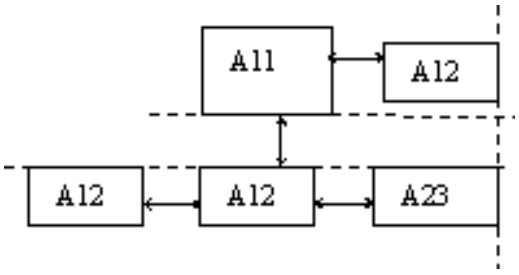
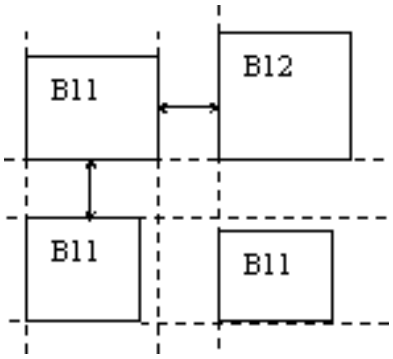


Figure 6-9 shows the result of applying the constraint to align a two-dimensional rectangular heterogeneous macro array.

Figure 6-9 Aligning a Rectangular 2-D Heterogeneous Array



- Pin of macro 1 and Pin of macro 2 – You can align cells in macro arrays having two macro cells by aligning a reference pin (a pin on the first cell in the array, which is the reference cell) and a pin on the other cell in the array.

  This option takes effect only if the array has two macro cells.

  Figure 6-10 shows the result of aligning cells in a macro array by using pins.

Figure 6-10 Result of Aligning Cells in Macro Arrays by Using Pins

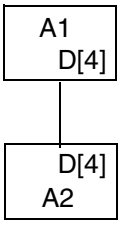
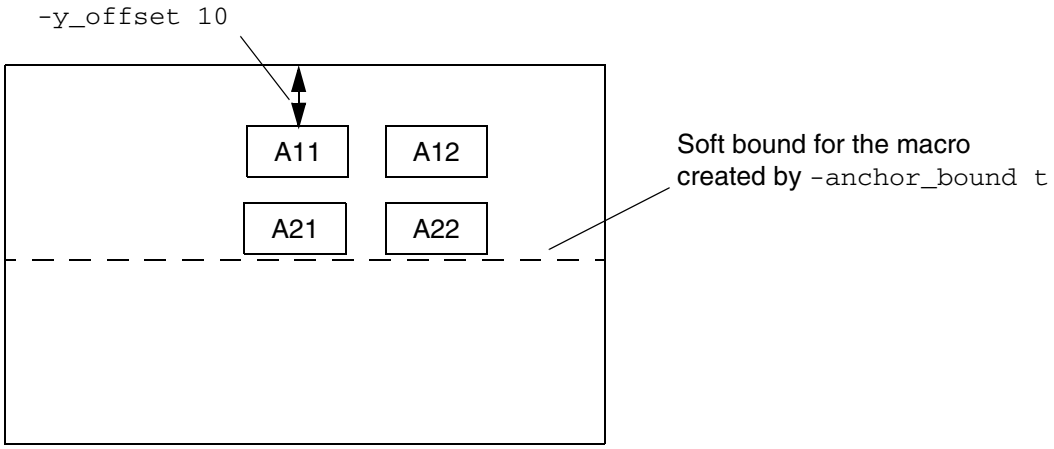


Figure 6-11 shows a two-dimensional array offset 10 microns from the top edge of the core area.

Figure 6-11 Two-Dimensional Array Offset 10 Microns From Core Top Edge



4. Click OK.

Setting Floorplan Placement Constraints On Macro Cells

During floorplanning, you can define placement constraints that restrict specified macro cells and macro arrays to particular regions, orientations, alignments, and so forth. This constrains the `create_fp_placement` command in placing the macro cells in the floorplan.

When you specify the constraints on macro cells and macro arrays, keep the following points in mind:

- The physical library and design must be loaded.
- Set these floorplan placement constraints before you run the `create_fp_placement` command.
- During placement, macro cells can be flipped or rotated according to legal orientation constraints from the library or by constraints you set.
- You must provide a list of macro cells to which the constraints apply. You can provide this list by using the `get_cells` command (for example, [`get_cells RAM`]). If you list a macro cell that is not found, an error occurs.
- To report the options that are set, use the `report_fp_macro_options` command. The report provides details about a particular macro cell or array or all the macro cells or arrays. Information includes the name of the macro cell or macro array, anchor bounds, offsets, and legal orientation for macro cells. (Legal orientation does not apply to macro arrays.)

To set floorplan placement constraints that restrict specified macro cells and macro arrays during virtual flat placement,

1. Choose Placement > Macro Constraints.

   The Macro Constraints dialog box appears.

   Alternatively, you can use the `set_fp_macro_options` command.

   Each constraint is displayed with a unique name, type, and list of macros or macro arrays.

2. To create a new macro options constraint, select the Options tab and then click the Add button.

   The Add Macro Options dialog box appears.

3. Set the options, depending on your requirements.

   - Macro objects – Enter a list of macro cells to be constrained.
   - Legal orientations – Select this option to restrict the orientation to a valid orientation that you specify. If you do not use this option, orientation is not restricted during placement. The default is enabled.

     You can provide a single orientation to force the placement engine to place the macro cell in a fixed orientation. After the cell orientation is fixed, the placement tool cannot change the orientation.

     The value can be one or more than one of the following orientations using the Design Exchange Format (DEF) syntax: N, W, S, E, FN, FW, FS, FE

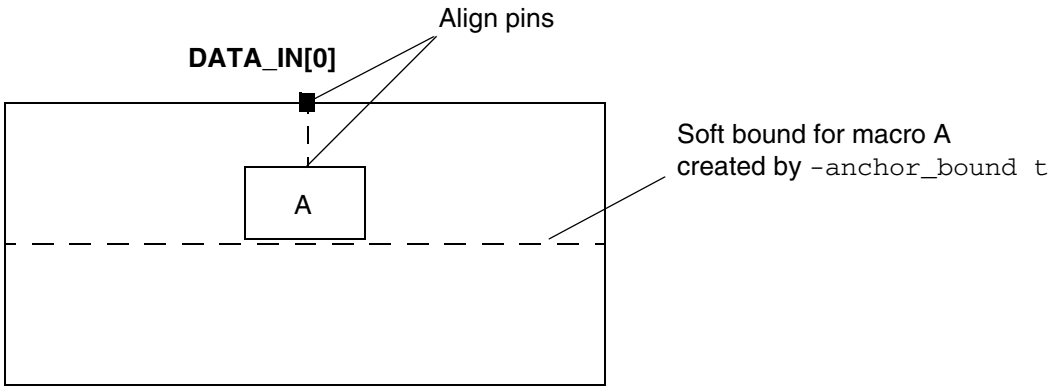
     The orientation you specify overrides the legal orientations defined in the physical library. If you do not specify this option, the macro cell's legal orientation is used in the placement.

   - Align Pins – You can align a specified port or pin to a pin of a constrained macro cell by using the this option. The argument is a list of two objects: reference port or pin followed by the constrained pin.

     Figure 6-12 shows the alignment between a port and a pin of a constrained macro cell.

     To constrain macro A so that port DATA_IN[0] of the block aligns with DATA[0] of macro A, the DATA_IN pins were previously grouped so that they align with the other pins of macro A.

Figure 6-12 Aligning a Port to a Pin of a Constrained Macro Cell



- Anchor bound – You can set an anchor bound for macro placement to constrain the macro placement to the boundary region you specify, forcing the macro cell to stay at one of the boundaries of the block. To do this, select the “Anchor bound” option, which sets a soft bound. Because the bound is soft, placement can move the macro cell out of the bound if there is no space and the cost is too high.

  You can set one of the following 13 designations used for 16 boundary regions, the area in which the macro is placed:

Keyword	Definition
br	bottom-right corner
tl	top-left corner
t	top
tr	top-right corner
r	right
b	bottom
bl	bottom-left corner
l	left
tm	top-middle
bm	bottom-middle
lm	left-middle

Keyword	Definition
rm	right-middle
c	center

Figure 6-13 shows boundary areas and values for the `-anchor_bound` option.

Figure 6-13 Boundary Areas and Values for the -anchor_bound Option

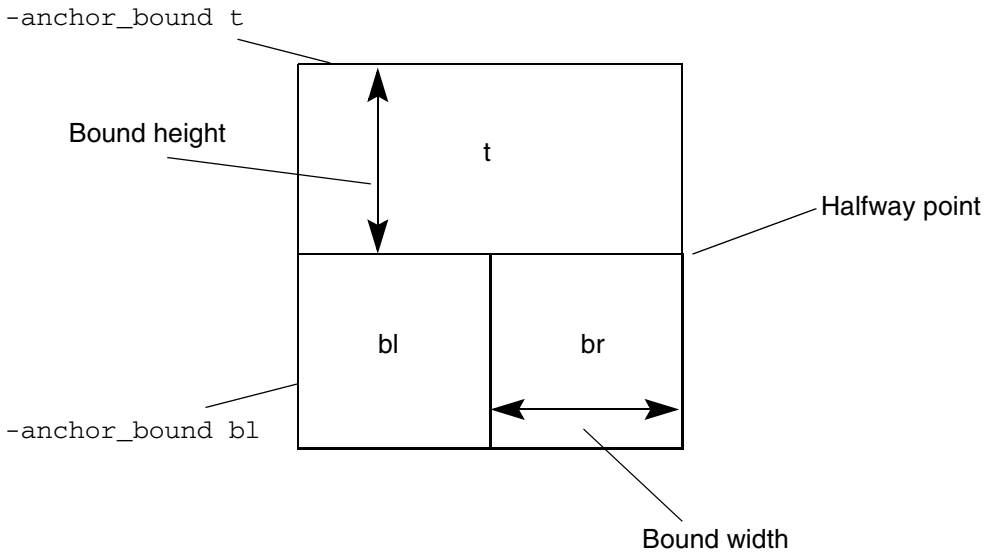
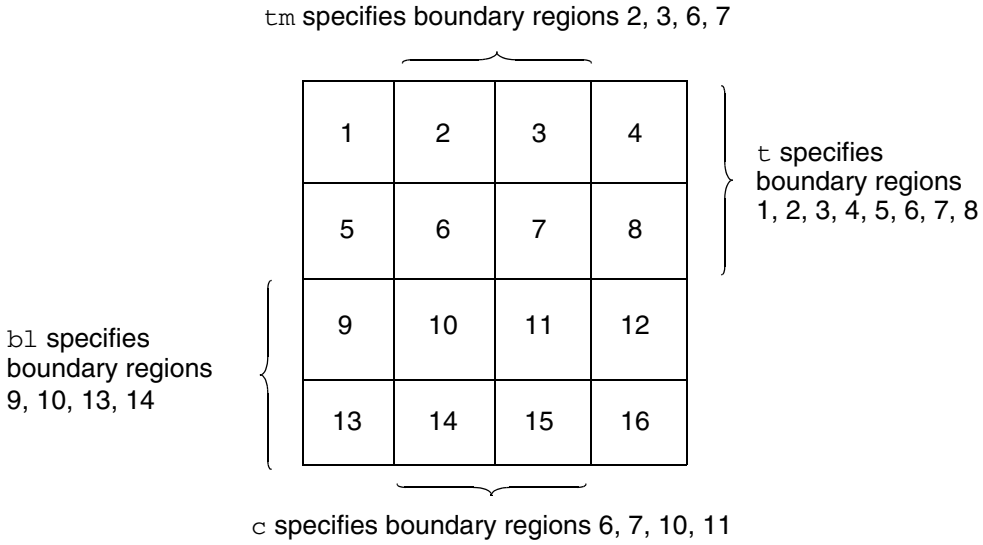


Figure 6-14 shows the key for determining how to set the value for each of the 16 boundary regions.

Figure 6-14 Key for Determining Boundary Region Values for the -anchor_bound Option



The default bound area is

1/4 of the core area when you use bl, tl, br, tr, tm, bm, lm, rm, or c

1/2 of the core area when you use t, b, l, or r

The entire core area if you do not specify an anchor bound

As shown in Figure 6-13 on page 6-12, specifying the anchor bound as t creates the move bound at the top-half of the core area, and specifying the anchor bound as bl creates a move bound at the quarter core area on the bottom-left area. If the half- or quarter-core area is not large enough to hold the macro cell in any dimension, the move bound is set to cell width or height in that dimension.

You can create two bounds with the same anchor point.

- X offset and Y offset – You can set a distance between the constrained macro and the core edges by selecting either the X offset or Y offset option or both options.

  You must use the “Anchor bound” option when you use these options. The available offset options depend on the type of anchor bound, as listed in Table 6-1.

Table 6-1 Anchor Bound and Offset Option

When the anchor bound is this	Use this offset
t, b, tm, or bm	Y offset
l, r, lm, or rm	X offset

Table 6-1 Anchor Bound and Offset Option (Continued)

When the anchor bound is this	Use this offset
tl, tr, bl, br, or c	X offset or Y offset or X offset and Y offset

- Side channels – You can create a side channel by using this option. Specify the desired channel size on the left, right, top, and bottom sides of the chip, respectively, in microns. The macros are then placed at least the specified distances away from the core edges.

  For example, to place a user-defined macro array at least 20 microns away from the left edge, 30 microns away from the right edge, and 30 microns away from the top edge of the core area, enter 20 in the “Left” text box, 30 in the “Right” text box, 30 in the “Top” text box, and 0 in the “Bottom” text box.

4. Click OK.

Placing a Macro Cell Relative to an Anchor Object

You can set a relative location constraint on a cell, which is referred to as the target cell with respect to another cell, which is referred to as the anchor cell, or on a corner of the core area or plan group. This fixes the cell in a certain location with respect to the core area.

The anchor in a relative location constraint must be fixed. For example, this could be a corner of the core area or plan group, a fixed cell, or a cell that is anchored by some other relative location constraint.

Note:
If the relative location constraint is called more than once on a cell, the last one overrides any relative location constraint previously set on the cell.

To place a macro cell relative to an anchor object,

1. Choose Placement > Macro Constraints.

   The Macro Constraints dialog box appears.

   Alternatively, you can use the `set_fp_relative_location` command.

   Each constraint is displayed with a unique name, type, and list of macros or macro arrays.

2. To create a new relative location macro constraint, select the Relative tab and then click the Add button.

   The Add Relative Location dialog box appears.

3. Set the options, depending on your requirements.
   - Constraint name – Enter a name of the relative location constraint.
   - Anchor object name – Enter the name of the anchor object. The object can be a plan group, a fixed macro cell, a macro cell that has its own relative location, or a core area.

     Specify the corner of the anchor object's bounding box that the constraint uses. You can choose from the following designations: bl (bottom-left corner), br (bottom-right corner), tl (top-left corner), and tr (top-right corner).
   - Target macro – Specify the name of the macro cell to which the constraint is applied.
   - Orientation – Specify the orientation in which to place the target cell. You can choose from the following values: N, S, E, W, FN, FS, FE, and FW. The default is N.
   - Corner – Specify the corner of the target cell on which to apply the constraint. You can choose from the following designations: bl (bottom-left corner), br (bottom-right corner), tl (top-left corner), and tr (top-right corner). The default is bl.
   - X offset – Specify the distance in microns from the target cell to the anchor corner of the anchor cell in the X-dimension. You can enter a positive, negative, or zero value. The default is 0.
   - Y offset – Specify the distance in microns from the target cell to the anchor corner of the anchor cell in the Y-dimension. You can enter a positive, negative, or zero value. The default is 0.
4. Click OK.

Reporting Relative Location Constraints

You can use the `report_fp_relative_location` command to report macro relative location constraints set by the `set_fp_relative_location` command.

Extracting Relative Location Constraints

From an existing placement, you can extract location constraints for a list of macro cells (target cells) relative to an anchor object by using the `extract_fp_relative_location` command. The extracted constraints can be output to a file.

The `extract_fp_relative_location` command uses the following syntax:

```
extract_fp_relative_location -target_cells cells
   [-target_corner bl | br | tl | tr] [-anchor_object object_name]
   [-anchor_corner bl | br | tl | tr] [-output_file file_name]
```

Table 6-2 describes how to use the relative location constraints.

Table 6-2 Relative Location Constraints

To do this	Use this option
Specify the names of the macro cells whose locations will be extracted in the format of relative location constraints. The default is all macro cells. (Required)	`-target_cells` *cells*
Specify the corner of the target cells on which to extract the relative location constraints. You can choose from the following designations: bl (bottom-left corner), br (bottom-right corner), tl (top-left corner), and tr (top-right corner). The default is bl. (Optional)	`-target_corner`
Specify the name of the anchor object. The anchor object can be a plan group, a fixed macro cell, or a core area. The default is core area. (Optional)	`-anchor_object` *object_name*
Specify the corner of the anchor object's bounding box that will be used in extracting the constraints. You can choose from the following designations: bl (bottom-left corner), br (bottom-right corner) tl (top-left corner), and tr (top-right corner). The default is bl. (Optional)	`-anchor_corner`
Specify the name of the file into which the extracted constraints are output. The default is output to a console log view.	`-output_file` *file_name*

Removing Relative Location Constraints

You can remove relative location constraints set on the specified target cells and any other cells that are anchored to these cells by using the `remove_fp_relative_location` command. By default, all relative location constraints are removed.

The `remove_fp_relative_location` command uses the following syntax:

```
remove_fp_relative_location [-name constraint name] [-target_cells cells]
```

Using a Virtual Flat Placement Strategy

This section describes the constraints you can use to control the virtual flat placement. These constraints are not persistent in the Milkyway database.

Use the `report_fp_placement_strategy` command to display floorplan-related option values set using the following commands.

They are grouped into four categories:

- Hard Macro
- Net Weighting
- Congestion
- Miscellaneous

Note:

You can set all the default values by using the `set_fp_placement_strategy -default` option.

Table 6-3 lists the hard macro control constraints.

Table 6-3 Hard Macro Control

Option	Description
`set_fp_placement_strategy -pin_routing_aware on | off`	Set this parameter to a value of on to provide pin aware information to virtual flat placement. Macro placement (`create_fp_placement`) takes this information into consideration and creates macro-only blockages along the edges of a plan group or a soft macro so that the macro cells do not block the edges of the plan groups or soft macros. This reserves enough space along the edges of the plan group or soft macro for pin assignment to place pins and for the router to route nets to the pins. The default is `off`.
`set_fp_placement_strategy [-macro_orientation automatic | all | N]`	Controls how macro orientations are determined. A value of `automatic` tells IC Compiler to analyze the macro's reference library cell and determine a good orientation for each reference cell, based on the reference cell's pins. This is the default. A value of `all` rotates macros to any legal orientation during placement. A value of `N` means all macros will be in one of the following orientations unless overridden by a reference library constraint or a user-set macro orientation constraint. 0 (0-degrees rotation) 180 (180-degrees rotation) 0-mirror (reflection in the y-axis) 180-mirror (180-degrees counterclockwise rotation, followed by a reflection in the y-axis)

Table 6-3 Hard Macro Control (Continued)

Option	Description
`set_fp_placement_strategy -macro_setup_only on | off`	When set to `on`, the placement engine places all macro arrays outside the core area for viewing purposes. You can then manually place the macro arrays inside the core area if desired. Macros with relative locations constraints are still placed within the core area. Standard cells are not placed. The default is `off`.
`set_fp_placement_strategy -auto_grouping none | user_only | low | high`	Controls how much automatic hard macro array packing will be done, based on design hierarchy information. • Wire length is reduced if a single net drives the same pin of more than one macro cell in an array. • The number of scattered hard macros is reduced, increasing the routing area. • The `-auto_grouping` option is hierarchy-aware, which means that macros in the same physical hierarchy will be arrayed (grouped) together. • User-created arrays are honored. Set the `-auto_grouping` option to `none` if you do not want to generate macro arrays. A value of `user_only` generates only user-defined macro arrays. A value of `low` (the default) generates macro arrays from small macros only, and a value of `high` generates macro arrays from all macros. (Note: You can also pack selected hard macros into a region. See "Packing Hard Macros Into an Area" on page 6-39.)

Table 6-3 Hard Macro Control (Continued)

Option	Description
`set_fp_placement_strategy -macros_on_edge on | off | auto`	Creates a placement with hard macros on the boundary edge of the chip or plan group edge. This option eliminates the fragment spaces that are created when hard macros are scattered, thereby creating a large open space in the core area for placement and routing. Set this option to `on` if a large percentage of the chip area contains hard macros. Moving hard macros to the edge reduces congestion and improves routability. For a hierarchical design, set this option to `off` until you have shaped the plan groups and placed them inside the core area of the chip. Set this option to `auto` to let the tool decide which macros to place along the edges. This option interacts with the hierarchy gravity option. If the hierarchy gravity option is off, the macros are placed on the edges of the chip. If the hierarchy gravity option is on, the macros are placed on the edges of the physical hierarchical boundaries to which they belong. For more information about hierarchical gravity see "Controlling the Placement" on page 6-36.

Table 6-3 Hard Macro Control (Continued)

Option	Description
`set_fp_placement_strategy -sliver_size` *distance*	Defines the minimum channel size that can be populated by standard cells. A sliver is a channel that is too small for placement of any standard cells. For example, if a `sliver_size` is 10, a channel of height 9 would not have any standard cells inside it. Slivers apply to channels created between any two of the following objects, provided that the object in question occupies at least 1/100 of the total chip area: fixed or movable hard macros, placement blockages, plan group edges, or core area edges. (Note that the keepout margin of hard macros is not included in the sliver size. Hard macro slivers are measured from the edge of the keepout margin, not from the edge of the macro.) During placement, blockages are dynamically created over narrow channels between macros as well as over channels between macros and chip edges. Use this option if there is congestion in narrow channels and if channels contain standard cells. If there are several channels containing standard cells that have a lot of congestion, set the `sliver_size` to the largest width of those channels. Legal values are any real numbers greater than or equal to 0.0 (the default).

Table 6-3 Hard Macro Control (Continued)

Option	Description
`set_fp_placement_strategy -fix_macros none | soft_macros_only | all`	Controls whether macros should be temporarily fixed in position during virtual flat placement. When set to `none` (the default), all macros are allowed to move during placement. When set to `soft_macros_only`, soft macros are fixed during virtual flat placement. The `create_fp_placement` command will not move them, regardless of their placement status in the Milkyway database. When set to `all`, all soft macros and hard macros are fixed during virtual flat placement. Use this option if your hard macros are in good positions in the design and you want to fix them during virtual flat placement. This parameter does not change the "fixed" attributes on macros.
`set_fp_placement_strategy -snap_macros_to_user_grid on | off`	A value of `on` snaps the lower-left corner of the hard macro's bounding box to the grid points defined by the `set_user_grid` command. After you set this value, during any future virtual flat placement, each hard macro is placed (snapped) with its lower-left corner on a grid point. The default is `off`.
`set_fp_placement_strategy -hierarchy_gravity_multi_level on | off {}`	A value of `on` places macros in close proximity based on macro connectivity at lower hierarchy levels as determined by the tool.
`set_fp_placement_strategy -hierarchy_gravity_blocks {`*`block_names`*`}`	If you specify this option, the tool skips automatic hierarchy detection and honors plan groups specified in the *block_names* list.

Table 6-3 Hard Macro Control (Continued)

Option	Description
`set_fp_placement_strategy -force_auto_detect_hierarchy on | off`	If your design contains existing plan groups either within the core area or outside the core area and this parameter is set to `on`, the placement engine analyzes the logical hierarchy nodes that are left at the top level of the design to determine if they are candidates for hierarchy gravity. If the tool detects additional logical nodes that are outside the preexisting plan groups and which contain top-level modules for grouping, it extracts those modules. The `create_fp_placement` command places the cell instances inside these modules together. If your design does not contain plan groups, this option has no effect. This option is based on the logical hierarchy only. Other placement constraints, such as power domains or relative placement groups, do not affect the results of auto hierarchy detection. The default is `off`. Figure 6-15 shows an example of the logical hierarchy of a design, in which there is a preexisting plan group consisting of modules top/A11 and top/A12. When the `-force_auto_detect_hierarchy` option is set to `on`, an additional logical hierarchy node, which is outside of the preexisting plan group and consists of modules top/A13, top/A13/B12, and top/A13/B13, is extracted. The `create_fp_placement` command places the cell instances inside these modules together.

Figure 6-15 Example of Auto Detect Hierarchy

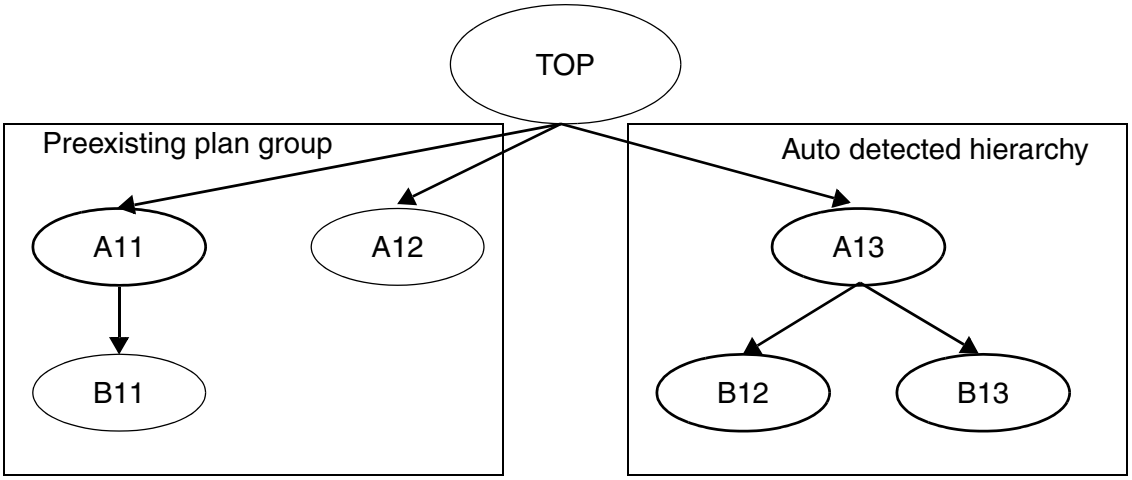


Table 6-4 lists the net weighting control constraints.

Table 6-4 Net Weighting Control

Option	Description
`set_fp_placement_strategy -IO_net_weight` *weight*	Controls the special weighting on I/O nets. This option is useful if the design requires a short distance between the I/Os and their connected standard cells or hard macro cells. The allowed range is from 0.0 through 10.0. The default is 1.0. Example: `IO_net_weight = 2.0` The nets connected to I/Os are weighted twice as high as the other nets, and therefore, are likely to be short. Example: `IO_net_weight = 0.0` The lengths of the nets are ignored. This is useful if no final I/O placement has been done and you want the placer to help determine the I/O locations.
`set_fp_placement_strategy -plan_group_interface_net_weight` *weight*	Specifies the net weight on interface nets that cross the edges of exclusive plan groups. This option is useful if the cells in your design need to be pulled closer to the plan group boundary when those nets cross the boundary. The allowed range is from 0.0 through 10.0. The default is 1.0.

Table 6-4 Net Weighting Control (Continued)

Option	Description
`set_fp_placement_strategy -honor_mv_cells on | off`	When set to `on`, attaches a constraint to level-shifter and isolation cells that brings them closer to the appropriate voltage area edge. After setting the placement strategy with the `set_fp_placement_strategy` command, you must use the `link` command for placement to recognize the `dont_touch` attribute on the cells. The default is `off`.

Table 6-5 lists the congestion control placement constraints.

Table 6-5 Congestion Control

Option	Description
`set_fp_placement_strategy -congestion_effort low | medium | high`	Controls the effort level of congestion-driven placement. When set to `high`, the tool uses the Zroute global router and the routing congestion map. If the effort level is set to `medium`, the tool uses the Zroute global router with a minimum effort setting and the routing congestion map. if the effort is set to `low`, or this option is not specified, the tool uses the placement congestion map. The default is `low`.
`set_fp_placement_strategy -adjust_shapes on | off`	If you set this option to on and your design has plan groups inside the core area, when you run the `create_fp_placement -incremental congestion_driven` command, the shapes of the plan groups are automatically adjusted to alleviate channel congestion.

Table 6-6 lists the option for multilevel constraint control. The format of the block constraint file is

```
fplModuleDestRegion plan_group_1 direction_1 [order_number]
fplModuleDestRegion plan_group_2 direction_2 [order_number]
...
```

where *plan_group_n* is the name of the module, *direction_n* is one of {N S E W NE NW SE SW}, and the optional *order_number* indicates the relative order of submodules along the specified edge. The *order_number* applies only when there is more than one submodule for the specified edge of the module. Note that the block constraints are obeyed only if the `-macros_on_edge` option is set to `on`.

Table 6-6 Multilevel Constraint Control

Option	Description
`set_fp_placement_strategy -block_constraint_file` *file_name*	Controls the reading of a file specifying positional constraints for lower-level modules within the hierarchy.

Table 6-7 lists miscellaneous placement constraints.

Table 6-7 Miscellaneous Control

Option	Description
`set_fp_placement_strategy -legalizer_effort low | high`	Controls the effort level of the legalizer. Set this option to `low` if the legalizer is running too long. The default is `high`.
`set_fp_placement_strategy -spread_spare_cells on | off`	Spreads spare cells that have no connection to any other cell evenly across the chip. If you set this option to `off`, the placement engine does not do anything special to spread the spare cells. The default is `on`.

Table 6-7 Miscellaneous Control (Continued)

Option	Description
`set_fp_placement_strategy -virtual_IPO on | off`	Controls whether to use virtual in-place optimization during timing-driven placement. Timing-driven virtual flat placement uses a net weighting method. It adds a higher weight on the more critical nets, based on critical path information from the timing analysis. Some nets, however, that might be critical at the time of placement might not be considered critical later on, because in-place optimization can easily fix the timing on these nets and paths. Using virtual in-place optimization during timing-driven virtual flat placement estimates the effects of timing optimization so that higher net weights are applied to the real critical nets. Virtual in-place optimization estimates possible buffer insertion and cell sizing and their effects on the timing during timing analysis. If you select a value of `on`, the virtual in-place optimization feature is turned on inside timing-driven virtual flat placement. A value of `off` (the default) turns it off. Using virtual in-place optimization during timing-driven virtual flat placement can help the placement engine recognize the critical paths that timing optimization might not be able to fix, and therefore, the placer will put more effort into improving the timing on these critical nets and paths.

Creating Macro Blockages for Hard Macros

If you do not want hard macros to be placed in certain areas of your design, you can create a macro blockage to restrict the placement of hard macros.

For example, there might be a channel between two plan groups or two hard macros that is large enough for standard cells to go through, but if a hard macro is placed in the channel, routing would be adversely affected. In this case, placing a macro blockage in the channel restricts the placement of the hard macro. Another example might be if a hard macro were to be placed near a power pad, which could have an adverse effect on the IR drop. In this case, placing a macro blockage around the power pad ensures that the hard macro is not placed close to the power pad.

To create a placement blockage for hard macros only,

1. Choose Floorplan> Create Placement Blockage.

   The Create Placement Blockage dialog box appears.

   Alternatively, you can use the `create_placement_blockage` command.
2. Select Hard macro blockage to create a macro blockage.
3. Click in the Name text box, and enter a unique name for the macro blockage.
4. Specify the bounding box (rectangle) in which to create the blockage. The rectangular coordinates are relative to the current design and identify the lower-left and upper-right corners of the rectangular area.

   You can also specify the area graphically in the layout window by selecting the button next to the Coordinates field.
5. Click OK or Apply.

Padding the Hard Macros

To avoid placing standard cells too close to macros, which can cause congestion or DRC violations, you can set a user-defined padding distance or keepout margin around the macros. You can set this padding distance on a selected macro's master cell. During virtual flat placement, no other cells will be placed within the specified distance from the macro's edges.

To set a padding distance (keepout margin) on a selected macro's master cell,

1. Choose Placement > Set Keepout Margin.

   The Set Keepout Margin dialog box appears.

   Alternatively, you can use the `set_keepout_margin` command.
2. Specify the type of keepout margin to be created, either hard or soft. The default is hard
3. Choose whether to set a user-defined padding distance on all macros (the default), on specified macros, or on specified instances. Enter the names of the specified masters or instances, if applicable.
4. If you selected "All macros," you can choose to either derive the amount of padding based on the number of pins and track width, or you can specify explicit keepout distances on all four sides. If you select "Specified masters" or "Specified instances", you must specify explicit keepout distances on all four sides.

If you choose “Derive outer keepout coordinates”, specify the following:

- The number of tracks per macro pin.

  If you have macros with lots of pins, you should set the padding to 0.5 to 1.0 track per pin (0.5 reserves one track for every two pins) to reserve some routing area around the macros and give the router enough space to route to the macro’s pin.

- The minimum padding per macro size.

  This sets the minimum padding that is applied when there are few pins on some sides of some macros. The default is 0.

- The maximum padding per macro size.

  The default is `-1`, which means that the placer would apply a 40x metal1 pitch limit on all macros.

If you choose “Use specified outer keepout coordinates”, enter the explicit padding in microns in the Top, Bottom, Right, and Left fields.

5. Click OK or Apply.

After the padding distance or keepout margin is defined on a master cell, it will follow the movement of the hard macros; that is, when a hard macro moves, the padding moves along with it.

The padding is automatically saved to the design database when the design is saved.

Automatic Padding

If you do not define a padding distance (keepout margin), the tool automatically generates a default keepout margin based on the pin count on the edges for all the hard macros. By default, the placement keepout margin on an edge is calculated as follows:

(number of pins on edge /2)* pitch

Note:
You can change this value by changing the “Tracks per macro pin” option value in the Set Keepout Margin dialog box.

User-defined padding overrides automatic padding.

Placing Hard Macros and Standard Cells

You can perform a virtual flat placement and obtain a flat placement of the hard macros and standard cells. Based on the placement results, you can then decide the relative locations, shapes, and sizes of the top-level logic blocks. If you have a design with plan groups or voltage areas, the refined placement honors the exclusiveness of the plan groups and voltage areas.

Note:

The main consideration when placing a large hard macro is congestion; wire length becomes secondary. A hard macro is considered large if it takes up 5 percent or more of the chip area or 40 percent or more of the chip width. To improve routability, place large hard macros at the edges of the chip boundary.

To place the hard macros and standard cells simultaneously,

1. Choose Placement > Place Macros and Standard Cells.

   The Place Macros and Standard Cells dialog box appears.

   Alternatively, you can use the `create_fp_placement` command.

2. Click Default, and then click OK.

This section includes the following topics:

- Planning the Location and Shape of Voltage Areas
- Controlling the Placement
- Moving Cells in the Core Area to a New Location
- Packing Hard Macros Into an Area
- Manually Adjusting the Hard Macros
- Performing Simultaneous Placement and Pin Assignment for Block-Level Designs

Planning the Location and Shape of Voltage Areas

You can automatically place and shape voltage areas, including voltage areas that are physically nested. You can also create voltage areas outside the chip area.

This section includes the following topics:

- Handling Nested Voltage Areas
- Supported Voltage Area Physical Boundary Scenarios

- Updating Voltage Areas in a Design
- Removing Voltage Areas

In IC Compiler, you can define voltage areas at any level of a logic hierarchy. Both rectangular and rectilinear voltage areas are allowed. Voltage areas cannot consist of disjoint rectilinear and rectangular shapes.

Power domains and voltage areas should maintain a one-to-one mapping relationship. A power domain and its matching voltage area should have the same name and same set of hierarchical cells. When a power domain is mapped with the voltage area, the associated logic information is automatically derived from the power domain.

Voltage areas can be explicitly associated with existing power domains, or they can be created without specifying any explicit association with power domains. They can also be created when no power domains have been defined for the design.

A given voltage area can contain one or more physically nested voltage areas. For more information, see "Handling Nested Voltage Areas" on page 6-33. A voltage area must completely contain its nested voltage areas. These nested voltage areas correspond to logically nested power domains. Voltage areas that would overlap physically must be resolved into nonoverlapping, abutted rectangular or rectilinear subareas. Intersecting voltage areas are not supported.

Planning the location and shape of voltage areas requires the following steps:

1. Create a voltage area of a specific geometry on the core area of the chip.

   Choose Floorplan > Create Voltage Area

   The Create Voltage Area dialog box appears.

   Alternatively, you can use the `create_voltage_area` command.

   - Name – Specify the name of the voltage area you want to create. The DEFAULT_VA is reserved and cannot be used.

     If you specify this option, you must also provide a list of hierarchical cells that are contained within the voltage area. The logic hierarchies are assigned to specific voltage areas when you create or update these voltage areas

   - Power Domain – If your design contains power domains, you can use this option with the corresponding power domain name to create a voltage area. Then the power domain name is also the voltage area name. After the voltage area is created, the power domain contains the corresponding boundary information.

     In this case, do not specify a list of hierarchical cells. The cell list is provided in the power domain definition. Power domains and voltage areas should maintain a one-to-one mapping relationship. A power domain and its matching voltage area

should have the same name and same set of hierarchical cells. IC Compiler places these cells in the given voltage area. Using the power domain option ensures the correct alignment of the logic hierarchies with the voltage areas.

- Horizontal guard band width and Vertical guard band width – You can use guard bands to ensure that no shorts occur at the boundaries of the voltage areas. Guard bands define hard keepout margins surrounding the voltage areas. No cells, including level shifters and isolation cells, can be placed within the guard band margins.

  For example, using guard bands is recommended when the rows are the same for all the voltage areas. The guard bands guarantee that the cells in different voltage areas are separated so that power planning does not introduce shorts.

  You can specify a guard band width in the horizontal and vertical direction when creating or updating voltage areas.

- Target utilization – Specifies the target utilization for the voltage area during shaping.

  You must specify a utilization value between 0.1 and 1.0. The default utilization is the same value used for the rest of the design. Larger utilization values result in smaller voltage areas, with less extra space allowed for routing. A voltage area is created outside the chip area using the given utilization.

  To change the target utilization, you must use the `remove_voltage_area` command to remove all voltage areas or specified voltage areas from the design, and then re-create the voltage area and the new target utilization value.

- Bounding Boxes – If you know an ideal location for a voltage area, you can direct it by specifying explicit coordinates. The voltage area geometry can be a rectangle or a rectilinear polygon.

  To define a rectangular voltage area, select the rectangular-shaped button, and type the x- and y-coordinates for the upper left and lower right corners of the rectangle in the Coordinates box.

  To define a rectilinear voltage area, select the rectilinear-shaped button, and type the x- and y-coordinates for each corner of the polygon in the Coordinates box.

  After the user-defined voltage areas are created, the tool automatically derives a default voltage area for placement of cells not specifically assigned to any voltage areas.

- Set as fixed – You can place newly created voltage areas inside the core in fixed locations. The voltage areas with a fixed attribute are ignored by the `shape_fp_blocks` command and are not reshaped. By default this option is off.

- Color cell instances – (Optional) Select a method for coloring cells in the voltage area.

  To disable cell coloring, select None.

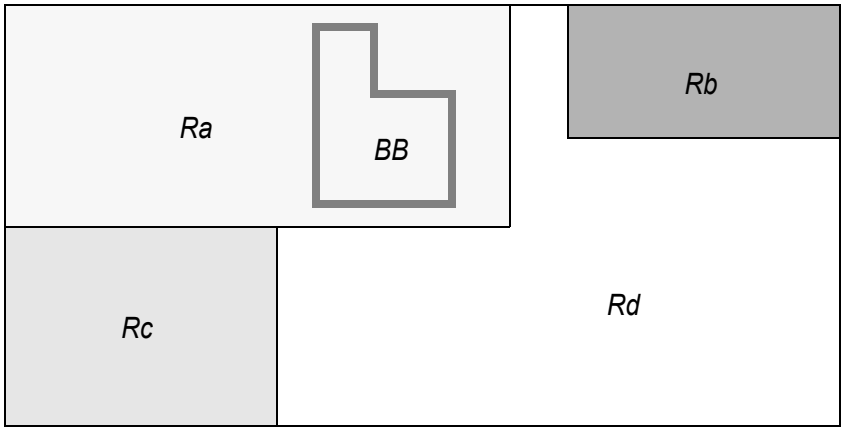
  To cycle through the available colors, select Cycle. This is the default.

To apply a specific color, select Specified and select a color in the color list.

2. Click OK or Apply.
3. Use the `create_fp_placement` command to place the design, keeping voltage area cells together. You can also choose Placement > Place Macro and Standard Cells. For more information, see "Placing Hard Macros and Standard Cells" on page 6-30.
4. Use the `shape_fp_blocks` command and options to automatically shape and place the voltage area boundaries. You can also choose Placement > Place and Shape Plan Groups. For more information, see "Automatically Placing and Shaping Objects in a Design Core" in Chapter 7.

Figure 6-16 shows a multivoltage design divided into several voltage areas.

Figure 6-16 Multivoltage Design With Several Voltage Areas



In this design, the floorplan has four voltage areas: Ra, Rb, Rc, and Rd. The default voltage area is Rd. Each voltage area has a corresponding logical partition, A, B, C, and D, that aligns with its respective voltage area. A hard bound can be defined inside a voltage area but not across voltage areas. In this example, a hard bound, BB, is defined inside voltage area Ra.

Special always-on site rows can be defined within a given voltage area. The standard cells placed in these site rows serve as the always-on cells of the shut-down power domain associated with the given voltage area.

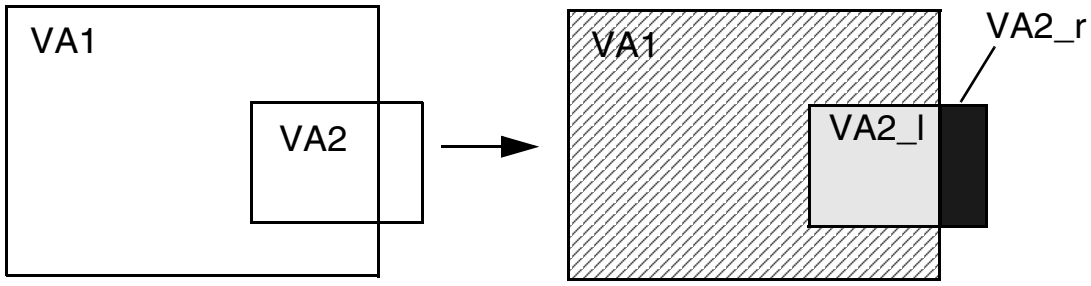
Handling Nested Voltage Areas

Voltage areas can be physically nested corresponding to the nested relationship of the logic hierarchy of nested power domains. You still use the `create_voltage_area` command to define the individual voltage areas. A given voltage area can contain one or more nested voltage areas. Note that a voltage area must completely contain its nested voltage areas. Intersecting voltage areas are not supported.

In the case where logically nested hierarchies would lead to overlapping voltage areas, you can use the `create_voltage_area` command to resolve the overlapping areas by defining a number of voltage areas consisting of separate, *abutted* rectangles or rectilinear shapes. Each shape would share at least one boundary with another shape of the given voltage area.

Figure 6-17 shows an example of areas intersecting and how you might resolve them into three physically abutted, separate voltage areas, VA1, VA2_l, and VA2_r. Voltage area VA1 is an eight-sided rectilinear area, and voltage areas VA2_l and VA2_r are four-sided rectangles.

Figure 6-17 Example of Physically Resolved Voltage Areas

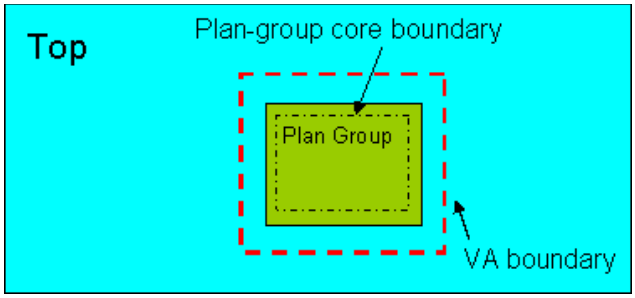


Supported Voltage Area Physical Boundary Scenarios

The following voltage area physical boundary scenarios are supported.

- The voltage area boundary encloses the plan group core boundary, as shown in Figure 6-18.

Figure 6-18 Voltage Area Boundary Encloses Plan Group Boundary

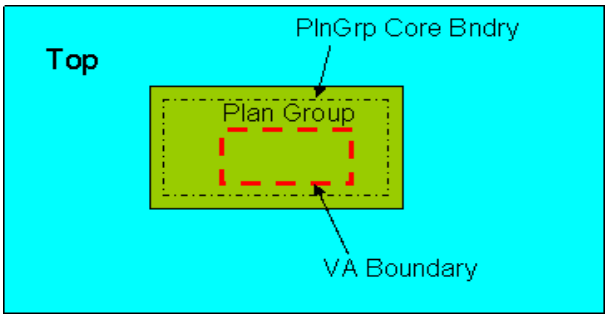


During commit hierarchy (`commit_fp_plan_groups` command), the physical voltage area boundary is not copied down into the soft macro, and child cell instances are not assigned logically to any non-DEFAULT_VA. The power domain is updated to swap the hierarchical cell with the new cell instance.

During uncommit hierarchy (`uncommit_fp_soft_macros` command), the soft macro child cell instances are pushed back up to the top CEL view, and are reassigned logically to the original top cell non-DEFAULT_VA.

- The voltage area boundary is contained by the plan group core boundary, as shown in Figure 6-19.

Figure 6-19 Voltage Area Boundary Contained by Plan Group Boundary



During commit hierarchy, the physical voltage area boundary is copied down into the soft macro, and the child cell instances are assigned logically to their respective non-DEFAULT_VA's. The effects on associated power domains are not considered.

During uncommit hierarchy, the soft macro child cells are assigned logically to the original top cell non-DEFAULT_VA's.

Updating Voltage Areas in a Design

After you have created voltage areas and associated hierarchical cells with them, you can use the `update_voltage_area` command to add or remove cells from specified voltage areas. You can use the `get_voltage_areas` command instead of explicitly specifying voltage area names. By adding or removing cells, you can adjust the utilization of the voltage area. However, you might prefer to change the size of the voltage areas and keep the utilization the same.

Removing Voltage Areas

Use the `remove_voltage_area` command to remove all voltage areas or specified voltage areas from the design. This includes removing any move bounds and guard bands belonging to the voltage area. If you remove a logically nested voltage area, the subhierarchy logic inherits the voltage properties of the parent hierarchy.

Controlling the Placement

This section describes the placement constraints that you can use to place hard macros and standard cells and control the placement results.

Choose Placement > Place Macro and Standard Cells. The Place Macro and Standard Cells dialog box appears.

Alternatively, you can use the `create_fp_placement` command.

- Selecting an effort level

  The effort level controls the trade-off between the QoR and the amount of CPU runtime spent performing an initial flat placement. Select the low-effort option (the default) for very fast results that provide good plan group locations and hard macro locations.

  Raising the effort level to high improves the quality of the placement, but incurs an increase in CPU runtime.

- Using the “Congestion driven” option

  You can perform congestion-driven placement by selecting the “Congestion driven” option. The default is off. Standard cells and hard macros are placed together to minimize wire length and congestion in the design and to avoid cell overlaps.

  Congestion-driven placement:

  - Reduces congestion by adding padding to cells in congested areas.

    The amount of padding needed for each cell depends on the congestion around each cell.

  - Moves hard macros and standard cells in congested areas.
  - Pads macros in automatically generated arrays, thereby changing the size of the array.

  If there is congestion between hard macros, congestion-driven placement increases the channel size and keepout region around the macros to reduce congestion. Additional cells are not placed in the channels. After placement, the command saves these keepouts to the file design_planning_blockages.tcl in the current working directory, and removes the added keepouts from the design. To reinsert the keepouts, source the file design_planning_blockages.tcl prior to the in-place optimization step.

- Using the “Timing driven” option

  You can perform incremental timing-driven placement to improve the timing result for your design by selecting the “Timing driven” option. The default is off.

  Timing-driven placement:

  - Improves the timing by moving cells on critical paths closer together.

  - Moves hard macros on critical paths.

- Using the “Hierarchical gravity” option

  The “Hierarchical gravity” option tends to keep cells of a hierarchical design logic block physically together in the chip layout. This type of grouping usually results in better placement because designs tend to partition well along hierarchical boundaries. However, you should deselect this option if you know that the logic hierarchy does not represent a good physical partitioning of the design. This allows the placement engine the flexibility to place cells of a hierarchical block far from other cells from the same block. The default is on.

  If there are no user-created plan groups, the placement engine analyzes the logic hierarchy of the design and uses the blocks with reasonable sizes as the default plan groups.

  The use of this option is also recommended for placement consistency. For example, if you make a minor change to your design but keep the same cell instances together and then use this option, those cell instances will remain near their original locations.

The following options are available in the Advanced Options dialog box:

- Choosing a maximum fanout

  Enter a maximum fanout as a positive integer. The wire lengths of any nets with a fanout higher than the number you specify are ignored by the placement. The default is 512.

- Performing an incremental placement

  If you want the placement to start with existing cell locations and incrementally improve on them during the placement, select the “Incremental” option. Use this option if you want to start with a good relative placement but do not want the cells to move too far from their current locations. (If cells need to be moved far from their current locations and you use this option, the result will be a poor placement.) This option is off by default, and the placer ignores existing cell locations and creates a placement from scratch.

  Another reason to use this option would be if you have already run a placement on the design with placed plan groups inside the core area and there is a congestion or timing problem.

  When plan groups are placed inside the core area, enable the Incremental option and then select from the following three suboptions to further refine the placement on selected parts of your design.

  All – Runs incremental placement on all the standard cells and hard macros in the design. This is the default.

  Top level cells only – Runs incremental placement only on the top-level cells that do not belong to a plan group.

Specified plan groups – Performs a refined placement on only the selected exclusive plan group cells. You can select one or more plan groups.

Specified voltage areas – Performs a refined placement on standard cells and hard macros within the specified voltage areas. The virtual flat placement engine will recognize voltage areas as physical constraints in the floorplan.

To create a floorplan with multiple voltage areas, you need to specify each logic module that belongs to the separate top-level power domains. Placement voltage areas are used to constrain the placement of cells to certain locations, based on the cell's power domain. A power domain is a set of cells with the same voltage requirements. This means that each delineated power domain should include only cells driven from the same power supply.

In a single-voltage design, the entire design operates under the same voltage, the default voltage. In a multivoltage design, you need to define areas where all cell instances use a higher or lower voltage than the default voltage. The rest of the design area (outside of any defined voltage areas) uses the default voltage. For multivoltage designs, the default voltage area equals the design boundary minus all the voltage areas.

- Legalizing the resulting placement

  Select the "Legalize resulting placement" option to resolve cell placement conflicts after doing initial placement. Overlaps are removed, and standard cells are placed at legal sites. Legalization is appropriate for the final placement. The default is on.

  Alternatively, you can use the `legalize_fp_placement` command.

- Optimize pins

  Select the "Optimize pins" option to perform simultaneous placement and pin assignment for block-level designs. The placement engine places the cells near the port locations according to the TDF pin constraints (side, side and order, and side and location) that you have set. The result is a simultaneous virtual flat placement and pin assignment for block-level designs with a smaller wire length of nets connected to the constrained pins

- Ignore scan chain connectivity

  Select this option if you want the placement engine to ignore scan chain connections during virtual flat placement.

- Specifying the number of CPUs

  Enter the number of CPUs to be used in parallel during initial virtual flat placement. The number you enter should be an integer value less than or equal to the number of free CPUs on your machine. The default is one CPU.

Moving Cells in the Core Area to a New Location

You can move (unplace) macro cells, standard cells, or all cells that reside in the core area of the chip, and place them in a new location by using the `remove_placement` command. The command does not unplace cells with `fixed` or `dont_touch` attributes assigned to them.

The `remove_placement` command uses the following syntax:

```
remove_placement
 -object_type standard_cell | macro_cell | all
 -new_location top_right | origin | center
```

For the `-object_type` option, the default moves (unplaces) all cell types.

For the `-new_location` option, the default moves the cells to the `top_right`. Macro cells are moved to top side of the chip, and standard cells are moved to the right side of the chip.

If you specify the `-origin` argument, the standard cell coordinates are set to (0,0).

If you specify the `-center` argument, the standard cell coordinates are set to the center of the chip.

Packing Hard Macros Into an Area

You can specify an area (region) after performing virtual flat placement and automatically pack selected hard macros along the edges of that selected area. A data structure is automatically built for packing the hard macros.

To automatically pack hard macros into an area,

1. Choose Placement > Pack Macro Cells.

   The Pack Macro Cells dialog box appears.

   Alternatively, you can use the `pack_fp_macro_in_area` command.
2. Set the options, depending on your requirements.
   - Pack design
   - Pack macro cells of specified plan groups - All hard macros belonging to selected plan groups are packed inside the plan group boundaries. If a plan group boundary is outside the core area, any macros belonging to that plan group are ignored. Top-level macros and macros belonging to other plan groups are also ignored.
   - Pack all macro cells - Select this option (the default) to pack all macro cells inside the core area. Any top-level macros are also packed into the remaining area.

- Pack specified macro cells - Select this option to pack only the specified macro cells into the core area. If a polygon or rectangle is selected (it can be on any layer except a blockage layer) together with hard macros, the hard macros are packed into the selected polygon or rectangle.
- Preferred edges

  You can specify the edge of the area (region): L (left), R (right), B (bottom), and T (top) against which to pack the hard macros. These are the preferred edges for placing the macro cells. For example, if you select Left, Top, and Bottom, the macro cells will be placed along those edges first, arranging them in a “C” shape.

  Selecting all four edges will cause the macro cells to be placed along all four edges, arranging them in an “O” shape. This is different from specifying no edges, which will cause placement to be based on other considerations, such as wire length.

3. Click OK or Apply.

Manually Adjusting the Hard Macros

To manually adjust the hard macros, you should observe the following principles, in addition to applying your own criteria:

- Follow the given relative locations from the hard macro placement command (`create_fp_placement`).
- Avoid isolated standard cell placement areas enclosed by a ring of hard macros.
- Avoid narrow standard cell placement areas in channels (slivers) between hard macros.
- Align similar hard macros.
- Push large hard macros to the core boundary.
- For standard cell placement, create a rectangular placeable area.

The “Align objects” and “Distribute objects” buttons on the Edit toolbar are particularly helpful for adjusting your hard macros.

Using the Align Objects Button

To use the “Align objects” button, first select one or more hard macros in the layout window, then move your cursor over the arrow next to the button, and click to display the left, right, top, bottom, vertical, and horizontal alignment icons.

Alternatively, you can choose Edit > Align in the GUI, or you can use the `align_objects` command.

Selecting the right-align icon, for example, moves all selected hard macros so that their right edges are aligned. If one hard macro is selected, its right edge is aligned with the right edge of the core. If more than one hard macro is selected, one of those hard macros is also selected as the anchor point and the right edge of the remaining hard macros is aligned with the anchor object's right edge. If any selected objects are fixed, the rightmost fixed hard macro is the anchor object; otherwise, the rightmost hard macro's edge is the anchor edge.

Using the Distribute Objects Button

To use the "Distribute objects" button, first select one or more hard macros in the layout window, then move your cursor over the arrow next to the button, and click to display the left, right, top, and bottom distribution icons.

Alternatively, you can choose Edit > Distribute in the GUI, or you can use the `distribute_objects` command.

Selecting the left distribute icon, for example, moves all selected hard macros so that they are laid out next to each other from left to right. The distance between the hard macros is specified by the value in the Offset box.

Performing Simultaneous Placement and Pin Assignment for Block-Level Designs

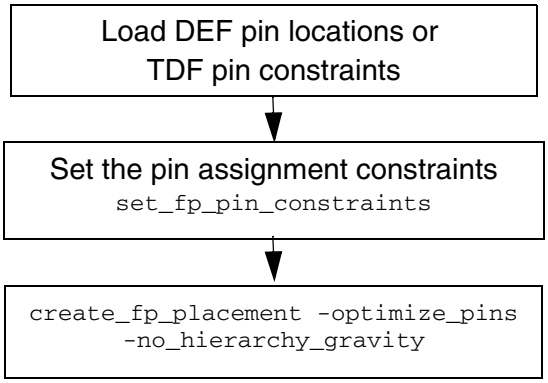
You can simplify your design flow by performing simultaneous placement and pin assignment for block-level designs. To do this, choose Placement > Place Macro and Standard Cells > Advanced Options. Select "Optimize pins" on the dialog box.

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

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

Figure 6-20 shows the design flow for the simultaneous placement and pin assignment for block-level designs.

Figure 6-20 Simultaneous Placement and Pin Assignment Flow for Block-Level Designs



Performing Fast Placement for Floorplan Exploration

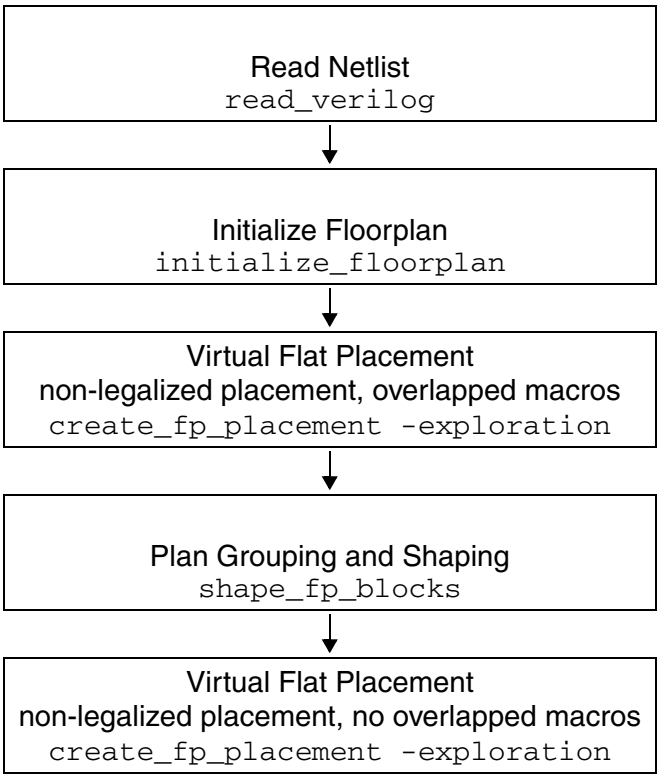
The fast floorplan exploration flow provides a methodology to quickly generate a trial floorplan for routability analysis. Using this flow minimizes runtime during the virtual flat and hierarchical placement steps and produces placement adequate for top-level routability analysis.

During the flat placement stage prior to plan group creation, you can use the `create_fp_placement -exploration` command to place hard macros and standard cells. The command groups together logic from the same hierarchy, as in the standard design planning flow, although overlaps might exist. This placement provides sufficient information to determine plan group size, shape, and location.

During the hierarchical placement stage after plan group creation, use the `create_fp_placement -exploration` command to place hard macros and standard cells. The command places hard macros with no overlap, but standard cells might overlap. More effort is applied to the placement of top-level cells, interface cells and hard macros inside the plan groups. At the end of hierarchical placement in exploration mode, the placement of hard macros, top-level cells, and interface cells has the same quality as regular hierarchical placement. The placement results are sufficient for you to perform the top-level routability analysis.

Figure 6-21 shows a typical timing-driven exploration flow using the fast placement exploration mode.

Figure 6-21 Timing-Driven Exploration Flow



Note:

The fast exploration flow does not support designs with multiply instantiated modules, black boxes, and multicorner-multimode.

The `-congestion_driven`, `-effort`, `-incremental`, `-optimize_pins`, and `-timing_driven` options of the `create_fp_placement` command are not compatible with the fast floorplan exploration flow.

In addition to the `create_fp_placement -exploration` command, the floorplan exploration flow is enabled by using other commands and options. The following paragraphs describe the flow steps and the commands to enable the floorplan exploration flow.

- High-fanout synthesis

  By default, the commands that perform high-fanout synthesis and in-place optimization of feedthroughs also run the legalizer, even though legalization is not necessary during the exploration flow. To disable the legalizer, set the `fpopt_no_legalize` variable to `true`

and use incremental high-fanout synthesis instead of regular high-fanout synthesis. To run incremental high-fanout synthesis, use the `set_ahfs_options -incremental true` and `optimize_fp_timing -hfs_only` commands.

- Incremental high-fanout synthesis is necessary for the exploration flow to reduce runtime
- Incremental high-fanout synthesis in the exploration flow is more efficient for analyzing the clumped placement of preexisting buffers.
- Incremental high-fanout synthesis does not analyze preexisting buffers and their placement.

- Routing

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

- Optimization

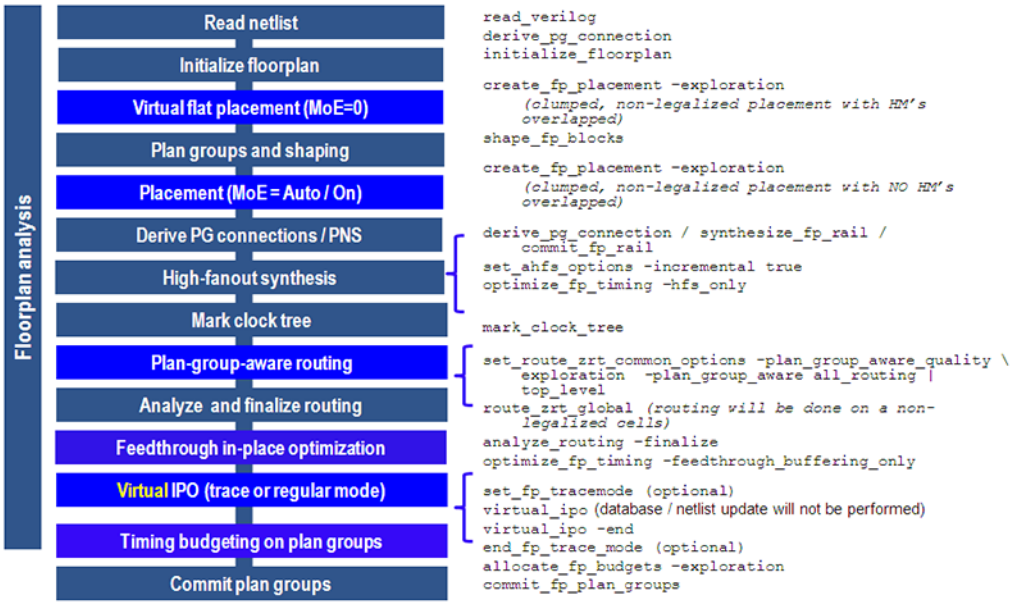
  The `optimize_fp_timing -feedthrough_buffering_only` command, followed by virtual in-place optimization, performs optimization without running the complete in-place optimization flow. The `virtual_ipo` and `virtual_ipo -end` commands perform in-place optimization for fast optimization during design exploration.

- Timing budgeting

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

Figure 6-22 shows the suggested flow.

Figure 6-22 Timing-Driven Exploration Flow



Supporting Relative Placement Groups in Initial Virtual Flat Placement

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

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

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

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

The `create_rp_group` command creates relative placement groups. A relative placement group is an association of cell instances, other hierarchical relative placement groups, and placement keepouts. A group is defined by the number of rows and columns that it uses. Use the `add_to_rp_group` command to add leaf cells to a relative placement group. Information about the relative placement groups is stored in the Milkyway database by using the `write_rp_groups` command. It contains the height and width of the top-level relative placement groups and the x and y offsets of the cell instances, placement keepouts, and hierarchical relative placement groups for each top-level relative placement group. The x and y offsets are used to anchor the relative placement cell instance with respect to the lower-left corner (the relative placement's origin) of its corresponding relative placement group.

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

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

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

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

Using the create_fp_placement Command to Place Cells in Relative Placement Groups

By default, relative placement is supported during initial virtual flat placement. The `create_fp_placement` command extracts information about the relative placement groups from the Milkyway database. A dummy cell, which contains both macros and standard cells in one relative placement group, is created for each top-level relative placement group. The Milkyway properties of the relative placement group are modified based on the new

locations of the placed dummy cells. All relative placement cell instances are temporarily fixed. Any dummy cells related to the top-level relative placement groups are removed. This allows the `create_fp_placement` command to place other standard cell instances over the free spaces inside the relative placement bounding boxes.

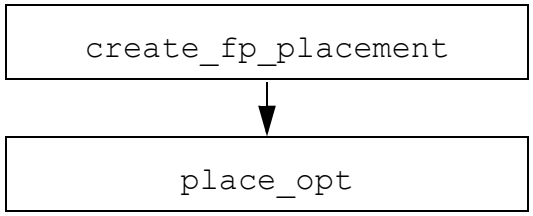
The `create_fp_placement` command honors the specified placement keepouts when it places the cell instances in the relative placement group, and no cell instances in the relative placement group are placed over the keepout areas.

Default Relative Placement Flow

Use the default relative placement flow if you do not want to fix the relative placement groups after you run the `create_fp_placement` command.

Figure 6-23 shows the default relative placement flow.

Figure 6-23 Default Relative Placement Flow



Use the `legalize_fp_placement` command to automatically legalize the standard cells. The standard cells in relative placement groups might be moved outside of the corresponding relative placement bounding box during legalization.

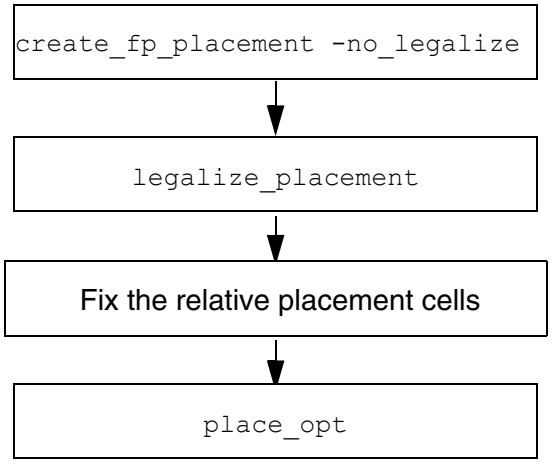
Using the Relative Placement Cells Flow

Use the relative placement cells flow if your design contains either macro-only or standard cell-only relative placement groups and you need to fix the relative placement groups after floorplanning to ensure the exact relative placement constraints are met.

1. Define the relative placement constraints.
   - Create the relative placement groups by using the `create_rp_group` command.
   - Add relative placement objects to the groups by using the `add_to_rp_group` command.
2. Run virtual flat placement by using the `create_fp_placement -no_legalize` command.
3. Mark all macros with the `is_fixed` variable.
4. Legalize your design by using the `legalize_placement` command.
5. Mark all relative placement cells with the `is_fixed` variable.
6. Perform physical placement by using the `place_opt` command.

Figure 6-24 shows the flow for fixing the relative placement cells.

Figure 6-24 Fixing the Relative Placement Cells Flow



Use this flow to perform virtual flat placement if your design contains relative placement groups that have mixed macros and standard cells.

1. Run virtual flat placement by using the `create_fp_placement -no_legalize` command.
2. Mark all macros with the `is_fixed` variable.

3. Legalize your design by using the `legalize_placement` command.
4. Mark all macros and standard cells in the relative placement groups with the `is_fixed` variable.
5. Continue the design planning flow.

Propagating Relative Placement Groups by Commit and Uncommit in a Hierarchical Flow

During the commit and uncommit hierarchy steps of the design planning flow, the `commit_fp_plan_groups` and `uncommit_fp_soft_macros` commands support relative placement groups, including hierarchical relative placement groups. The `commit_fp_plan_groups` command converts plan groups into new soft macros. If the plan group includes relative placement groups, they are automatically propagated from the top cell to soft macros. The `uncommit_fp_soft_macros` command converts soft macros into top-level plan groups. If the soft macro includes relative placement groups, they are automatically propagated to the plan group that is in the top level.

Committing Relative Placement Groups

As described in the following steps, for each plan group that the `commit_fp_plan_groups` command processes, the relative placement groups are searched to determine if they belong to the plan group that is being committed.

1. All relative placement groups in the top cell are examined and for each relative placement group, the cell instances of the hierarchy are examined to see if they belong to a descendant of a plan group's logical hierarchy.
2. If cell instances that belong to each relative placement group are determined to be logical descendants of the plan group's logical hierarchical cell instance, the relative placement group is processed by the `commit_fp_plan_groups` command, and an exact copy of it is created in the new soft macro, according to its placement within the plan group boundary.

   Any keepout blockages associated with the relative placement group are pushed down, and the cell instances are assigned to the relative placement group. The keepout blockages are also checked to see if they are included within the plan group boundary. If they extend outside the plan group boundary, a warning message will be issued, but the commit process will continue.
3. After all relative placement groups that belong to the plan group are pushed down, a link step occurs which populates hierarchical attachment lists in some of the relative placement groups that have hierarchical trees. Relative placement group instances are also instantiated.

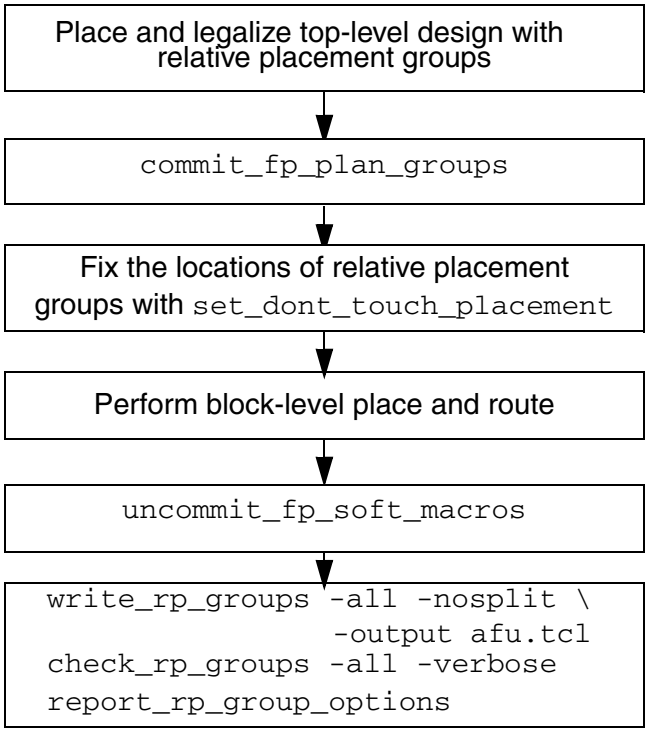
Uncommitting Relative Placement Groups

For each soft macro that the `uncommit_fp_soft_macros` command processes, all relative placement cells in the relative placement groups are replicated in the top cell according to the hierarchical cells and the hierarchical cell instance masters within the soft macro.

Design Flow for Hierarchical Relative Placement Groups

Figure 6-25 shows the flow for designs containing hierarchical relative placement groups.

Figure 6-25 Hierarchical Relative Placement Group Design Flow



Use the following flow for designs that contain hierarchical relative placement groups.

1. Place and legalize the design with hierarchical relative placement groups by using the `create_fp_placement -no_legalize` and `legalize_placement` commands.
2. Convert plan groups into soft macros, also called child cells or physical blocks, by using the `commit_fp_plan_groups` command.

   The command extracts relative placement groups that belong to the plan group from the top CEL view and pushes them down to soft macros. Each soft macro is a CEL view of the block in the same Milkyway design library.
3. Fix the locations of the relative placement groups, which include macros, by using the `set_dont_touch_placement` command.

4. Perform place and route on each soft macro independently.
5. Convert all soft macros into plan groups by using the `uncommit_fp_soft_macros` command.

   During uncommit hierarchy, the soft macro child cell instances are pushed back up to the top CEL view.
6. After the uncommit process, you can check the validity of the relative placement groups by using the following commands:

```
write_rp_groups -all -nosplit -output afu.tcl
check_rp_groups -all -verbose
report_rp_group_options
```

Placing Multiply Instantiated Modules

You can perform virtual flat placement of multiply instantiated modules. Multiply instantiated modules are physically represented in the design as plan groups and have the same reference. A set of these instantiated modules is called a multiply instantiated module group. A design can have more than one multiply instantiated module group. If for example, modules A1 and A2 have reference A, and modules B1 and B2 have reference B, two multiply instantiated module groups are formed. Many designs use multiple instances of the same logic block to create the required functionality. By using the same block for similar operations, the design time is reduced.

This section includes the following topics:

- Determining Which Plan Groups and Soft Macros are Multiply Instantiated Modules
- Identifying Multiply Instantiated Module Plan Groups
- Shaping the Multiply Instantiated Module Plan Groups
- Placing and Analyzing the Multiply Instantiated Module Plan Groups
- Flipping the Placement of Multiply Instantiated Module Plan Groups
- Performing a QoR Analysis of Multiply Instantiated Module Plan Groups

Determining Which Plan Groups and Soft Macros are Multiply Instantiated Modules

Before you can place the multiply instantiated modules, you must first determine which plan groups and soft macro are to be considered as multiply instantiated modules. This can be done by using the `uniquify_fp_mw_cel -store_mim_propert`y command to store the multiply instantiated module information in the Milkyway database as a multiply instantiated module property. After uniquification, this makes it possible for other IC Compiler design planning commands to access the multiply instantiated module information and treat the cell instances with these properties as multiply instantiated modules in the virtual flat placement design planning flow.

The `uniquify_fp_mw_cel` command uses the following syntax:

```
uniquify_fp_mw_cel -store_mim_property cell_instances
```

For example:

```
uniquify_fp_mw_cel -store_mim_property [get_cells {instA instB instC}]
```

If you select the -`store_mim_property` option, the tool stores a multiply instantiated module property on the cell instances you specify during uniquification. Specifically, it stores the name of the original unique master for every specified hierarchical cell instance as a multiply instantiated module property.

Removing the Multiply Instantiated Module Property

You can use the `remove_mim_property` command to remove the multiply instantiated module data for selected cell instances. You must specify a list of hierarchical cells for which the multiply instantiated module data is to be removed. If a cell is not found, a warning message is issued.

The cell instances are no longer treated as multiply instantiated modules; each cell instance is now referred to as a unique master.

The `remove_mim_property` uses the following syntax:

```
remove_mim_property collection_of_cells
```

For example:

```
remove_mim_property [get_cells instA]
```

Identifying Multiply Instantiated Module Plan Groups

Use the `report_mim` command to print a list of all properly identified multiply instantiated module plan groups in the design. Soft macro multiply instantiated modules are also identified by this command, although for placement, they will be treated as hard macros.

Here is an example of a report:

```
icc_shell > report_mim
****************************
Report   :MIM
Design   :test.CEL;1
Version  :B-2008.09-ICC-CS
Date     :Mon Aug 25 17:01:51 2008
*********************************
Master BLENDER is instantiated to:
    Soft Macro: 1_ORCA_TOP/1_BLENDER_4
    Soft Macro: 1_ORCA_TOP/1_BLENDER_3
    Soft Macro: 1_ORCA_TOP/1_BLENDER_2
    Soft Macro: 1_ORCA_TOP/1_BLENDER_1
    Soft Macro: 1_ORCA_TOP/1_BLENDER_5
    Reference Blender is instantiated 5 times.
  Total 1 MIM references and 5 MIM instances.
  MIM plan group flipping grid:x offset=425.5,x step=0.410
                               y offset=425.565,y step=7.380
  MIM Report End
```

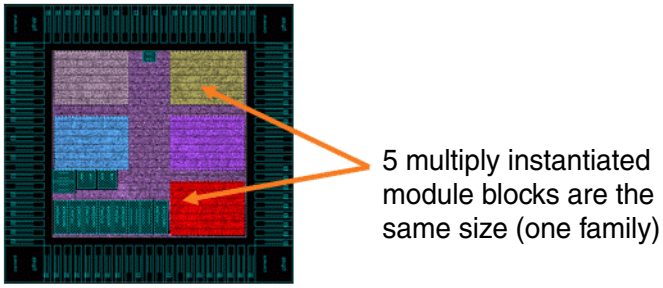
The report also provides information about the status of the copied placement data and suggests a multiply instantiated module plan group flipping grid on which to place an multiply instantiated module plan group. When the plan group is flipped, the standard cell placement remains legal.

Shaping the Multiply Instantiated Module Plan Groups

After running virtual flat placement using the `create_fp_placement` command, you can use the `shape_fp_blocks` command to shape the plan groups in each multiply instantiated module group automatically. For more information, see “Automatically Placing and Shaping Objects in a Design Core” in Chapter 7.

The plan groups in each multiply instantiated module group are shaped and sized identically, as shown in Figure 6-26.

Figure 6-26 Plan Groups of Multiply Instantiated Module Groups in the Same Size and Shape



All corners of the multiply instantiated module plan groups are aligned on their row and column placement boundaries and are placed in such away that they can be legally flipped, provided the rows are uniform.

Note:
The default plan group locations might not be aligned.

Placing and Analyzing the Multiply Instantiated Module Plan Groups

Run the `create_fp_placement` command to place the macros and standard cells into fixed placement plan group boundaries. However, the placement in the multiply instantiated module groups will not be identical.

You can use the `copy_mim` command and options to perform various manipulations during the virtual flat placement stage on the multiply instantiated modules to determine which of the plan groups in each multiply instantiated module group is the master of that multiply instantiated module group.

The placement of the master is copied to the other plan groups in the multiply instantiated module group. The copied information, which is saved to the Milkyway database, contains the original placement of all the cells in the copied plan multiply instantiated module plan groups relative to the lower-left corner of the plan group and the orientation of the plan group.

The `copy_mim` command uses the following syntax:

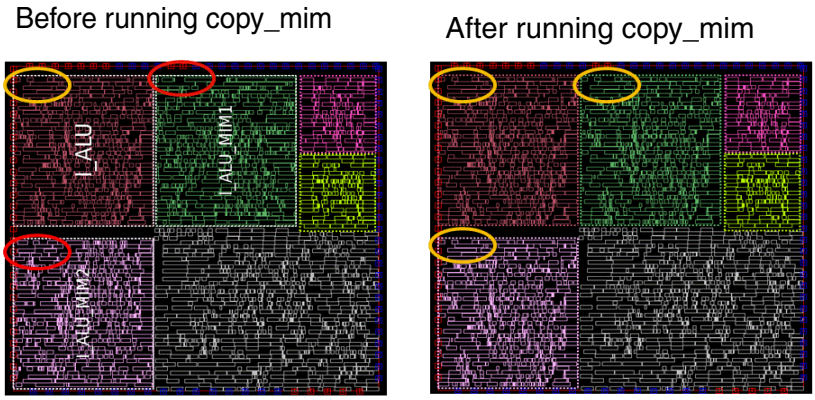
```
copy_mim [-type placement | blockage | boundary] [-restore_placement]
collection
```

A *`collection`* specifies a list containing one multiply instantiated module plan group.

As shown in Figure 6-27, the placement is different in every plan group before the `copy_mim` command is run. After the `copy_mim` command is run, the placement argument copies the cell placement of the given multiply instantiated module to the other multiply instantiated modules in the same group, as shown in the following example:

```
icc_shell> copy_mim -type placement 1_ALU
```

Figure 6-27 Before and After Running the copy_mim Command



You can create multiply instantiated modules with the same cell placement, blockages, and shapes as described in the following sections.

Copying the Cell Placement of a Plan Group

Use the `copy_mim -type placement` option to copy the cell placement of a plan group specified as a *`collection`* to all other plan groups in its multiply instantiated module group where the plan group in the list is the master. The copied information contains the original placement of all the cells in the copied multiply instantiated module plan groups relative to the lower-left corner of the plan group and the orientation of the plan group. This is the default.

Note:

Placement blockages are not touched by this command. Errors are reported if the *`collection`* is not part of an multiply instantiated module group or the plan groups in the plan group's multiply instantiated module group do not have the same size and shape.

You can make more than one copy on an multiply instantiated module group; however, you will lose a previous copy because only one level of undo is supported.

You can print out information about which multiply instantiated module groups are copied and the master plan group for that copy by using the `report_mim` command. If the shapes or sizes of the plan groups changed from the previous `copy_mim -type placement` command, this information is also reported.

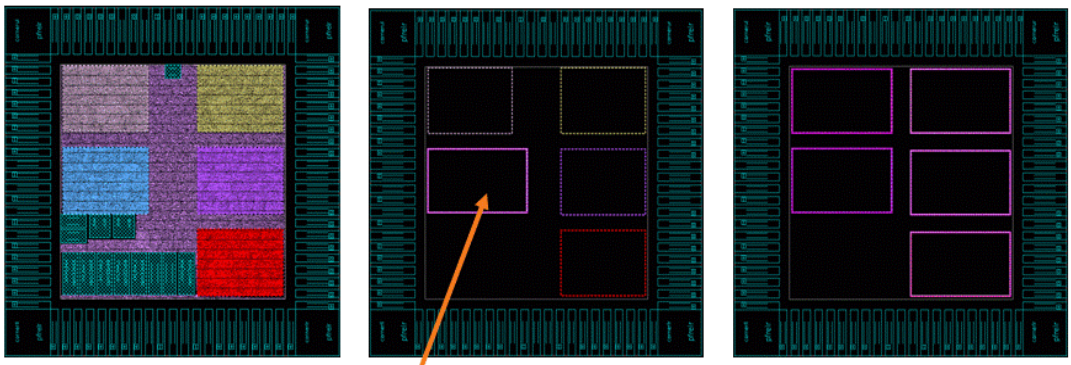
Copying Placement Blockages and Boundaries to Other Plan Groups

Use the `copy_mim -type blockage` option to copy the placement blockages that intersect the plan group's specified *`collection`* to all the other plan groups in the *`collection`* multiply instantiated module group. This command uses the plan group orientation you set with the `flip_mim` command. For more information, see "Flipping the Placement of Multiply Instantiated Module Plan Groups" on page 6-57.

Use the `copy_mim -type boundary` option to reshape each plan group that belongs to a multiply instantiated module group so that their boundaries are identical to the specified plan group in the *`collection`*, while at the same time preserving the orientation of the individual plan groups.

Figure 6-28 shows an example of a new modified boundary applied to all the other multiply instantiated modules in the same group.

Figure 6-28 Modified Boundary Applied to Other Multiply Instantiated Modules in the Same Group



Restoring the Placement of Cells in Plan Groups

Use the `copy_mim -restore_placement` option to restore the placement of cells in all the plan groups specified in the multiply instantiated module group as it was when the last `copy_mim -type placement` was done.

The orientation or flipping about the x- or y-axis of *`PG_list`* is also restored to what it was during the `copy_mim -type` placement command.

Note:

Placement blockages are not touched by this command. Errors are reported if any size or shape does not correspond to the original size or shape in the *`PG_list`* multiply instantiated module group.

Flipping the Placement of Multiply Instantiated Module Plan Groups

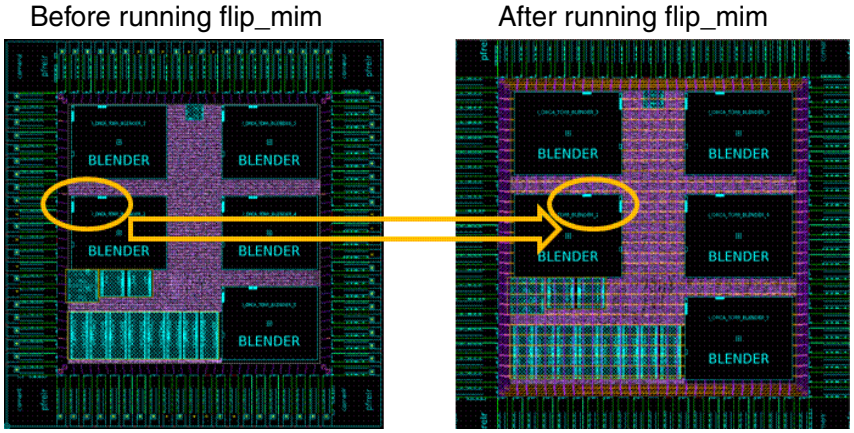
Use the `flip_mim` command to flip the placement of all cells and their shapes in the multiply instantiated module plan group *`collection`* list and set the orientation. The resulting pins on the multiply instantiated module are also flipped. After running the `flip_mim` command, the resulting placement is the same as if the plan group was converted to a soft macro, flipped, and then converted back to a plan group.

The `flip_mim` command uses the following syntax:

```
flip_mim[-direction x | y]collection
```

Figure 6-29 shows a before and after example of running the `flip_mim` command.

Figure 6-29 Before and After Running the flip_mim Command



Only mirroring, flipping about the x- or y-axis, is allowed for multiply instantiated module plan groups. If the *`collection`* list is not part of an multiply instantiated module group and if the plan group is not rectangular, an error is reported.

During flipping, the bounding box of the plan group does not move.

Placement blockages are not flipped; however, the `copy_mim -type blockage` command recognizes the flipped orientation.

Horizontal Requirements for Plan Group Locations

Consider the following horizontal requirements for plan group locations, when running the `flip_mim` command. The requirements ensure that the cells in the plan group have the exact same relative distance to the left and right plan group boundaries after flipping the plan group horizontally.

- Snap the left and right boundaries of a plan group to the edge of a unit tile or to a location that is one-half the unit tile width.
- Ensure that the plan group width is a multiple of the unit tile width.

Vertical Requirements for Plan Group Locations

Consider the following vertical requirements for plan group locations when running the `flip_mim` command. The requirements ensure that the cells in a plan group have the exact same relative distance to the top and bottom plan group boundaries after flipping the plan group vertically. The cells are flipped and placed legally at the rows with the opposite orientation.

- Snap the top boundary of a plan group to the top-side of a row and snap the bottom boundary of a plan group to the bottom-side of a row. Alternatively, you can snap both the top and bottom boundaries of the plan group to a location that is one-half the row height
- Ensure that plan group height covers an even number of rows. (It must not be an odd number of rows.)

Using the Multiple Instantiated Module GUI

You can run multiple instantiated module operations from a single dialog box by choosing Floorplan > Multiple Instantiated Modules in the GUI. The MIM Flow dialog box appears.

Performing a QoR Analysis of Multiply Instantiated Module Plan Groups

Use the `get_fp_wirelength` command to run a quality of results (QoR) analysis of the different multiply instantiated module plan groups.

The following options determine which plan group out of a set of multiply instantiated module plan groups has the best placement with respect to timing.

- Plan Group Interface Nets – Reports the wire length of interface nets for all plan groups. Interface nets have at least one pin inside the plan group and one pin outside the plan group boundary.
- Plan Group Internal Nets – Reports the wire length of internal nets for all plan groups. Internal nets have all their pins inside the plan groups.

Generating a QoR Placement Report

You can generate a quality of results (QoR) report for virtual flat placement by using the `report_fp_placement` command. It includes the following information:

- Total wire length
- Number of regions with high density
- Number of regions with low density
- Number of overlaps between hard macros
- Number of overlaps between hard macros and standard cells
- Number of cells that are not inside their respective plan groups (or core area)
- Number of overlaps between plan groups
- Number of cells violating the core area

If your design does not have channels, you can use the `-check_abutment` option to check for the abutment of each soft and hard macro with the surrounding macro cell instances. The abutment check reports gaps detected between the macro cell instances.

Checking the Total Wire Length Estimate

You can use the `get_fp_wirelength` command to evaluate the over all placement quality. The command generates vertical, horizontal, and total wire length statistics for the top-level signal nets in your design. It also calculates the virtual route wire length as a minimum length needed to connect all the pins.

Performing Floorplan Editing

You can use IC Compiler to perform the following floorplan editing operations:

- Create objects
- Delete objects
- Undo and redo edit changes
- Move objects
- Change the way objects snap to a grid
- Align movable objects

- Edit rectilinear objects
  - Turn snapping on or off
  - Cut and add object shapes
  - Change object shapes to rectangle, T-shape, L-shape, cross, or U-shape
  - Move and resize objects
- Resize objects manually
- Expand objects
- Distribute objects
- Spread objects
- Split objects
- Fix and unfix objects for edits

For detailed information, see the “Editing a Design” topic in IC Compiler online Help.

Using Constraints With the Core and Die Editing Commands

You can use the `-keep_placement` and `-keep_pad_to_core_distance` options with the editing commands that are relative to the core and die. Using these options makes it easier and more convenient to refine and debug core and die commands.

These options are effective only when they are applied to the following core and die editing commands:

`set_object_boundary, resize_objects, set_object_shape, window_stretch move_objects,` and `cut_objects`

When you move or change a core or die, you can use the `move_objects -keep_placement` command to ensure that standard cells and hard macros so that they remain within the boundary of the parent object (for example, the core, the die, or a plan group).

You can use the `-keep_pad_to_core_distance` option to maintain the distance between the core and the I/O pad cells and by implication, the distance between the core and the die. This forces changes to the core and die to be made in tandem. If the I/O pad cells are absent, the core-to-die distance is still maintained.

Note:

After moving or resizing objects, the resulting placement is not legalized. You must explicitly legalize the placement by using the `legalize_placement` command.

7

Creating and Shaping Plan Groups

This chapter describes how to create plan groups for logic modules that need to be physically implemented as a group. Plan groups restrict the placement of cells to a specific region of the core area. This section also describes how to automatically place and shape objects in a design core, add padding around plan group boundaries, and prevent signal leakage and maintain signal integrity by adding modular block shielding to plan groups and soft macros.

This chapter includes the following sections:

Creating Plan Groups

You create plan groups to manipulate the hierarchy. (See “Analyzing and Manipulating the Hierarchy” on page 7-8.) A plan group is a physical partition with a physical boundary. Each plan group represents a module in the logic hierarchy that you want to implement as a physical block and contains cells that belong to that module. A plan group can have many logical modules. The physical block inherits the size and shape (rectangular or rectilinear) of the plan group. Timing budgeting, pin cutting, and commit hierarchy are applied on plan groups.

Plan group boundaries are created for the modules you selected and are placed outside the chip boundary. Each plan group boundary is displayed in a different color in the layout window and in the hierarchy tree to illustrate the relationship between the logical hierarchy and the physical groups of standard cells.

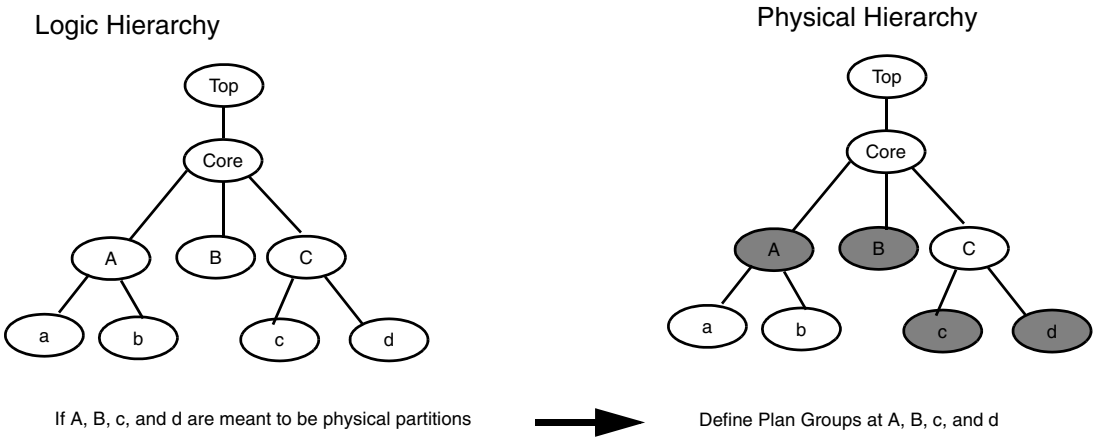
Plan groups require all their cells to be placed inside them and prohibit the placement of other cells in the same area.

In the early stage of the design flow when your design contains the intended plan group partitions, you can analyze the pins and feedthrough pins to help identify issues before you create the pins and fix the pin locations. Using this information, you can redefine route guides or constraints on the router to achieve better pin locations.

During commit hierarchy (`commit_fp_plan_groups` command), the plan groups are converted to soft macro cells, also called a child cell, or simply a physical block, and the pins on each soft macro are cut. (See “Converting Plan Groups to Soft Macros” in Chapter 14.) After the conversion is complete, the floorplan will contain only the top-level standard cells and soft macro cells. Place and route can then work on each soft macro independently. These soft macros are then propagated to the top-level in a FRAM view, and place and route is run on the whole chip design.

You can create plan groups for the logic modules that exist at any level in the logic hierarchy. Figure 7-1 on page 7-3 shows the relationship between the logic hierarchy and the physical hierarchy.

Figure 7-1 Relationship Between Logic Hierarchy and Physical Hierarchy



To create a plan group,

1. Choose Floorplan > Create Plan Group.

   The Create Plan Group dialog box appears.

   Alternatively, you can use the `create_plan_groups` command.

2. Click in the “Hierarchical cell” text box and enter the hierarchical cell names. A plan group is created for each valid hierarchical cell name in the list.

3. Define the plan group shape by doing one of the following:

   - Select the Aspect option to define a plan group shape by its aspect ratio and utilization. This is the default.

     Enter an aspect ratio (height divided by width) for the plan group area. If you specify a ratio of 1.00 (the default), the height and width are the same, and therefore the core is a square. If you specify a ratio of 3.00, for example, the height is three times the width.

     Enter a utilization value. This number is calculated as the total plan group area divided by the area occupied by the cells of the plan group.

   - Select the Dimension option to define a plan group shape by its dimensions and enter the width and height in microns. The plan groups are placed outside the core area.

   - Select the Region option to define a plan group by using coordinates.

     To define a rectangular plan group shape, select the Rectangle button and draw the rectangle in the layout view, or enter the lower-left and upper-right coordinates in the Coordinates field.

     To define a rectilinear plan group shape, select the Rectilinear button and draw the rectilinear shape in the layout view, or enter the lower-left and upper-right coordinates in the Coordinates field.

4. (Optional) You can apply a specific color to a plan group. To do this, deselect the “Cycles colors” option to assign a new color to each new plan group automatically (the default). Select the “Specified” option to specify a particular color for the plan group color from the list. Select “None” for no color

   By default, IC Compiler automatically assigns a color to the plan group.

5. Click OK or Apply.

Removing the Plan Groups

To remove (delete) plan groups from the current design, choose Floorplan > Delete Floorplan Objects > Delete All Plan Groups. This command deletes all the plan groups.

Alternatively, you can use the `remove_plan_groups` command, which lets you selectively delete plan groups.

Note:

Deleting a plan group means cancelling the grouping for future placement. The existing placed cells are not affected.

Adding Padding to Plan Groups

To prevent congestion or DRC violations, you can add padding around plan group boundaries. Plan group padding sets placement blockages on the internal and external edges of the plan group boundary to prevent cells from being placed in the space around the plan group boundaries. Internal padding is equivalent to boundary spacing in the core area. External padding is equivalent to macro padding.

You can pad the plan groups to ensure that the placed standard cells do not extend over the plan group boundary and that cells belonging to the plan group remain inside the plan group. The cells that do not belong to the plan group remain outside the plan group when the physical hierarchy is committed. The boundary space is also needed for pins and the routing to those pins.

You should pad plan groups before refining the placement to ensure that the placement honors the plan group boundaries.

Padding the plan groups also reserves space for the pins on soft macros.

When the physical hierarchy is committed, the padding is transferred to a soft macro cell as core-boundary spacing and not as placement blockages in the child soft macro.

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

To add padding to plan groups,

1. Choose Floorplan > Create Plan Group Padding.

   The Create Plan Group Padding dialog box appears.

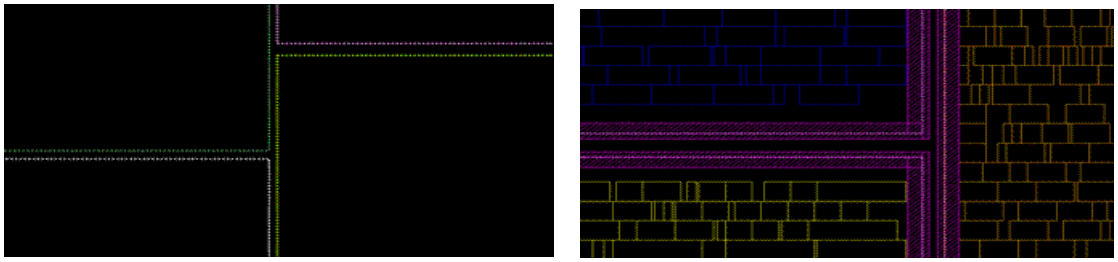
   Alternatively, you can use the `create_fp_plan_group_padding` command.
2. Specify the plan groups for which you want to add padding.
   - All – Select this option to add padding for all plan groups. This is the default.
   - Specified – Select this option to add padding for selected plan groups.

     Enter the names of the plan groups.
3. Specify whether the padding is internal, external, or both.
   - Internal – Select this option to add padding inside the plan group boundaries. Enter a width value. Internal padding should be at least 1 micron (the default) wide to allow space for pins and pin routing.
   - External – Select this option to add padding outside the plan group boundaries. Enter a width value. The default is 0.

   If you want to add both internal and external padding, select the “Internal” and “External” options and enter the width values.
4. Click OK or Apply.

Figure 7-2 shows an example of with and without plan group padding.

Figure 7-2 Example Floorplan With and Without Plan Group Padding



Removing the Plan Group Padding

To remove (delete) both external and internal padding for the plan groups, choose Floorplan > Delete Floorplan Objects > Delete Plan Group Padding. The Delete Plan Group Padding dialog box appears.

Alternatively, you can use the `remove_fp_plan_group_padding` command.

You can delete padding for selected plan groups or for all plan groups. The default is all plan groups.

Adding Block Shielding to Plan Groups or Soft Macros

When two signals are routed parallel to each other, crosstalk can occur between the signals, leading to an unreliable design. You can protect signal integrity by adding modular block shielding to plan groups and soft macros.

The shielding creates rectangular metal layers around the outside of the soft macro boundary or plan group in the top level of the design, and around the inside boundary of the soft macro or plan group, or both.

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

To handle signal integrity when creating hierarchical designs, the router must be prevented from laying down routes that run collinear to the soft macro or plan group boundary. To do this, rectangles that the router creates are placed along and inside the soft macro's or plan group's edges on metal layers whose preferred direction crosses the soft macro or plan group boundary. If the plan groups are committed, the inner-boundary shielding is then transferred to the soft macro.

The rectangles act as hierarchical signal shielding by allowing only routes that are perpendicular to the soft macro or plan group boundary to pass through the metal layers. By preventing collinear routing along the soft macro and plan group boundaries, signal crosstalk among routing in the soft macro and routing in the top-level channels is minimized.

To add block shielding for plan groups or soft macros,

1. Choose Floorplan > Create Module Block Shielding.

   The Create Module Block Shielding dialog box appears.

Alternatively, you can use the `create_fp_block_shielding` command.

2. Select the type of objects to which you want to add signal shielding.

   You can add shielding for all plan groups and soft macros, or you can specify selected plan groups or soft macros. The default is “All”.

3. Select a scope option.

   - Inside – Select this option if you want to create metal route blockages (shielding) around the boundary inside the plan groups or soft macros.
   - Outside – Select this option if you want to create metal route blockages (shielding) around the boundary outside the plan groups or soft macros.
   - Both – Select this option if you want to create metal route blockages (shielding) around the boundary both inside and outside the plan groups or soft macros. This is the default.

4. Specify a shielding width.

   Click in the Width box and a enter a value. The shielding width is determined by multiplying the layer pitch with the value you specify. The default multiplier is 3.0. You can also specify a value in microns.

5. Select the metal layers.

   Specify the metal layer or layers on which the rectangles (shielding) will be created.

6. Specify the sides on which to create the shielding.

7. Click OK or Apply.

Removing Module Block Shielding

You can remove the signal shielding (route blockages) created by modular block shielding. Choose Floorplan > Delete Floorplan Objects > Delete Module Block Shielding. The Delete Module Block Shielding dialog box appears.

Alternatively, you can use the `remove_fp_block_shielding` command.

You can remove shielding rectangles from all plan groups and soft macros or from specified plan groups or soft macros. You can also select the sides as well as the metal layers on which shielding should be removed.

Analyzing and Manipulating the Hierarchy

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

If you are not familiar with a design, you can explore the logic hierarchy to understand its structure and gather information about the design objects. You can also use the hierarchical browser to

- Highlight cells or blocks (and their leaf cells) with different colors in both the hierarchy browser and the layout window
- Perform floorplan tasks such as creating plan groups and estimating black boxes
- Display design schematics of hierarchical cells
- View the hierarchical data interactively and switch between the layout view and the hierarchy browser view for selection and highlighting
- Perform probing between the logical hierarchy tree and the physical layout. This allows you to select cell instances, highlight and unhighlight cells, turn flylines on and off, and display the connectivity of selected modules and plan groups.

This section includes the following topics:

Opening the Hierarchy Browser

To open the hierarchy browser, do one of the following:

- In the layout window, choose Partition > New Hierarchy Browser View
- In the main window, choose Hierarchy > New Hierarchy Browser View

The view window consists of two panes with an instance tree on the left and an object table on the right. The instance name of the top-level design appears at the top of the instance tree.

Note:

After the instance tree is loaded, it exists in memory. If you close the hierarchical browser, and then open it later, the existing instance tree is loaded automatically, which saves you a significant amount of time, particularly if you have a multimillion cell design.

Indicating Object Types

The hierarchy browser indicates object types by displaying icons next to the cell names.

- Logical
  - T – Top-level design
  - BB – Black boxes
  - H – Hierarchical module
  - HM – Hard macro
  - IO – I/O cell
  - SC – Standard cell
  - ILM – Interface logic model
- Physical
  - P – Plan group
  - SM – Soft macro

If you highlight cell hierarchies with colors, the hierarchy browser displays the cell colors on the icons.

Exploring the Hierarchical Structure

You can explore the complete hierarchical structure of your design and observe how many hierarchical blocks are present. To explore the design hierarchy, you can

- Click the expansion button (plus sign) next to an instance name to expand the instance tree, showing the names of the subblocks at the next level of hierarchy.

  You can also right-click and select “Expand Tree Level” from the menu to display the Expand Tree dialog box. You can expand and display the instance tree up to a user-specified level; the root is a level 1, which shows the blocks one level below the current level. A number greater than 1 must be specified in the “Expand to level” field to expand the instance tree and see the subblocks at the next level.

- Select an instance to display leaf cell information in the object table.

  Shift-click or Control-click instance names to select combinations of instances. The information includes cell instance names, cell types, and cell reference names.

- Display port and pin names or net names by selecting a port, pin, or net (rather than a cell) in the field above the object table list on the right.

  You can specify what types of information are displayed in the object table list. Right-click inside the table and then choose Columns > More. You can then select the details you want to display from the objects listed in the dialog box.

Note:
For more information about the hierarchy browser, see IC Compiler online Help.

Manipulating the Hierarchy

This section describes how to create a new level of hierarchy in the early design planning or prototyping phase and how to remove a level of hierarchy from the current design.

This section includes the following topics:

- Merging the Hierarchy
- Flattening the Hierarchy

Merging the Hierarchy

You can select any number of cells in the current design and merge (group) them together into a new cell, thereby creating a new level of logic hierarchy.

To create a new level of hierarchy,

1. Select Partition > Merge Hierarchy.

   The Merge Hierarchy dialog box appears.

   Alternatively, you can use the `merge_fp_hierarchy` command.

2. Set the options, depending on your requirements.

   - New cell name – Enter the name of the new cell instance that replaces the merged cells in the design.
   - New design name – Enter the reference name of the new cell design created by merging the existing cells.

     The name you enter cannot already exist in the current design.

- Cells – Enter a list of cells (modules, plan groups, black boxes, hard macros, soft macros, or leaf cell instances) that you want to merge together into a new level of logic hierarchy.

  The cells you select must be from the same parent and reside at the same level of hierarchy.

- Update SDC – During the creation of the logical module, you might want to update your original Synopsys Design Constraints (SDC) file so that it matches the new logical hierarchy. Select this option to update the SDC file.

  Click in the “Input SDC file” box and enter the name of the original SDC file to be updated to reflect the netlist changes. The file should contain valid SDC commands for the current design.

  Click in the “Output SDC file” box and enter the name of the resulting SDC output file. This file should contain the same information as the original SDC file but with updated top-level timing constraints and new hierarchical names for all the objects in the netlist.

- Load back updated SDC file – Select this option to reload the updated SDC file. The new SDC file is automatically reloaded into the CEL view.

- Create wrapper – Select this option to create a wrapper on any existing leaf cell or hierarchical logic modules. A wrapper is used to selectively expose internal connections and dangling pins onto a wrapper interface so that any future changes made to these internal objects do not impact the wrapper interface.

  The new wrapper module retains the pin names from the logic modules. (Note that if the new pin name on the wrapper is different from the one connected inside the original cell’s pins, a file describing the mapping of the pin names is automatically generated.

  You can also use this option to create a wrapper module for a merged plan group before committing it to a soft macro.

  Note:

  Pin assignment and feedthrough generation occur only at the wrapper module; the original module under the wrapper will not be affected by any feedthrough insertion changes.

  If there are changes in the netlist of the original logic modules, but the module interface is still the same, you can swap the new netlist into the wrapper module, create a new block CEL view, and then link it to the top-level design. You do not have to run pin assignment or generate feedthroughs again for the new netlist.

- Port merging – Deselect this option if you do not want the ports connected by the same interface net to be merged under the wrapper.

3. Click OK or Apply.

The Milkyway hierarchical netlist is updated and a new hierarchical cell is created in the Hierarchical Preservation data.

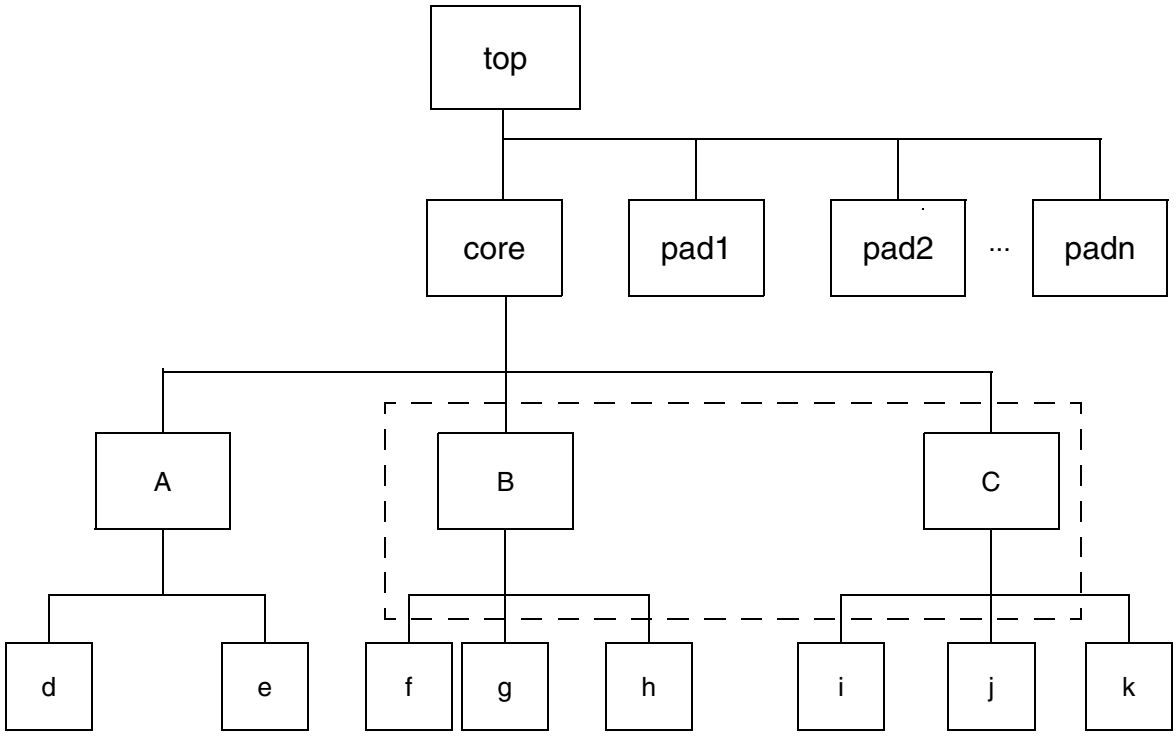
This new logical module can be converted into plan groups, which you can then later commit (`commit_fp_plan_groups`) to soft macros (physical blocks).

Logic Netlist Grouping (Merging) Examples

You can group (merge) modules at the same level of hierarchy and of the same parent in the logic hierarchy. This section has five examples, showing ways that modules can and cannot be grouped.

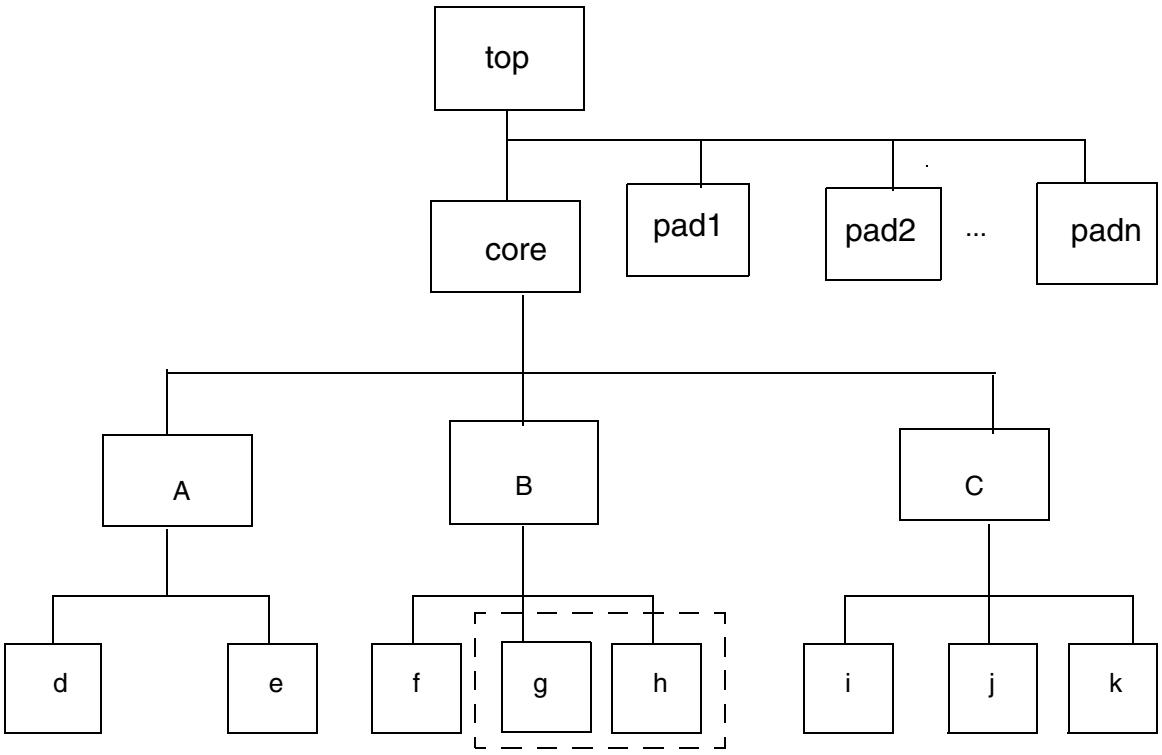
In example 1, shown as Figure 7-3 on page 7-12, standard cells at the same level as A, B, and C are included in the merge.

Figure 7-3 Example 1: Logic Netlist Merging: Same Level of Hierarchy and Same Parent



In example 2, shown as Figure 7-4 on page 7-13, standard cells at the same level as f, g, and h and under B, are included in the merge. Standard cells and hard macros of B and f, are placed at the top level.

Figure 7-4 Example 2: Logic Netlist Merging: Same Level of Hierarchy and Same Parent



The remaining three examples (Figure 7-5, Figure 7-6, and Figure 7-7) show that you cannot merge modules that are in multiple levels of the hierarchy or that do not have the same parent.

Figure 7-5 Example 3: Logic Netlist Merging: Multiple Levels of Hierarchy (Not Possible)

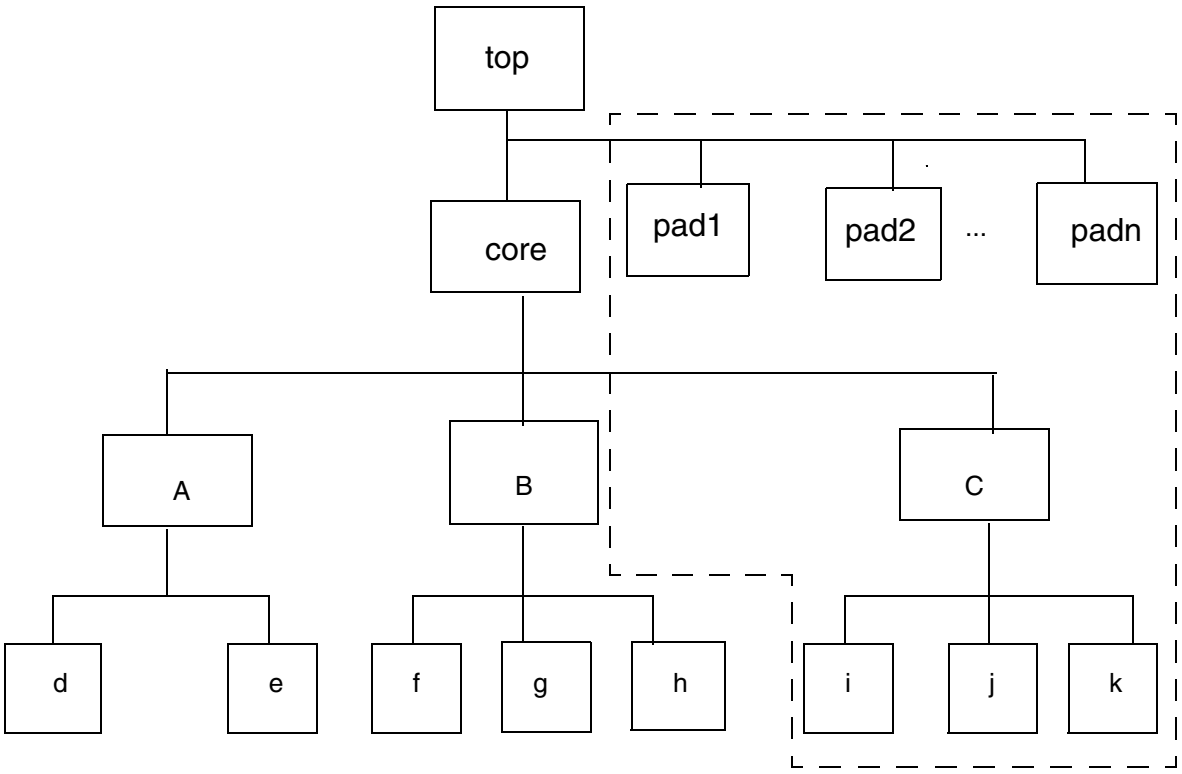


Figure 7-6 Example 4: Logic Netlist Merging: Multiple Levels of Hierarchy and Not the Same Parent (Not Possible)

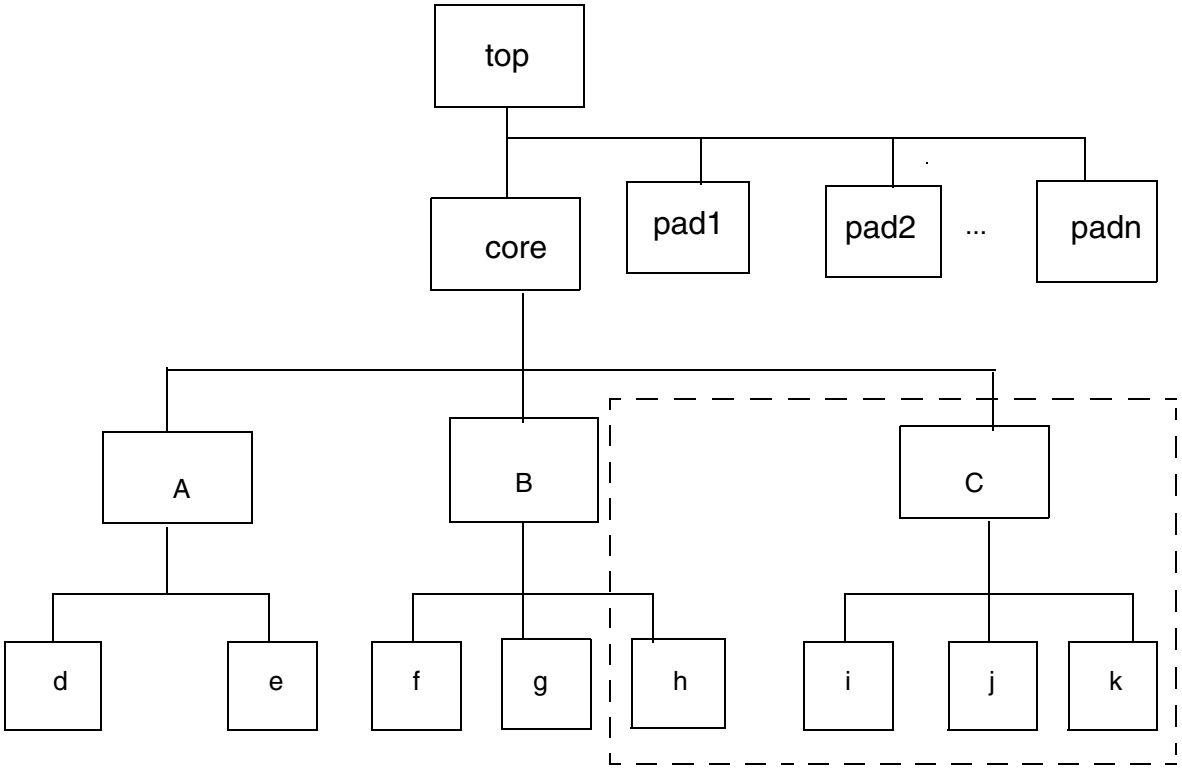
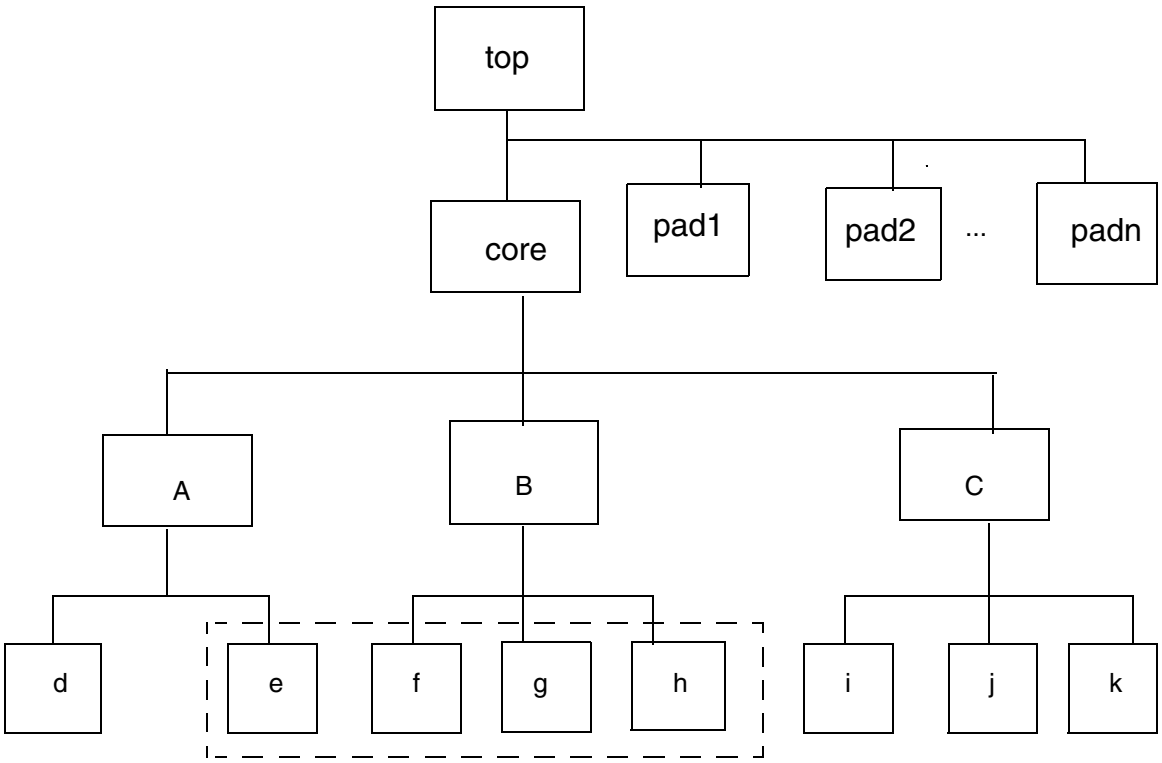


Figure 7-7 Example 5: Logic Netlist Merging: Not the Same Parent (Not Possible)



Flattening the Hierarchy

You can flatten (remove) a single level of hierarchy from your current design so that the logic modules at the next lower level of hierarchy are merged into the current level of hierarchy. This replaces a single hierarchical block with multiple blocks brought up from the lower level, which raises all of those blocks up by one level of hierarchy.

To flatten a level of hierarchy,

1. Select Partition > Flatten Hierarchy.

   The Flatten Hierarchy dialog box appears.

   Alternatively, you can use the `flatten_fp_hierarchy` command.

2. Set the options, depending on your requirements.

   - Cells – Enter a list of cells (modules, plan groups, black boxes, hard macros, soft macros, or leaf cell instances) in the current design that are to be flattened.
   - Update SDC – Select this option to update the SDC file to reflect the changes to the netlist.

Click in the “Input SDC file” box and enter the name of the SDC file to be updated. The file should contain valid SDC commands for the current design.

Click in the “Output SDC file” box and enter the name of the resulting SDC output file into which the updated SDC timing constraints are written.

- Load back updated SDC file – Select this option to reload the updated SDC file. The updated SDC constraints are also automatically reloaded into the CEL view.
- Name Prefix – Select this option to specify the prefix to use when naming flattened cells.

  Simple name – Select this option to indicate that simple, nonhierarchical names are to be used for cells that are flattened. The cells maintain their original names. This is the default.

  No backslash – Select this option to indicate that a backslash should not be added before the hierarchy delimiter.

  Specified prefix – Select this option to specify the prefix to be used when naming flattened cells. The naming style is:

  <specified_prefix><original_cell_name>

3. Click OK or Apply.

Automatically Placing and Shaping Objects in a Design Core

You can automatically place and shape plan group boundaries, black boxes, voltage areas, and other soft macros in a design core.

This section includes the following topics:

- Controlling the Placement and Shaping of Objects
- Using Relative Placement Constraints to Guide the Placement and Shaping of Objects

Plan groups are automatically shaped, sized, and placed inside the core area based on the distribution of cells resulting from the initial virtual flat placement. Blocks (plan groups, voltage areas, and soft macros) marked as fixed (they have the `is_fixed` property set to `true`) remain fixed; the other blocks, whether or not they are inside the core, are subject to being moved or reshaped.

Note:
An initial virtual flat placement must have been previously run on the design.

To automatically place and shape objects in the design core,

1. Choose Placement > Place and Shape Plan Groups.

   The Place and Shape Plan Groups dialog box appears.

   Alternatively, you can use the `shape_fp_blocks` command.

2. Select the remaining options as needed.

   - Prefer rectilinear shapes – If you do not select this option, the `shape_fp_blocks` command creates only rectangular plan groups or soft macros boundaries unless they would overlap fixed objects, in which case the shape might need to be rectilinear if it needs to be cut out around fixed objects.

     If you select this option, the virtual flat placer tries to put the plan group boundary around the set of preplaced cells associated with that plan group. However, in cases where you have one small block and one large block in your design, the small block is placed by one of the corners of the large block, creating a rectilinear L-shaped boundary. L-shaped plan groups can be created even if no fixed objects are encountered on the floorplan.The default is off.

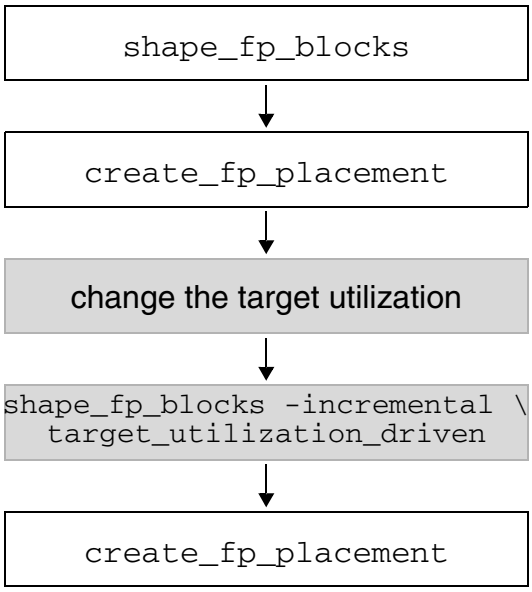
   - Run incrementally – When you select this option, the floorplan is adjusted to meet the new plan group or voltage area target utilizations or to reduce the placement congestion in channels between blocks or inside the blocks themselves. The input is a legal, nonoverlapping floorplan and a congestion map if you select the congestion driven mode. The `shape_fp_blocks` command incrementally resizes and reshapes the blocks (plan groups, voltage areas, and soft macros) in the design, changing the floorplan as little as possible, so that the estimates based on the congestion map are satisfied.

     The run incrementally option runs in two modes: target utilization driven and congestion driven.

     Target utilization driven – If you select this mode, the `shape_fp_blocks` command adjusts the floorplan so that it meets the newly specified target plan group or voltage area utilizations. The command should achieve target utilizations within the `set_fp_shaping_strategy -util_slack` settings. The new floorplan should be similar to the original one. This mode is useful when you are satisfied with the overall locations of the plan groups, but realize that the sizes of the plan groups need to change.

     Figure 7-8 on page 7-19 shows the target utilization flow.

Figure 7-8 Target Utilization Shaping Flow

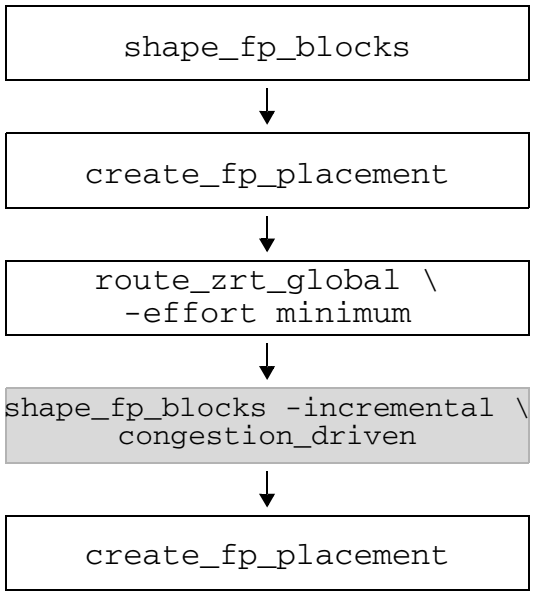


Congestion driven – If the channels between the blocks (plan groups, voltage areas, and soft macros) are too narrow or if the blocks themselves are too small, congestion in the design can often result. If you select this mode, the `shape_fp_blocks` command first attempts to use the congestion map created by global routing. If a congestion map is not available, it tries to use a placement congestion map. If neither of the two congestion maps is available, an error message is generated. After a congestion map exists, you can use it to estimate the required sizes of the channels and blocks that will be needed to help reduce congestion.

The channels and blocks are resized and reshaped, changing the floorplan as little as possible, to reduce congestion in the channels between blocks and inside the blocks themselves so that the estimates based on the congestion map are satisfied. The objective is to reduce congestion by resizing channels as close to an optimal width as possible based on the existing congestion map.

Figure 7-9 on page 7-20 shows the congestion-driven shaping flow.

Figure 7-9 Congestion-Driven Shaping Flow



Occasionally, standard cells move into widened channels which causes additional connections to pass through the channels and therefore, making the initial channel congestion estimates invalid. To limit the instances of this occurring, you can use the `set_fp_shaping_strategy -add_channel_blockages` parameter to automatically control the creation of blockages in the widened channels. You can use hard placement blockages or partial blockages with the congestion density reflecting the cell area that was initially inside the channel.

Note:

Because the placement legalizer does not obey partial blockages, using hard placement blockages in the channels is generally better, but only if there is enough space.

- Create routing channels – If you select this option, routing channels are created between plan groups, black boxes, and soft macros. The channel widths are based on pin counts with nonperpendicular connections to objects on the associated object edge. The default is off.

  If you want channels between the top-level blocks in your design, but the design is not pad limited in size, set the initial core utilization slightly lower than the initial block utilizations. A utilization of 0.65 for the core and a utilization of 0.7 for the blocks is recommended.

- Refine placement afterwards – If you select this option, the placement is refined after the placement and shaping of all unfixed plan groups, soft macros, and black boxes. (This is the equivalent of running the `create_fp_placement` command without any arguments.) The default is off.

- Top-down block placement mode – If you select this option, the command sets variables before it runs so that the placement and shaping of objects is constrained to top-down placement. Initial virtual flat placement is not assumed when you use this option. After the command runs, any variables that it set are restored to their previous settings. The default is off.
- Place sub macros – If you select this option, the command places hard macros in the top-level cell and in all unfixed soft macros. The default is off.
- Block small placement area – If you select this option, placement areas between blockages or fixed objects or both which are smaller than the input threshold value (in microns) are blocked out during shaping. This avoids skinny “fingers” that can occur when there are many blockages or when there are blockages with small channels between each other or the design core.

  Select the “Narrower than (microns)” option and enter a number in microns. By default, the value is -1, which means that the tool automatically calculates a reasonable value for this option.
- Use placement constraint file – If you select this option, you must enter the name of the file that contains the path and name of an ASCII file containing relative placement constraints to guide the placement and shaping of objects.

  The format of the constraint file is one constraint per line. The first word in the line is always the name of the constraint. For example,

  ```
  fplModuleDestRegion plan_group_name region
  ```

  where `region` is one of N, S, E, W, NE, NW, SE, SW, or a coordinate specified as `{x,y}`

  For a description of the relative placement constraints, see “Using Relative Placement Constraints to Guide the Placement and Shaping of Objects” on page 7-24.

3. Click OK or Apply.

Controlling the Placement and Shaping of Objects

You can use the `set_fp_shaping_strategy` command, as shown in Table 7-1, to control how the `shape_fp_blocks` command places and shapes objects in the design core.

Note:
These settings are not saved in the Milkyway design library.

Table 7-1 Controlling the Placement and Shaping of Objects

Option	Usage
`set_fp_shaping_strategy -default`	Sets all the default parameter values.
`set_fp_shaping_strategy -avoid_power_grid on | off`	If you set this parameter to `on`, the plan group edges will avoid the power grid during shaping. The default is `off`. This parameter applies to the `shape_fp_blocks` command without the `-incremental` option. During incremental shaping, it applies only to the edges that the `shape_fp_blocks` command decides to move. The command will not move an edge just to “legalize” its status with respect to the power grid, but it will place all edges it does move into a legal location.
`set_fp_shaping_strategy -distance_to_power_grid` *float*	Specifies the minimum required distance between the plan group boundary and the power grid. If the value is negative, the tool uses one row height as the minimum distance. The default is `-1`, which means the placement will automatically decide the distance between the boundary and the power grid.
`set_fp_shaping_strategy -keep_top_level_together on | off`	If you set this parameter to `on`, it forces the `shape_fp_blocks` command to keep the top-level area of the design contiguous during nonincremental shaping. Keeping the top level contiguous enables the signals to reach all of the top-level area without going inside the plan groups, which is particularly useful if no feedthroughs are allowed. The default is `off`.

Table 7-1 Controlling the Placement and Shaping of Objects (Continued)

Option	Usage
`set_fp_shaping_strategy -min_channel_size` *float*	Specifies the minimum channel size between two plan groups if the `-channels` option is used with `shape_fp_blocks` command. The default is 0, which means plan groups can be abutted.
`set_fp_shaping_strategy -util_slack` *float*	Specifies how much a plan group is allowed to exceed the target utilization during incremental shaping. The default is 0.1, which means the target utilization can be exceeded by 10 percent.
`set_fp_shaping_strategy -add_channel_blockages none | partial | soft | hard`	Controls the types of blockages that are added to the congested channels during incremental shaping. If adding hard or soft blockages will result in over utilization, partial blockages are added instead. The default is `soft`.
`set_fp_shaping_strategy -adjust_macro_locations on | off`	If you set this parameter to `on`, when you run the `shape_fp_blocks -incremental congestion_driven` command, the macros are moved slightly to make sure their placement is legal. The default is `off`.
`set_fp_shaping_strategy -preserve_abutment on | off`	If you set this parameter to `on`, when you run the `shape_fp_blocks -incremental target_utilization_driven` command, the abutment of plan groups is preserved. The default is `off`.
`set_fp_shaping_strategy -max_shape_complexity` *integer*	Controls the maximum number of rectangles that plan group shapes can consist. For example L-shaped plan groups consist of 2 rectangles and U-shaped plan groups consist of 3 rectangles. The default is 0, which means there is no limit.

Using Relative Placement Constraints to Guide the Placement and Shaping of Objects

Relative placement constraints considered in a constraint file for guiding the placement and shaping of objects are described as follows:

- `fplNamedGroup` *`groupName obj`*

  Note:
  Groups must first be defined by using the `fplNamedGroup` constraint.

  If *groupName* is not a name of an existing group, this constraint creates one and places *obj* in it. *Obj* can be a name of a block or another (already created) group. The block is a soft macro, black box, or plan group. Grouping, in the context of these relative constraints, can be viewed as a "pseudo hierarchy" because these are physical groups that do not have to be related to the logic hierarchy of the netlist.

- `fplRel` *`relType obj1 obj2`*

  `relType` (a type of relation)

  Supported types are

  - `BT` (bottom top)–This means *obj1* should be below *obj2*.
  - `LR` (left right)–This means *obj1* is to be placed to the left of *obj2*.

  Care should be taken to ensure consistency of these relative constraints. There is no constraint "conflict resolution" up front when running the `shape_fp_blocks` command. For example, do not use "fplRel BT o1 o2" and "fplRel BT o2 o1" or other, more complicated and conflicting constraint sets. Likewise, constraining some elements of group1 above elements of group2 while placing other elements of group1 below elements of group2, will give unpredictable results. Do not make relationships between an object and its group. Generally, relations should be on the same grouping level.

- `fplModuleDestRegion` *`plan_group_name region`*

  where *region* is one of `{ N | S | E | W | NE | NW | SE | SW}` or an x- and y-coordinate surrounded by braces, for example `{10.700 1200.000}`

  The `fplModuleDestRegion` constraints can be applied to both top level and lower level modules.

Here is an example constraint file:

```
fplNamedGroup red plan_group1
fplNamedGroup red black_box_foo

fplNamedGroup blue plan_group2
fplNamedGroup blue plan_group3
fplNamedGroup green plan_group4
```

```
fplNamedGroup green black_box_bar

fplNamedGroup root red
fplNamedGroup root blue
fplNamedGroup root green

fplRel BT plan_group1 black_box_foo
fplRel LR plan_group3 plan_group2

fplRel BT green blue
fplRel LR red green
```

DFT-Aware Design Planning Flow

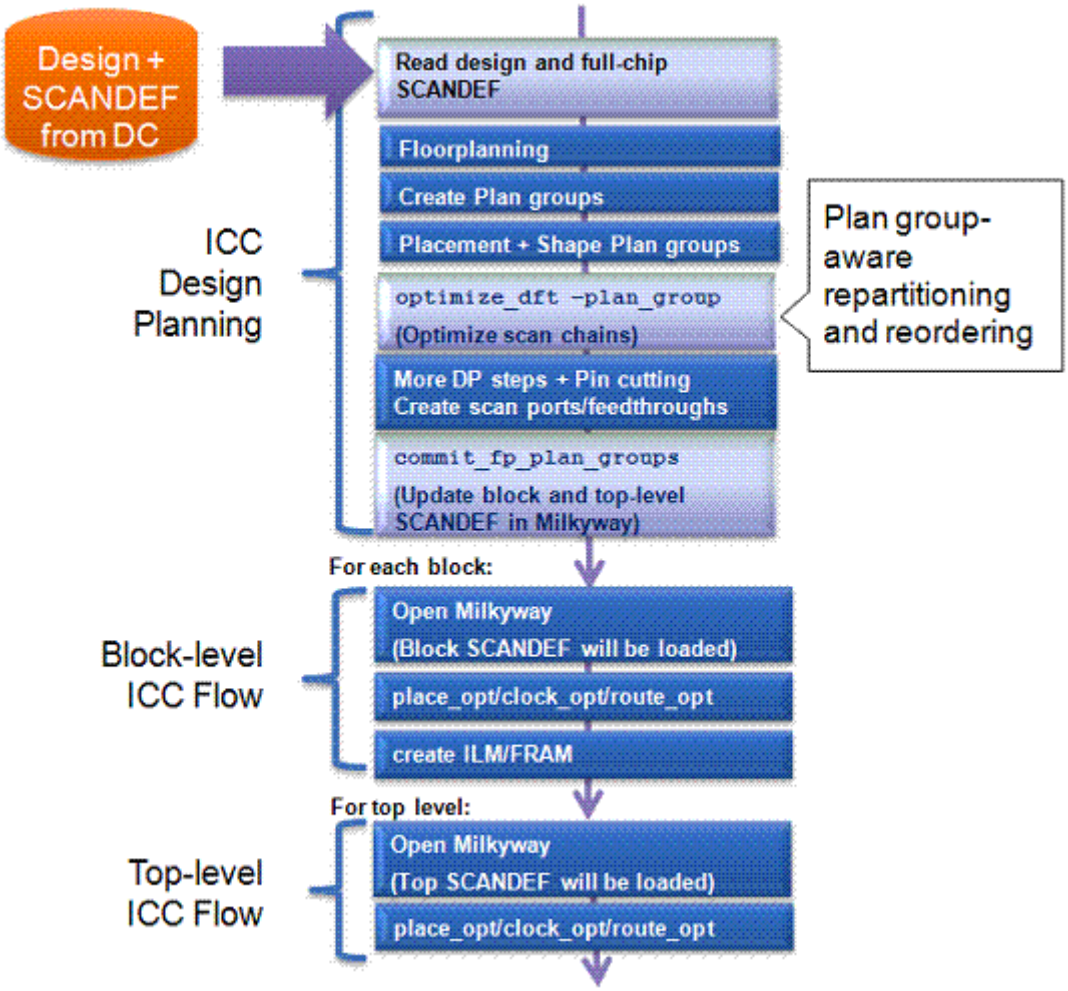
In the IC Compiler design planning flow, you can use the `optimize_dft -plan_group` command to optimize scan chains based on the physical hierarchy after you place and shape the plan groups but before you perform pin cutting. This is useful when logical hierarchy-based scan chains are suboptimal for the best physical design. The plan group-aware scan chain takes advantage of the flexibility of scan chain connections to repartition and reorder the scan cells in a way that reduces the number of scan ports and feedthroughs.

Note:

MIM is not supported together with scan chain `optimize_dft -plan_group`.

Figure 7-10 shows where the `optimize_dft -plan_group` command is used in the DFT-aware design planning flow.

Figure 7-10 DFT-Aware Design Planning Flow



Generating and Using SCANDEF

DFT Compiler uses the `write_scan_def -expand` *`list_of_instances`* command to write the chip-level SCANDEF file with all the scan cells of the subblocks. You can load the full-chip SCANDEF at the beginning of the design planning process in IC Compiler in one of two ways:

- If the design input is .ddc, read the .ddc file with the SCANDEF embedded in it.
- If the design input is Verilog, read the ASCII SCANDEF file, using the `read_def` command.

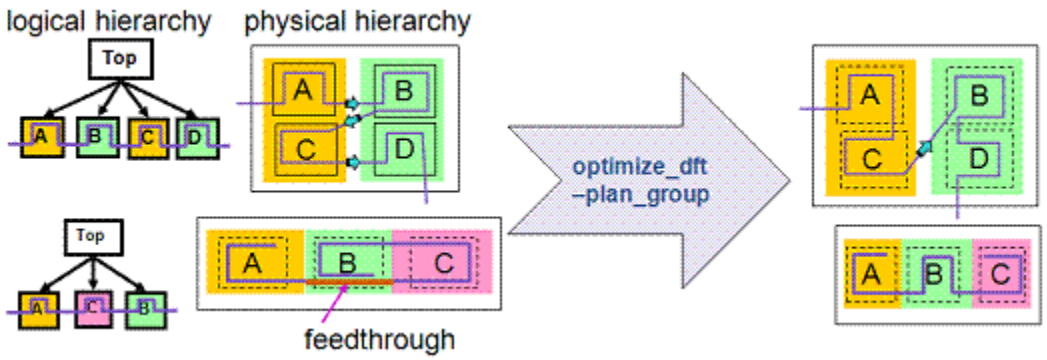
Reading the full-chip SCANDEF information enables scan chain elements to have `size_only` attributes applied. This prevents the scan chain elements from being reoptimized during in-place optimization. As a result, the consistency between the scan chain netlist and SCANDEF is maintained through the design planning stage.

Performing Plan Group-Aware Scan Chain Optimizations

Using the `optimize_dft -plan_group` command performs scan chain repartitioning and reordering. Repartitioning reallocates the scan cells to chains so that scan cells in the same physical hierarchy that are close together form new scan chains. Reordering ensures that the scan chains in the same physical hierarchy are adjacent to other to minimize net lengths that cross physical hierarchies. It provides the following benefits:

- The number of unnecessary top-level scan-chain wires crossing between the physical hierarchies is minimized during design planning.
- Congestion is minimized and routability is improved.
- After scan chain optimizations, the scan cells of the same plan group are clustered together. The entry and exit points of the plan group and the top-level scan wires and ports are minimized, as shown in Figure 7-11.
- An optimal order of the physical hierarchies is determined for each top-level scan chain to minimize excessive feedthroughs created for scan wires, as shown in Figure 7-11.

Figure 7-11 Minimizing Top-Level Scan Wires and Feedthroughs



Using Plan Group-Aware Repartitioning

You can use the following command to control the plan group-aware repartitioning of the scan chains during design planning:

```
set_optimize_dft_options -repartitioning_method \
         none|single_directional|multi_directional|adaptive
```

For more information about this command and options, see the man page.

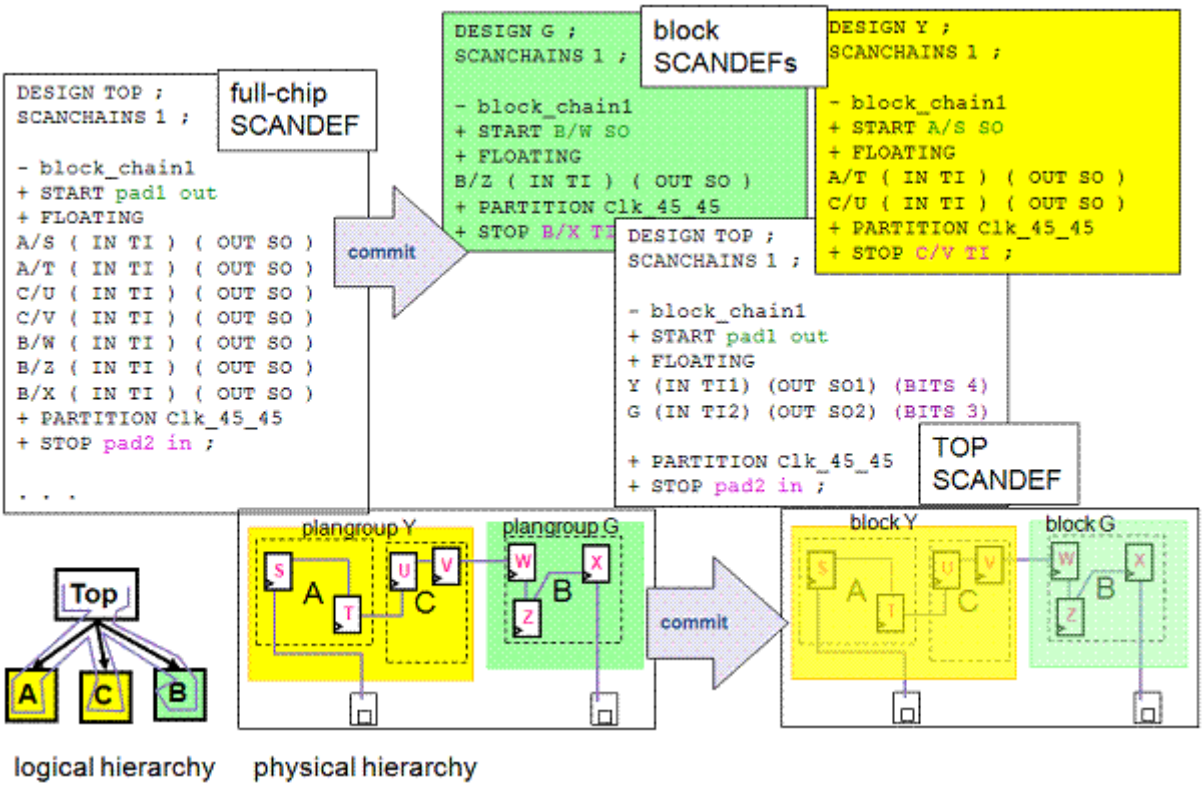
You can turn off repartitioning and perform only plan group-aware scan chain reordering by using the following setting prior to running the `optimize_dft -plan_group` command:

```
set_optimize_dft_options -repartitioning_method none
```

Committing Physical Hierarchy Changes After Scan Chain Optimization

When committing the physical hierarchy changes by using the `commit_fp_plan_groups` command, the SCANDEF information is updated based on the plan groups. The original logical hierarchy-based SCANDEF information is automatically updated to correspond to the physical hierarchy in the design planning phase, and block-level SCANDEF information is pushed down into the hierarchy, and the updated SCANDEF is stored in the Milkyway CEL views for the blocks and top level, based on the physical hierarchy, as shown in Figure 7-12.

Figure 7-12 Updating SCANDEF Data During Commit Hierarchy



When using the `uncommit_fp_soft_macros` command, note that the SCANDEF might be different from the original file. Nevertheless, this SCANDEF is functionally equivalent to the original.

Checking the Consistency of the SCANDEF Data Against the Netlist

IC Compiler maintains the SCANDEF data that describes the scan-chain characteristics and constraints, as well as the scan-stitched netlist. The netlist and SCANDEF data must be consistent with each other. To validate their consistency, you can use the `check_scan_chain` command in both the top level and the blocks. By default, a summary of the consistency of all the chains is displayed.

Consistency checking is performed on elements between the START and STOP points of the chain stubs. The check starts at the STOP point and proceeds backward. Each scan chain stub is given a status after the checks are complete.

8

Performing Power Planning

Power planning, which includes power network synthesis and power network analysis, is required in order to create a design with good power integrity. A design with a robust power and ground (PG) grid mitigates the impact of IR drop and electromigration by providing an adequate number of power and ground pads and rails. As part of the power planning process, power network synthesis generates a power mesh based on a specified IR drop. The power plan can be used to assess the routing resources consumed by the power nets and to determine the impact on routability due to the power plan. You can experiment with different power plans or fine-tune the existing power plan by modifying the replay script and regenerating the power plan.

Power network analysis extracts the resistance characteristics of the power network and performs an IR drop analysis. You can use the analysis results to modify the power plan and reduce the IR drop to meet your specification. IC Compiler also supports In-Design Rail Analysis with PrimeRail; you can use PrimeRail to perform voltage drop analysis, electromigration analysis, inrush current analysis, and other types of analysis on the power network.

This chapter includes the following sections:

- Viewing the Analysis Results
- Running PrimeRail Within IC Compiler
- Performing Power and Ground Routing

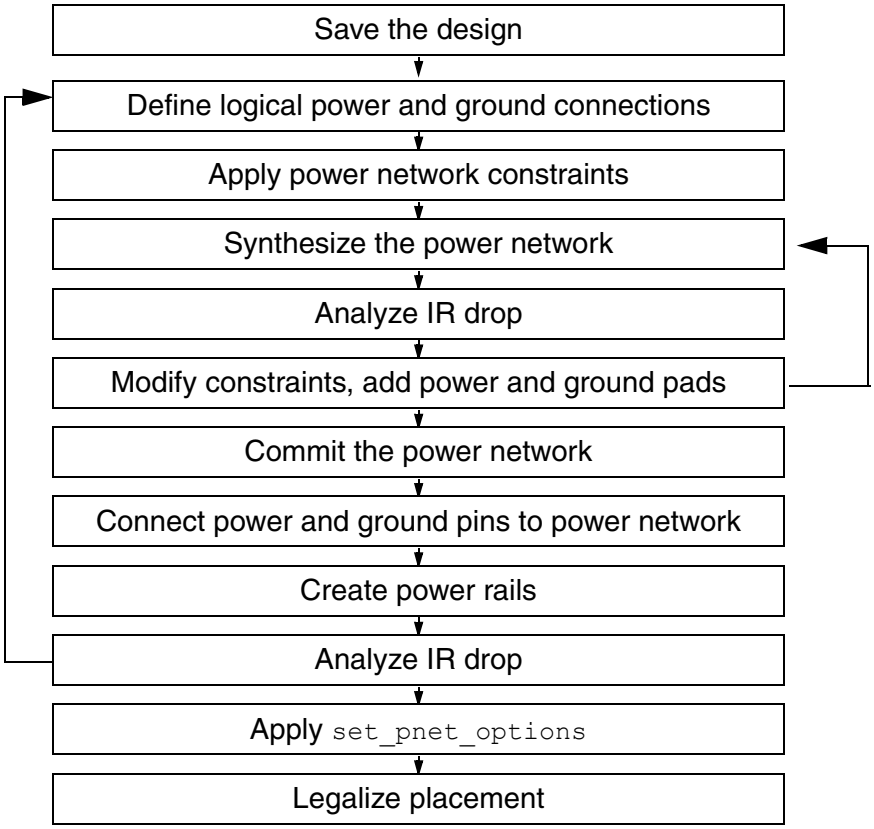
Performing Power Network Synthesis on Single-Voltage Designs

IC Compiler supports different flows for single-voltage designs and multivoltage designs. For information about performing power network synthesis on multivoltage designs, see "Performing Power Network Synthesis on Multivoltage Designs" on page 8-16. The following section outlines the steps to perform power network synthesis for a single-voltage design.

- Saving the Design
- Applying Power Network Synthesis Constraints
- Synthesizing the Power Network
- Creating Virtual Pads
- Committing the Power Plan
- Generating Rails for Standard Cells and Macros
- Legalizing Placement After Adding Power Rails

The following illustration shows a typical power network synthesis flow for single-voltage designs.

Figure 8-1 Power Network Synthesis for Single-Voltage Designs



Saving the Design

Prior to running power network synthesis on your design, you should save the design by using the `save_mw_cel` command. If you are not satisfied with the results of power network synthesis, you can revert to the saved version and resynthesize using different constraints. The following command saves the design prior to power network synthesis.

```
icc_shell> save_mw_cel -as floorplan_prepns
```

Defining Logical Power and Ground Connections

Use the `derive_pg_connection` command to create logical connections between power and ground pins on standard cells and macros and the power and ground nets in the design. For more information about the `derive_pg_connection` command, see “Connecting Power and Ground Ports” on page 2-5.

Applying Power Network Synthesis Constraints

You must specify constraints before synthesizing the power network. The constraints specify the width, direction, spacing, offset, and other parameters the tool uses when creating the rings and straps for the power network. Use the `set_fp_rail_constraints` command or choose one of the GUI forms by choosing Preroute > Power Network Constraints in the GUI and selecting one of the GUI forms to set the constraints.

Table 8-1 describes the constraint settings for power and ground straps; these options can also be set by choosing Preroute > Power Network Constraints > Strap Layer Constraints in the GUI. For more information about these options, see the man page.

Table 8-1 set_fp_rail_constraints Options for Power Straps

Command option	Description
`-add_layer` ("Layer" box in the GUI)	Adds a power strap layer constraint.
`-direction vertical | horizontal` ("Direction" radio button in the GUI)	Specifies the direction for the power strap.
`-layer` *`layername`* ("Layer" box in the GUI)	Specifies the layer to which to add the constraint.
`-max_strap` *`number`* ("Density: By strap number: Max" text box in the GUI)	Specifies the maximum number of power and ground straps. Use the `-min_strap` *`number`* option to specify the minimum number of straps.
`-max_pitch` *`number`* ("Density: By pitch: Max" text box in the GUI)	Specifies maximum routing pitch for the power and ground straps. Use the `-min_pitch` *`number`* option to specify the minimum pitch.
`-max_width` *`distance`* ("Width: Max" text box in the GUI)	Specifies the maximum width of the power straps. Use the `-min_width` *`distance`* option to specify the minimum width.
`-spacing distance | minimum | interleaving` ("PG spacing:" radio button in the GUI)	Constrains the spacing between power and ground wires to allow adequate spacing for signal routing.
`-offset` *`distance`* ("Offset:" text box in the GUI)	Specifies the offset from the I/O pad to the nearest power strap.

The following example uses METAL2 as the vertical power rail and METAL3 as the horizontal power rail. The first command removes the constraints on all layers and then the next two commands constrain the power rails to limit the number of straps between 10 and 20 and sets a minimum width of 0.4 microns.

```
icc_shell> set_fp_rail_constraints -remove_all_layers
icc_shell> set_fp_rail_constraints -add_layer  -layer METAL2 \
  -direction vertical -max_strap 20 -min_strap 10 \
  -min_width 0.4 -spacing minimum
icc_shell> set_fp_rail_constraints -add_layer  -layer METAL3 \
  -direction horizontal -max_strap 20 -min_strap 10 \
  -min_width 0.4 -spacing minimum
```

Table 8-2 describes the `set_fp_rail_constraints` command options that determine the power ring configuration in your design. These constraints specify the layers, ring width, ring spacing, and ring offset for the power ring. The `-extend_strap` option is also described in the following table. Use the `set_fp_rail_constraints` command or choose Preroute > Power Network Constraints > Strap Layer Constraints in the GUI to set the ring constraints. For more information about these options, see the man page.

Table 8-2 set_fp_rail_constraints Options for Power Rings

Command option	Description
`-extend_strap core_ring | boundary | pad_ring` (“Extend straps to” radio buttons in the GUI)	Extends the existing power straps to the core ring, top-level cell boundary, or the existing pad ring.
`-horizontal_ring_layer` *`layer`* (“Horizontal layers” box in the GUI)	Specifies the horizontal metal layers for the power ring. Use the `-vertical_ring_layer` *`layer`* option to specify the vertical metal layers for the power ring.
`-ring_max_width` *`width`* (“Ring width Max” text box in the GUI)	Specifies the maximum permissible core ring width. Use the `-ring_min_width` *`width`* option to specify the minimum allowable core ring width.
`-ring_width` *`width`* (“Ring width Fixed” text box in the GUI)	Specifies the fixed width for the core ring.
`-ring_offset` *`offset`* (“Ring offset to IO or PNS region” text box in the GUI)	Specifies the offset from the power ring to the I/O pad.
`-ring_spacing` *`spacing`* (“Ring spacing” text box in the GUI)	Specifies the spacing between the core rings.

Table 8-2 set_fp_rail_constraints Options for Power Rings (Continued)

Command option	Description
`-set_ring` ("Generate rings" check box in the GUI)	Adds power ring constraints for power ring generation by using the `synthesize_fp_rail` command.
`-skip_ring` (Uncheck the "Generate rings" check box in the GUI)	Disables power ring generation by using the `synthesize_fp_rail` command.
`-nets` *net_names* ("Power Ground nets" text box in the GUI)	Specifies the net names used to construct the power rings.

Table 8-3 describes the `set_fp_rail_constraints` command options that apply globally to the design. These constraints specify the routing strategy for designs that contain plan groups, hard macros, and soft macros. Use the `set_fp_rail_constraints -set_global` command or choose Preroute > Power Network Constraints > Global Constraints in the GUI to set the global constraints. For more information about these options, see the man page.

Table 8-3 set_fp_rail_constraints Global Options

Command option	Description
`-keep_floating_segments` (Uncheck the "Remove floating segment" check box in the GUI)	Retains floating wire segments that are cut by hard macros or blockages.
`-no_same_width_sizing` (Uncheck the "Force same width sizing" check box in the GUI)	Allows the tool to create wires with different widths for power and ground nets.
`-no_stack_via` (Uncheck the "Generate stacked vias" check box in the GUI)	Prevents the tool from creating a via that connects more than two layers.
`-optimize_tracks` ("Optimize tracks usage" check box in the GUI)	Sizes wires to allow sufficient space for signal routes in tracks adjacent to the power wires.
`-ignore_blockages` ("Ignore all the blockages" check box in the GUI)	Allows the tool to create power wires that ignore hard macro blockages.

Table 8-3 set_fp_rail_constraints Global Options (Continued)

Command option	Description
`-no_routing_over_plan_groups` (“No routing over plan groups” check box in the GUI)	Prevents the tool from creating power wires over plan groups, even if no blockages exist.
`-no_routing_over_hard_macros` (“No routing over hard macros” check box in the GUI)	Prevents the tool from creating power wires over hard macros.
`-no_routing_over_soft_macros` (“No routing over soft macros” check box in the GUI)	Prevents the tool from creating power wires over soft macros.
`-keep_ring_outside_core` (“Keep rings outside of core area” check box in the GUI)	Prevents the tool from creating the power ring within the core area.

The `set_fp_block_ring_constraints` command defines the constraints for the power and ground rings that are created around plan groups and macros by using the `synthesize_fp_rail` command. Table 8-4 describes the `set_fp_block_ring_constraints` command options; you can also choose Preroute > Multi-Voltage Power Network Constraints > Block Rings Constraints in the GUI to set the constraints. For more information about these options, see the man page.

Table 8-4 set_fp_block_ring_constraints Command Options

Command option	Description
`-add` (“Set” button in the GUI)	Adds or updates the block ring constraints.
`-all_blocks` (“All plan groups”, “All voltage areas”, and “All cells” radio buttons in the GUI)	Selects all blocks of the specified type for ring creation.
`-block` *`blocks`* (“Specified plan groups” text box in the GUI)	Specifies the blocks around which the tool creates the ring.

Table 8-4 set_fp_block_ring_constraints Command Options (Continued)

Command option	Description
`-block_type master | instance | plan_group | voltage_area` (“Specified plan groups”, “Specified voltage areas”, “Specified cell instances”, and “Cell masters” radio buttons in the GUI)	Specifies the type of block around which to create the ring.
`-horizontal_layer` *`layer`* (“Horizontal layer” box in the GUI)	Specifies the horizontal layer used to create the block ring. The `-vertical_layer` *`layer`* option specifies the vertical layer used to create the block ring.
`-horizontal_offset` *`distance`* (“Offset” text box for “Horizontal layer” in the GUI)	Specifies the horizontal offset from the block boundary to the block ring. The `-vertical_offset` *`distance`* option specifies the vertical offset from the block boundary to the block ring.
`-horizontal_width` *`distance`* (“Width” text box for “Horizontal layer” in the GUI)	Specifies the width for the horizontal wires for the block ring. The `-vertical_width` *`distance`* option specifies the width for the vertical wires for the block ring.
`-load_file` *`file_name`* (“Load” text box in the GUI)	Loads the block ring constraints from the specified file.
`-nets` *`nets`* (“Power Ground nets” text box in the GUI)	Specifies the power and ground nets used for the generated rings.
`-remove` (“Remove” button in the GUI)	Removes the selected or specified constraint.
`-remove_all` (“Remove All” button in the GUI)	Removes all block ring constraints.
`-save_file` *`filename`* (“Save” button in the GUI)	Saves the current block ring constraints to the specified file.
`-spacing` *`distance`* (“Spacing between block ring nets” text box in the GUI)	Specifies the spacing between block rings.

The following example adds a ring constraint for the current design. The horizontal layer is constrained to the METAL5 layer with a width of 3 microns and an offset of 0.600 microns. The vertical layer is constrained to the METAL6 layer with a width of 3 microns and an offset of 0.600 microns. The ring constraint is applied around all instances with the cell names ALU, CPU, or RAM using the VDD and VSS nets.

```
icc_shell> set_fp_block_ring_constraints -add \
 -horizontal_layer METAL5 -vertical_layer METAL6 \
 -horizontal_width 3 -vertical_width 3 \
 -horizontal_offset 0.600 -vertical_offset 0.600 \
 -block_type master -nets {VDD VSS} \
 -block { ALU CPU RAM }
```

Synthesizing the Power Network

You can perform power network synthesis on your design by using the `synthesize_fp_rail` command. This command synthesizes the power network based on the constraints you define. To create the power network, use the `synthesize_fp_rail` command or choose Preroute > Synthesize Power Network in the GUI. Table 8-5 describes the `synthesize_fp_rail` command options. For more information about these options, see the man page.

Table 8-5 synthesize_fp_rail Command Options

Command option	Description
`-analyze_power` (“Analyze power” radio button in the GUI)	Performs power analysis on each specified power net.
`-create_virtual_rails` *layer* (“Create virtual rails” check box in the GUI)	Creates pseudo straps on the specified metal layer. The straps are not added to design after commit. This option is useful for designs that do not contain standard cell rails.
`-honor_conn_view_layers` *conn_layer* (“Honor CONN view layers” check box in the GUI)	Uses the connectivity view on the specified metal layers during extraction.
`-ignore_blockages` (“Ignore blockages for virtual pads” radio button in the GUI)	Ignores blockages when connecting virtual pads to the power network.
`-lowest_voltage_drop` (“Lowest” radio button in the “Target IR drop” section of the GUI)	Generates a power network with the lowest possible IR drop consistent with the power network constraints.

Table 8-5 synthesize_fp_rail Command Options (Continued)

Command option	Description
`-nets` *nets* ("Synthesize power network by nets" box in the GUI)	Specifies the nets on which to perform power network synthesis.
`-output_directory` *directory_name* ("Output directory" directory selection box in the GUI)	Specifies the output directory used to save the analysis result files.
`-pad_masters` *pad_reference_cells* ("Specified pad masters" check box in the GUI)	Specifies the power pad cells and their associated nets.
`-power_budget` *power* ("Power budget" text box in the GUI)	Specifies the power budget for the entire design.
`-read_default_power_file` *power_file_name* ("Power info" check box and "Input power consumption file" file selection box in the GUI)	Assigns power budgets to instances based on the specification in the *power_file_name* file.
`-read_pad_instance_file` *pad_instance_file_name* ("Instances" radio button and file selection box in the GUI)	Assigns pads to power and ground nets based on the specification in the *pad_instance_file_name* file.
`-read_pad_master_file` *pad_reference_cell_file_name* ("Masters" radio button and file selection box in the GUI)	Assigns pads to power and ground nets based on the specification in the *pad_reference_cell_file_name* file.
`-read_power_compiler_file` *power_compiler_report_file*	Assigns power to each instance based on the specified Power Compiler report file generated with the `report_power` command.
`-read_prime_power_file` *prime_power_report_file*	Assigns power to each instance based on the specified PrimePower or PrimeTime PX report file.
`-synthesize_power_pads` ("Synthesize power pads" check box in the GUI)	Synthesizes power pads in the design. This option is useful for power network synthesis on a top-level design without existing power pads.

Table 8-5 synthesize_fp_rail Command Options (Continued)

Command option	Description
`-synthesize_power_plan` ("Synthesize power plan" check box in the GUI)	Performs power network synthesis on the design.
`-target_voltage_drop` *`target_voltage`* ("Specified (mV)" radio button and text box in the GUI)	Specifies the target IR drop for the power network.
`-top_level_only` ("Top level cells only" radio button in the GUI)	Ignores cell instances inside soft macros during power network synthesis.
`-use_pins_as_pads` ("Use top level design pins as pads" check box in the GUI)	Treats power and ground pins on the periphery as pads.
`-use_strap_ends_as_pads` ("Use strap ends as pads" check box in the GUI)	Uses the ends of straps at the design boundary or core ring as power pads. This option is useful for power network synthesis on blocks without existing power or ground pads.
`-voltage_supply` *`voltage`* ("Supply voltage" text box in the GUI)	Specifies the supply voltage for single-voltage designs.

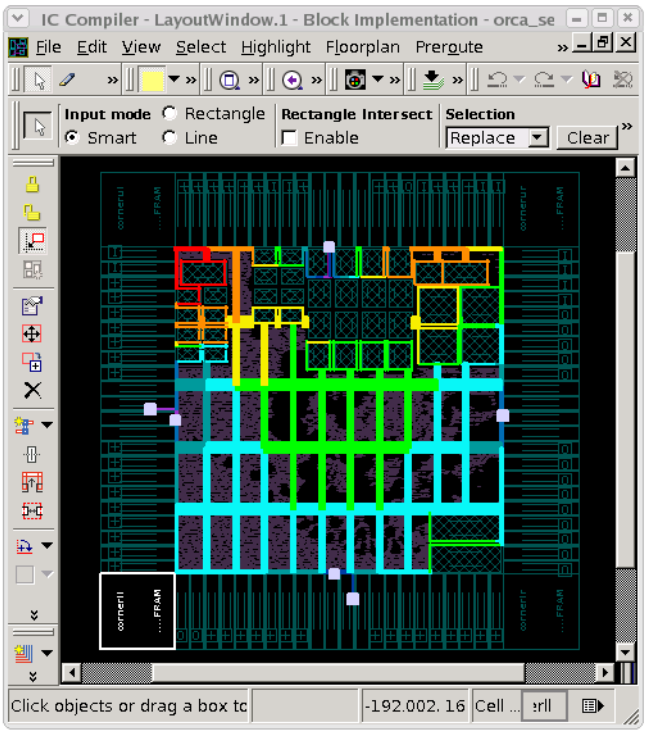
The following example synthesizes the power plan based on the constraints set previously by using the `set_fp_rail_constraints` command. The tool synthesizes a power plan using the VDd power net, the VSS ground net, and a supply voltage of 1.32V. The command writes the voltage drop and electromigration results to the *powerplan.dir* directory.

```
icc_shell> synthesize_fp_rail -power_budget 800 \
  -voltage_supply 1.32 \
  -output_directory powerplan.dir \
  -nets {VDD VSS} \
  -synthesize_power_plan
```

The `synthesize_fp_rail` command generates a sample power plan based on the constraints you provide. Rail analysis results for the design are saved to the ./pna_output directory; you can use these results to examine the voltage drop, resistance, and electromigration and perform other analyses on the power network. See "Analyzing the Power Network" on page 8-33 for more information. If the target IR drop is greater than your specification, change the power network constraints by using the

`set_fp_rail_constraints` and `set_fp_block_ring` commands. Figure 8-2 on page 8-13 shows the sample power plan created by running the `synthesize_fp_rail` command.

Figure 8-2 Power Network Synthesis Results Display in IC Compiler



Creating Virtual Pads

You can create virtual pads to serve as temporary power sources for your design. Virtual pads provide additional sources of current for the power network and do not require modification of the floorplan. After experimenting with power network analysis using different placements and numbers of virtual pads, you can modify your floorplan and insert additional power pads based on your analysis.

To insert virtual pads into your design, use the `create_fp_virtual_pad` command or choose Preroute > Create Virtual Power Pad in the GUI. Table 8-6 describes the `create_fp_virtual_pad` command options. For more information about these options, see the man page.

Table 8-6 create_fp_virtual_pad Command Options

Command option	Description
`-nets` *net_name* (“Power or Ground net” box in the GUI)	Specifies the net name of the net to attach the virtual pad.
`-layer` *layername* (“Specified” box in Layer area in the GUI)	Specifies the layer used to create the virtual pad.
`-point` *{x y}* (“Coordinates” text box in the GUI)	Specifies the x- and y-locations for the virtual pad.
`-load_file` *filename* (“Load” text box in the GUI)	Specifies the file name containing the list of virtual pads.
`-save_file` *filename* (“Save” text box in the GUI)	Specifies the file name used to write the location and net connections for the virtual pads.

The following example creates two virtual pads for the VDD net at coordinates 100,100 and 2000,2000.

```
icc_shell> create_fp_virtual_pad -nets VDD -point {100.000 100.000}
icc_shell> create_fp_virtual_pad -nets VDD -point {2000.000 2000.000}
```

You can save the current virtual pads to a file by using the `-save` option. The following example saves the current virtual pads to the *pads.rpt* file and displays the content of the file. To re-create the virtual pads, use the `create_fp_virtual_pad` command with the `-load_file` option.

```
icc_shell> create_fp_virtual_pad -save_file pads.rpt
Saving the virtual pad file pads.rpt successfully

icc_shell> exec cat pads.rpt
VDD
100.000 100.000 auto
2000.000 2000.000 auto
```

To remove the virtual pads created by using the `create_fp_virtual_pad` command, use the `remove_fp_virtual_pad` command. The `remove_fp_virtual_pad -all` command removes all virtual pads; you can also use the `remove_fp_virtual_pad` command with the `-net` and `-point` options to remove individual virtual pads. The following command removes all virtual pads from the design.

Committing the Power Plan

After creating a satisfactory power plan that meets your specifications for IR drop, you can commit the power plan to convert the virtual power straps and rings to actual power wires, ground wires, and vias. Use the `commit_fp_rail` command or click the Commit button on the Preroute > Synthesize Power Network form in the GUI.

Generating Rails for Standard Cells and Macros

The `synthesize_fp_rail` command synthesizes a power and ground network, but does not generate rails for standard cells nor connect macros to the power network. To create a more accurate power plan, you must create these rails and connections. Use the `preroute_instances` command, or select Preroute > Preroute Instances in the GUI, to connect instance pins on macros to the power and ground network. Use the `preroute_standard_cells` command, or select Preroute > Preroute Standard Cells in the GUI, to add standard cell power rails and connect standard cell power pins to the rail. For more information about these commands, see the man pages.

You can use the `set_preroute_special_rules` command to create named rules that are used when creating the macro connections with the `preroute_instances` command. See the man page for `set_preroute_special_rules` for more information.

The following example uses the `preroute_instances` command to connect macro pins to the power network and uses the `preroute_standard_cells` command to create power rails for the standard cells.

```
icc_shell> preroute_instances
icc_shell> preroute_standard_cells
```

After adding the power rails for the standard cells and connecting the macro pins, you should analyze the power plan to determine the new maximum IR drop for the design. To analyze the power network, use the `analyze_fp_rail` command or choose Preroute > Analyze Power Network in the GUI. The following command performs power network analysis on the VDD and VSS nets using a supply voltage of 1.32V and a power budget of 760 mW.

```
icc_shell> analyze_fp_rail -analyze_power -nets {VDD VSS} \
  -voltage_supply 1.32 -power_budget 760
```

For more information about analyzing the power network, see “Analyzing the Power Network” on page 8-33.

Legalizing Placement After Adding Power Rails

The `synthesize_fp_rail` and `preroute_standard_cells` commands generate power and ground straps; these commands might create shorts between the power rails and standard cells. Use the `set_pnet_options` command to enable checking for shorts on the new metal layer straps, and use the `legalize_fp_placement` command to shift the violating cells away from the power straps. The following example uses the `set_pnet_options -partial` command to enable a check for shorts between the standard cells and the power nets on the METAL2 and METAL3 layers. The `legalize_fp_placement` command moves the standard cells with shorts.

```
icc_shell> set_pnet_options -partial "METAL2 METAL3"
icc_shell> legalize_fp_placement
```

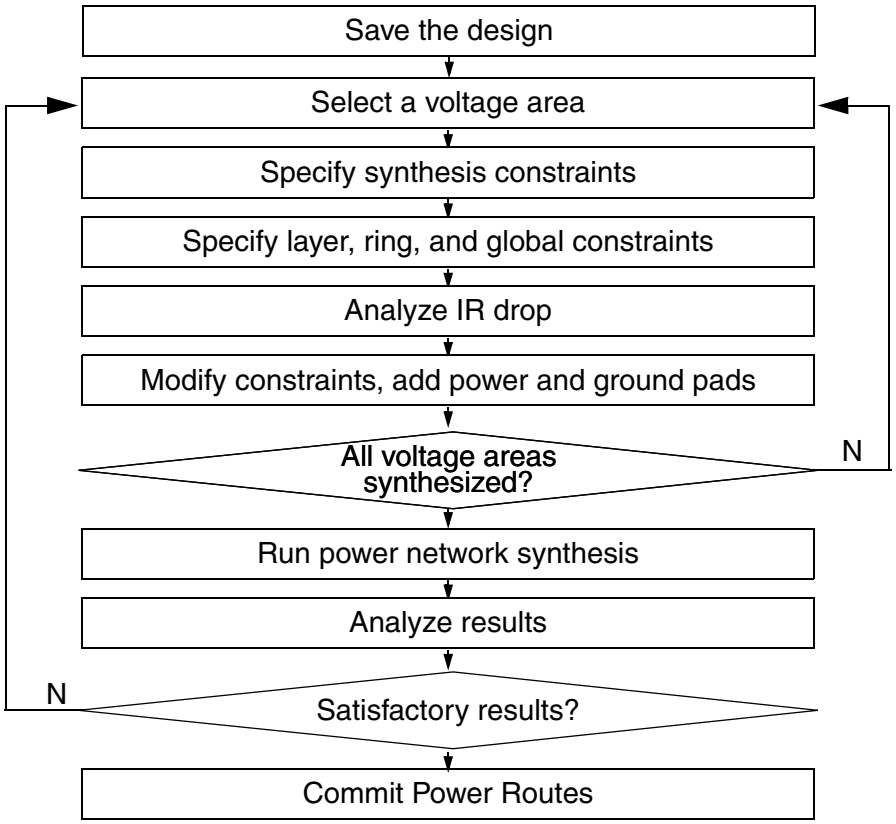
Performing Power Network Synthesis on Multivoltage Designs

The following section outlines the steps necessary to implement a power network in a multivoltage design.

- Saving the Design
- Selecting a Voltage Area and Specifying Synthesis Constraints
- Setting Layer Constraints
- Setting Ring Constraints
- Setting Global Constraints
- Synthesizing the Power Network

The following illustration shows a typical power network synthesis flow for a multivoltage design.

Figure 8-3 Multivoltage Power Network Synthesis Flow



Saving the Design

Prior to running power network synthesis on your design, you should save the design by using the `save_mw_cel` command. If you are not satisfied with the results of power network synthesis, you can revert to the saved version and resynthesize using different constraints. The following command saves the design prior to power network synthesis.

```
icc_shell> save_mw_cel -as floorplan_prepns
```

Selecting a Voltage Area and Specifying Synthesis Constraints

Begin power network synthesis by selecting a voltage area and setting synthesis constraints on that area. You must set synthesis constraints before setting the layer, ring, and global constraints on the voltage area. Synthesis constraints include settings for the power and ground net names, supply voltage, power budget, target IR drop, and power-switch name. Use the `set_fp_rail_voltage_area_constraints` command, or choose Preroute > Multi-Voltage Power Network Constraints > Voltage Area Constraints in the GUI to select the

voltage area and set the constraints. Table 8-7 describes the `set_fp_rail_voltage_area_constraints` command options used to set synthesis constraints for a voltage area.

Table 8-7 set_fp_rail_voltage_area_constraints Command Options for Power Network Synthesis

Command option	Description
`-voltage_area` *areaname* ("Voltage Area" selection box in the GUI)	Specifies the name of the voltage area.
`-nets` *net_names* ("Power Ground nets" text box in the GUI)	Specifies the power and ground net names.
`-voltage_supply` *voltage* ("Supply Voltage" text box in the GUI)	Specifies the supply voltage in volts for this voltage area.
`-power_budget` *power* ("Power Budget" text box in the GUI)	Specifies the power budget in mW for the voltage area.
`-target_voltage_drop` *target* ("Target IR drop" box in the GUI)	Specifies the target voltage drop in mV for the voltage area.
`-power_switch` *switchname* ("Power switch name" in the GUI)	Specifies the name of the power switch for this voltage area.

Note:

To set the constraint for a power-down voltage area, you must specify the name of the permanent power net followed by the name of the virtual power net. For example, use the `set_fp_rail_voltage_area_constraints -voltage_area VA1 -nets VDD+VDD_Virtual` command to assign a constraint on the VDD net for the power-down voltage area named VA1. IC Compiler synthesizes both nets connected by power-switch cells.

For a UPF design, the tool inserts the power and ground net names automatically, although you must provide supply voltage information. If you do not set the power budget, power network synthesis assigns a power budget based on the ratio of the instance area for this voltage area divided by the area of the entire design.

The following example sets a power network synthesis constraint on the VA1 voltage area for the VDD and VSS power nets using a supply voltage of 1.5 V, a power budget of 200 mW, a target IR drop of 160 mV, and a power switch named SWITCH1.

```
icc_shell> set_fp_rail_voltage_area_constraints -voltage_area VA1 \
  -power_budget 200 -power_switch SWITCH1 -voltage_supply 1.5 \
  -target_voltage_drop 160 -nets  {VDD VSS}
```

Setting Layer Constraints

To assign layer constraints for a voltage area, use the `set_fp_rail_voltage_area_constraints` command or choose Preroute > Multi-Voltage Power Network Constraints > Voltage Areas Constraints in the GUI and click the Layer constraints button. The layer constraints define the layer, direction, strap width, strap spacing or density, and net type for the voltage area. Table 8-8 describes the options for the `set_fp_rail_voltage_area_constraints` command.

Table 8-8 set_fp_rail_voltage_area_constraints Options for Layer Constraints

Command option	Description
`-layer` *layer* (“Layer” text box in the GUI)	Specifies the layer to which to apply the constraint.
`-direction` *direction* (“Direction” radio buttons in the GUI)	Specifies the routing direction, horizontal or vertical, for the layer.
`-max_strap` *number* (“Density,By strap number,Max” text box in the GUI)	Specifies the maximum number of power and ground straps in the voltage area. Use the `-min_strap` option to specify the minimum number of straps.
`-max_pitch` *distance* (“Density,By pitch,Max” text box in the GUI)	Specifies the maximum pitch of the power and ground straps in the voltage area. Use the `-min_pitch` option to specify the minimum pitch.
`-max_width` *distance* (“Width,Max” text box in the GUI)	Specifies the maximum width of the power and ground straps in the voltage area. Use the `-min_width` option to specify the minimum strap width.
`-spacing minimum | interleaving | distance` (“PG spacing” radio buttons in the GUI)	Specifies the spacing between power and ground straps in the specified voltage area.
`-offset` *offset* (“Offset” text box in the GUI)	Specifies the distance from the voltage area boundary to the left or bottom power strap.

Table 8-8 set_fp_rail_voltage_area_constraints Options for Layer Constraints (Continued)

Command option	Description
`-mtcmos_net_type permanent | virtual` (“Net type” check box and radio buttons in the GUI)	Specifies the power switch array synthesis net type, permanent or virtual, for the layer.

The following example specifies a power network layer constraint on the VA1 voltage area for layer METAL6. The layer direction is vertical, the maximum number of power straps is 128, the minimum number is 4, the minimum width is 0.1 microns, and the spacing is minimum.

```
icc_shell> set_fp_rail_voltage_area_constraints -voltage_area VA1 \
  -layer METAL6 -direction vertical -max_strap 128 -min_strap 4 \
  -min_width 0.1 -spacing minimum
```

Setting Ring Constraints

Ring constraints specify the layer, ring width, ring spacing, and strap extension parameters used when creating power and ground rings around a voltage area. To assign ring constraints for a voltage area, use the `set_fp_rail_voltage_area_constraints` command with the appropriate options or choose Preroute > Multi-Voltage Power Network Constraints > Voltage Areas Constraints in the GUI and click the Ring and Straps constraints button. Table 8-9 describes the command options for the `set_fp_rail_voltage_area_constraints` command used to create power rings around voltage areas.

Table 8-9 set_fp_rail_voltage_area_constraints Options for Ring Constraints

Command option	Description
`-ring_nets` *net_names* (“Power Ground nets” in the GUI)	Specifies the power and ground net names for the power ring.
`-horizontal_ring_layer` *layers* (“Horizontal layers” box in the GUI)	Specifies the layers to use for horizontal routes for power and ground nets. Use the `-vertical_ring_layer` option to specify the layers for vertical routes for power and ground nets.
`-ring_width` *width* (“Ring width, Fixed” text box in the GUI)	Specifies the power and ground ring width. Use the `-ring_max_width` and `-ring_min_width` options to specify a maximum and minimum width.

Table 8-9 set_fp_rail_voltage_area_constraints Options for Ring Constraints (Continued)

Command option	Description
`-ring_spacing` *spacing* (“Ring spacing” text box in the GUI)	Specifies the distance between the power and ground rings in the voltage area.
`-ring_offset` *offset* (“Ring offset to IO or PNS region” text box in the GUI)	Specifies the distance from the voltage area boundary to the closest ring net.
`-extend_strap voltage_area_ring | core_ring | boundary` (“Extend straps to” radio buttons in the GUI)	Specifies how to extend power and ground nets in the voltage area.
`-num_extend_straps` *number* (“Number of straps ...” text box in the GUI)	Specifies the number of power straps extended from each side of the voltage area to the core ring.
`-extend_strap_direction {left right top bottom all}` (“Direction for extending straps” check boxes in the GUI)	Specifies the directions in which to extend the straps from the voltage area to the core ring or boundary.

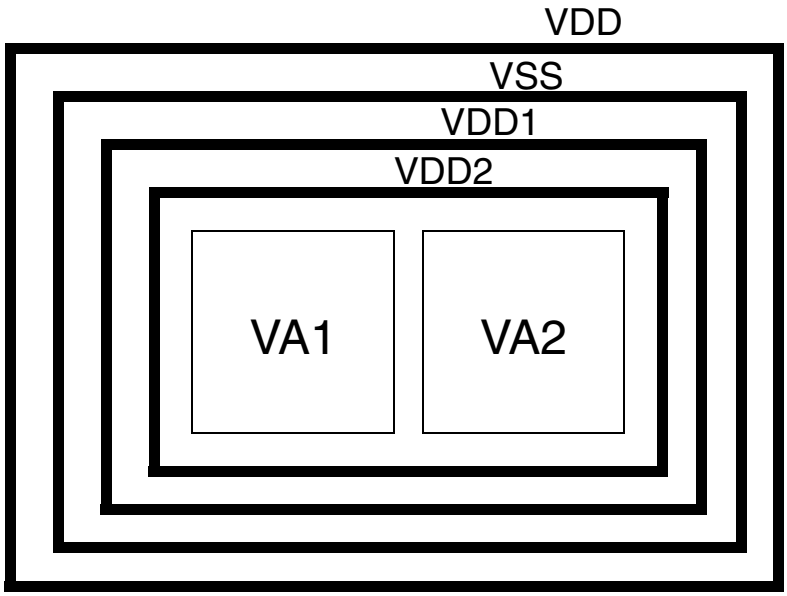
Note:

You can define constraints for the core ring only for the default voltage area. To generate the core ring, select the default voltage area in the Voltage Area selection box, select the “Generate core rings” check box in the GUI, and specify the ring power and ground net names in the Power Ground nets text box. You can generate a permanent ring in the default voltage area.

Figure 8-4 shows a set voltage rings created by using the following `set_fp_rail_voltage_area_constraints` command:

```
icc_shell> set_fp_rail_voltage_area_constraints \
  -voltage_area DEFAULT_VA -ring_nets  {VDD VSS VDD1 VDD2} \
  -horizontal_ring_layer { M9 } -vertical_ring_layer { M8 }
```

Figure 8-4 Core Rings for Default Voltage Area



The tool supports different techniques for extending power straps from the voltage area. The `-extend_strap voltage_area_ring` option (“Voltage area ring” radio button in the GUI) extends the straps to the power and ground ring for the voltage area. The `-extend_strap core_ring` option (“Core ring” radio button in the GUI) extends the power and ground straps to the power ring for the default voltage area. The `-extend_strap boundary` option extends the straps to the top-level cell boundary. The `-extend_strap pad_ring` option extends the straps to the top-level pad ring, and the `-extend_strap voltage_area_boundary` option extends the straps to the boundary of the voltage area set by using the `create_voltage_area -coordinate` command.

You can also control the direction used for extending the straps. Set the direction for power rail extension by using the `-extend_strap_direction` option or by selecting the appropriate check boxes in the “Direction for extending straps” section of the GUI. By default, the tool extends the power straps from all sides of the voltage area, unless they are blocked by a constraint, a hard marco, or another voltage area.

The ground net is not extended out from each voltage area to the core ring because the ground net is synthesized as a common net in the design. As a result, the ground net in each voltage area is extended to the nearest ground straps outside the voltage area boundary.

Setting Global Constraints

Define global constraints by using the `-global` option with the `set_fp_rail_voltage_area_constraints` command, or by clicking the Global constraints button in the GUI. The list of global constraint options is the same as the list for the `set_fp_rail_constraints` command; see Table 8-3 on page 8-7 for a description of the options used to set global constraints.

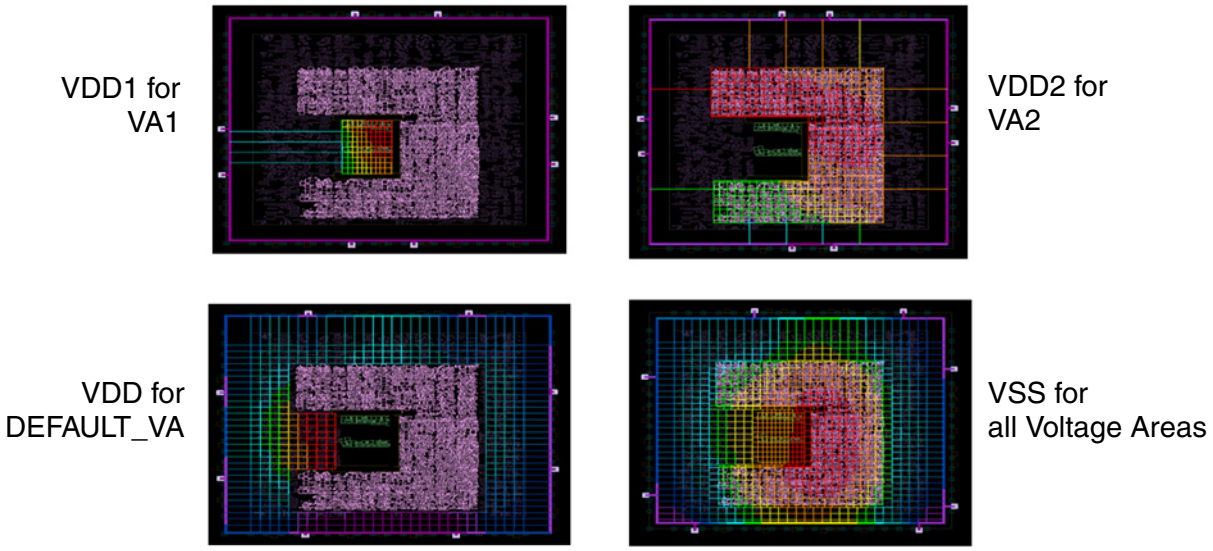
Synthesizing the Power Network

After you set the power network constraints by using the `set_fp_rail_voltage_area_constraints` command, you can perform power network synthesis on your multivoltage design. The tool supports synthesis of all voltage areas at the same time, or synthesis of each voltage area individually. Use the `synthesize_fp_rail` command, or choose Preroute > Synthesize Power Network in the GUI.

To synthesize the power network for the voltage areas, select the “Synthesize power network by voltage areas” radio button in the GUI. You can synthesize all voltage areas at the same time by selecting the “All voltage areas” radio button. To synthesize a specific set of voltage areas, select the “Specified voltage areas” radio button and enter the voltage area name in the text box. Use the `synthesize_fp_rail -synthesize_voltage_areas -voltage_areas { area_names }` command to synthesize the power network for the specified voltage areas from the command line, or do not use the `-voltage_areas` option to synthesize all voltage areas.

Figure 8-5 shows the results of power network synthesis on multiple voltage areas. The figure contains an IR drop map for each power and ground net in the different voltage areas.

Figure 8-5 IR Drop Maps for Different Nets and Voltage Areas



Creating Power-Switch Arrays and Rings for Multithreshold-CMOS Designs

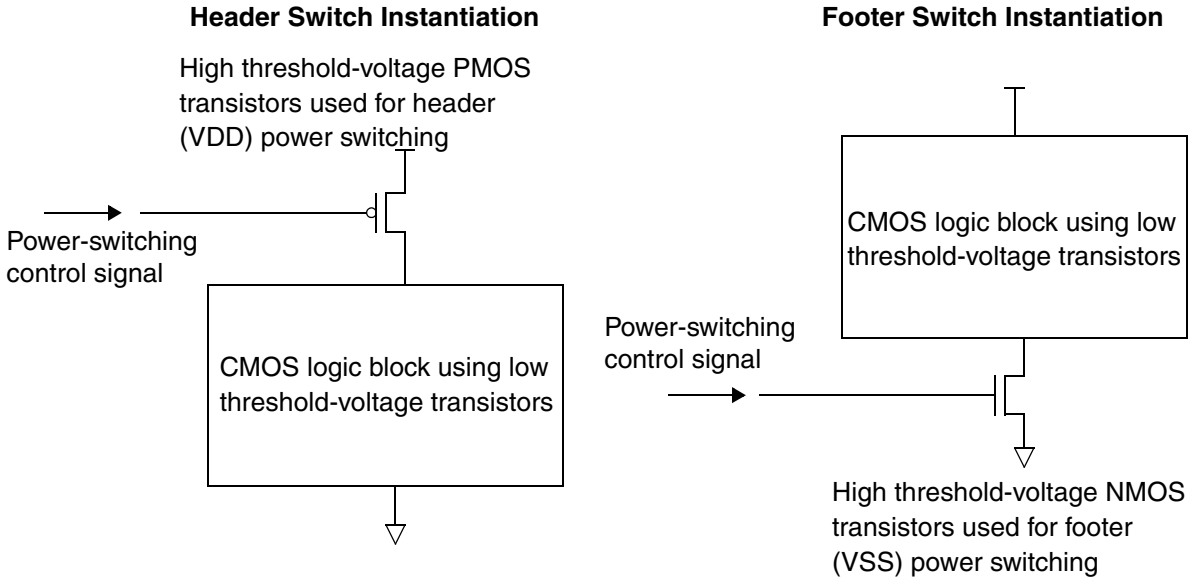
As process technology continues to scale to smaller and smaller geometries, the leakage current of CMOS devices increases exponentially. An effective method of controlling leakage power is to use multithreshold CMOS technology with shutdown transistors. The following sections illustrate the use of multithreshold CMOS and power gating to reduce leakage power in CMOS designs. The following sections describe the implementation of a power-switch network in a multivoltage design.

- Power Switch Overview
- Selecting the Proper power-switch Strategy
- Creating a Power-Switch Array
- Creating a Power-Switch Ring
- Exploring Different Switch Configurations
- Optimizing Power Switches
- Connecting Power Switches

Power Switch Overview

You can implement power switching in your design during floorplanning to reduce static leakage power to idle circuit blocks. Power switching is a technique that reduces leakage power by cutting the flow of current between VDD and VSS while the circuit block is idle. There are two types of power switches: header switches and footer switches. Header switches with PMOS transistors are placed between VDD and the block power supply pins; footer switches with NMOS transistors are placed between VSS and the block ground pins. In many designs, only one type of switch is used. Switch cells are encapsulated within standard cells.

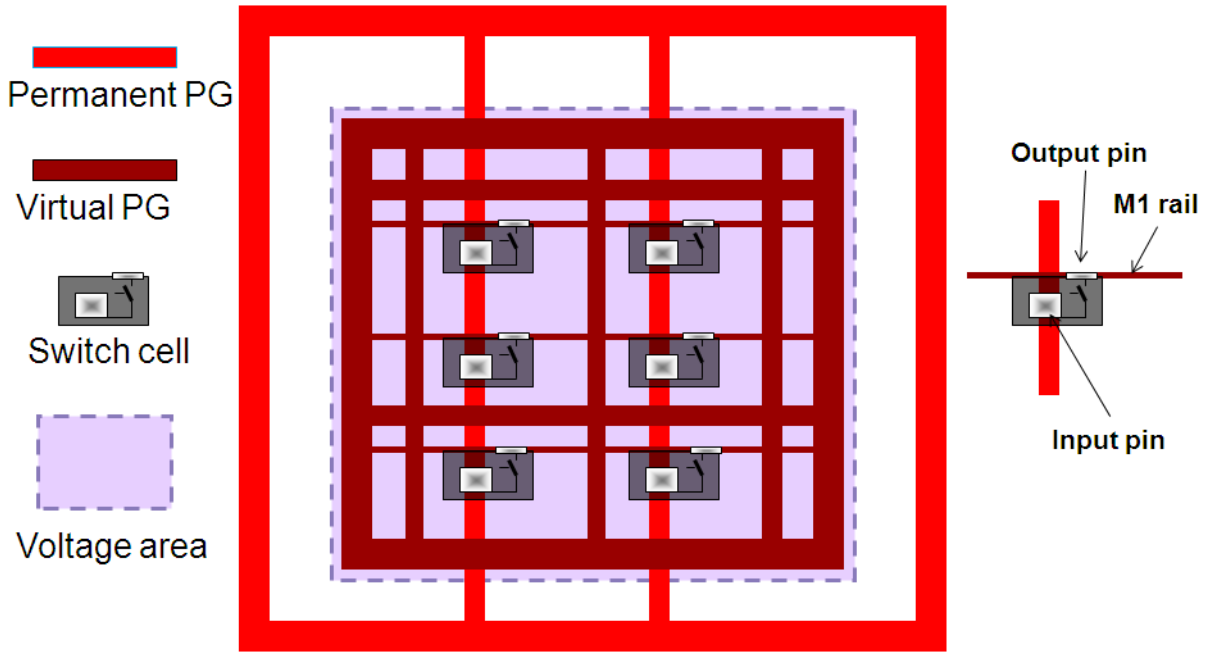
Figure 8-6 Power-Switching Network Transistors



The switching strategy shown in Figure 8-6 is a coarse-grain strategy; this strategy applies power switching to the entire block through a single control. Multiple transistors in parallel drive a common supply net for the block. In a fine-grain strategy, each library cell has its own power switch, allowing fine-grain control over which cells are powered down. The fine-grain approach has better potential for power savings, but requires more area.

The following illustration shows the layout view of the power-switch array used to shut down power to a circuit block. The power-switch network controls the current flow from the permanent power network to the virtual power network. Permanent power is a constant supply that originates from the top-level circuit pads and virtual power is switched by the power switch.

Figure 8-7 Power Switch and Power Rails



Selecting the Proper power-switch Strategy

Although power gating uses relatively large power-gating cells, the actual current available to the switched power network might be less than that available on the permanent power supply rails. The current reduction depends on the dynamic power requirements of the shutdown block, the size of the voltage area, and the number of power-gating cells. The physical placement of the power gating cells is critical to achieving an acceptable IR distribution within the shutdown block.

Two primary placement styles are used when placing power-gating cells: array-style placement and ring-style placement. In the array-style approach, power-gating cells are placed in a uniform grid within the voltage area. Array-style placement provides high drive current and minimizes resistance, which in turn impacts IR drop. The benefits of an array-style placement include

- Greater control over IR drop distribution
- Better optimization that allows for fewer gating cells than the ring-style method

The shortcomings of the array-style approach include

- Less available cell placement area
- More difficult or impossible implementation with hard macros, as the existing placement must be modified to accommodate power gates inside the voltage area
- Potentially more congestion or increase in area
- Potentially greater power grid complexity

In a ring-style approach, the tool places power-gating cells in a ring that surrounds the shutdown voltage area. The benefits of a ring-style placement include:

- Easier power grid creation for the block
- Easier application to designs that contain hard macros, without the need for modification inside the block. No extra cells or routing is required, with the exception of support for retention, isolation, level shifting, and always-on logic.
- Easier connection of power and daisy chains by using abutment

A major limitation of the ring-style approach is that the IR drop impact might be unacceptable toward the center of the voltage area as the distance increases from the power-gating cells. The severity of the IR drop depends on the voltage area size, shape, and internal dynamic power requirements of the block. For some designs, it might be impossible to implement a ring-style approach without violating IR drop and inrush current requirements.

Creating a Power-Switch Array

You can use the `create_power_switch_array` command to add a power switch in a uniform grid array inside the voltage area. After inserting the power switches, the command legalizes the power switches and locks their position by setting the `is_fixed` attribute on each power switch. Table 8-10 describes the `create_power_switch_array` command options. For more information about these options, see the man page.

Table 8-10 create_power_switch_array Command Options

Command option	Description
`-lib_cell` *`cellname`*	Specifies the library cell or power switch cell to be used.
`-bounding_box {`*`left bottom right top`*`}`	Specifies the coordinates for the rectangle in which to create the power-switch cell array.
`-design` *`hierarchy_name`*	Specifies the hierarchy level name in which to create the power-switch cell array.

Table 8-10 create_power_switch_array Command Options (Continued)

Command option	Description
`-offset_to_va` *offsets*	Specifies the offset to maintain between the the voltage area boundary and the power-switch array.
`-orientation N | W | S | E | FN | FE | FS | FW`	Specifies the placement orientation of the switch cells.
`-pattern normal | staggered`	Specifies the instantiation pattern for the switch cells.
`-prefix` *prefix_name*	Specifies a name prefix for the inserted switch cells.
`-relative_to_voltage_area`	Sets the origin of the bounding box coordinates at the lower-left corner of the voltage area.
`-respect` *object_type*	Specifies object types that cannot have power-switch cells overlapping the object.
`-snap_to_row_and_tile`	Snaps standard cell placement to row and tile.
`-start_column` *index*	Specifies the column index used when creating the instance path name. Use the `-start_row` *index* option to specify the row index.
`-voltage_area` *voltage_area*	Specifies the voltage area in which to place the switch cells.
`-x_increment` *deltax*	Specifies the incremental step in the x-direction for power-switch cell placement. Use the `-y_increment` *deltay* option to specify the y-direction incremental step.

The following example creates a power-switch array by using the `create_power_switch_array` command. The command instantiates the POWERSWITCH1 cell in a staggered pattern within the V_DOMAIN1 voltage area with a spacing between cells of 120 microns in the y-direction and 120 microns in the x-direction. The command attaches the V_DOMAIN1_ prefix to each inserted power-switch cell.

```
icc_shell> create_power_switch_array -lib_cell POWERSWITCH1 \
  -voltage_area V_DOMAIN1 -y_increment 120 -x_increment 120 \
  -prefix V_DOMAIN1_ -pattern staggered
```

Creating a Power-Switch Ring

The `create_power_switch_ring` command creates a full or partial ring of switch cells around the voltage areas or macro cells in your design. You can use command options to control the cell density of the switch cell ring, the type of filler cells that are inserted between the power-switch cells, the start point, the orientation, and other settings. The command places only the power-switch cells, it does not connect the power, ground, and control pins. The command does not honor the blockages or existing cells. However, the tool issues a warning message if the inserted cells create conflicts. Table 8-11 describes the `create_power_switch_ring` command options. For more information about these options, see the man page.

Table 8-11 create_power_switch_ring Command Options

Command option	Description
`-switch_lib_cell` *cellname*	Specifies the library cell to be used as the switch cell.
`-area_obj` *name*	Specifies the voltage area or macro around which to create the ring of switch cells.
`-check_overlap`	Checks whether the inserted cells overlap with other objects.
`-continue_pattern`	Continues the pattern on adjacent sides.
`-density` *density*	Specifies the cell density of the power-switch ring.
`-design` *hierarchy_name*	Specifies the hierarchy level name in which to create the power-switch cell array.
`-end_point` *{point}*	Specifies the endpoint of the ring.
`-filler_lib_cell` *cell_names*	Specifies the ordered list of filler cell names.
`-inner_corner_lib_cell` *cell_name*	Specifies the cell to be placed at the inner corners of the ring. Use the `-outer_corner_lib_cell` option to specify the cell for the outer corners of the ring.
`-inner_corner_orientation N | W | S | E | FN | FE | FS | FW`	Specifies the orientation of the cells placed at the inner corners of the ring. Use the `-outer_corner_orientation` option to specify the cell orientation for the outer corners.
`-offset` *offset_values*	Shrinks or enlarges the boundary of the ring with respect to the voltage area or macro boundary by the specified amount.

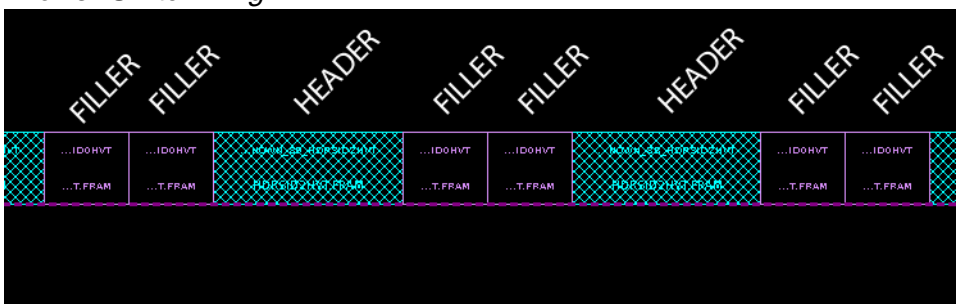
Table 8-11 create_power_switch_ring Command Options (Continued)

Command option	Description
`-place_pattern` *pattern_syntax*	Specifies the insertion pattern for cells on the ring.
`-polygon { points }`	Specifies a rectilinear polygon around which to insert the power-switch cells.
`-prefix` *prefix_string*	Specifies the name prefix for the cells inserted by this command.
`-start_point {`*point*`}`	Specifies the startpoint of the ring.
`-switch_orientation N | W | S | E | FN | FE | FS | FW`	Specifies the orientation of the switch cells placed on the bottom edge of the ring.

The following example creates a power-switch ring around the DOMAIN1 power domain. The command uses the SWITCH_HEADER power-switch cell and creates a ring pattern using the FILLER and HEADER library cells. Figure 8-8 shows the ring created by using the command in the example.

```
icc_shell> create_power_switch_ring -area_obj DOMAIN1 \
  -switch_lib_cell SWITCH_HEADER \
  -place_pattern { { FILLER HEADER FILLER } * }
```

Figure 8-8 Power-Switch Ring



Exploring Different Switch Configurations

There are many possible configurations you can select when creating a power-switch array network; the challenge is to determine the optimal configuration. You can use the `explore_power_switch` command to estimate the IR drop of various power-switch array configurations. The `explore_power_switch` command invokes both power network synthesis and power network analysis and requires power network synthesis constraints set by using the `set_fp_rail_constraints` command.

To perform power switch exploration, provide the power-switch library cells for insertion, the switch array x- and y-pitches, and the permanent and virtual supply names. The `explore_power_switch` command automatically loops through the configurations you provide and performs the following tasks:

- Inserts header or footer arrays based on your specification
- Legalizes placement after power switch insertion
- Logically connects permanent and virtual PG nets to power switches
- Performs multivoltage power network synthesis and creates a virtual power and ground mesh for all voltage areas
- Connects power switches automatically to the global supply
- Preroutes standard cells if the `-create_virtual_rail` option is not specified
- Performs power network analysis on the current configuration by using the `analyze_fp_rail -nets {vdd+vdd_local}` command

The `explore_power_switch` command creates a report containing a sorted list of maximum IR drop values for all configurations. Based on the results of the command, you can select the appropriate power switch configuration to achieve the desired IR drop for your design.

The following command performs power-switch cell exploration on the current design. The command explores the IR drop in the VA1 voltage area by instantiating the FOOTER8, FOOTER16, and FOOTER32 power-switch cells using the x-spacing values of 25, 50, and 100 microns. The VIRTUAL_VSS virtual ground net and VSS real ground net are used in the analysis. See the man page for a complete list of command options.

```
icc_shell> explore_power_switch \
  -lib_cells {"FOOTER8" "FOOTER16" "FOOTER32"} \
  -x_increment {25 50 100} -voltage_area VA1 \
  -virtual_pg_net VIRTUAL_VSS -real_pg_net VSS
```

Optimizing Power Switches

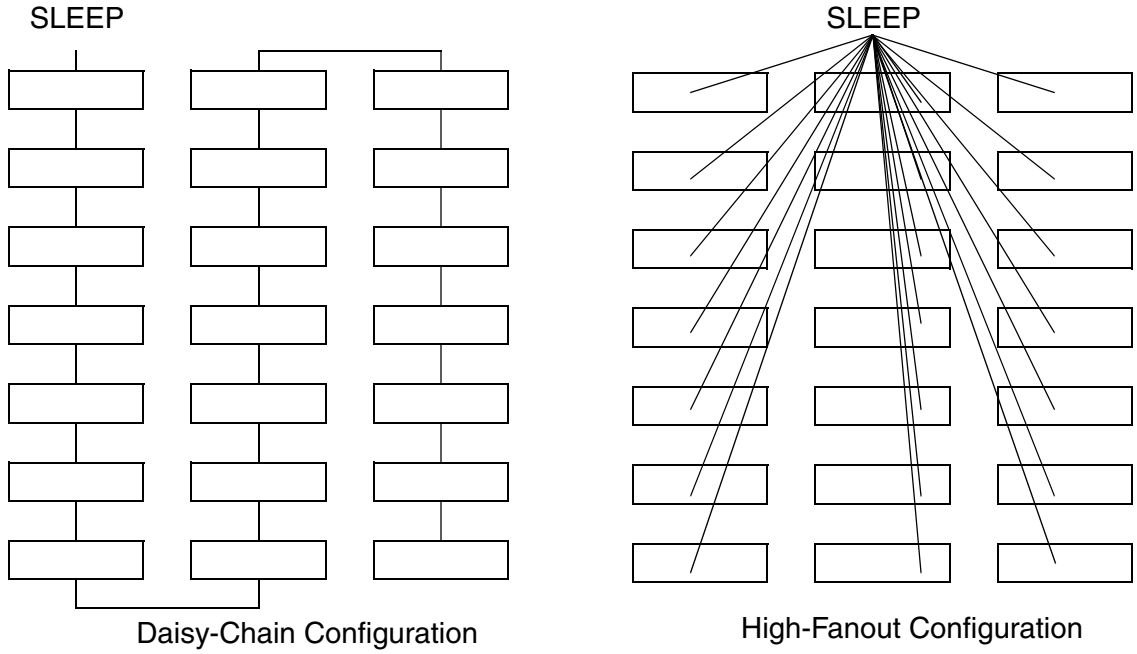
After you insert power switches into a design, you can optimize the power switches by using the `optimize_power_switch` command. The command sizes the power-switch cells to attempt to reduce the IR drop to the target level set by using the `set_mtcmos_pna_strategy` command. The following example uses the `optimize_power_switch` command to optimize the power-switch network to achieve a target IR drop of 120 mV.

```
icc_shell> set_mtcmos_pna_strategy -target_voltage_drop 120
icc_shell> optimize_power_switch -virtual_pg_net VIRTUAL_VDD \
  -real_pg_net VDD
```

Connecting Power Switches

The control pins of the cells within the power-switch cell network must be connected to a master sleep control signal. There are two connection styles: high fanout and daisy chain. The high-fanout connection style uses a single net and buffers to simultaneously power up the switch-cell network. The daisy-chain connection style enables the switch cells in a sequential fashion and results in a lower inrush current. Figure 8-9 shows a schematic view of these two connection schemes.

Figure 8-9 Daisy-Chain and High-Fanout Connection Styles



To connect the sleep control pins on the power-switch cells in daisy-chain mode, use the `connect_power_switch -mode daisy` command. In the following example, the `connect_power_switch` command connects the sleep control pins by using a daisy chain.

```
icc_shell> connect_power_switch -source SLEEP \
  -port_name sleep_port -mode daisy \
  -direction horizontal -verbose \
  -voltage_area { VA1 VA2 }
```

Analyzing the Power Network

You can perform power network analysis on your design to predict the IR drop of different floorplan and power routing configurations. Power network analysis extracts and analyzes the power rail network connected to the net names you specify. You can analyze IR drop and electromigration effects during initial power grid planning while the floorplan is incomplete and the power ports of the standard cells are not yet connected. After detailed placement and power rail routing, you can verify that the size and quantity of the power nets are adequate to power the design. The following section describes the steps to perform power network analysis on your design.

- Performing Power Network Analysis
- Creating Connectivity Views
- Performing Power Analysis With Switching Activity Information

Performing Power Network Analysis

To perform power network analysis on your design, use the `analyze_fp_rail` command or choose Preroute > Analyze Power Network in the GUI. Table 8-12 describes the command options for the `analyze_fp_rail` command.

Table 8-12 analyze_fp_rail Command Options

Command option	Description
`-analyze_power` (“Analyze power” check box in the GUI)	Runs the power analysis engine on the current design.
`-create_virtual_rails` *`layer`* (“Create virtual rails” check box in the GUI)	Creates virtual power straps for the standard cell pin connections.
`-ignore_blockages` (“Ignore blockages” check box in the GUI)	Ignores blockages for virtual pads.
`-ignore_conn_view_layers` *`layer`* (“Ignore CONN view layers” check box in the GUI)	Ignores the specified metal layers in the CONN view.

Table 8-12 analyze_fp_rail Command Options (Continued)

Command option	Description
`-nets` *net_names* (“Power Ground nets” text box in the GUI)	Specifies the names of the power or ground nets on which to perform power network analysis.
`-output_directory` *directory_name* (“Output” text box in the GUI)	Specifies the directory name in which to save the analysis results file.
`-pad_masters` *master_nets* (“Specified pad masters” check box in the GUI)	Specifies the power net name and associated power pad.
`-power_budget` *power* (“Power budget” text box in the GUI)	Specifies the total power budget.
`-read_default_power_file` *file_name* (“Default” radio button in the GUI)	Specifies the file name that contains power information in default format.
`-read_pad_instance_file` *file_name* (“Instances” radio button in the GUI)	Specifies the file name that contains pad instance names and optional net names.
`-read_pad_master_file` *file_name* (“Masters” radio button in the GUI)	Specifies the file name that contains pad master names and optional net names.
`-read_power_compiler_file` *file_name* (“Power Compiler/...” button in the GUI)	Specifies the file that contains the Power Compiler report.
`-read_prime_power_file` *file_name* (“PrimePower/...” button in the GUI)	Specifies the file that contains the PrimePower report.
`-top_level_only` (“Top level cells only” radio button in the GUI)	Specifies that only top-level cells are used in the power calculation. Cells inside soft macros are ignored.

Table 8-12 analyze_fp_rail Command Options (Continued)

Command option	Description
`-use_pins_as_pads` ("Use top level design pins as pads" check box in the GUI)	Treats power pins as power pads during power analysis.
`-voltage_supply` *`voltage`* ("Supply voltage" text box in the GUI)	Specifies the supply voltage for the specified power net.

Creating Connectivity Views

Use the `create_connview` command to invoke the Hercules connectivity engine to create CONN views and current source files for hard macro cells in your design. Power network analysis uses the `create_connview` command options and values that you specify, together with those specified in the technology file for the Milkyway design library, to create a Hercules runset.ev file. The Hercules connectivity engine uses the runset file and creates a SMASH view in the database. After the SMASH view is created, the command uses this data to create a CONN view for the hard macro cells. Power network analysis updates the runset file each time you run the `create_connview` command.

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

Performing Power Analysis With Switching Activity Information

The amount of power consumed by a design depends on the switching activity that occurs within the design. You must provide an accurate model of the switching activity to produce an accurate estimate of the power consumption for the design. Switching activity can be annotated on design nets, ports, and cell pins.

After capturing the switching activity and annotating your design, you can invoke power network analysis by using the `analyze_fp_rail -analyze_power` or `synthesize_fp_rail -analyze_power` command. These commands calculate the dynamic power value of each cell instance by considering the specified switching probabilities and by using the obtained power values for IR drop analysis. For more information, see "Performing Power Network Analysis for Each Cell Instance" on page 8-37

Annotating Switching Activity

You can annotate the switching activity with one of the following methods:

- Use the `set_switching_activity` command to annotate particular design objects, such as nets, pins, ports, and cell instances, of the current design.

  Switching activity information consists of the static probability and the toggle rate. The static probability is the probability that the value of the design object has a logic value 1. It defines the percentage of time the signal is at a high state; the default value is 0.5. The toggle rate is the rate at which the design object switches between logic values 0 and 1 within a time period. The following example defines a toggle rate of 0.25 and a static probability of 0.4 for the CLK signal.

  ```
  icc_shell> set_switching_activity -toggle_rate 0.25 \
    -static_probability 0.4 CLK
  ```

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

- Generate one or more switching activity interchange format (SAIF) files by using RTL or gate-level simulation. Use the `read_saif` command to annotate the switching activity information onto nets, pins, ports, and cells in the current gate-level design.

  A SAIF file is typically generated within a simulation flow that contains a testbench. The SAIF file contains the switching activity information organized in a hierarchical fashion, where the hierarchy of the SAIF file reflects the hierarchy of the simulation testbench. The following example uses the `read_saif` command in verbose mode to read the data.saif file for the current design. The design appears as cpu/system_controller in the SAIF file.

  ```
  icc_shell> read_saif -input data.saif \
    -instance_name cpu/system_controller
  ```

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

- Use default switching activity.

  If no simulation data is available, IC Compiler applies the built-in default toggle rate to analyze the power consumption.

  The switching activity of primary inputs and black box outputs cannot be propagated. You can set the default toggle rate (0.1) and default static probability (0.5) for primary inputs and black box outputs by using the `power_default_toggle_rate` and `power_default_static_probability` variables. The default toggle rate value is the value of the `power_default_toggle_rate` variable multiplied by the related clock frequency. The clock can be specified by using the `-clock` option of the `set_switching_activity` command. If no related clock is specified on the net, the clock with the highest frequency is used. The default switching activity applied to these objects is propagated throughout the design for power network analysis.

Performing Power Network Analysis for Each Cell Instance

After capturing the switching activity and annotating your design, you can invoke power network analysis by using the `analyze_fp_rail -analyze_power` or `synthesize_fp_rail -analyze_power` command to analyze the power information from each cell instance in your design based on static probability and toggle rates. These commands calculate dynamic power values for each cell instance by using the switching probabilities and toggle rates that you specify. The power values are used to perform IR drop analysis. Power is not calculated unless you specify the `-analyze_power` option.

In the following example, the voltage drop on the VDD net is analyzed by using a supply voltage of 1.2 volts.

```
icc_shell> analyze_fp_rail -analyze_power -nets { VDD } \
  -voltage_supply 1.2
```

You can use the `-analyze_power` option concurrently with the `-power_budget` and the `-read_default_power_file` options.

Note:

If you use the `-analyze_power` option concurrently with the `-read_default_power_file` option, the power specified in the default power file has a higher priority than that calculated by the `-analyze_power` option.

Viewing the Analysis Results

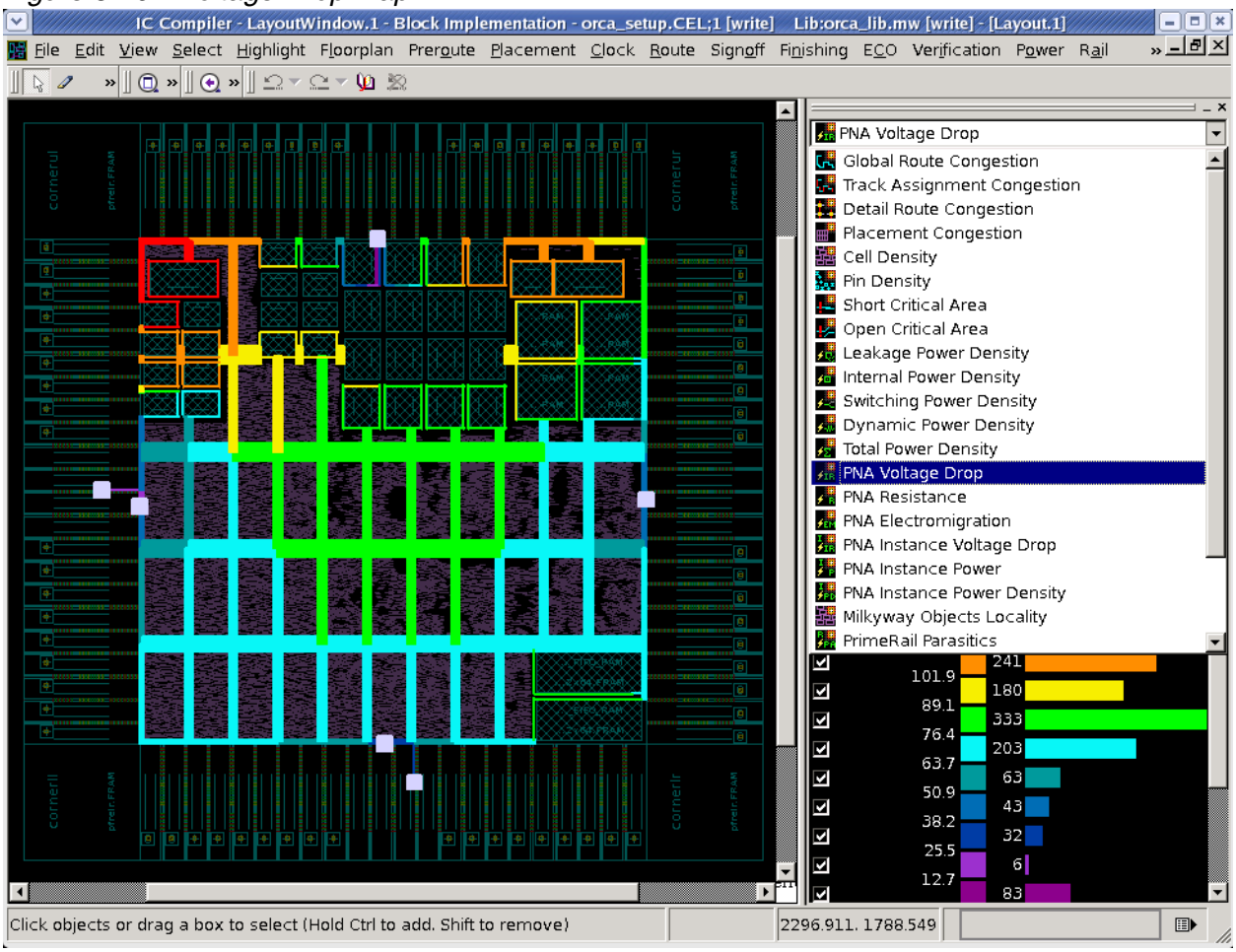
After you complete power and rail analysis, you can check for power-related violations in your design. IC Compiler provides several analysis maps that overlay on the layout view of your design, including a voltage drop map, a resistance map, an electromigration map, an instance voltage drop map, an instance power map, and an instance power density map. The following sections outline the steps to display the different power and rail analysis maps available in IC Compiler:

- Displaying the Voltage Drop Map
- Displaying the Resistance Map
- Displaying the Electromigration Map
- Displaying the Instance Voltage Drop Map
- Displaying the Instance Power Map
- Displaying the Instance Power Density Map

Displaying the Voltage Drop Map

After synthesizing the power network by using the `synthesize_fp_rail` command or by choosing Preroute > Synthesize Power Network in the GUI, you can review the voltage drop for different metal layers and locations in the layout. Choose Preroute > Power Network Voltage Drop Map in the GUI to view the voltage drop map, or use the `load_fp_rail_map` command. IC Compiler reads the results data saved to the ./pna_output directory, constructs a color-coded voltage drop overlay, and displays the overlay on the layout view. The command also builds a color histogram that displays the distribution of voltage drop values. Figure 8-10 shows a sample voltage drop analysis overlay in the IC Compiler GUI.

Figure 8-10 Voltage Drop Map



You can configure the histogram and voltage drop map display by selecting threshold voltages and metal layers in the GUI. To select only certain metal layers for IR drop analysis, select or deselect the check boxes in the layers box in the PNA Voltage Drop panel of the GUI. To enter a maximum or minimum threshold value for display in the histogram and

voltage drop map, select the “Max threshold” or “Min threshold” check box and enter the value in the text box. To limit the voltage drop bins that are displayed, select or deselect the check boxes in the histogram box in the PNA Voltage Drop panel of the GUI. After you change the selection for either threshold or metal layers, click the Apply button to display the updated histogram and voltage drop map.

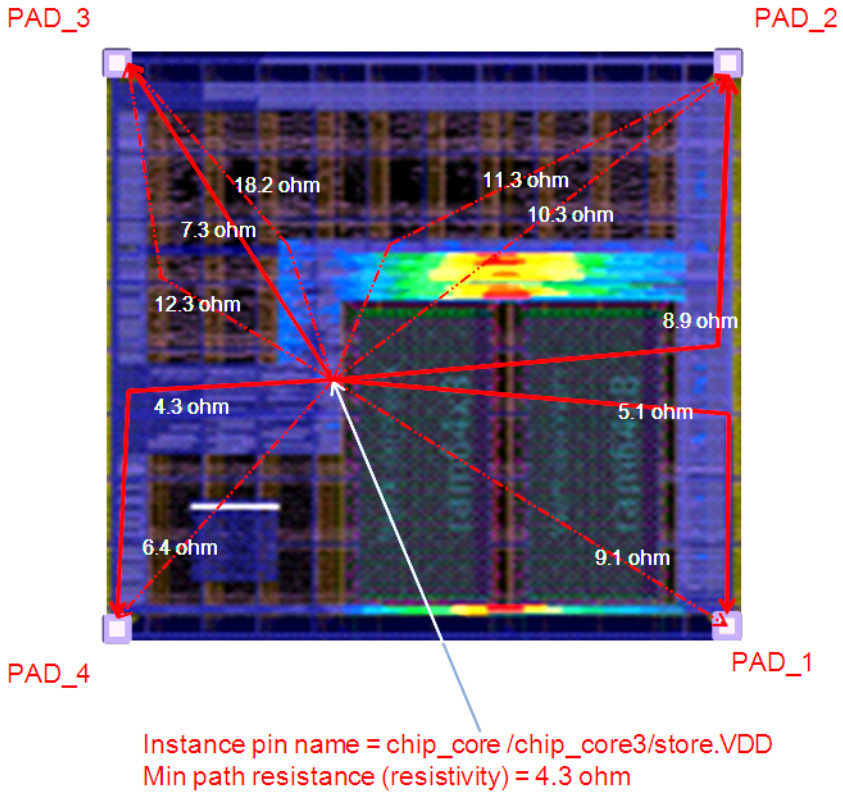
To use the mouse to examine the voltage drop at individual locations in the layout, click the Query button and move the mouse onto the layout view. As you move the mouse over different points in the layout, the tool displays the voltage drop at that point in the query panel in the lower-left area of the layout.

To change the net for analysis, click the Reload button and choose a net name from the Net drop-down box. Click OK to view the results in the GUI.

Displaying the Resistance Map

The resistance map displays a map of resistivity values for the power network in your design. The resistivity value at any instance pin or power grid location is the minimum path resistance value of all independent path resistances to any voltage source location. Figure 8-11 shows a series of paths to the chip_core/chip_core3/store.VDD pin; the lowest resistivity value of 4.3 represents the resistivity for the node. You can use the resistance map to check for unusually high values of resistivity, which might indicate missing vias or disconnected nets in the power network.

Figure 8-11 Minimum Path Resistance



Use the `load_fp_rail_map -nets net_name -type R` command or choose Preroute > Power Network Resistance Map in the GUI to load the map and histogram. The operation of the GUI for changing the threshold values, histogram, metal layer selection, and query text are similar to that for the voltage drop map; see "Displaying the Voltage Drop Map" on page 8-38 for information about operating the GUI.

Displaying the Electromigration Map

Electromigration is the permanent physical movement of metal in thin wire connections resulting from the displacement of metal ions by flowing electrons. Electromigration can lead to shorts and opens in wire connections that can cause a functional failure. You can view the electromigration analysis results for your design by using the `load_fp_rail_map -nets net_name -type EM` command or by choosing Preroute > Power Network Electromigration Map in the GUI. The operation of the GUI for changing the threshold values, histogram, metal layer selection, and query text are similar to that for the voltage drop map; see "Displaying the Voltage Drop Map" on page 8-38 for information about operating the GUI.

Displaying the Instance Voltage Drop Map

The instance voltage drop map displays voltage drop values for each instance in the design. You can view instance voltage drop analysis results for your design by using the `load_fp_rail_map -nets net_name -type InstIR` command or by choosing Preroute > Power Network Instance Voltage Drop Map in the GUI. The operation of the GUI for changing the threshold values, histogram, and query text are similar to that for the voltage drop map; see "Displaying the Voltage Drop Map" on page 8-38 for information about operating the GUI.

Displaying the Instance Power Map

The instance power map displays power values for each instance in the design. You can view instance power analysis results for your design by using the `load_fp_rail_map -nets net_name -type P` command or by choosing Preroute > Power Network Instance Power Map in the GUI. The operation of the GUI for changing the threshold values, histogram, and query text are similar to that for the voltage drop map; see "Displaying the Voltage Drop Map" on page 8-38 for information about operating the GUI.

Displaying the Instance Power Density Map

The instance power density map displays power density values for areas of the design. You can view instance power density analysis results for your design by using the `load_fp_rail_map -nets net_name -type PD` command or by choosing Preroute > Power Network Instance Power Density Map in the GUI. The operation of the GUI for changing the threshold values, histogram, and query text are similar to that for the voltage drop map; see "Displaying the Voltage Drop Map" on page 8-38 for information about operating the GUI.

Running PrimeRail Within IC Compiler

You can run PrimeRail In-Design Rail Analysis to perform power network verification on your design, including voltage drop, electromigration, and other rail analysis. This section includes the following topics:

- IC Compiler In-Design Rail Analysis Flow
- Running PrimeRail Within IC Compiler
- Checking Power Network Integrity
- Performing Rail Analysis

- Loading and Deleting PrimeRail Analysis Maps
- Performing Decoupling Capacitance Analysis and Insertion With PrimeRail
- Analyzing Internal Power Nets
- Performing Inrush Current Analysis on Multithreshold-CMOS Cells In IC Compiler

IC Compiler In-Design Rail Analysis Flow

The PrimeRail tool is integrated with IC Compiler through In-Design Rail Analysis; you can run rail integrity checking, voltage drop analysis, and electromigration analysis in the IC Compiler environment. The IC Compiler and PrimeRail integration also supports advanced analysis features such as decoupling capacitor analysis and inrush analysis. When checking and analysis processes are complete, you can display the generated resistivity map, voltage drop map, and electromigration map. These maps help you to discover problem areas and determine corrective action without leaving the IC Compiler environment. Error cells are also available for display through the IC Compiler error browser.

Running PrimeRail Within IC Compiler

You can execute PrimeRail commands from within the IC Compiler environment by using the GUI or by using the command line.

Note that the PrimeRail and PrimeTime PX tools require a separate tool license apart from IC Compiler. In addition, you must add the directory paths for the PrimeRail and PrimeTime PX executables to your Linux or UNIX PATH environment variable. Because the `analyze_rail` command is available only for cell-level static rail analysis, you must obtain the following minimum license configuration to invoke PrimeRail within IC Compiler: PrimeRail-static, MDataPrep, Milkyway, and PrimeTime PX.

The following IC Compiler commands are used to perform PrimeRail tasks: `set_rail_options`, `analyze_rail`, and `report_rail_options`. Use the `set_rail_options` command or choose Rail > Set Rail Options in the GUI to set options

for PrimeRail analysis. Table 8-13 describes the `set_rail_options` command options, see the man page for more information. See the *PrimeRail User Guide* for specific information about input and output file formats for PrimeRail.

Table 8-13 set_rail_options Command Options

Command option	Description
`-output_dir` *`dir_name`* (Environment tab > "Output directory" text box in the GUI)	Specifies the output directory for the `analyze_rail` command.
`-use_pins_as_pads true | false` (Input tab > "Use pins as pads" check box in the GUI)	Treats top-level power and ground pins as ideal voltage sources in the block-level simulation.
`-pad_master_file` *`file_name`* (Input tab > "Pad master file" text box in the GUI)	Specifies an optional pad master file.
`-pad_instance_file` *`file_name`* (Input tab > "Pad instance" text box in the GUI)	Specifies an optional pad instance file.
`-user_defined_tap_file` *`file_name`* (Input tab > "Tap file" text box in the GUI)	Specifies an optional file that defines the coordinates and layer numbers of the ideal voltage sources to be used during rail analysis.
`-packaging_file` *`file_name`* (Input tab > "Package file" text box in the GUI)	Specifies an optional SPICE file that contains linear circuit elements for modeling packaging parasitics.
`-power_scale_factor` *`value`* (Analysis tab > "Power Scale Factor" text box in the GUI)	Specifies the power scaling factor for rail analysis, where the scaled power is the total power multiplied by the factor.
`-power_scale_value` *`value`* (Analysis tab > "Power Scale Value" text box in the GUI)	Specifies the target total power for rail analysis.
`-vd_threshold` *`value`* (Analysis tab > "Voltage drop" text box in the GUI)	Specifies the voltage drop threshold in mV for violation checking.

Table 8-13 set_rail_options Command Options (Continued)

Command option	Description
`-switching_activity {file_format` *`file_name`* `[`*`strip_path`*`]}` (Input tab > “Switching activity” text boxes and radio buttons in the GUI)	Specifies an optional switching activity file format and name to be used in rail analysis.
`-host` *`machine_name`* (Environment tab > “Hosts” text box in the GUI)	Specifies a computer host name. This option is used when running PrimeRail on a different machine.
`-pr_exec_dir` *`dir_name`* (Environment tab > “PrimeRail” text box in the GUI)	Specifies the path to the PrimeRail executable.
`-pt_exec_dir` *`dir_name`* (Environment tab > “PrimeTime” text box in the GUI)	Specifies the path to the PrimeTime executable.
`-sdc` *`file_name`* (Input tab > “SDC” text box in the GUI)	Specifies an optional SDC script file.
`-spef` *`file_name`* (Input tab > “Parasitics” text box in the GUI)	Specifies an optional parasitic SPEF file.
`-verilog` *`file_name`* (Input tab > “Verilog” text box in the GUI)	Specifies an optional Verilog netlist file.
`-upf` *`file_name`* (Input tab > “UPF” text box in the GUI)	Specifies an optional UPF script file.
`-analysis_mode static | dynamic` (Analysis tab > “Analysis mode” radio button in the GUI)	Specifies the rail analysis mode to use: static or dynamic.
`-start_time` *`time_value`* (Analysis tab > “Start time” text box in the GUI)	Specifies the start time for a simulation run on a specific subwindow of activity. Use the `-end_time` *`time_value`* option to specify the simulation end time.

Table 8-13 set_rail_options Command Options (Continued)

Command option	Description
`-filler_lib_cells` *lib_cell_list* (Analysis tab > "Filler cell names" text box in the GUI)	Specifies the names of the filler cells.
`-decap_lib_cells` *lib_cell_list* (Analysis tab > "Decap cell names" text box in the GUI)	Specifies the names of the decoupling capacitor cells.
`-config_file` *file_name* (Environment tab > "PrimeRail configuration file" text box in the GUI)	Specifies a configuration file for PrimeRail.
`-signal_parasitics_output_format SBPF | SPEF` (Input tab > "Parasitics type" text box in the GUI)	Specifies the output parasitic file format, SBPF or SPEF, used when generating parasitic files.
`-reuse {power pg_extraction setup_variables setup_files}`	Specifies rail analysis results to reuse from the previous `analyze_rail` run.

Checking Power Network Integrity

Prior to running a comprehensive power network analysis, you can check the design for common power connectivity problems by using the `check_rail` command. This command performs a full-chip connectivity check by using PrimeRail, and then saves the connectivity report to a log file. The `check_rail` command checks the design for the following common connectivity issues:

- Floating power and ground pins
- Supply nets that do not connect to an instance
- Power pins that connect to a ground or signal net, and ground pins that connect to a power or signal net
- Power nets with a zero or unassigned voltage

By default, the `check_rail` command purges and rebuilds the RAIL view to ensure consistency between the CEL and RAIL design views. To use the existing RAIL view generated by the `analyze_rail` command without purging, use the `set_rail_options -reuse setup_variables` command.

Performing Rail Analysis

You use the `analyze_rail` command to perform rail analysis using PrimeRail. When executed, the command generates setup data for PrimeRail and runs a PrimeRail script within the IC Compiler session. The `analyze_rail` command supports power and ground network integrity analysis, voltage drop analysis, and electromigration analysis. When the analysis is finished, the command generates a script file that contains PrimeRail commands. If an error occurs, you can modify the script file and use it in a subsequent run. For debugging purposes, you can run the script in PrimeRail outside of the IC Compiler environment.

Table 8-14 describes the `analyze_rail` command options. For more information, see the man page.

Table 8-14 analyze_rail Command Options

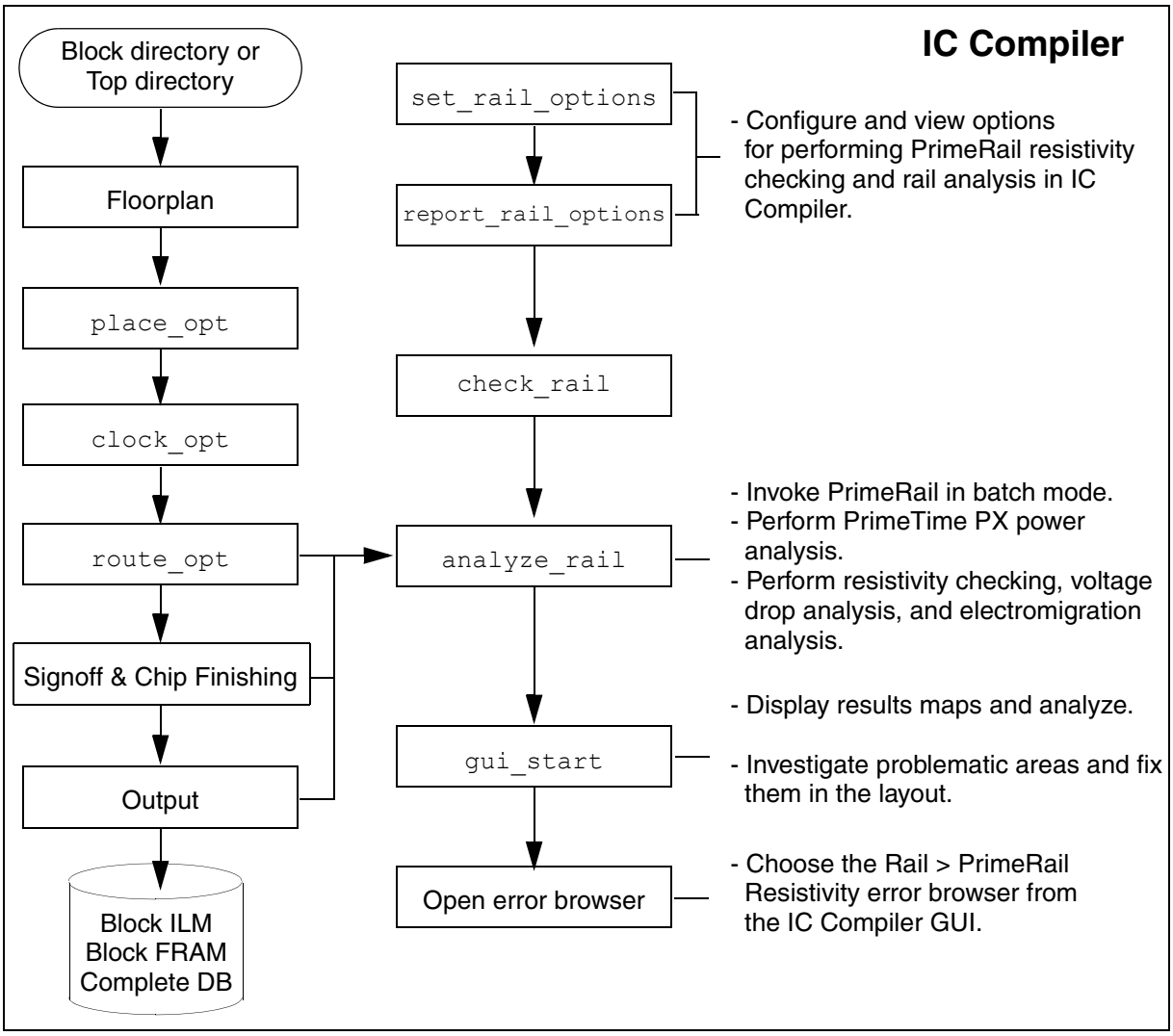
Command option	Description
`-decap` ("Decoupling capacitance analysis" check box in the GUI)	Performs decoupling capacitance analysis for the top-level design.
`-electromigration` ("Electromigration analysis" check box in the GUI)	Performs electromigration analysis on the specified power and ground nets.
`-inrush` ("In-rush current analysis on specified PG nets" check box in the GUI)	Performs inrush current analysis on the specified power and ground nets.
`-integrity` ("PG network weakness analysis" check box in the GUI)	Performs connectivity checking on the specified power and ground nets.
`-primerail_script_file` *`script_name`* ("Run user script" text box in the GUI)	Specifies a script for the PrimeRail analysis run.

Table 8-14 analyze_rail Command Options (Continued)

Command option	Description
`-script_only` (“PrimeRail script generation only” check box in the GUI)	Generates a PrimeRail script for the target analyses without actually performing the analysis run.
`-voltage_drop` (“Voltage drop analysis” check box in the GUI)	Performs voltage drop analysis on the specified power and ground nets.

Figure 8-12 describes the commands used in block-level rail analysis within the IC Compiler flow.

Figure 8-12 PrimeRail Chip-Level Rail Analysis Command Flow Within IC Compiler



Loading and Deleting PrimeRail Analysis Maps

When you perform power network analysis by using the `analyze_rail` command, PrimeRail generates map files for the analysis mode you specify. You can load the map file into the IC Compiler GUI by using the `read_rail_maps` command or by choosing the appropriate item from the Rail menu in the IC Compiler GUI:

- Rail > PrimeRail Parasitics to view the parasitics map.
- Rail > PrimeRail Resistivity to view the resistivity map.

- Rail > PrimeRail Power to view the power map.
- Rail > PrimeRail Voltage Drop to view the voltage drop map.
- Rail > PrimeRail Electromigration to view the electromigration map.

Use the `remove_rail_maps` command to remove the rail map data from the current IC Compiler session.

Performing Decoupling Capacitance Analysis and Insertion With PrimeRail

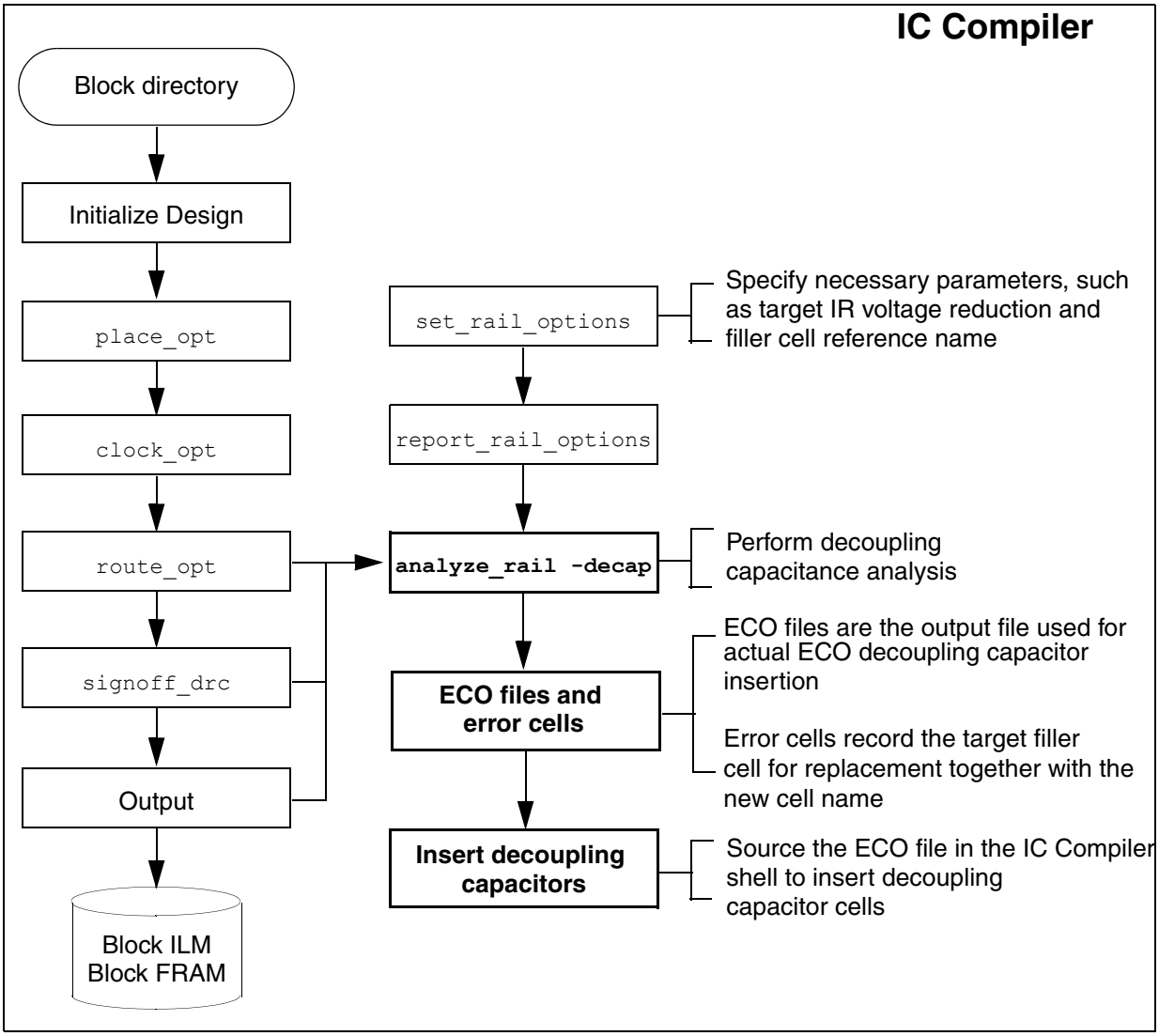
Decoupling capacitor cells reduce the peak voltage drop and attenuate noise on the power supply rails. You can perform decoupling capacitance analysis and insertion by using the `set_rail_options` and `analyze_rail` commands within IC Compiler. The PrimeRail tool can provide placement suggestions for decoupling capacitor cells to minimize power supply noise. In addition, IC Compiler can automatically place the decoupling capacitor cells to achieve a target voltage drop reduction.

This section includes the following topics:

- Analyzing Filler Cell and Decoupling Capacitor Cell Replacements
- Performing Decoupling Capacitance Analysis in IC Compiler
- Performing Decoupling Capacitor Insertion in IC Complier

Figure 8-13 illustrates the decoupling capacitance analysis and insertion flow in IC Compiler.

Figure 8-13 Steps for Running Decoupling Capacitance Analysis in IC Compiler



Analyzing Filler Cell and Decoupling Capacitor Cell Replacements

Before you can insert decoupling capacitor cells, you must load the design in IC Compiler and set the necessary parameters for the PrimeRail decoupling capacitor cell analysis and insertion.

- Use the `set_rail_options -filler_lib_cells` *`lib_cell_list`* command to specify the names of the filler cell references to be replaced with decoupling capacitor cells. Wildcard usage is not supported.

Note that the filler cell references must already be placed or routed in IC Compiler by using the `insert_stdcell_filler` command.

- Use the `set_rail_options -decap_lib_cells` *`lib_cell_list`* command to specify the names of the decoupling capacitor cell used to replace the filler cells. Wildcard usage is not supported.
- Use the `-vd_threshold` option to specify the target reduction in IR drop. The default reduction is 0 percent. This option is not required.

The `set_rail_options` command saves the data required for the decoupling capacitance analysis and insertion flow in the Milkyway design library.

Performing Decoupling Capacitance Analysis in IC Compiler

Decoupling capacitance analysis focuses on achieving the user-specified target IR voltage drop while minimizing the decoupling capacitor insertion cost. The cost is a function of the total area and leakage of the inserted decoupling capacitor cells. When you perform decoupling capacitance analysis and insertion within IC Compiler, the PrimeRail tool selects available preplaced filler cells in the design and then virtually replaces them with decoupling capacitor cells. During the analysis, the PrimeTime tool removes unneeded decoupling capacitor cells to minimize area and leakage, while meeting the user-specified voltage drop reduction target.

- Use the `analyze_rail -decap` option to perform decoupling capacitance analysis for the top-level design in IC Compiler. This command must be run after setting the dynamic rail analysis options by using the `set_rail_options` command. The PrimeRail tool iterates on the design to determine which filler cells can be replaced with decoupling capacitor cells to satisfy the IR drop target.

The following example shows how to perform decoupling capacitor insertion on the VDD net. The design uses the *`FillerCell1`* and *`FillerCell2`* filler library cells and the *`DecapCell1`* and *`DecapCell2`* decoupling capacitor library cells. The voltage threshold target is 1.5 V.

```
icc_shell> set_rail_options -filler_lib_cells "FillerCell1 FillerCell2" \
  -decap_lib_cells "DecapCell1 DecapCell2" -vd_threshold 1.5
icc_shell> analyze_rail VDD -decap
```

After you perform decoupling capacitance analysis, the PrimeRail tool writes out an ECO file. The file contains a list of filler cells that need to be swapped along with the corresponding decoupling capacitor cells. The tool saves the ECO file to the pr*_design_name* PrimeRail run directory and uses the file name `.decap_ude_`*`n`*, where *n* is the number of the decoupling capacitance analysis iteration. The tool writes the ECO file one time per analysis iteration.

The following is an example of the ECO file:

```
change_link [get_cells -hierarchy -all xofiller!FILL16!9053] DCAP16
change_link [get_cells -hierarchy -all xofiller!FILL16!9054] DCAP16
```

In this example, xofiller!FILL16!9053 and xofiller!FILL16!9054 represent the existing filler instance names. DCAP16 represents the replacement decoupling capacitor library cell.

Performing Decoupling Capacitor Insertion in IC Complier

After PrimeRail generates the ECO file, you can open the design in IC Compiler and perform decoupling capacitor cell insertion by sourcing the ECO file, .decap_ude_*n*, in the IC Compiler shell. The commands in the ECO file replace filler cells with decoupling capacitor cells.

During decoupling capacitor insertion, the tool saves physical information from the filler cell. This information includes the location, size, orientation, and information about the power and ground nets. After saving the information, the tool deletes the filler cell, replaces it with a new instance of the decoupling capacitor cell, and attaches the physical attributes of the filler cell.

Analyzing Internal Power Nets

In-Design Rail Analysis in IC Compiler can analyze a virtual power and ground net that is located within a soft macro cell. The internal power net connects to the main power net through a power management cell or switch cell. To perform analysis on an internal power or ground net in IC Compiler, you must specify both the internal power or ground net name and its supply voltage.

To analyze the internal power net, the specified net name should be the full path name from the top cell. Because Milkyway allows the symbol “/” to be used as both the instance name and the hierarchy separator, the tool revises the hierarchy separators internally to match the physical cell hierarchy in IC Compiler. To perform analysis with the internal power net in IC Compiler, specify both the internal power net name and its supply voltage by using the following command syntax:

```
analyze_rail {net_name {internal_pwr_net_name pwr_net_supply_voltage}}
```

For example:

```
icc_shell> analyze_rail {VDD {inst/VDDV 1.08}}
```

where inst/VDDV is the internal power net name in IC Compiler format and 1.08 is the supply voltage. PrimeRail updates IC Compiler’s hierarchy separators for the internal power net name in order to match the physical Milkyway cell hierarchy.

The tool supports internal power and ground net analysis in both static and dynamic mode, including resistivity analysis.

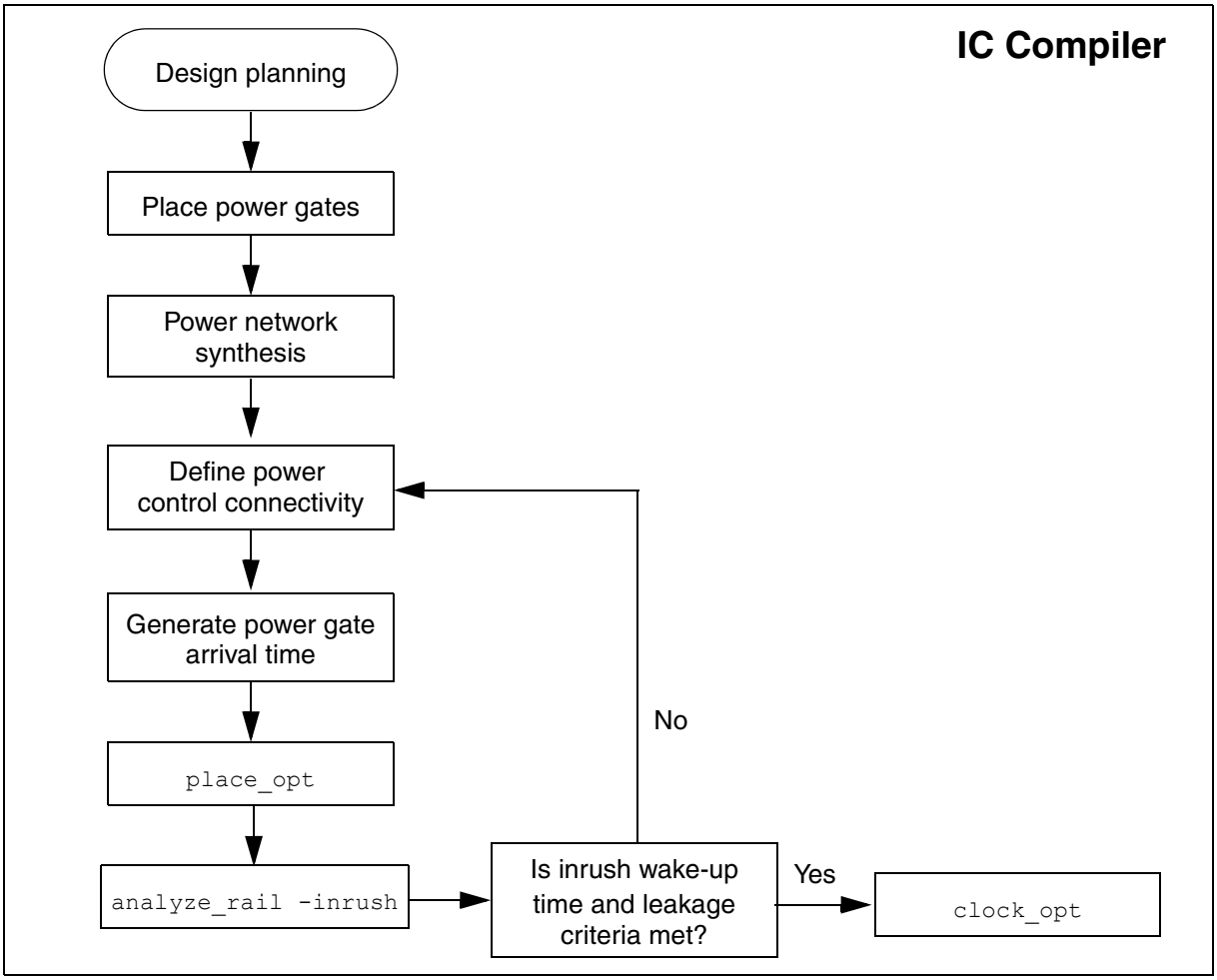
Performing Inrush Current Analysis on Multithreshold-CMOS Cells In IC Compiler

You can perform PrimeRail inrush current analysis on multithreshold-CMOS cells in IC Compiler. For more information about Multithreshold-CMOS cells, see “Creating Power-Switch Arrays and Rings for Multithreshold-CMOS Designs” on page 8-24.

PrimeRail provides a power-up sequence flow to control the simultaneous switching noise (inrush current) by turning on power management circuits sequentially. Accordingly, the number of power management cells, their drive strength, and the timing sequence of the control signals are the design parameters that must be optimized and verified through accurate circuit simulation.

Figure 8-14 illustrates the steps to perform inrush current analysis in IC Compiler.

Figure 8-14 Inrush Analysis Flow in IC Compiler



The following section describes a common inrush analysis methodology illustrated in Figure 8-14.

1. Create the power domains during initial synthesis, using the IEEE 1801 Unified Power Format (UPF) specification. To minimize complexity, you should ensure that the following requirements are met:
   - Power domains correspond directly to logical blocks or groups of logical blocks with the same power requirements.
   - Power domains correspond directly to physical voltage areas.
   - A power-gated block corresponds 1-to-1 with a power domain and voltage area.
   - Supply nets are connected to the standard cells in the design by using the `derive_pg_connection` command.

2. Insert and place the power-gating cells.
3. After connecting the power-gating cells, export the power management cell or switch cell arrival times to PrimeRail for inrush current analysis, using the static timing arrival time available in IC Compiler. In IC Compiler, you can query the static timing information and write out the arrival time information in PrimeRail syntax to a file. Otherwise, PrimeRail can also generate the file with arrival time information from the PrimeTime PX report that is generated during current waveform generation.

   The arrival time information is loaded into a file which is known as the “event file” for switch cells. The following example shows an event file from the VCD-based inrush current analysis:

   ```
   definePortInstancePMCellEvent (geGetEditCell)
     "CPU/SWITCH1" "VDDT" 1
     '(EN_in) 1 '(7.991152 r)
   definePMCellPortInstTransition (geGetEditCell)
     "CPU/SWITCH1" "EN"
   ```

4. Run the `set_rail_options -config_file` command and specify the PrimeRail `WAKEUP_THRESHOLD` and `SWITCH_CELL_EVENT_FILE` Scheme options in the specified configuration file for performing inrush analysis.

   In the configuration file, specify the `WAKEUP_THRESHOLD` and `SWITCH_CELL_EVENT_FILE` options in the following syntax:

   ```
   define WAKEUP_THRESHOLD float
   define SWITCH_CELL_EVENT_FILE file_name
   ```

   The `WAKEUP_THRESHOLD` option is used to generate the power management control file. The `WAKEUP_THRESHOLD` is the percentage of supply voltages considered as the stable controlled voltage level after power-up in inrush analysis. This option is not required. The default value is 0.01.

   The `SWITCH_CELL_EVENT_FILE` option specifies the Scheme formated switch cell power-up event or sequence file. The following example shows the syntax for the file. You can specify one configuration file at a time. Multiple files are not supported.

   ```
   ...Scheme Syntax
   ,,,
   ;definePortInstancePMCellEvent (geGetEditCell) "inst_name"
   "pg_port_name" num_port '(p1 p2...) num_event '(t0 s0 t1 s1...)

   ;definePMCellPortInstTransition (geGetEditCell) "inst_name"

   "signal_port_name" riseSlew fallSlew
   ```

   You can use the `definePortInstancePMCellEvent` Scheme command to annotate multiple events on multiple control pins of a power-switch cell instance. The second command argument is the switch cell instance name; the third is the switch input power pin; the fourth is the number of control pins; the fifth is the switch enable input pin list; the

sixth is the number of events; and the seventh is an event pair list of an arrival time, either a “r” for rise or “f” for fall or a “1” for 0 or “x” state to indicate the event for all the control pins.

The following example illustrates a sample event file.

```
definePortInstancePMCellEvent (geGetEditCell) "Mult/header1_C0"
"TVDD" 2 ’(NSLEEPIN1 NSLEEPIN2) 2 ’(15.258717 r0 123.856032 1r)
```

where

’r0’ indicates `NSLEEPIN1` is in the “r” (rise) state and `NSLEEPIN2` is in the “0” state.

5. Run the `analyze_rail -inrush` command to perform inrush analysis, based on the arrival time of the power-switch control pin.

   When you use the `-inrush` option, the `analyze_rail` command calculates the inrush current and ramp or wake-up time, based on the power management cell control file that was created earlier. The tool performs dynamic voltage drop analysis and displays the voltage drop analysis in a power map file.

   Note:
   The inrush analysis ignores the `set_rail_options -analysis_mode static` setting and performs the analysis in dynamic mode.

   After the inrush analysis is complete, PrimeRail writes out a text file that contains an analysis summary on the leakage current, peak inrush current, and wake-up time estimation.

   The tool generates the following outputs in the ./pr_run directory:

   - Fast Signal Database (FSDB) that contains the following waveform information:

     - Total rush current waveform for the switch cell subgroup

     - Rush current waveform through each power-switch cell

     - The voltage waveform for the virtual power and ground net

     By default, the generated FSDB file is named inrush.fsdb and can be viewed in a waveform viewer, such as WaveView or nWave.

   - Log file: The log file generated during inrush analysis includes the following information:

     - Wake up time: The time required for the virtual power and ground net to reach the same voltage value as the actual power and ground net. The following report is generated by switch cell analysis in PrimeRail and shows the wake-up time for a specific switching domain.

       ```
       Power Management Analysis Report

       Controlled voltage domain 1: DSPVDD
       240 power management cell channels are used to control 142711 std
       ```

```
cells.
      Total OFF leakage current from PM cells is 248.4e-3 (uA).
      Total ON leakage current from std cells is 2.973 (uA).

0% (3.1e6) 10% 20% 30% 40% 50% (3.19e6) 60% 70% 80% 90% (3.194e6)
Stable PG net voltage 1.18(V) is reached at wakeup time 3.1e6
   [start at 3.1e6 duration 8.5e3] (ps).
Peak current 271.459e-3 (A) is required at 3.193e6 (ps).
Peak controlled voltage domain current 135.2e-3 (A) is observed at
3.1e6 (ps).

Max voltage difference 66.317 (mV) occurs at 3.193e6 (ps).
```

  - Leakage Information: The PrimeRail power management report also provides the total OFF-state leakage current and ON-state leakage current for each switching group.

- Map files: Because running inrush analysis also includes rail analysis, the tool also generates voltage drop and electromigration maps when the analysis is complete. You can view these map files with the `read_rail_maps` command in IC Compiler.

6. In the rush current profile, you can redesign the turn-on sequence if the wake-up time or leakage does not meet your requirements for on-peak current, leakage value, or desired wake-up time or if you want to redefine power control connectivity. The power management analysis report typically contains the number of power-switch cells used, the peak rush current value, the wake-up time, the off-state leakage, and the on-state leakage values. Based on the first default analysis results of peak rush current, off-state leakage, and wake-up time, you can derive the desired number of power management cells required.
7. In the rush current profile, if the wake-up time and leakage information is acceptable, go to the subsequent clock optimization stage with the `clock_opt` command.
8. Finish the detail route stage with the `route_opt` command and perform dynamic voltage drop analysis using inrush effects with the `analyze_rail -voltage_drop` command.

Performing Power and Ground Routing

To provide adequate routing resources for power and ground routes in your design, you should create power and ground rings and straps and perform prerouting of standard cell power and ground pins during design planning. Prerouting helps to ensure that enough space exists for power and ground routes during the detail routing phase of the flow. The following sections describe the steps that implement a preroute power plan in your design.

- Creating Logical Power and Ground Connections
- Adding Power and Ground Rings

- Adding Power and Ground Straps
- Prerouting Macro Power and Ground Pins
- Prerouting Standard Cell Power and Ground Pins
- Creating Preroute Vias
- Creating Pad Rings
- Other PG Editing Commands

Figure 8-15 illustrates a typical power and ground preroute structure implemented in IC Compiler.

Figure 8-15 Power and Ground Preroute Structure

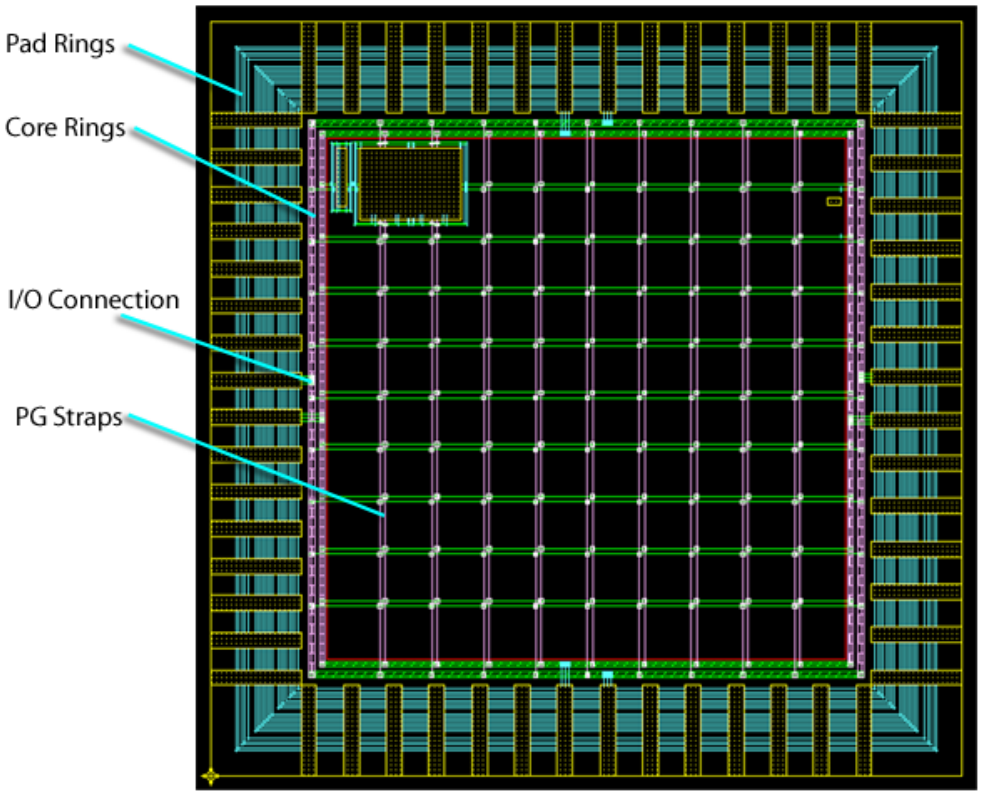
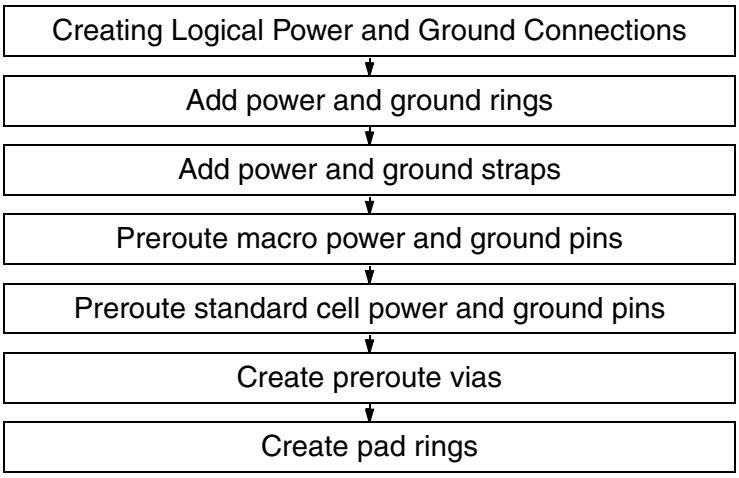


Figure 8-16 outlines the typical steps used to generate power and ground routes. The final step, creating pad rings, applies only to chip-level power and ground route generation.

Figure 8-16 Power and Ground Routing Flow



Creating Logical Power and Ground Connections

The `derive_pg_connection` command creates logical connections between power and ground pins on standard cells and macros and the power and ground nets in the design. For additional information about the `derive_pg_connection` command and its options, see "Connecting Power and Ground Ports" on page 2-5.

For single-voltage designs and designs without a UPF specification, the `derive_pg_connection -tie` command operates in manual mode and connects tie-off pins based on the `-power_net` and `-ground_net` options you specify. For UPF multivoltage designs, the `derive_pg_connection -tie` command operates in automatic mode and connects tie-off pins to the proper power nets consistent with the power connection for the cell instance.

The `derive_pg_connection` command checks for conflicts and other potential problems in the PG network. For multivoltage designs with power networks created by using a IEEE 1801 Unified Power Format (UPF) specification, the `derive_pg_connection -create_nets` command detects the following types of PG network issues:

- Incomplete PG network with a UPF conflict
- Signal tie net name that is inconsistent with the UPF supply net name
- Signal net with a PG name but without a PG source
- Signal net with a PG name at a middle hierarchy level connected to a signal at the top level

For multivoltage designs, the `derive_pg_connection -resolve_conflict` command modifies a PG network to make it compatible with the defined UPF power intent and reports the network changes. If the changes are satisfactory, you can use the `derive_pg_connection -create_nets` command, followed by the `derive_pg_connection` command, to complete the PG connections in your design according to the UPF power intent. You can also update the design by using editing or ECO commands to implement the changes manually. For more information about power planning for multivoltage designs, see "Performing Power Network Synthesis on Multivoltage Designs" on page 8-16.

Adding Power and Ground Rings

You can use the `create_rectangular_rings` command to create power and ground rings around the macros in your design. Table 8-15 describes the command options for the `create_rectangular_rings` command; these options can also be set by choosing Preroute > Create Rings in the GUI and selecting the Rectangular tab. For more information, see the man page.

Table 8-15 create_rectangular_rings Command Options

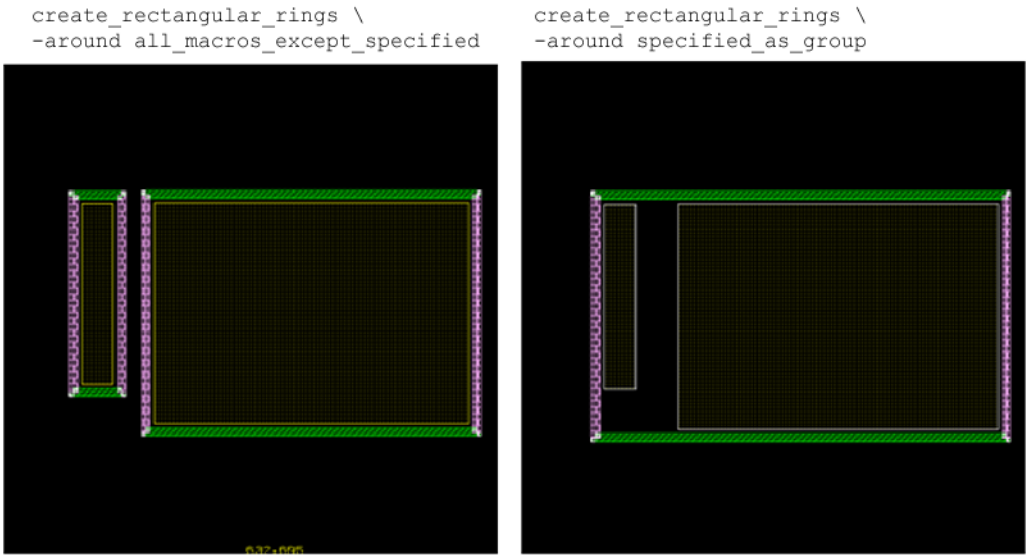
Command option	Description
`-advanced_via_rules` ("Use advanced via rules" check box in the GUI)	Enables the via insertion rules set by using the `set_preroute_advanced_via_rule` command.
`-around` *specification* ("Around" group of radio buttons in the GUI)	Defines the area around which to create a ring. The valid values for this option are `core`, `specified`, `specified_as_group`, `all_macros_except_specified`, and `rectangle`.
`-top_offset` *distance* ("Top" check box in the GUI)	Specifies the offset for the top side of the ring. The `-bottom_offset`, `-left_offset`, and `-right_offset` options specify the offset settings for the other segments of the ring.
`-top_segment_layer` *layer* ("Layer" box in the GUI)	Specifies the metal layer to use for the top segment of the ring. The `-bottom_segment_layer`, `-left_segment_layer`, and `-right_segment_layer` options specify the metal layers for the other segments of the ring.
`-top_segment_width` *width* ("Width" text box in the GUI)	Specifies the width of the top segment of the ring. The `-bottom_segment_width`, `-left_segment_width`, and `-right_segment_width` options specify the width of the other segments in the ring.

Table 8-15 create_rectangular_rings Command Options (Continued)

Command option	Description
`-skip_top_side` (Uncheck the “Top” check box in the GUI)	Skips creation of the top ring segment. The `-skip_left_side`, `-skip_right_side`, and `-skip_bottom_side` options block creation of segments on other sides of the ring.
`-cells` *`cell_list`* (“Macros” text box in the GUI)	Specifies the list of instances around which to create the ring or rings.
`-extend_bh` (“Right” check box in Bottom row in the GUI)	Extends rings to the top cell boundary or to the first target on the same net. The `-extend_ll`, `-extend_lh`, `-extend_rl`, `-extend_rh`, `-extend_bl`, `-extend_tl`, and `-extend_th` options extend the rings in different directions. The `-extend_for_multiple_connections` and `-extension_gap` options extend the rings past the first connection.
`-nets` *`net_names`* (“Nets” check box in the GUI)	Specifies the nets used to create the rings.
`-within` *`coordinates`* (“Coordinates” text box in the GUI)	Specifies the coordinates around which to create the ring.

Figure 8-17 illustrates two strategies for creating rings around macros by using the `create_rectangular_rings` command. The image on the left shows two separate rings created to surround each macro. The image on the right shows a single ring around the same group of macros.

Figure 8-17 create_rectangular_rings Ring Creation Strategies



If a blockage does not allow the tool to place the ring in the area you specified, the tool places the ring in the closest legal position. You can prevent the tool from automatically adjusting the ring position; however, the tool does not create the segment of the ring if the placement causes a DRC violation. You can change the internal application of the design rules by using the `set_preroute_drc_strategy` command. For more information about this command, see the man pages.

The following example creates a rectangular ring for the VDD net. The left and right sides of the ring use the METAL4 layer with a width of 1. The bottom and top sides of the ring use the METAL5 layer with a width of 1. The command extends the lower segment of the left side and the upper segment of right side with an offset of 0.5 from the core on all four sides.

```
icc_shell> create_rectangular_rings -around core -nets {VDD} \
 -left_segment_layer METAL4 -right_segment_layer METAL4 \
 -bottom_segment_layer METAL5 -top_segment_layer METAL5 \
 -left_segment_width 1 -right_segment_width 1 \
 -bottom_segment_width 1 -extend_ll -extend_rh \
 -left_offset 0.5 -right_offset 0.5 -top_offset 0.5 \
 -bottom_offset 0.5
```

You can also create rectilinear rings around the core area or macro cell instances in your design by using the `create_rectilinear_rings` command. Table 8-16 describes the command options for the `create_rectilinear_rings` command; these options can also be set by choosing Preroute > Create Rings in the GUI and selecting the Rectilinear tab. For more information, see the man page.

Table 8-16 create_rectilinear_rings Command Options

Command option	Description
`-around core | macros | all_macros_except_specified` (Radio buttons in the "Around" section of the GUI)	Defines the area where the rings are to be created.
`-create_bridges {`*`offset spacing`*`}` ("Create bridges" check box in the GUI)	Defines the offset and spacing for single wire segments that connect neighboring sections of the ring.
`-exclude_instances {`*`cells`*`}` ("Exclude specified macros and I/O pads" radio button on the GUI)	Specifies the macro cells to be excluded from the ring.
`-extension_of_ excluded_instances {`*`x y`*`}` ("Extend boundaries of excluded instances" check box in the GUI)	Specifies an additional boundary for excluded instances.
`-ignore_parallel_targets` ("Ignore parallel targets" check box in the GUI)	Prevents a ring from connecting to an object of the same net located over or under the segment.
`-layers {`*`h_layer v_layer`*`}` ("Layer" selection boxes in the GUI)	Specifies the layers to use when creating the vertical and horizontal ring segments.
`-macro_cells {`*`cells`*`}` ("Macros" text box in the Around section of the GUI)	Specifies the macros to encircle when using the `-around macros` option or the macros to exclude when using the `-around all_macros_except_specified` option.
`-nets {`*`nets`*`}` ("Nets" text box in the GUI)	Specifies the power and ground nets to use to create the rectilinear rings.

Table 8-16 create_rectilinear_rings Command Options (Continued)

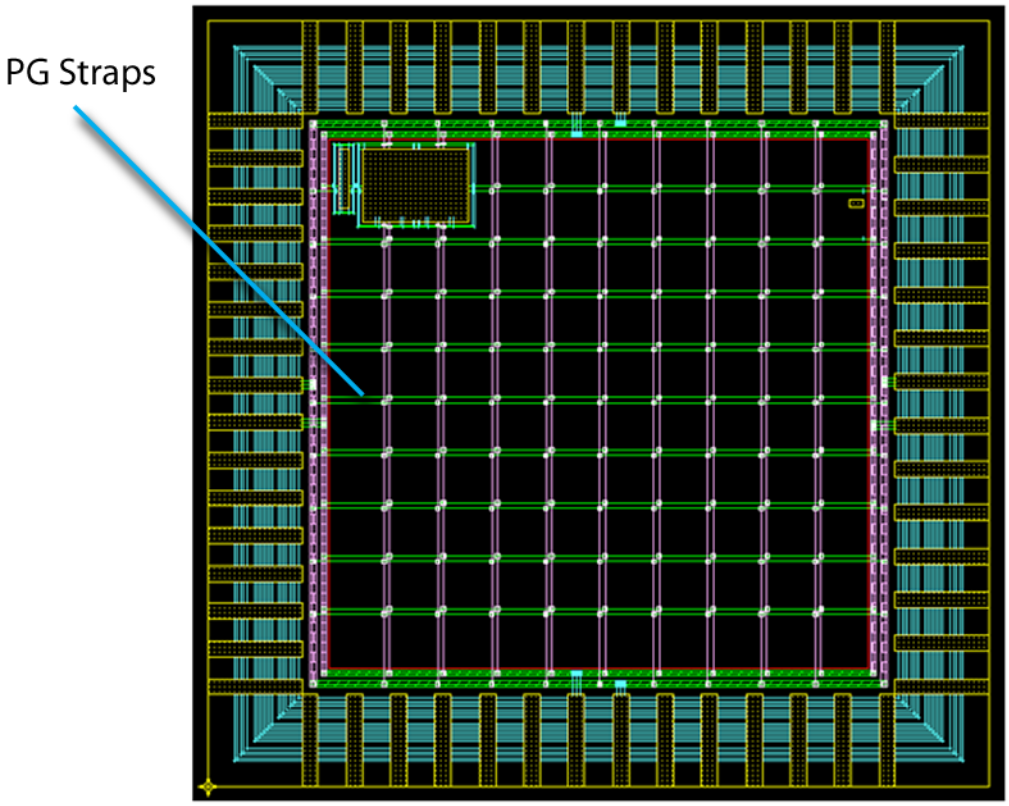
Command option	Description
`-offset {x_offset y_offset}` ("Offset" text boxes in the GUI)	Specifies the offset of the inner boundary of the ring from the boundary of the area around where the rings are created. Use the `-space` option to specify the minimum spacing between ring segments. Use the `-width` option to specify the width of the ring segments.

The `create_rectilinear_rings` command supports a list of macro cell instances to be excluded from the area bounded by the power and ground rings. You can use the command options to control the location of the rings and the width and layers of their segments. Note that the `create_rectilinear_rings` command does not support extension of the rings to the boundary as does the `create_rectangular_rings` command.

Adding Power and Ground Straps

After you add power and ground rings, use the `create_power_straps` command to create power and ground straps for your design. The `create_power_straps` command can automatically connect the straps to the closest power or ground ring at the ends of the straps. The `create_power_straps` command creates the straps and via arrays to form a power mesh. The command also extends the straps and connects them to the power rings. Figure 8-18 shows a design with a power mesh created by using the `create_power_straps` command.

Figure 8-18 Floorplan With PG Straps



All pins generated by prerouting commands are marked as fixed to prevent the `create_power_straps` command from moving them during other operations. You can use the `set_preroute_drc_strategy` command to change the internal application of the design rules. For more information about the `create_power_straps` and `set_preroute_drc_strategy` commands, see the man page.

The following example creates vertical straps on the METAL5 layer with a 1 micron width and a 4 micron spacing between each strap. The first strap is placed at x=7 microns and the number of straps to be placed is 10. You can use the `-step` option to control the spacing between the straps. Note that the step size is equal to the strap width plus the strap spacing.

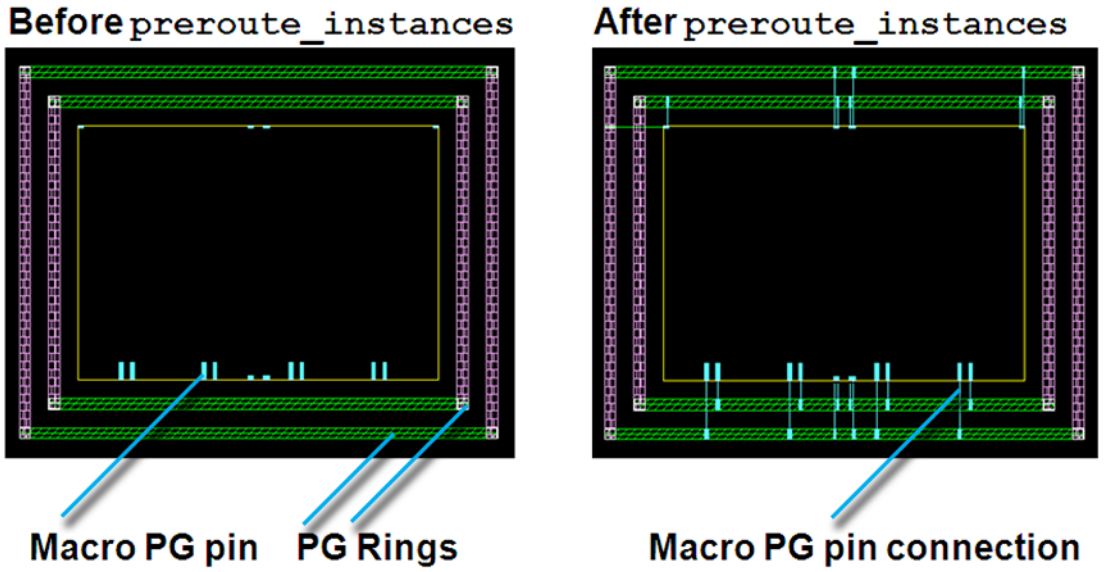
```
icc_shell> create_power_straps -direction vertical -nets {VDD} \
 -layer METAL5 -width 1 -configure groups_and_step -start_at 7 -step 5 \
 -num_groups 10
```

Prerouting Macro Power and Ground Pins

Use the `preroute_instances` command to build physical connections between power and ground pins on macros and pads to other pins on the same net. Figure 8-19 shows a design after running the `preroute_instances` command. You can change the design rules that are

used to check the connections by using the `set_preroute_drc_strategy` command. For more information about the `preroute_instances` and `set_preroute_drc_strategy` commands, see the man page.

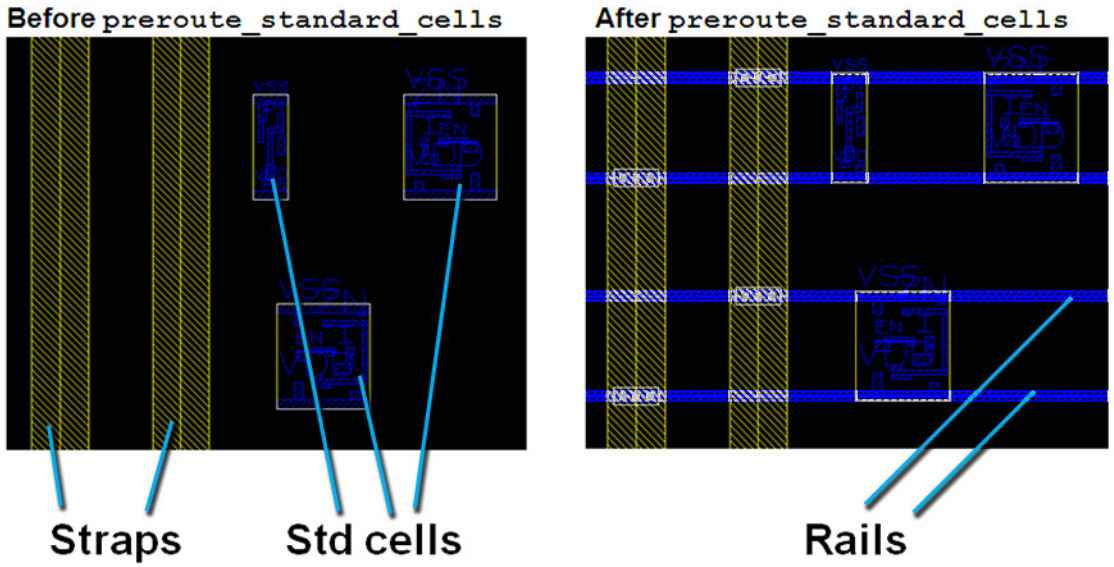
Figure 8-19 Floorplan After preroute_instances Command



Prerouting Standard Cell Power and Ground Pins

You can connect the power and ground pins on standard cells to the straps and rings in the power network by using the `preroute_standard_cells` command. To avoid possible obstructions during global routing due to power and ground routes to standard cells, use the `preroute_standard_cells` command before performing global routing. Figure 8-20 shows a design after running the `preroute_standard_cells` command.

Figure 8-20 Floorplan After preroute_standard_cells Command



If the design does not contain pad cells, you can use the `-extend_to_boundaries_and_generate_pins` option to create power and ground route extensions to the cell boundary and generate a power and ground pin. This option can be used if the following conditions are met:

- The power and ground pin is not already connected to power and ground in the direction in which IC Compiler would create the extension wire.
- Creating the power and ground extension wire does not cause a design rule violation. Pins are marked as fixed to prevent them from being moved by place and route operations.

Creating Preroute Vias

You can use the `create_preroute_vias` command to create vias between the specified layers on specified power and ground nets. The command can also be used to create vias between specified layers and other nets in the design.

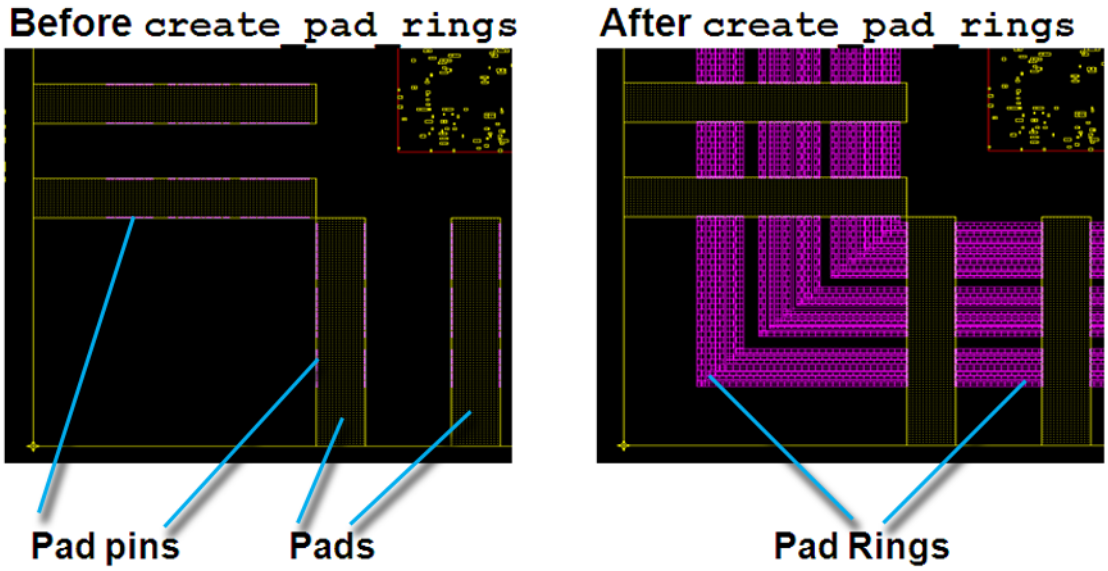
The following example creates preroute vias between layers METAL1 and METAL6 from the VDD power straps to the VDD pins of the standard cells within the rectangular region 100,100 and 500,500. Note that you must specify at least one source object and one target object when using the `create_preroute_vias` command.

```
icc_shell> create_preroute_vias -nets VDD -from_layer METAL1 \
 -to_layer METAL6 -to_object_strap -from_object_std_pin_connection \
 -within {{ 100.0 100.0 } { 500.0 500.0 }}
```

Creating Pad Rings

You can use the `create_pad_rings` command to create a pad ring and ensure connectivity between pad cells that do not abut. The `create_pad_rings` command creates pad rings only for pins in the boundary pads; the rings created by the command do not cover the pad pins. Figure 8-21 shows a pad ring generated by using the `create_pad_rings` command.

Figure 8-21 Floorplan after create_pad_rings Command



Other PG Editing Commands

IC Compiler supports several commands for editing PG nets and performing tie-off connections in your design. Table 8-17 summarizes the PG editing commands.

Table 8-17 Other PG Editing Commands

Command	Description
`derive_pg_connection`	Performs the automatic connection of power and ground pins and direct rail-tie connections.
`connect_net`	Connects the specified PG net to the specified pins or ports.

Table 8-17 Other PG Editing Commands (Continued)

Command	Description
`disconnect_net`	Breaks the connections between a net or a net instance and its pins or ports.
`connect_tie_cells`	Specifies the cell and port names to use when connecting tie-off inputs in your design.
`recover_tie_connection`	Disconnects and reconnects tie-off connections from PG nets to tie-high and tie-low pins.
`report_tie_nets`	Reports the number of tie-high nets and tie-low nets and the connected pin and port for each tie net in the current design.
`report_pg_net`	Reports information about the power and ground nets for the currently opened Milkyway design.
`create_net`	Creates power and ground nets, tie-off nets, as well as signal nets in the current design
`create_pg_network`	Creates a new hierarchical power or ground network at any level of hierarchy. The newly created PG network connects multiple hierarchical cell instances across selected hierarchical boundaries.
`remove_pg_network`	Removes hierarchical power and ground connections from specified hierarchical cell instances and their child modules or from the entire design.
`remove_tie_cells`	Removes the tie cells specified by a given collection. The signal pins that were originally driven by the deleted tie cells are connected to the default Milkyway tie-high or tie-low nets.

9

Performing Prototype Global Routing

You can perform prototype global routing to get an estimate of the routability and congestion of your design. Global routing is done to detect possible congestion hot spots that might exist in your floorplan due to the placement of the hard macros or inadequate channel spacing.

This chapter includes the following sections:

Setting Up for Routing

You can specify general routing setups for IC Compiler to use whenever you perform routing. For information about setting up for routing, see the “Routing Using Zroute” chapter in the *IC Compiler Implementation User Guide* or the “Setting up for Routing” chapter in the *IC Compiler Classic Router User Guide*.

Running Global Prototype Routing

During global routing, IC Compiler assigns nets to the global routing cells through which they pass. For each global routing cell, the routing capacity is calculated according to the blockages, pins, and routing tracks inside the cell. Although the nets are not assigned to the actual wire tracks during global routing, the number of nets assigned to each global routing cell is noted. IC Compiler calculates the demand for wire tracks in each global routing cell and reports the overflows, which are the number of wire tracks still needed after IC Compiler assigns nets to the available wire tracks in a global routing cell.

IC Compiler might reduce overflows by detouring nets around congested areas and increasing the wire length. You can examine the global routing report that appears in the command window and display congestion maps to help you decide whether your design can be routed.

The global router considers spacing and wide-wire variable routing rules as well as shielding variable routing rules, when calculating congestion.

Global routing is divided into the following phases:

- One initial routing phase, where all the unconnected nets are routed
- One or more rip-up and reroute phases, where for a selected set of nets, the routing results from the previous phase are deleted and nets are rerouted to reduce the congestion

Note:

If, after the second routing phase, the maximum overflow in any direction is greater than 50, both global routing and prototype routing stop because the design is too congested and it is now unroutable.

To perform global routing,

1. Choose Route > Global Route.

   The Zroute dialog box appears.

2. Select the “Minimum” radio button in the Effort section of the GUI. This option enables prototype global routing.
3. Click OK or Apply.

Alternatively, you can use the `route_zrt_global -effort minimum` command.

10

Performing Clock Planning

This chapter describes how to reduce timing closure iterations by performing clock planning on a top-level design during the early stages of the virtual flat flow, after plan groups are created and before the hierarchy is committed. You can perform clock planning on a specified clock net or on all clock nets in your design.

For hierarchical timing closure, clock planning can also be used to generate realistic clock latency through timing budgeting to fix timing violations.

Clock planning tries to minimize clock skew by running block-level and top-level clock tree synthesis during the early stages of the design flow to determine the clock budgets, allocate resources for clock buffers and clock routes, determine optimal clock pin locations for soft macros, and provide an estimate of the block-level insertion delays and skew for each plan group prior to finalizing the floorplan. Having optimal clock pin locations is a key factor in meeting the final clock skew and insertion delay numbers with the optimal number of buffers.

This chapter includes the following sections:

- Performing Plan-Group-Aware Clock Tree Synthesis in Clock Planning
- Supporting Abutted Floorplans in Hierarchical Clock Planning

Setting Clock Planning Options

Before you can compile clock trees inside the plan groups and build clock trees at the top level, you must first set different clock planning options such as anchor cell insertion, specifying nets for clock planning, and whether or not to route the clock nets after clock planning. You can also modify the clock tree constraint settings.

To set clock planning options,

1. Choose Clock > Set Clock Plan Options.

   The Set Clock Plan Options dialog box opens.

   Alternatively, you can use the `set_fp_clock_plan_options` command.

2. Set the options, depending on your requirements.

   - Clock Nets – Enter the name of the clock nets on which to do clock planning.
   - No Feedthroughs in Plan Groups – Enter a list of plan groups on which you do not want buffers placed. During the top-level clock tree synthesis phase of clock tree planning, no buffers are placed on the plan groups, unless they drive sinks inside those plan groups. This minimizes the creation of feedthroughs.
   - Anchor Cell – Enter the name of the cell that is inserted as an anchor cell for all the plan groups. The anchor cell is a clock buffer cell. All plan groups should use the same buffer type. Inverters are not supported. This option is required. If you do not specify the name of the anchor cell, the IC Compiler tool issues an error message.

     For each clock interface net (nets that cross plan group boundaries), an anchor cell (driver cell) is inserted for each plan group on every input clock net that crosses a hierarchical block. This partitions the plan group level clock subtree from the top-level clock tree.

     An isolation cell is also inserted at the top level (for each output clock port on the plan group) to isolate the plan group level clock subtree from the top-level clock tree.

     The anchor cell is inserted inside the plan group at the center of the mass of flip-flops that are connected to each clock net to isolate the top-level clock net from the clock net that is inside the plan group. The input pin of the anchor cell is driven by the clock net, and the output pin of the anchor cell drives the root clock pin of the block.

   - Route Mode – Choose whether or not to route the top-level clock nets after clock planning. Based on the routing information, clock pins are created and assigned a location where the route crosses the plan group boundary.

Global route – Select this option to perform global routing on the clock nets.

Detailed route – Select this option to perform detail routing on the clock nets.

None – Select this option if you do not want to route the clock nets. This is the default.

- Output directory – Enter the name of the directory in which to write out all constraint files, log files, and generated reports from clock planning. The default directory is /cp_output.
- Keep block level tree – Select this option if you want to keep the top-level clock tree buffers inside the plan groups during clock planning. The default is to remove the block level clock tree when clock planning is complete.

  Clock tree planning might, for example, show a large insertion delay and skew value on one of the blocks in your design. By keeping the clock tree buffers inside the plan groups, you can more easily analyze them to determine why clock tree planning shows such a large value.
- Set clock tree options – Select this option to open the Set Clock Tree Options dialog box.

3. Click OK or Apply to set the clock planning options.

Reporting Clock Planning Options

You can get a report of the clock plan options by using the `report_fp_clock_plan_options` command. If no clock plan options have been set, this command reports the default values.

Removing Clock Planning Options

You can remove (reset) the database entries for the clock planning options you set by using the `reset_fp_clock_plan_options` command.

Performing Clock Planning Operations

Clock planning is done during the early stages of the virtual flat flow (after plan groups have been created and before the hierarchy is committed) to allocate clock resources and provide an estimate of the block-level insertion delay and skew for each plan group, prior to finalizing the floorplan.

You can perform clock planning operations to compile the clock trees inside plan groups and build clock trees at the top level based on the options you have selected in the Set Clock Plan Options dialog box (`set_fp_clock_plan_options` command).

To perform clock planning operations,

1. Choose Clock > Compile Clock Plan.

   The Compile Clock Plan dialog box appears.

   Alternatively, you can use the `compile_fp_clock_plan` command.

2. Set the options, depending on your requirements.

   - Operation Condition – Select the operating conditions (Max, Min, or Min/Max) for top-level clock tree synthesis and optimization. The default is Max.
   - Insert Anchor only – If you select this option, the clock planning tool only inserts anchor cells on the input ports of the plan groups, and does not synthesize the clock plan.

     By default, the clock planning tool inserts anchor cells on the input ports of the plan groups plan groups, and then synthesizes the clock plan.

3. Select OK or Apply.

   During clock planning, the following operations are performed:

   - Anchor cells are inserted on the input ports of the plan groups to isolate the clock trees inside the plan groups from the top-level clock tree.
   - Fast clock tree synthesis is run inside each plan group to estimate the insertion delay and skew values at the input of the anchor cells.
   - The clock tree synthesis results are annotated on floating pins, which are defined on the anchor cells.
   - The top-level clock tree is synthesized to the anchor cell floating pins.
   - Detail routing is run on the clock interface nets.

Generating Clock Tree Reports

You can use clock skew analysis to generate a skew report for a specified clock (or for all the clocks within a design) before or after routing. You can view the report in a text window or write it to a specified file.

To generate clock tree reports, choose Clock > Report Clock Tree or use the `report_clock_tree` command. By default, the global skew is reported for all the generated clock trees.

Generating Clock Network and Source Latency for Each Clock Pin of Each Plan Group

For hierarchical timing closure, clock planning is also used to generate realistic clock latency through budgeting to fix timing violations. Top-level timing violations result not only from delays on combinational logic between registers, but also from clock skew between launching and capturing registers.

When you perform clock planning operations, it enables the timing budgeter to generate clock network and clock source latencies for each clock pin of each plan group in the design. For more information, see “Performing Clock Latency Budgeting” on page 13-29.

- Clock network latency is the time a clock signal (rise or fall) takes to propagate from the clock definition point in the design to a register clock pin.

  Example:

  ```
  set_clock_latency 2 -max -rise [get_clocks CLK]
  ```

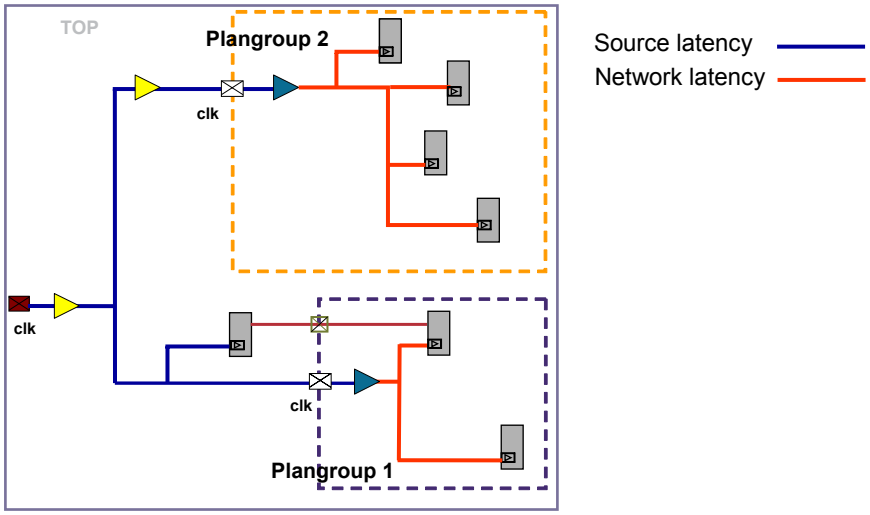
- Clock source latency is the time a clock signal takes to propagate from its ideal waveform origin point to the clock definition point in the design.

  Example:

  ```
  set_clock_latency 2 -source -max -rise [get_clocks CLK]
  ```

Figure 10-1 shows an example of the clock network latency and the clock source latency.

Figure 10-1 Clock Network and Source Latency



Creating a Virtual Clock for I/O Paths

For each plan group, a virtual clock can be created to describe clock latency outside of the plan group

The format for defining virtual clocks created for clock latencies outside the plan group is

- Clocks launching flops of input paths:

  *clock_name*_v_in

- Clocks capturing flops of output paths:

  *clock_name*_v_out

Note:
To minimize the number of virtual clocks, only the worst-case clock latency is created for all input and output pins of plan groups launched or captured by a virtual clock.

Figure 10-2 on page 10-7 shows a virtual clock example.

- In plan group 1, the name of the input path is in1.

  clk1_v_in launching flop A on the top level has source latencies only.

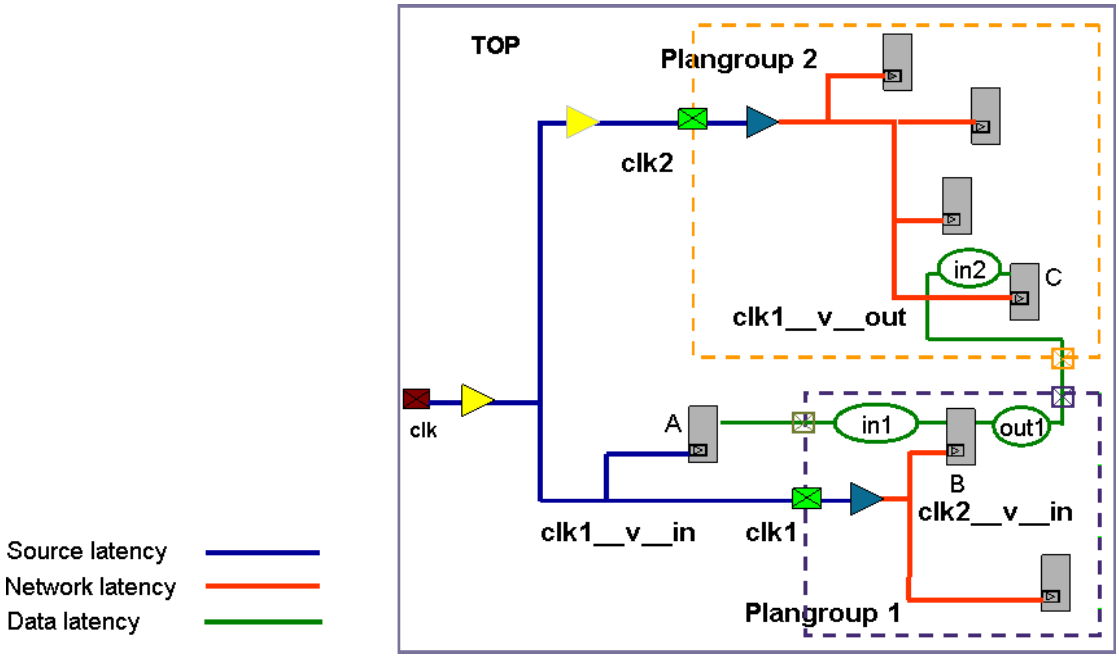
- In plan group 1, the name of the output path is out1.

  clk1_v_out (clk2) capturing flop C in plan group 2 has both source and network latencies.

- In plan group 2, the name of the input path is in2.

  clk2_v_in (clk1) launching flop B in plan group 1 has both source and network latencies.

Figure 10-2 Virtual Clock Example



Using Multivoltage Designs in Clock Planning

Clock planning supports multivoltage designs. Designs in multivoltage domains operate at various voltages. Multivoltage domains are connected through level-shifter cells. A level-shifter cell is a special cell that can carry signals across different voltage areas.

Clock planning isolates each plan-group-level clock from the top-level clocks by inserting an anchor cell. However, if you are using multivoltage designs in your clock planning, level-shifter cells should have already been inserted into the voltage area of the plan group.

Interpreting Level-Shifter Cells as Anchor Cells During Clock Planning

If there is a level-shifter cell for an interface clock net (a net that comes from the top-level clock net into a plan group), it is interpreted as an anchor cell. Then, during clock planning, anchor cells are inserted for each interface clock net only as long as they do not cross voltage areas. This prevents the insertion of buffers and inverters into the wrong voltage areas. Both plan-group-level clocks and voltage areas now isolated from the top-level clocks.

Note:
Feedthroughs created by clock planning might not honor the voltage area constraints. For all designs in multivoltage domains, you should specify the “No Feedthroughs in Plan Groups” option on the Set Clock Plan Options dialog box (`set_fp_clock_plan_options` command).

Performing Plan-Group-Aware Clock Tree Synthesis in Clock Planning

You can perform plan-group-aware clock tree synthesis in clock planning. With this feature, clock tree synthesis can:

- Generate a clock tree that honors the plan groups while inserting buffers into the logic hierarchy tree or, if it exists, into the corresponding physical region. The physical region can be a voltage area, an exclusive move bound, or a plan group.
- Prevent new clock buffers from being placed on top of a plan group unless they drive the entire subtree inside that particular plan group. This results in a minimum of clock feedthroughs, which makes the design easier to manage during partitioning and budgeting.

To perform plan-group-aware clock tree synthesis in clock planning with fully abutted floorplans, set the following variable:

```
set cp_full_abut_cts_region_aware true
```

To control plan-group-aware clock tree synthesis in clock planning, click in the text box next to the “No Feedthroughs in Plan Groups” option on the Set Clock Plan Options dialog box and enter a list of plan groups where clock feedthroughs should not be generated.

Supporting Abutted Floorplans in Hierarchical Clock Planning

You can perform hierarchical clock planning (`compile_fp_clock_plan` command) on designs with fully abutted floorplans.This can help solve possible DRC violations on nets going through different plan groups.

To do this, clock tree synthesis must be plan-group-aware (See “Performing Plan-Group-Aware Clock Tree Synthesis in Clock Planning”).

From a placed design, set the following variable to perform hierarchical clock planning on fully abutted floorplans:

```
set cp_in_full_abut_mode true
```

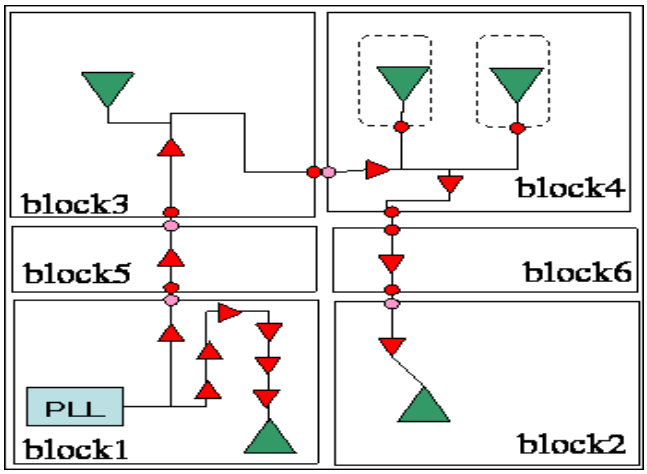
Run the `compile_fp_clock_plan` command to compile the clock trees inside plan groups and build clock trees at the top level.

Note:

Clock tree planning should resolve any DRC violations on nets going through different plan groups in the design, and the ideal clock buffer location is far from the plan group that is associated with the logical hierarchy to which the buffer is added.

Figure 10-3 shows an example of a fully abutted floorplan.

Figure 10-3 Fully Abutted Floorplan



11

Performing In-Place Timing Optimization

In-place timing optimization is used to improve the timing of a given design, and in particular, to meet the timing constraints on the design. It is an iterative process based on virtual routing. If you run in-place optimization after global routing, the tool deletes the global routes and optimizes the design based on the virtual route timing.

This chapter includes the following sections:

The optimization process starts with the most significant changes on multiple violation paths that will quickly improve overall timing. Next, the scope is narrowed to focus on individual paths. Netlist changes are committed only if the timing improves. The output is a legally placed netlist that is plan group aware.

Three types of optimizations are performed: timing improvement, area recovery, and fixing timing design rule violations. These optimizations preserve the netlist's logical hierarchy, and the physical locations of most cells change as little as possible.

Many methods for improving timing are used, including

- Inserting buffers and inverters (legal locations are found automatically)

  In the case of inverters, it is guaranteed that the polarity of the signals is preserved

- Increasing or decreasing cell size (If the cell size is increased, the larger cell is automatically adjusted to ensure that all cells are placed in legal locations.)
- Moving cells

  Similar to cell sizing, the cells are placed in legal locations

Running In-Place Timing Optimization

To perform in-place timing optimization,

1. Choose Timing > In Place Optimization.

   The In Place Optimization dialog box appears.

2. Set the options in the GUI depending on your requirements.

   - Add buffers for feedthrough nets only

     Enable this option to only add buffers for feedthrough nets. Optimization is not performed and timing is not updated. Using this option minimizes design changes in the late stages of design planning. The default is disabled.

     You must run the `analyze_fp_routing` command before you use this option; otherwise, the tool issues an error message and stops.

     The tool automatically selects medium-sized buffers and inserts them near each pin (port) location determined by the `analyze_fp_routing` command. During the optimization phases, in-place optimization might resize or move these buffers. The buffers are added in the same hierarchy as the plan groups.

- Perform high fanout synthesis optimization only

  Enable this option to perform only high-fanout synthesis (HFS) on the design. You can combine this option with the “Don’t add any new cells at top level” option when implementing fully abutted or narrow channel designs.

  The default is disabled.

  Use the `optimize_fp_timing -hfs_only` command to perform normal high-fanout synthesis. For incremental high-fanout synthesis, use the `set_ahfs_options -incremental true` command, before running the `optimize_fp_timing -hfs_only` command. Note the following limitations for the regular and incremental high-fanout synthesis flows:

  - Regular high-fanout synthesis is recommended for the normal flow
  - Incremental high-fanout synthesis is required for the exploration flow in order to reduce runtime
  - Regular high-fanout synthesis in the exploration flow is not well suited for analyzing the clumped placement of pre-existing buffers
  - Incremental high-fanout synthesis does not analyze preexisting buffers and their placement
  - Incremental high-fanout synthesis is optional for the regular high-fanout synthesis flow

  These recommendations are based on the following QoR metrics: worst negative slack (WNS), total negative slack (TNS), design rule check (DRC), and runtime.

- Effort

  You can specify how much effort is used to minimize the worst negative slack in the design. If you select an effort level of high, more effort is expended to improve the timing of the design, resulting in more CPU time. In-place optimization stops when it finds the worst negative slack in the design cannot be further improved. The output is a legally placed netlist.

  The default effort is medium.

- Fix design rules violations

  Enable this option to fix design rules. The default is disabled. Design rule violations, such as maximum transition, maximum capacitance, and maximum fanout, are fixed by use of buffer insertion, gate sizing, and automatic high-fanout net synthesis for handling medium- and high-fanout nets.

When you run in-place optimization at the virtual route stage, only obvious design rule violations are fixed because wire locations and interconnect are estimated at this stage since timing analysis cannot be completely accurate and runtime is shorter. After you finish global routing, optimization takes longer to run, but the results are based on more accurate timing information.

- Enable area recovery

  Enable this option to direct the in-place optimization engine to invoke additional operations to try to reduce the cell area without causing timing violations. The area is optimized by removing cells and decreasing the size of the cells on noncritical timing paths. This allows more space for optimization on critical paths.

  The default is disabled.

  Note:

  Area recovery is an operation that tries to reduce the total area of cells used in the design. Area recovery will not cause timing to become worse on the critical paths, but some paths might see an increase in timing as long as it is nonviolating. If the path has positive slack, area recovery might reduce this slack to zero. Area recovery does not cause nonviolating paths (paths with nonnegative slack) to become violating paths.

  Area recovery, however, is a difficult and expensive operation, resulting in longer runtime but with a smaller area being used. Designs with high utilization will benefit from area recovery but with a cost that results in higher runtimes.

- Keep global routes

  Use this option if you want to preserve the global routes after timing optimization. By default, the tool removes global routes from the design.

- Don’t add any new cells at top level

  Enable this option to prevent new cell insertion at the top level of the design during in-place optimization. You can combine this option with the “Perform high fanout synthesis optimization only” option when implementing fully abutted or narrow-channel designs.

  By default, new cells can be inserted at the top level of the design.

- Report quality of results

  Enable this option to get reports on the worst negative slack (WNS) and total negative slack (TNS) of the design, the utilization percentage of each plan group and top level, the number of buffers added, and the number of cells sized for each plan group and top level.

  By default, the command does not report the quality of results.

3. Click OK or Apply.

Alternatively, you can use the `optimize_fp_timing` command.

Running Trace Mode In-Place Optimization

Use the `optimize_fp_timing` command or choose Timing > In-Place Optimization in the GUI to perform in-place optimization. To improve capacity and runtime, you can run the in-place optimization in trace mode by setting the `set_fp_trace_mode` command. In this mode, the tool optimizes all top-level and interface paths or interface timing paths only.

The syntax is

```
set_fp_trace_mode
[-include_top_logic]
[-verbose]
```

The default is to consider only interface logic.

Note:
To check if a design is in trace mode, use the `get_fp_trace_mode` command.

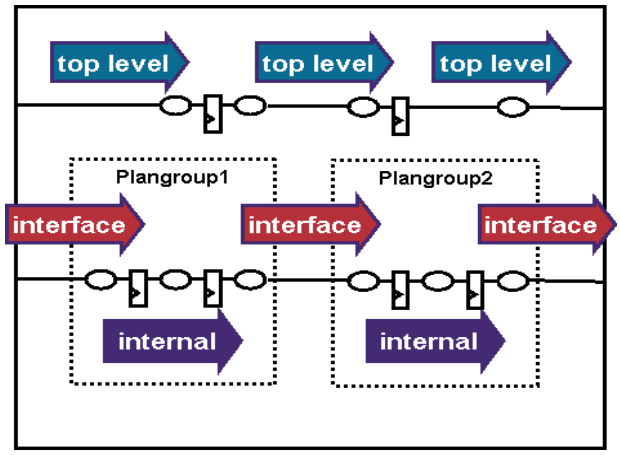
In trace mode, the tool considers only the top-level and interface paths even though the entire design is loaded.

When trace mode timing is used in a flat design with plan groups defined for soft macros, trace mode timing selectively filters paths. This is done by tracing only the nets and cells of timing paths at the top level which cross plan group boundaries. Accordingly, the `report_clock_timing` command reports only the timing violations on those paths. By using trace mode timing analysis, you can detect and fix these timing violations in the early phases of the design flow. This enables faster timing analysis by focusing on interface nets.

During the trace mode in-place optimization stage, only the interface logic or interface logic plus top-level logic is optimized. This results in good QoR and might improve runtime.

Figure 11-1 on page 11-6 illustrates the top-level, internal, and interface paths for trace mode timing.

Figure 11-1 Top-level, Interface, and Internal Paths for Trace Mode Timing



Reporting Trace Mode Status

You can report the current option settings for trace mode by using the `report_fp_trace_mode_options` command. The command shows whether top-level and interface logic are included in the timing optimization. The following example shows a report generated by the `report_fp_trace_mode_options` command.

```
icc_shell> report_fp_trace_mode_options
Trace mode is set with following options:
  interface   : included
  top         : not included
  verbose     : off
1
```

Removing the Trace Mode

To remove trace mode timing and return to the normal mode for timing analysis, use the `end_fp_trace_mode` command. This command removes the trace mode design from memory. After the next command that accesses the netlist of the design is used, it triggers a reload of the entire design netlist.

Using In-Place Optimization With Multivoltage Designs

The `optimize_fp_timing` command detects multivoltage settings and performs multivoltage-aware optimization. By following predefined multivoltage design rules, in-place optimization improves timing by swapping cell instances. The command replaces cell instances with cells of a different sizes from within the same power domain and inserts buffers that obey multivoltage design rules.

IC Compiler handles always-on nets and honors power guides created by the `set_power_guide` command during feedthrough buffer optimization. If a feedthrough net has a power guide setting, IC Compiler uses the appropriate buffers to satisfy the setting. IC Compiler uses normal buffering on nets that do not have a power guide setting.

Performing In-Place Optimizations Based on Pin Locations

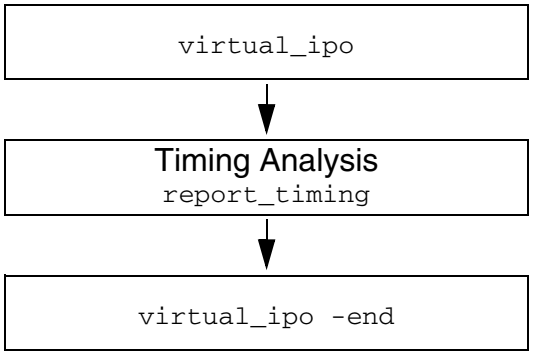
You can use the `optimize_fp_timing` command after pin cutting. Timing optimization understands pin locations decided by the previous pin cutting command when performing timing optimization. It chooses medium-sized buffers and inserts them near each pin (port) location determined by the `analyze_fp_routing -finalize_pins_feedthroughs` command.

Virtual In-Place Optimization

The `virtual_ipo` command performs virtual in-place optimization on the design and predicts timing after physical synthesis. The command estimates the timing of the design by using analytical models and does not change the placement or the netlist. As a result, the `virtual_ipo` command can complete faster than the `optimize_fp_timing` command. The output from the `virtual_ipo` command contains estimated and annotated delays of cells and nets along paths that violate timing. When you run the exploration flow, you should first add buffers on feedthrough nets by using the `optimize_fp_timing -feedthrough_buffering_only` command before you run the `virtual_ipo` command.

To remove timing data after performing virtual in-place optimization, use the `virtual_ipo -end` command, which removes all back-annotation data inserted during the previous virtual in-place optimization run. This operation is required after you have performed timing closure using virtual in-place optimization, before proceeding to physical optimization as shown in Figure 11-2.

Figure 11-2 Virtual In-Place Optimization Flow



For more information about virtual in-place optimization, see Table 6-7 on page 6-26.

12

Performing Routing-Based Pin Assignment

You can create top-level soft macro pins in hierarchical designs, constrain the pins during pin creation, edit and modify the soft macro pins, and analyze the quality of the pin assignment results. Both pad and pin assignment are supported concurrently.

IC Compiler provides two ways to perform pin assignment: on soft macros using the traditional pin assignment or on plan groups, using the pin-cutting flow.

This chapter includes the following sections:

- Removing Feedthrough Ports and Nets From Blocks and Voltage Areas
- Refining the Pin Assignment

Setting Pin Assignment Constraints

You can set constraints prior to performing pin assignment. These pin constraints are honored during pin assignment on soft macros during traditional pin assignment and on plan groups in the pin-cutting flow. The pin assignment constraints are saved in the Milkyway database.

This section includes the following topics:

- Specifying Physical Design Constraints on Pins
- Reporting Pin Assignment Constraints

Various pin constraint settings are available and can be applied to all soft macros or to a selected set of soft macros. Pin creation layers can be limited to a set of selected layers. Pins can be constrained to avoid corner placement by various factors. You can also set the spacing between pins and preroutes.

To assign pin constraints,

1. Choose Pin Assignment > Pin Constraints.

   The Pin Assignment Constraints dialog box appears.

   You can apply constraints to all soft macros and plan groups or to specified soft macros or plan groups. By default, the specified constraints apply to all soft macros and plan groups.

   You can apply constraints to block-level pins instead of soft macro pins. To do this, select the “Block level” option. The default is disabled.
2. Click the Add button to open the Add Pin Assignment Constraints dialog box.
3. Click the Settings tab and complete the necessary constraint settings.

   Allowed layers from – Select this option to specify the range of consecutive metal layers on which to place soft macro pins.

   - Specify the range of allowed layers, from the lowest to highest allowed layers.

   Retain existing pins – Select this option when you do not want the pin assignment to delete movable pins. The default is off.

   Pin stacking – Specifies how to handle pin overlaps across layers. You can select the following options:

- No Stacking – Select this option when you do not want to overlap pins on two different layers (for example, metal1 pins with metal3 pins). This is the default.
- Stacking OK – Select this option if you want signal, and power and ground pins to be stacked.
- Allow signal-signal Stacking – Select this option if you want only signal pins to be stacked.
- Allow signal-power/ground Stacking – Select this option if you want both signal, and power and ground pins to be stacked.

Hard constraint for pin – Select this option if you want the pin-to-pin spacing constraint,location or layer to be treated as a hard constraint rather than a soft constraint. Pin assignment always tries to honor the constraints you set. However, if a design is very congested and there are not enough wire tracks to allocate all the pins, the pin-to-pin spacing constraint might be violated. To prevent this violation, select this option.

The default is off and the pin spacing constraint is treated as a soft constraint.

- Spacing – If you select this option, the pin-to-pin spacing constraint is treated as a hard constraint. If there are insufficient pin slots to honor the constraint, pin assignment issues an error message.
- Location – If you select this option, the pin location constraint is treated as a hard constraint. Pins are created at the exact locations. If there are pin spacing violations, pin assignment issues a warning message.
- Layer - If you select this option, Pins are created at the specified layer based on the physical constraints.

Pin spacing – Pins that are created are always snapped to wire tracks. Specify the number of wire tracks in the spaces between adjacent pins. The default pin-to-pin spacing is 1 extra wire track between pins.

Pin preroute spacing – Specify the number of wire tracks between adjacent preroutes and pins. The default distance between preroutes and pins is 3 wire tracks.

To allow enough space for routing, no pins are created on the section of a soft macro edge that is parallel to a preroute if that edge is closer to the preroute than to the number of wire tracks you specify. Pins can be created on other sections of the soft macro edge if the distance from that section of the soft macro edge is at or at least equal to the pin-to-pin spacing.

Prevent corner pins by – Specify the distance from the corners of soft macros where pins should begin. You can specify the distance two ways:

- Number of wire tracks – Specify the corner keepout spacing as the number of wire tracks from the corner where the pin assignment should not create pins. The default is 5 wire tracks.

The pin assignment engine takes the number of wire tracks from the corners specified in the dialog box, multiplies it by the maximum metal layer pitch (the maximum wire track distance for metal layers from minimum to maximum), and then uses the resulting distance as the corner spacing. For example, if the minimum layer is metal3 and the maximum layer is metal6, and metal6 pitch is the largest, the corner spacing is calculated as metal6 pitch multiplied by the number of wire tracks from the corners of soft macros.

- Percentage of Macro side length (%) – Specify the corner keepout spacing as a percentage of the edge length where the pin assignment should not create pins. The default is 5.0 percent.

Keep bus bits together – Select this option if you want pins associated with bus bits to be grouped together. The default is off.

Order bus bits – During automatic ordering of bus pins, IC Compiler honors all the bits of a bus as one individual entity and follows the ordering of bits when assigning pins. All pins of a bus are grouped together on a single edge. Bit ordering can be done in one of the following ways:

- From lsb to msb– Pins are ordered from the least significant bit to the most significant bit and from left to right on a horizontal edge and bottom to top on a vertical edge.

  For example:

  ```
  a[0], a[1], a[2], ... a[31] for the 32-bit bus "a"
  ```

- From msb to lsb – Pins are ordered from the most significant bit to the least significant bit and from left to right on a horizontal edge and bottom to top on a vertical edge.

  For example:

  ```
  a[31], a[30], ... a[1], a[0] for the 32-bit bus "a"
  ```

- Scrambled – Bits are first sorted in increasing order and then ordered in the following fashion:

  ```
  skip count = 1
  a[0], a[2], a[4],...a[30], a[1], a[3], a[5],...a[31]
  skip count = 2
  a[0], a[3], a[6],...a[1], a[4], a[7],... a[2], a[2],
  a[5], a [8],...
  ```

  The default skip count is 1. You can specify a different skip count by using the "Scramble skip count" option.

- Consistent wire length – Bits are ordered so that they equalize the wire lengths of the bus bits between soft macros.

4. Click the Feedthroughs tab and complete the necessary feedthrough settings.

Allow feedthroughs – Select this option when you want pin assignment or pin-cutting to create at least two feedthrough ports in the child cell. Each top-level net for which a feedthrough port is created is split into a set of new top-level nets and child-level nets (if feedthrough nets were created for the child net). Directions are assigned for the newly created feedthrough ports. Because new ports are created in soft macros, the netlist is modified as well. The default is off.

Exclude feedthroughs on clock nets – Select this option to exclude feedthrough ports on clock nets.The default is on.

Exclude scan chain nets – Select this option to exclude feedthrough ports on scan chain nets. The default is on.

Exclude network – Select this option to exclude not only the nets specified by the other exclusion options, but also the nets connected through combinational logic to those specified nets. These nets are also excluded from feedthrough creation during pin assignment. The default is on.

Exclude feedthroughs on nets with fanout greater than – Select this option to exclude feedthroughs generated on nets with fanout greater than the specified number. The default number is 0. You can change the high-fanout threshold in the associated text box. The default is off.

Exclude feedthroughs on specified nets – Select this option to exclude feedthroughs on specified nets. The file should contain top-level nets, and the format is one net per file. The default is off.

Write feedthrough map file - Click Blocks to write a feedthrough map file named feedthroughMapOut into the current directory. Click Side to write out the side number of the block on which routing passes through to connect to the other object. Click Offset to write out the distance in microns from the starting point of a specific edge to the routing cross point on that edge. Click Layers to write out the routing layers at the cross point on the block edge when connecting to another object.

Read feedthrough map file - Select this option to enable the router to read a feedthrough map file. The file is named feedthroughMapIn and must exist in the current directory. The format of a feedthrough map file is a series of net names followed by a colon, with pairs of linked objects. The objects can be a block or plan group, a top-level port, or any cell instance pin.

5. Click Apply to write the pin constraint settings to the CEL views of the selected set of soft macros.

Alternatively, you can use the `set_fp_pin_constraints` command.

Specifying Physical Design Constraints on Pins

You can use the `set_pin_physical_constraints` command to set constraints on individual pins or nets. Use the `set_fp_pin_constraints` command to set global constraints for a block. If a conflict arises between the individual pin constraints and the global pin constraints, the individual pin constraints have higher priority. The constraints are stored in the Milkyway database.

The `set_pin_physical_constraints` command uses the following syntax:

```
set_pin_physical_constraints
 objects | -pin_name pin_name [-cell cell_name] | -nets nets
[-layers layers]
[-width pin_width]
[-depth pin_depth]
[-side side_number]
[-offset offset_distance]
[-order order_number]
[-pin_spacing spacing]
[-exclude_sides exclude_side_numbers]
[-off_edge center | location | auto]
[-location point]
```

The `set_pin_physical_constraints` command uses the following arguments to set the physical constraints per pin instance.

Table 12-1 set_pin_physical_constraints Command Options

Command option	Description
`-pin_name` *pin_name*	Specifies the name of the pin on which the physical constraint is applied. Valid values: Name of any pin in the design
`-cell` *cell_name*	Specifies the name of the Milkyway cell to which the pin belongs. This option also requires the pin name.
`-nets` *net_names*	Specifies the name of the net to apply the physical constraint. The constraint is applied to all pins, ports, and feedthrough ports on the specified net.
`-layers` *layer_names*	Specifies the name of the metal layers on which you want to place the pin.
`-width` *pin_width*	Specifies the width of the pin in microns.
`-depth` *pin_depth*	Specifies the depth of the pin in microns.

Table 12-1 set_pin_physical_constraints Command Options (Continued)

`-side` *side_number*	Specifies the side number of the block on which the pin is placed so that it abuts the specified side of the block. Side numbering begins at 1 from the lower-left side of the block.
`-offset` *offset*	Specifies the distance in microns that the pins must be offset from the starting point of the specific edge.
`-order` *order_number*	Specifies the placement order for the relative positions of the pins.
`-off_edge center | location | auto`	Specifies the type of off-edge pin. Specify `center` to place the pin at the center of the cell. Specify `location` together with the `-location` option to specify exact coordinates for the pin. Specify `auto` to place the pin in the center of the cell or in the position guided by the pin guide for the pin.
`-location` *point*	Specifies the coordinates of the off-edge pins. You must also specify the `-off_edge location` option.
object	Specifies the pin to which the specified pin constraints apply.

Honoring the Pin Physical Constraints

By default, pin assignment honors any preexisting pin physical constraints that are set by the `set_pin_physical_constraints` command or the `read_pin_pad_physical_constraints` command. To change this behavior, use the `set_fp_pin_constraints -use_physical_constraints off` command. See "Specifying Physical Design Constraints on Pins" on page 12-6.

Reporting Pin Assignment Constraints

You can use the `report_fp_pin_constraints` to display the pin assignment constraints that you have set for specified soft macros, plan groups, or ports. These constraints control both pin assignment and pin-cutting.

The syntax is:

```
report_fp_pin_constraints
   [objects]
   [-block_level]
```

Use the *objects* argument to include a collection of blocks or ports.

Note:
A mix of blocks and ports is currently not allowed.

Blocks specify the soft macros and plan groups for which the pin constraints are shown. Ports specify the ports for which the block level non-edge pin constraints are shown. When you specify the ports, you must also specify the `-block_level` option.

Use the `-block_level` option to report the block-level pin assignment constraints on the currently open cell instead of the pin constraints for blocks contained in the cell.

Concurrent Pad and Pin Placement

IC Compiler supports designs that contain mixed pads and ports through the proper placement of I/O cells and terminals. Terminals are placed directly on top of and on the same layer as pad cells and bump cells, and on soft macro pins that are internally connected to an I/O cell pin. This reduces the RC delay between the primary signal I/O and the I/O cell to zero. When the I/O cell is located within a soft macro child cell, a soft macro pin is placed on top of the I/O cell.

If multiple I/O cell pins are assigned to the same pin connection, electrically equivalent terminals are created at the corresponding I/O cell pin locations. Except for bump cell pins, an error occurs if the port to I/O cell pin connection is not one-to-one. Terminal placement honors any specified physical pin constraints.

Setting Voltage Area Feedthrough Constraints

You can set voltage area feedthrough constraints that are honored during both global routing and finalize routing. The voltage area constraints are saved in the Milkyway database.

To set the voltage area feedthrough constraints,

1. Choose Pin Assignment > Voltage Area Constraints.

   The Voltage Area Constraints dialog box appears.

   Alternatively, you can use the `set_fp_voltage_area_constraints` command.

2. Select the “All” option if you want the feedthrough constraints to apply to all the voltage areas (the default) in your design, or select the “Specified” option and enter a list of voltage areas to which the specified feedthrough constraints apply.

   For example, to create feedthrough nets on voltage areas A, B, and C, select the “Specified” option and enter the following in the text box:

   ```
   [get_voltage_areas A B C]
   ```

3. Select the “All feedthroughs” option if you want global routing and routing analysis to create feedthrough ports in the logical child cell.

   Each top-level net for which a feedthrough port is created is split into a set of new top-level nets and child-level nets if feedthrough nets were created for the child net. Pin directions are assigned for the newly created feedthrough ports.

   The default is off; no feedthrough ports are created.

4. Select the “Exclude feedthroughs on specified nets” option if you want to exclude feedthroughs on specified nets.The file should contain top-level nets, and the format is one net per file. The default is off.

5. Click the Set button in the dialog box to apply the feedthrough constraints on the voltage areas.

Reporting Voltage Area Constraints

Use the `report_fp_voltage_area_constraints` command to display the options used to control feedthrough creation on the voltage areas in your design.

Removing Voltage Area Constraints

Use the `remove_fp_voltage_area_constraints` to reset the feedthrough constraints for the specified voltage areas to their default values.

Performing Traditional Pin Assignment

Note:

The pin assignment constraints must be set prior to running pin assignment. (See “Setting Pin Assignment Constraints” on page 12-2.)

Traditional pin assignment automatically assigns top-level or block-level pins on both rectangular and rectilinear soft macros that mark the points where nets can pass in and out of soft macros.

For top-level pin assignment, the pin assignment engine considers the top-level connections to plan groups, macros, and pad locations when it determines where to assign the pins.

For block-level pin assignment, the pin assignment engine considers the cell placement inside the soft macro when it assigns pins to minimize the wire lengths to the internal connections within the soft macro. Use block-level pin assignment in a bottom-up design flow.

This section includes the following topics:

- Performing Block-Level Pin Assignment
- Performing Block Level Non-Edge Pin Placement

Note:
The pin assignment results are stored at the child level within the cell master. If you open a child cell after pin assignment and then close it without saving it, the pins are also discarded.

To assign soft macros pins,

1. Choose Pin Assignment > Assign Pins.

   The Assign Pins dialog box appears.

   Alternatively, you can use the `place_fp_pins` command.

2. Choose whether to assign pins in the current design to all soft macros or to selected soft macros. The default assigns pins to all soft macros.

3. Select the effort used to determine how the pins are placed.

   Low – This effort is connectivity driven (flyline-based) and runs very quickly. This is the default.

   - For top-level pin assignment, the pin assignment engine considers all the top-level connections for a soft macro pin before choosing the macro side (rectangular or rectilinear) to which the pin is assigned. It also considers the wire track assignments from the routing grid to place pins created for various layers on wire track positions corresponding to those layers.
   - For block-level pin assignment, the pin assignment engine considers all the internal connections for a soft macro pin before choosing the macro side (rectangular or rectilinear) to which the pin is assigned.

   High – This effort performs a global route. The pin assignment constraints are strictly honored. A high effort generally produces better results.

   - For top-level pin assignment, the pin assignment engine performs global routing of the top-level soft macro connections and considers routing congestion as pins are placed.
   - For block-level pin assignment, the pin assignment engine performs global routing of the internal soft macro connections to determine the side of the macro on which to assign the pins.

   Note:
   If feedthrough creation is required (top-level pin assignment only), you must select the high effort.

4. (Optional) Select the Verbose option to control the amount of output messages that are written to the log file during pin assignment.

   If the Verbose mode is disabled (off), pin assignment writes out the following messages to a log file during pin assignment:

   - Informational messages tracking the progress and stages of the pin creation.
   - Informational messages on the success or failure of setting pin assignment constraints on soft macros.
   - Error messages regarding issues detected in design data or constraint settings that prevent successful pin creation, such as non-Manhattan soft macro shapes or soft macro sides without enough space to assign all the pins.
   - Informational messages on feedthrough generation statistics, such as the number of original ports, and the number of feedthrough ports and feedthrough nets that were created in each soft macro.
   - Informational messages on the total number of pins that were created.

   If verbose mode is enabled (on), the tool writes the following additional warning and informational messages to the log file:

   - Warning messages regarding issues detected in design data or constraint settings that do not prevent pins from being created, but that might affect the quality of the pin assignment results, such as ports that are not connected, nets with no drivers, and nets with multiple drivers.
   - Detailed information about feedthroughs, including the names of the feedthrough nets and the ports that are created.
   - Detailed information about the created pins, including the name of each created pin and its location.

5. Click OK or Apply to assign pins to the soft macros.

Performing Block-Level Pin Assignment

You can assign pin locations based on the current cell placement. Select the “Block level pins” option on the Assign Pins dialog box.

Alternatively, you can use the `place_fp_pins -block_level` command.

Zroute is used when `place_fp_pins -effort high` is specified, otherwise the classic router is used.

Block-level pin assignment is performed separately and independently on each soft macro in the design. The pin assignment engine assigns pins at the child cell level within the soft macro cell, taking into account the cell placement, internal connections, and pin constraints inside the soft macro cell.

Performing Block Level Non-Edge Pin Placement

You can use the `place_fp_pins -block_level` command to place assign pins inside the top-level block directed by the pin guides by using the `create_pin_guide` command and the pin assignment constraints by using the `set_fp_pin_constraints` command.

These non-edge pins are placed within the pin guide region in a location that minimizes the wire length for the particular nets. Specifically, the pins are placed inside the pin guide region at the intersection of the preferred wire track direction on the pin layer with the preferred wire track direction on any layer.

Figure 12-1 and Figure 12-2 on page 12-13 shows examples of region constraints on non-edge pins.

Figure 12-1 Region Constraints on Non-Edge Pins

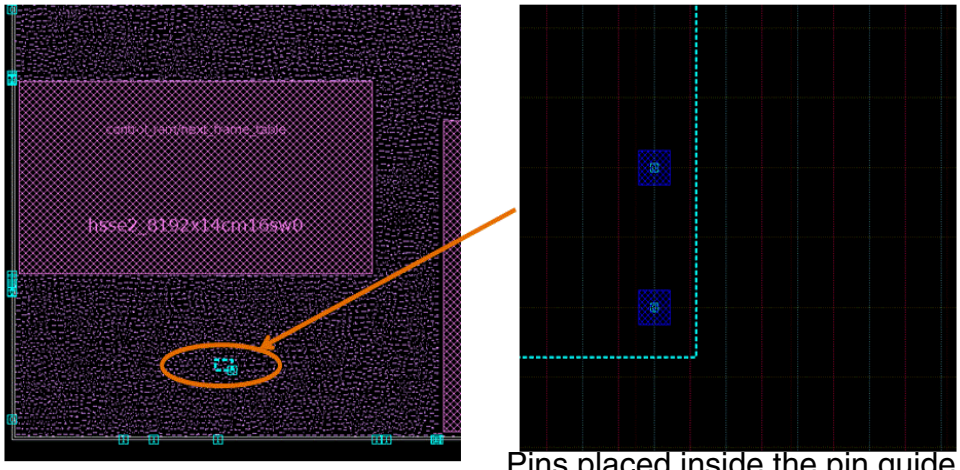
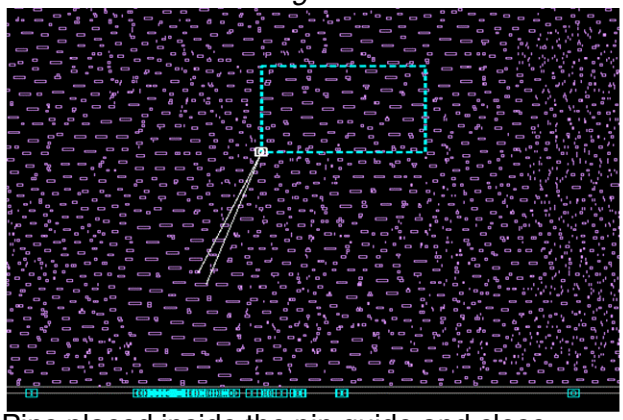


Figure 12-2 Region Constraints on Non-Edge Pins



Pins placed inside the pin guide and close to the standard cell connections to minimize the wire length

Specifying Layers for Non-Edge Pin Placement

Use the `set_fp_pin_constraints -allowed_layers` *`layers`* command to specify a set of metal layers to be used for pin placement. The *`layers`* argument is a collection of metal layers ordered consecutively. For information about creating pin guides, see "Creating Rectangular or Rectilinear Pin Guides" on page 12-35.

Checking the Non-Edge Pin Placement

You can check for pins that are not on an edge by running the `check_fp_pin_assignment -off_edge` command, and you can check for pins that are not inside the associated pin guide by running the `check_fp_pin_assignment -outside_pin_guide` command.

Aligning Soft Macro Pins

You can align a set of soft macro pins relative to the pins on a reference macro.

To align soft macro pins,

1. Choose Pin Assignment > Align Soft Macro Pins.

   The Align Soft Macro Pins dialog box appears.

   Alternatively, you can use the `align_fp_pins` command.

2. Click in the Reference Macro text box and enter the cell names of the macros whose pins are to be used as references for alignment.

3. Select the direction in which you want to align the edges of the soft macro pins. This makes a difference only if the reference pins and the pins you want to align have different widths.

   - Left – Aligns the left edges of the soft macro pins. This is the default.
   - Right – Aligns the right edges of the soft macro pins.
   - Top – Aligns the top edges of the soft macro pins.
   - Bottom – Aligns the bottom edges of the soft macro pins.

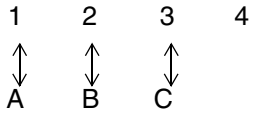
   If the pins you want to align sit on the bottom or top edges of their macros and you specify a left direction, the left edges of the pins will be aligned. Conversely, if you specify a right direction, the right edges of the pins will be aligned. In this case, do not specify a bottom or top direction. If you do, the value automatically defaults to left.

   If the pins you want to align sit on the left or right edges of their macros, the default value is bottom.

4. Identify the method to use for ordering the reference pins.

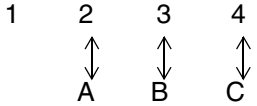
   - Low to high – Aligns pairs of soft macro and reference pins in order, starting with pins on the lowest x-coordinate axis for horizontal pins or y-coordinate for vertical pins. This is the default.

     If the alignment is done on a horizontal edge, as shown in the following example, and there are four reference pins and three pins on the other macro to be aligned, the fourth reference pin, which is the reference pin with the largest horizontal coordinate, is not used.

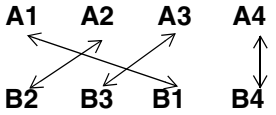


   - High to low – Aligns pairs of soft macro and reference pins in order, starting with pins on the highest x-coordinate axis for horizontal pins or y-coordinate for vertical pins.

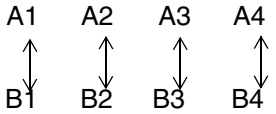
     If the alignment is done on a horizontal edge, as shown in the following example, and there are four reference pins and three pins on the other macro to be aligned, the first reference pin, which is the reference pin with the smallest horizontal coordinate, is not used.



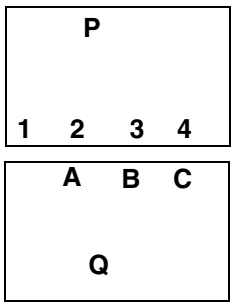
- Net connection – Aligns the soft macro pins to the reference pins if they are connected to the same net, as shown in the following example. A1 and B1 are connected, A2 and B2 are connected, A3 and B3 are connected, and A4 and B4 are connected.



After executing the “Net connection” option, the soft macro pins are aligned as follows. (Suppose macro A is the reference soft macro.)



The following example shows the effects of the “Net connection” and the “High to low” options in which selected pins 2, 3, and 4 of soft macro P are aligned with pins A, B, and C of soft macro Q.



5. Select other options as needed.
   - Change layer and width to match reference pin – Select this to change the pin metal layers and widths of the reference pins.
   - Align with child macro pins – Select this option to align the pins with the child pins on the reference macro.

     The reference macro pins are hard macro pins inside the hard macro. When you use this option, you must also select the “Net connection” option to order the pins by their logical net connections.
   - Mark pins as fixed – Select this option to fix the pins after aligning them. By default, the pins are unfixed, and they can be moved, using the Edit tool, after you align them.

Removing Soft Macro Pin Overlaps

You can remove pin overlaps between soft macro pins or you can remove overlaps for top level or soft macro pins. Pins are spread so that they do not overlap and do not cause spacing violations. The location of fixed pins are not moved. Pins inside placement blockages are considered illegal and are moved.

The new locations of the pins will be on routing tracks. Selective layer blockage is honored.

Note:
If there are two pins with overlapping pin boundaries but on different layers (for example, metal1 and metal3), it is not considered an overlap.

To remove soft macro pin overlaps,

1. Choose Pin Assignment > Remove Pin Overlaps.

   The Remove Pin Overlaps dialog box appears.

   Alternatively, you can use the `remove_fp_pin_overlaps` command.
2. To remove pin overlaps for soft macro, select the All option to instruct the tool to remove overlaps for all top level and soft macro pins. This is the default.

   To remove pin overlaps for specific soft macros, select the Specified option and enter the names of the macro cells.
3. To remove overlaps for pins, select the All option to instruct the tool to remove overlaps for all pins on the specified macro cells. This is the default.

   To remove overlaps for specific pins, select the Specified option and enter the names of the pins.
4. Click OK or Apply.

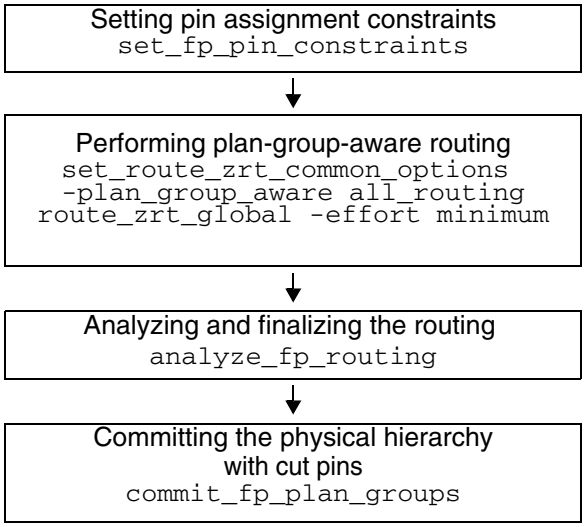
Note:
Pin overlap removal might fail if insufficient routing tracks are available or the pin placement is overconstrained. Look for error messages in the console window after performing this operation.

Performing Pin Assignment on Plan Groups (Pin-Cutting Flow)

To run the pin-cutting flow, as shown in Figure 12-3, you can assign pins to plan groups by using the `commit_fp_plan_groups` command. In the pin-cutting flow, the plan-group-aware global router generates a routing pattern that is consistent with the future physical hierarchy and with the pin constraints used for assigning pins.

The pin-cutting flow provides you with a much better correlation between the results obtained for physically flat and hierarchical designs after you commit the hierarchy. Pin-cutting allows you to floorplan during placement and fix any floorplan issues early in the design cycle.

Figure 12-3 Pin-Cutting Flow



This section includes the following topics:

- Setting Pin Assignment Constraints
- Reserving Narrow Channels for Special Nets
- Performing Plan-Group-Aware Global Routing
- Analyzing the Pin and Feedthrough Routing
- Committing the Hierarchy With Cut Pins

Setting Pin Assignment Constraints

You must first set your pin assignment constraints with the `set_fp_pin_constraints` command before running plan-group-aware global routing. The plan-group-aware router honors the pin assignment constraints for pin-cutting during the commit hierarchy stage.

You should exclude feedthroughs for the critical nets of the design, such as

- Clock nets
- Reset, set, and enable signals

- Global DFT signals (for example, test mode and scan enable)
- Scan in and scan out signals

Reserving Narrow Channels for Special Nets

The `-reserved_channel_threshold` *`channel_size`* option allows you to specify the maximum size of the space between plan groups or soft macros. This space is considered a channel reserved for special nets, such as clocks. This feature is targeted for floorplans where there are only narrow channels between the plan groups.

The default channel size is 0.0, which means the feature is turned off. Negative numbers are not valid. If you set a *channel_size* greater than 0.0, any space between adjacent plan group edges that is separated by less than this amount is considered a channel reserved for special nets. The `-allow_feedthroughs` option should be set to exclude feedthroughs on these special nets. Generally, feedthroughs are allowed for other nets.

During global routing, these special nets can then route freely through any space between the plan groups. If the size of the space you specified is less than or equal to the threshold, other nets are prevented from routing parallel to the direction of the channel as much as possible. However, any net that has a pin in the channel will have to enter the channel, regardless of whether or not it is a special net. For example, the channel might contain pins because there are standard cells or small macros in the channel area or because there are pins on a block at the very edge of a plan group. In such cases, the routing to these targets must enter the channel.

When this option is used during pin assignment, pins are discouraged on blocks that face the channel unless they are perfectly aligned across the channel or are pins that belong to special nets, such as clocks. Nets that are not in the special group will find other routing paths, creating feedthrough pins for these nets in preference to entering the channels.

Note:

Two voltage areas that face each other do not generate a reserved channel for special nets. However, if your floorplan consists entirely of plan groups with narrow channels between them and each plan group is a voltage area, you should set the `set_fp_voltage_area_constraints -allow_feedthroughs` command to `true`. Otherwise, a situation is created where neither feedthroughs nor channel routing is permitted.

You can use the `-reserved_channel_nets` *`nets`* option to specify a collection of nets which are permitted in the reserved channels. By default, this is the set of all the nets that are marked as clock nets in the database.

If you use this option, the collection you specify must include all the nets that you want routed in the reserved channels. These nets are implicitly marked as “exclude feedthroughs” because the intent is to route them through the channels. The default or any previous selection is overridden by this option.

Performing Plan-Group-Aware Global Routing

The `set_route_zrt_common_options -plan_group_aware` command enables plan-group-aware Zroute global routing in the pin-cutting flow.

You should run plan-group-aware Zroute global routing by using the following flow on a hierarchical placed design.

- `set_fp_pin_constraints -allow_feedthroughs on [-read_feedthrough_map on]`

  When the `-allow_feedthroughs` option is set to `on`, Zroute honors the logical constraints on voltage area feedthroughs to ensure port punching on the voltage areas. If you include the `-read_feedthrough_map` option, the command reads the feedthroughMapIn file from the current directory. The feedthroughMapIn file contains feedthrough constraints for the design.

- `set_route_zrt_common_options -plan_group_aware all_routing`

  When you set the `plan_group_aware` common route option, the Zroute global router becomes plan-group-aware. The Zroute global router reads any plan group definitions and honors any pin placement constraints placed on those plan groups. The `all_routing` setting routes all the nets in the design and preserves the hierarchy and pin constraints for the pin-cutting flow.

  Note:

  If you set the `set_route_zrt_common_options -plan_group_aware` command to `off`, the Zroute global router ignores the plan groups and routes the design as flat. The default is `off`.

- `route_zrt_global`

  Run the `route_zrt_global -effort minimum` command to perform global routing on the design.

Honoring the plan group boundaries and pin assignment constraints through the application of the following rules results in more accurate pin assignment and better use of valuable channel routing resources.

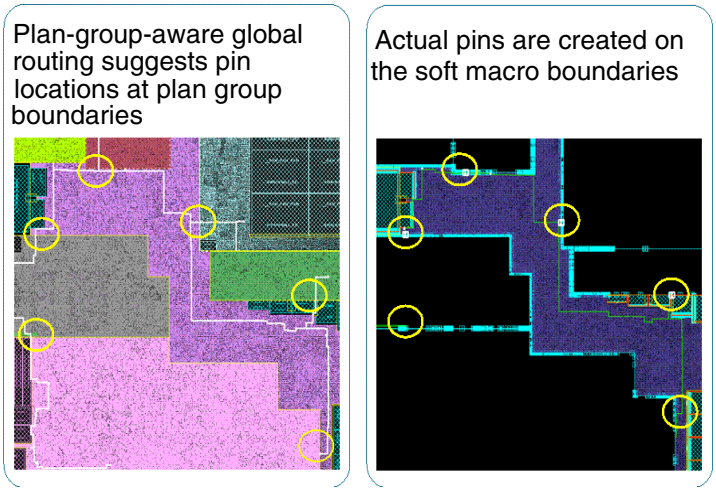
- Routing nets that exist only in the context of the modules associated with the plan group hierarchy, completely inside the plan group

- Not routing nets without a connection to the logic modules associated with a plan group over that plan group, unless feedthrough creation is allowed
- Ensuring that interface nets are routed across the plan group boundary at least as many times as the number of ports on those nets for the associated logic module.

  The router also tries to minimize the number of routes across the plan group boundary, so that they exceed the number of ports only when necessary.
- Minimizing jogs in the channels between plan groups
- Routing nets with the clock net type first, followed by the remaining signal nets

Figure 12-4 shows an example of plan-group-aware global routing and pin assignment.The picture on the left shows pins cut at the boundaries where the routes cut the plan group boundaries, as shown by the areas that are circled.The picture on the right shows the actual pins created at the boundaries of the soft macros, as shown by the areas that are circled. These are the exact locations where the routing crosses those boundaries.

Figure 12-4 Plan-Group-Aware Global Routing and Pin Assignment



Detecting Buses

If the “Keep bus bits together” option is enabled in the Add Pin Assignment Constraints dialog box, the plan-group-aware router automatically detects buses based on user-defined bus groups in the Milkyway database.

Detecting Clock Nets

Before you set the pin assignment constraints, you can ensure that there are optimal locations reserved for the pin of clock nets by using the `mark_clock_tree -clock_net` command. The plan-group-aware router checks the net type in the Milkyway database for any nets with the “clock” net type. Nets with the “clock” net type are routed first, followed by

the remaining signal nets. If no clocks are detected during the setting of the pin constraints, a warning message is issued saying that no clock nets were found and recommends that you run the `mark_clock_tree -clock_net` command to mark the clock nets in the Milkyway database.

Excluding Feedthroughs on Clock Nets

If you want to exclude feedthrough generation on clock nets, you can enable the “Exclude feedthroughs on clock nets” option on the Add Pin Assignment Constraints dialog box.

Alternatively, you can use the `set_fp_pin_constraints -exclude_clock_feedthroughs` command.

Increasing the Routing Speed

To increase the routing speed, you can set the `plan_group_aware` Zroute common route option to `top_level_routing_only` during floorplanning. With this setting, the subsequent `route_zrt_global` and `place_fp_pins` commands do not route any nets that are entirely contained within the plan groups.

```
icc_shell> set_route_zrt_common_options \

   -plan_group_aware top_level_routing_only
```

To use the fast exploration mode, specify the `-exploration true` option to the `route_zrt_global` command. The command runs in a lower-effort mode, ignoring blockages to more quickly return routing results. Use the congestion map to identify routing congestion hotspots.

```
icc_shell> route_zrt_global -exploration true
```

Note:

Interplan routing is not strictly needed for creating feedthroughs and pin locations in the subsequent `analyze_fp_routing` and `commit_fp_plan_groups` commands. However, if the routing congestion inside the plan groups is significant, ignoring intraplan group nets could lower the quality of the pin placement.

Using the Classic Router

To enable the classic router, set the Zroute mode option to false:

```
icc_shell> set_route_mode_options -zroute false
```

You can use plan-group-aware global routing to suggest pin locations using the classic router. To enable plan-group-aware global routing, use the `set_fp_flow_strategy` command:

```
icc_shell> set_fp_flow_strategy -plan_group_aware_routing true
```

Note:

If you are running the virtual flat or pin-cutting flow, you must set this value to `true`. If global routing is run without the “awareness” of plan groups, the `analyze_fp_routing` and `commit_fp_plan_groups` commands will not work properly.

By default, plan-group-aware routing is set to false.

When you set the `-plan_group_aware_routing` option to `true`, the classic router becomes plan-group-aware. This means that the `route_global` command reads any plan group definitions and honors any pin assignment constraints placed upon the plan groups, such as pin spacing and pin blockages.

The `route_global` command also honors the hierarchical rules for plan group boundary crossings by recognizing those plan groups and the routes internal to a plan group do not cross the plan group boundaries. The routes that cross the plan groups make minimal intersections with the plan group boundaries and do not crisscross. The intersection becomes the pin location of the plan groups. You can analyze the pin assignment by checking the topology of related global routes.

Performing Global Routing

After you have enabled plan-group-aware routing by using the `set_fp_flow_strategy -plan_group_aware_routing true` command, use the `route_global` command to perform global routing to get an estimate of the pin locations and congestion of your design.

Specifying Net and Feedthrough Topology

You can provide user-specified feedthrough topology information to the router to control pin cutting. The topology information is specified in a mapping file and includes information about how the tool should route a net, which blocks the route should pass through and the block ordering for the feedthrough connections. IC Compiler can also generate a mapping file that describes the net and feedthrough topology for the design. You can edit the generated file and read in the updated file to perform subsequent routing iterations. The net and feedthrough topology described in the file will be implemented by the router.

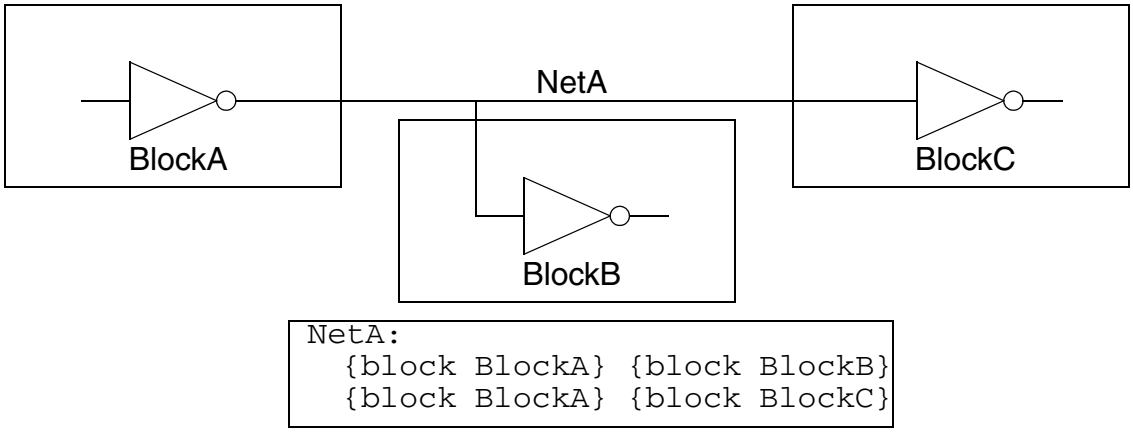
The general format of the mapping file is

```
Net_name1:
  {object_type object_name} {object_type object_name}
  {object_type object_name} {object_type object_name}
  ...
Net_name2:
  {object_type object_name} {object_type object_name}
  {object_type object_name} {object_type object_name}
  ...
```

where `object_type` can be a block, a plan group, an I/O port, soft macro pin, hard macro pin, or I/O pad cell pin and *`object_name`* is a plan group name, a top-level port name, a macro port name, or a pin name for an I/O pad.

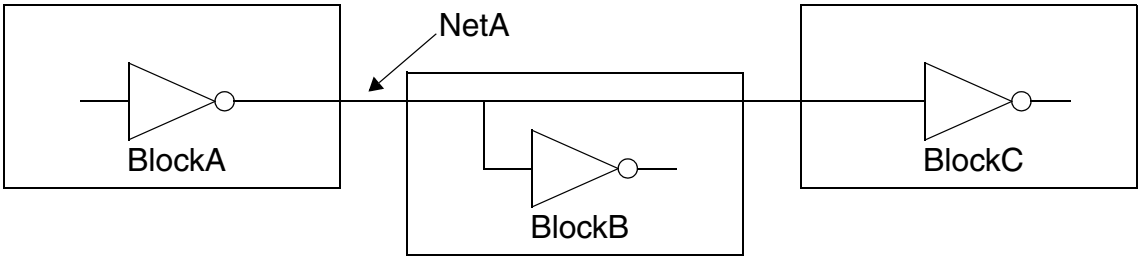
Figure 12-5, Figure 12-6, and Figure 12-7 show example topologies and the control files needed to produce the desired result.

Figure 12-5 Feedthrough Topology Example 1



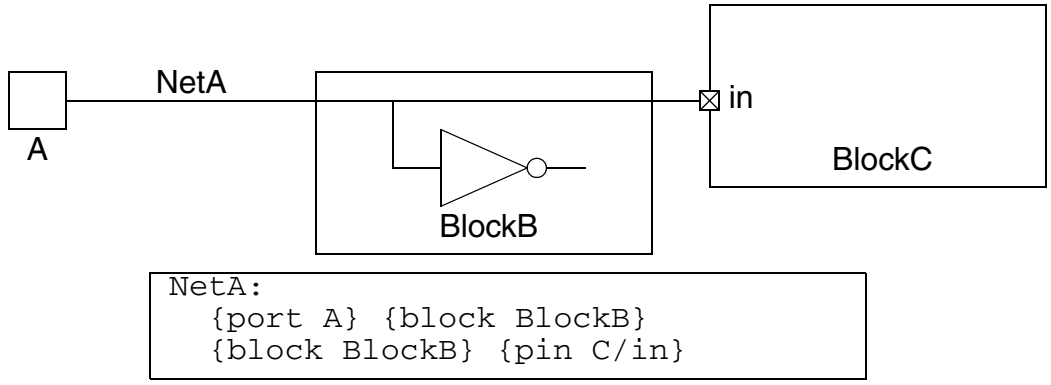

```
NetA:
  {block BlockA} {block BlockB}
  {block BlockA} {block BlockC}
```

Figure 12-6 Feedthrough Topology Example 2




```
NetA:
  {block BlockA} {block BlockB}
  {block BlockB} {block BlockC}
```

Figure 12-7 Feedthrough Topology Example 3



Note that you should specify a mapping file that does not overly constrain the tool; otherwise, the QoR might degrade.

To read a mapping file for the nets in the current design, use the following command:

```
set_fp_pin_constraints -allow_feedthroughs on \
                       -read_feedthrough_map on
```

The tool reads the feedthrough constraints from the file named feedthroughMapIn. To write out mapping information to the feedthroughMapOut file, use the following command:

```
set_fp_pin_constraints -allow_feedthroughs on \
                       -write_feedthrough_map on
```

Note that both the feedthroughMapIn and feedthroughMapOut file names are fixed and cannot be changed.

Analyzing the Pin and Feedthrough Routing

You can analyze the routing pattern generated by the plan-group-aware router to determine the pin locations after global routing. (The pins are physically created during the commit hierarchy stage.) The global routing is analyzed with respect to routes crossing the boundaries of the plan groups or voltage areas. This information can be used to output a list of feedthroughs nets on plan groups or voltage areas.

Feedthroughs are created on plan groups or on hierarchical modules that belong to voltage areas.

The behavior of pin and feedthrough routing analysis is highly dependent on both the pin and voltage area constraints set previously and the last global route. (see “Setting Pin Assignment Constraints” on page 12-2 and “Setting Voltage Area Feedthrough Constraints” on page 12-8)

During routing analysis, you can view the list of nets that will have feedthroughs created for them when a design with plan groups is converted to soft macros. When you click a net in the list, all pins connected to the net are cross-highlighted in the layout (CEL) view.

To analyze the pin and feedthrough routing,

1. Choose Pin Assignment > Finalize Pin/Feedthrough Routing.

   The Finalize Pin/Feedthrough Routing dialog box appears.

   Alternatively, you can use the `analyze_fp_routing -finalize_pins_feedthroughs {plan_groups | voltage_areas}` command.

2. Select the “Report feedthroughs only” option to output a list of all the feedthrough nets to the specified file.

   If you selected the “PlanGroup” option, a list of feedthrough nets with their associated plan groups is reported. This is the default. The location, level, and direction of the pins, including the feedthrough pins, is reported.

   If you selected the “Voltage areas” option, a list of feedthrough nets on voltage areas is reported.

   Note:
   No changes are made to the Milkyway database. The output list allows you to preview the nets whose routing implies feedthroughs on plan groups or voltage areas.

   The actual layout data is not modified until you “finalize the routing,” thereby allowing you to reroute and reanalyze your design until you are satisfied with the routing results. When you are satisfied with the results, move on to step three.

3. Finalize the routing. Choose Pin Assignment > Pin and Feedthrough Analysis.

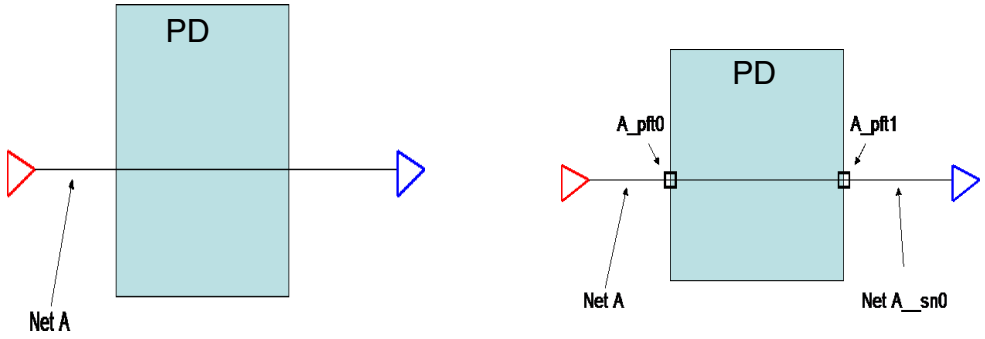
   The Pin and Feedthrough Analysis dialog box appears.

   Select the Pre-Finalized tab and click the Finalize button.

   - The logical netlist is modified and the feedthrough nets are updated and inserted into the hierarchy. The update is based on the existing global routing and where plan group crossings do not have corresponding connections in the hierarchy. The pin locations for each boundary location are also calculated and saved in the Milkyway database as a guide to pin placement later when the hierarchy is committed.

     Figure 12-8 shows an example of a simple feedthrough insertion before and after analyzing the routing.

Figure 12-8 Simple Feedthrough Insertion



Before analyze routing

After analyze routing

- If you selected the “PlanGroup” option, the pin locations for each feedthrough net crossing a plan group are also calculated and saved in the Milkyway database. This is the default behavior.
- If you selected the “Voltage areas” option, pure feedthrough nets and ports are created for nets with routes crossing between a global route and the voltage area’s boundaries. Only logical feedthrough nets and ports are created at the top level of the logical hierarchy within the voltage area.

  After the feedthroughs for voltage areas are created, you can run always-on synthesis as needed to optimize the feedthrough nets.

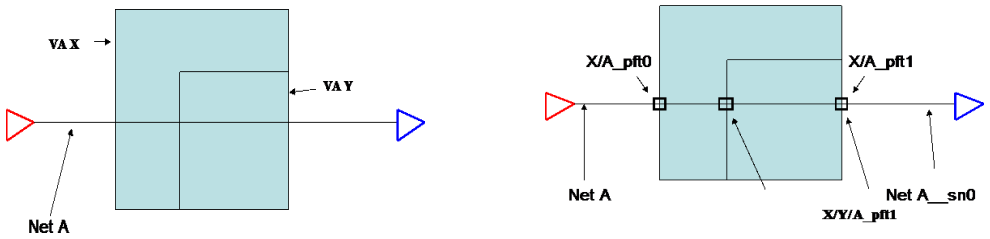
  If the routes cross a plan group which also belongs to a voltage area, a feedthrough is created on that plan group; otherwise a hierarchical module is selected within the voltage area and a feedthrough is created on the module.

  Note:

  There is no physical cell that corresponds to the voltage area, and therefore the physical location of the boundary crossing is not recorded.

  Voltage areas can be nested during feedthrough insertion. Figure 12-9 shows an example of nested voltage area feedthrough insertion before and after analyzing the routing.

Figure 12-9 Nested Voltage Area Feedthrough Insertion



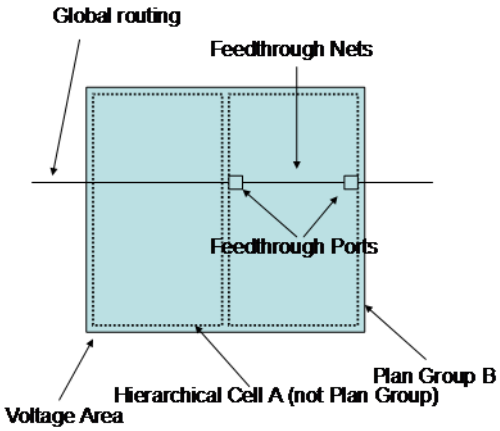
Before analyze routing

After analyze routing

Voltage areas can correspond to different hierarchical cells.

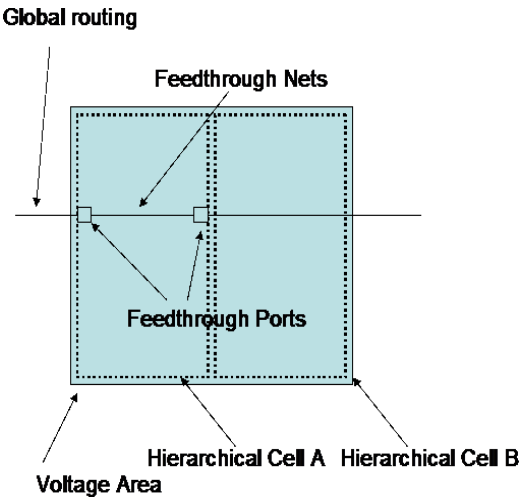
If a global route crosses a voltage area and there is a plan group within the voltage area that overlaps with the global routing, a pure feedthrough is created on that plan group and the other hierarchical cells within the voltage area remain intact. An example of this scenario is shown in Figure 12-10.

Figure 12-10 Plan Group Overlaps with Global Routing; Hierarchical Cells Remain Intact



If there is no plan group overlap with global routing, a hierarchical cell that belongs to the voltage area is randomly selected, and pure feedthrough nets and ports are created in this hierarchical cell. Because the hierarchical modules inside the voltage area have no placement bounds, their cells can freely intermix. An example of this scenario is shown in Figure 12-11. In this example, hierarchical cell A is selected.

Figure 12-11 No Plan Group Overlap with Global Routing; Hierarchical Cells Intermix



During “finalize routing,” the Milkyway database is also updated by inserting feedthroughs into hierarchical modules that belong to voltage areas when the router crosses a voltage area.

Note:

After you finalize the routing, any changes made to the global routes no longer affect pin assignment.

When “finalize routing” is complete, a summary of the block names, the number of original ports on the blocks is displayed in the console log view. A text file called “VA_Feedthrough.port” is automatically written with the names of all the feedthrough ports created on the voltage areas.

Committing the Hierarchy With Cut-Pins

Pin-cutting occurs during commit hierarchy when the actual physical hierarchy, the soft macros, are created from exclusive plan groups. The soft macros use the plan group’s placement. Pin-cutting during commit hierarchy is based on plan-group-aware global routing and pin constraints-aware routing results. Pin-cutting during commit hierarchy also honors the bus pin assignment constraints set by the `set_fp_pin_constraints` command.

To commit the hierarchy,

1. Choose Partition > Commit Plan Groups

   The Commit Plan Groups dialog box appears.

   Alternatively, you can use the `commit_fp_plan_groups` command.

2. Specify the plan groups to be committed to soft macros. The default is all.

3. Select the “Push down power and ground straps into the child cell” option if you want to push down the power and ground preroutes into the soft macros. It removes them from the top level.
4. Select the “Commit to a new top cell” option if you want to create a new top cell. Changes will occur at the top level. You must enter the name of the top-level cell.

During commit hierarchy the exclusive plan groups are converted to soft macros, and the following occurs:

- The design is converted to a two-level hierarchical design (the design netlist is partitioned to blocks and to the top level).
- CEL views are created for each soft macro.
- Nets are created in the soft macro based on the Hierarchy Preservation data.
- Cell rows (placement tiles) are cut down and pushed into the soft macro CEL views.
- (Optional) Power and ground straps and standard cell preroutes are pushed down into each soft macro CEL view.
- Placement blockages are pushed down to each soft macro.
- Pin placement is created based on the finalize routing results.

Note:
Commit hierarchy can take longer to run if power and ground straps and standard cell preroutes are pushed down because it calls repair hierarchy for each soft macro.

When pin-cutting during commit hierarchy is complete, a summary of the block names, the number of original ports on the blocks, and the feedthrough ports and nets that were created on those blocks are displayed in the console log view and the log file. This process also checks the Milkyway database for any nets with the “clock” net type and assigns the pin slots for the clock nets first. If no clocks are detected by pin-cutting during commit hierarchy, a warning message is issued saying that no clock nets were found.

Using Pin-Cutting With Multiply Instantiated Modules

The pin-cutting flow supports designs with multiply instantiated modules.

This section includes the following topics:

- Setting Pin Assignment Constraints for Multiply Instantiated Modules
- Performing Plan-Group-Aware Routing for Multiply Instantiated Modules
- Analyzing Routes and Finalizing the Routing for Multiply Instantiated Modules

- Selecting a Master Multiply Instantiated Module Plan Group
- Committing a Master Multiply Instantiated Module Plan Group

Setting Pin Assignment Constraints for Multiply Instantiated Modules

Use the `set_fp_pin_constraints` command or choose Pin Assignment > Pin Constraints in the GUI to apply pin constraints for multiply instantiated modules. For more information, see "Setting Pin Assignment Constraints" on page 12-2. The same pin assignment constraints used in the pin-cutting flow are assigned on all instances of a multiply instantiated modules set to avoid conflicts and potential problems during plan-group-aware routing, and pin-cutting during commit hierarchy. If a multiply instantiated modules instance of a multiply instantiated modules set has different pin assignment constraint settings, a warning message is issued during the `set_fp_pin_constraints` step.

Note:

No feedthroughs are allowed for multiply instantiated modules.

Performing Plan-Group-Aware Routing for Multiply Instantiated Modules

The plan-group-aware router performs global routing for multiply instantiated modules. (See "Analyzing the Pin and Feedthrough Routing" on page 12-24.) You can analyze the results for each plan group based on wire length, timing, and congestion reports. Based on this analysis, you should be able to determine which of the multiply instantiated modules plan groups to use as the master instance in the multiply instantiated modules set.

The plan-group-aware router automatically turns off feedthroughs on multiply instantiated modules blocks.

Note:

If any multiply instantiated or unique modules are instantiated as CEL views in the design and pin assignment has not been run on them, the plan-group-aware router issues an error message.

Analyzing Routes and Finalizing the Routing for Multiply Instantiated Modules

Run the `analyze_fp_routing` command or choose Choose Pin Assignment > Finalize Pin/Feedthrough Routing in the GUI to analyze the routing pattern generated by the plan-group-aware router. (See "Analyzing the Pin and Feedthrough Routing" on

page 12-24.) When you are satisfied with the routing results, choose Pin Assignment > Pin and Feedthrough Analysis. Select the Pre-Finalized Feedthrough tab and click the Finalize Pin Routing button to finalize the routing.

The `analyze_fp_routing` command determines the pin locations on the multiply instantiated modules during commit hierarchy based on the plan-group-aware routing results for the selected master instance.

Note:

If any multiply instantiated or unique modules are instantiated as CEL views in the design and pin assignment is run using the `place_fp_pins` command, finalize routing considers their pin placement as fixed and finalizes the routing for the remaining plan groups.

Selecting a Master Multiply Instantiated Module Plan Group

Before committing a master multiply instantiated module plan group, you must first choose a master plan group by using the `select_mim_master_instance` command to designate a plan group as the “master” (template) plan group among a set of multiply instantiated module plan groups that will use the same cell instance master for all multiply instances.

The syntax is:

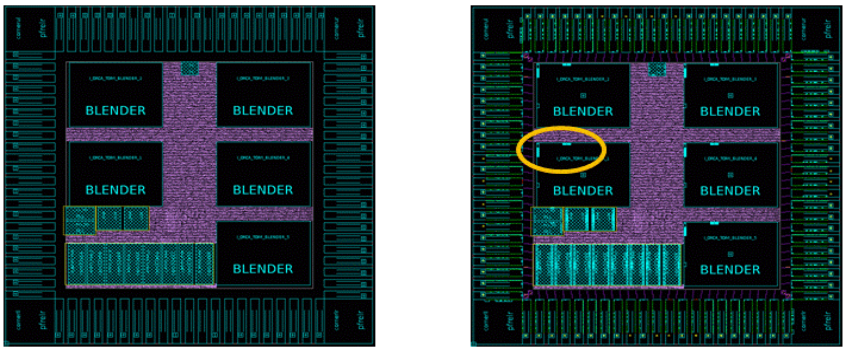
```
select_mim_master_instance plan_group
```

Use the *`plan_group`* argument to specify which plan group among a set of multiply instantiated module plan groups the `commit_fp_plan_groups` command should treat as the master plan group.

A multiply instantiated module group is a set of plan groups that retain a common database property. This database property associates the plan groups with each other and identifies the name of the hierarchical cell instance reference. When a multiply instantiated module group exists with this common property, the `commit_fp_plan_groups` command identifies one plan group among this multiply instantiated module group as the master plan group and then only creates a CEL view once for the plan group designated as the master of its multiply instantiated module group.

Figure 12-12 on page 12-32 shows pin assignment results after committing the hierarchy.

Figure 12-12 Pin Assignment After Commit Supports Multiply Instantiated Modules



Commit hierarchy Results of pin assignment

Note:

If you do not select an multiply instantiated module master plan group before committing the hierarchy, the `commit_fp_plan_groups` command issues a warning and then randomly chooses the first multiply instantiated module plan group in the multiply instantiated module list as a master plan group.

The following example specifies a particular plan group as the master of its multiply instantiated module group.

```
select_mim_master_instance [get_cells {planGroupA}]
```

The set of multiply instantiated module plan groups was previously grouped by the `uniquify_fp_mw_cel -store_mim_property` command. (See "Determining Which Plan Groups and Soft Macros are Multiply Instantiated Modules" on page 6-52.) This command creates multiply instantiated module plan groups that are identified as sharing a common hierarchical reference cell.

The master instance information is saved as a plan group property in the Milkyway database.

Committing a Master Multiply Instantiated Module Plan Group

Use the `commit_fp_plan_groups` command to convert a master multiply instantiated module plan group to a soft macro. Other multiply instantiated module plan group instances from the same multiply instantiated module set are also converted to instances of the master soft macro at the same time.

Pin-cutting occurs during commit hierarchy for multiply instantiated modules blocks based on the pin locations determined by the `analyze_fp_routing` command for the chosen master instance and the plan-group-aware routing results. Pin-cutting supports multiply instantiated modules blocks mirrored along both the x- and y-axes.

Performing Incremental Pin-Cutting

You can incrementally assign pins in the pin-cutting flow.

To perform incremental pin-cutting,

1. Perform pin-cutting on a group of selected nets by using the `set_fp_pin_constraints -nets` command.
2. Perform pin-cutting again on a second group of nets, or on the rest of the nets, by using the `set_fp_pin_constraints -incremental on -nets` command.

IC Compiler appends the results of the pin locations from the second (or final) group of nets to the already calculated results of the first group of nets.

By default, performing pin-cutting (`analyze_fp_routing` command) overrides all previously recorded pin locations.

Note:

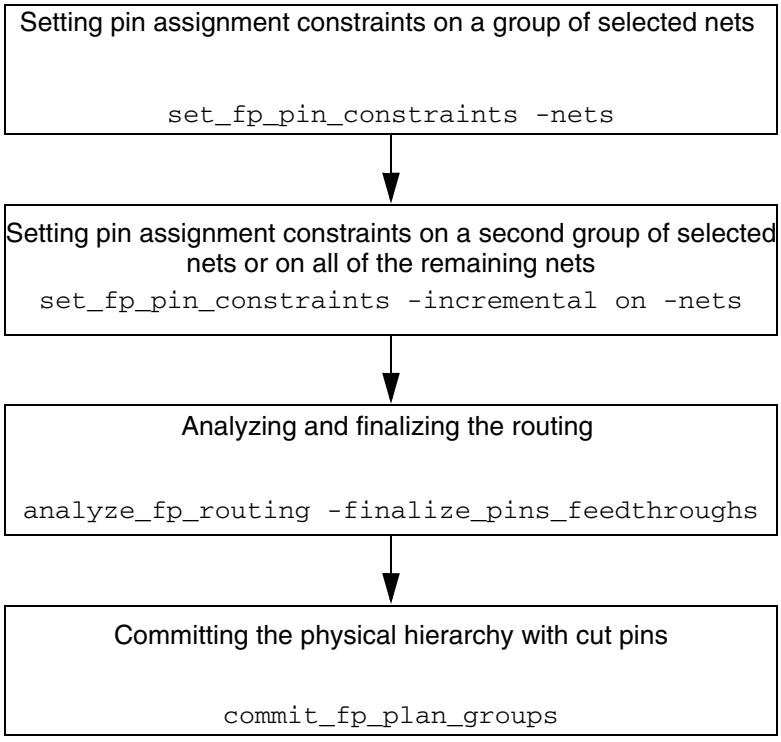
When performing incremental pin-cutting, keep in mind that although it provides you with a lot of flexibility and the ability to place pins incrementally, a cost is also incurred because it becomes necessary to run global routing (`route_zrt_global -effort minimum` command) and analyze routing (`analyze_fp_routing` command) multiple times.

This section includes the following topics:

- Setting Pin Assignment Constraints on a Group of Selected Nets
- Setting Pin Assignment Constraints on a Second Group of Selected Nets or on All Remaining Nets
- Analyzing and Finalizing the Routing on a Separate Group of Nets
- Committing the Hierarchy With Cut Pins

Figure 12-13 shows the incremental pin-cutting flow.

Figure 12-13 Incremental Pin-Cutting Flow



Setting Pin Assignment Constraints on a Group of Selected Nets

Set the pin assignment constraints for the first group of selected nets on which pin-cutting will be run. These nets are processed and pin locations are recorded in a pin-cutting attach file.

```
icc_shell> set_fp_pin_constraints -nets
```

Setting Pin Assignment Constraints on a Second Group of Selected Nets or on All Remaining Nets

Change the pin assignment constraints to indicate pin-cutting will be run on a second, separate group of nets, or on all of the remaining nets. Turn on the `-incremental` option. When this option is on, existing soft macro pins are retained.

```
icc_shell> set_fp_pin_constraints -incremental on -nets
```

Analyzing and Finalizing the Routing on a Separate Group of Nets

Analyze the existing global routing to determine the pin locations and feedthroughs needed on each plan group by using the `analyze_fp_routing` command. The global routing is analyzed with respect to routes crossing the plan groups. The nets should now be considered fixed.

When you are satisfied with the results, finalize the routing.

```
analyze_fp_routing -finalize_pins_feedthroughs {plan_groups |
voltage_areas}
```

The second group of selected nets is finalized, and the pin locations are added to the first pin-cutting attach file. The attach file now contains the pin locations for both groups of nets.

IC Compiler appends the results of the pin locations from the second (or final) group of nets to the already calculated results of the first group of nets.

Committing the Hierarchy With Cut Pins

Pin-cutting occurs during commit hierarchy (`commit_fp_plan_groups` command) when the actual physical pins are created, and is based on plan-group-aware global routing. The `commit_fp_plan_groups` command converts the specified plan groups into soft macros. For each plan group that is processed, a soft macro is created to replace it.

Creating Rectangular or Rectilinear Pin Guides

You can create rectangular or rectilinear pin guide shapes (pin keep-in regions) on soft macro boundaries, on plan groups, or on the current top-level cell to control the placement of pins or ports on soft macros or plan groups.

This section includes the following topics:

- Accessing Information About the Pin Guides
- Creating Pin Guides for Feedthroughs

Because the plan groups do not yet have ports, these constraints are applied to nets rather than to individual pins. The pin assignment engine (`place_fp_pins` command) will then determine the most appropriate location for those pins in the pin guide (keep-in region) based on routability and wire length.

Note:

Pin guides must intersect the module boundary in one contiguous area. Multiple intersections imply that you want a feedthrough on the net. If there are two or more external connections for the net, it is interpreted as a “real” feedthrough, that is, one of the ports is the original port on the block, and the other is the new port.

The routing for the nets that have ports associated with the pin guides is constrained to cross the corresponding soft macro or plan group boundaries through the pin guide regions. As a result, pin assignment honors the pin guide regions and places pins within the corresponding soft macro edge segment. If the pin guide overlaps more than one edge of the soft macro, the router can choose to go through any edge segment that overlaps that pin guide.

The pin guide is a polygon associated with a group of ports or pins and the soft macro or plan group it overlaps with. Only one pin guide can be associated with any given pin on a soft macro or plan group. If the pin guide does not overlap any soft macro or plan group, or if it does not have any ports associated with it, it is disregarded by the router and the pin assignment engine.

To create a pin guide,

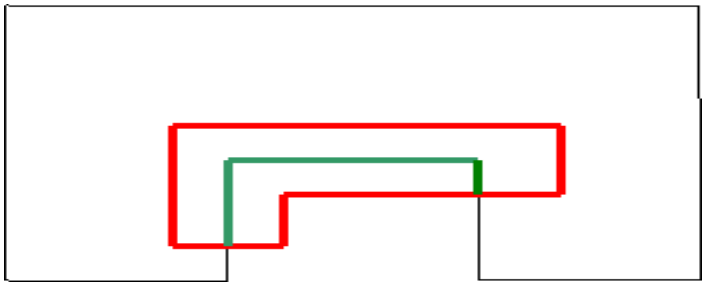
1. Choose Pin Assignment > Create Pin Guide.

   The Create Pin Guide dialog box appears.

   Alternatively, you can use the `create_pin_guide` command.
2. In the Name text box, enter the name of the pin guide you want to create. If you do not use this option, a default name is given to the newly created pin guide.
3. Select the object (soft macro, plan group, black box, or top cell) on which to create a pin guide. The default is soft macro. Enter the name of the soft macro, plan group, black box or top cell in the text box.
4. Under the “soft macro pins” heading, select the type of guide (Pin, Nets, or Ports) and enter a comma, or space-separated list of pins, nets, or ports.
5. Specify the rectangular or rectilinear boundary points of the rectangular or rectilinear pin guide. You can enter the coordinates of the corner points or draw the area graphically in the layout window.

   For a rectilinear plan group, you can draw a rectilinear shape to define a pin guide, as shown in Figure 12-14 on page 12-37. The pin can reside anywhere within the green area.

Figure 12-14 Rectilinear Shape Defining a Pin Guide



Accessing Information About the Pin Guides

You can access information about specific pin guides by using the `report_pin_guides` command. Select the `-pins` option to get detailed information about pin guides associated with specified pins or select the `-nets` option to get detailed information about pin guides associated with specified nets.

Creating Pin Guides for Feedthroughs

You can specify the locations of feedthroughs on a block (soft macro or plan group) for a given net by creating pin guides for the feedthroughs.

Note:

The feedthrough guide only controls where the pins are placed on the block boundaries. It does not control the routing inside the block.

If a pin guide shape passes through a soft macro or plan group, and if the net does not already touch the soft macro or plan group, a “pure” feedthrough is created. A “pure” feedthrough is unconnected internally; it has no connection in the block it passes through, except for the ports of the block. Conversely, if the feedthrough net has one of more cell instances within the block (soft macro or plan group) that the net is connected to, then one side of the feedthrough is counted as the original port, and the other side is a “regular” (connected) feedthrough. To ensure that the connectivity of the feedthrough is clear, the pin guide shape should touch the port, plan group, or soft macro that is connected to the net.

Feedthrough generation takes place only under the following two conditions:

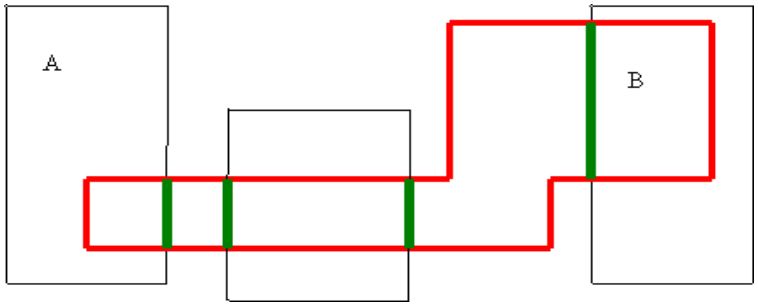
- The allow feedthroughs option must be selected (select the “Feedthrough Settings” tab on the Pin Assignment Constraints dialog box).

  You can also specify the `set_fp_pin_constraints -allow_feedthroughs on` command.

- The block must be identified in the list of “parents” (`create_pin_guide -parents` *`object`*) when the pin guide is created.

As shown in Figure 12-15, port A on the left plan group connects to port B on the right plan group. The center plan group does not contain any ports in the net. The pin guide shape (shown in red) will constrain the pins on the left and right plan groups to the green areas and will cause feedthroughs to be created on the center plan group in the green areas. It is not necessary to include the actual location of the instance ports in the plan groups in the pin guide shape.

Figure 12-15 Pin Guide for Feedthrough



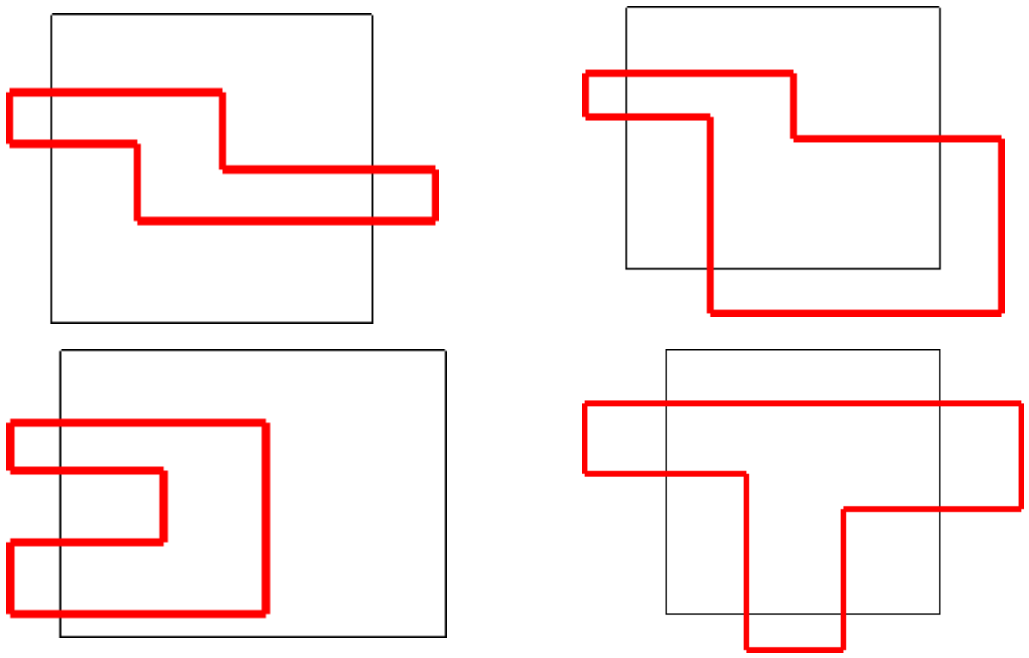
Supported Feedthrough Guides

IC Compiler supports the following rectilinear feedthrough guide shapes, as shown in Figure 12-16 on page 12-39.

A feedthrough guide on a block must have at least two connections on the net outside the block. If it does not, the feedthrough guide is ignored and a warning message is issued.

A feedthrough guide must intersect the module boundary in two disjoint areas.

Figure 12-16 Rectilinear Feedthrough Guide Shapes



Performing Pin and Feedthrough Analysis

You can inspect the physical paths of nets between their endpoints and through the hierarchy by tracing and highlighting the connectivity of specified top-level soft macro nets and pins, and the complete feedthrough path of all feedthrough nets and pins generated for plan groups and soft macros in your design.

You can perform feedthrough analysis on voltage areas and buffered feedthrough nets that are split by inserting buffers during timing optimization.

You can explore detour nets, including both pin detour severity and route detour severity.

You can display all the nets that belong to the traced path and cross-highlight them in the layout window. You can also view multiple nets at the same time, using different colors to do topology analysis.

To perform soft macro net-based or pin-based feedthrough analysis,

1. Choose Pin Assignment > Pin and Feedthrough Analysis.

   The Pin and Feedthrough Analysis dialog box appears.

You can use the Pin and Feedthrough Analysis dialog box to do the following:

- Trace all the nets for a path or all the feedthrough nets in the design.
- Highlight multiple nets and use different highlight colors to view the net locations in the layout view.
- Select the nets you want to gather information about by using the Query tool.
- Explore the physical path of a net by coloring individual net segments or pins in the layout view.
- Export the contents of the explore window to an ASCII file.
- Report feedthrough nets, detours, and pin alignment based on the active tab in the dialog box.

2. Select the Finalized tab.
3. Select an analysis mode option.

   To analyze feedthrough nets for plan groups and soft macros, including black boxes, select the Blocks option.

   To analyze feedthrough nets for voltage areas, select the “Voltage areas” option.

   To include feedthrough nets that are split into multiple sections due to the introduction of buffer cells during timing optimization, select the “Buffered feedthrough” option.
4. Click the Analyze button.

   The Finalized Options dialog box appears.
5. Identify the original plan group nets or soft macro nets or pins you need to trace by doing one of the following:

   - To trace all the nets in the design, select the “All nets” option.
   - To trace all the feedthrough nets in your design, select the “All feedthroughs” option This is the default.

     To trace feedthrough nets only for specific blocks, select the “Specified blocks” option and either select or browse for the block or type their names in the “Specified blocks” text box. If you enter more than one name, use a space or a comma to separate the names.
   - To trace all the non-feedthrough nets in your design, select the “All non-feedthroughs” option.

     To trace non-feedthrough nets only for specific blocks, select the “Specified blocks” option and either select or browse for the block or type their names in the “Specified blocks” text box. If you enter more than one name, use a space or a comma to separate the names.

- To trace all the bus nets in your design, select the “Bus nets only” option.

  To trace bus nets only for specific blocks, select the “Specified blocks” option and either select or browse for the block or type their names in the “Specified blocks” text box. If you enter more than one name, use a space or a comma to separate the names.

- To trace one or more nets, select the “Specified nets” option and either select or browse for the nets or type the net names in the “Specified nets” text box.

- To trace the net connections for one or more pins, select the “Specified pins” option, and either select or browse for the pins or type the pin names in the “Specified pins” text box.

6. Click OK in the Finalized Options dialog box.

   The tool filters the net or pin connections and displays the names of the original top-level nets in the Names nets list. These are the nets that existed in the design prior to design planning and pin assignment before they were split into several internal and external soft macro nets.

7. (Optional) If you want to display status information for the nets, select the “Show status columns” option. The list displays the information in the following columns:

   Pin Detour Severity
   Original Feedthroughs
   Pure Feedthroughs
   Regular Feedthroughs

8. Select the names of one or more nets in the nets list that you want to highlight or select.

   You can click an individual name in the list or use Shift-click or Control-click to select combinations of names.

   By default, the current highlight color yellow is applied to the nets in the layout window.

   - To highlight the nets when the “Auto color” option is not selected, click the Color button.

     Highlighted nets that are not selected appear in the current highlight color in the layout view. The highlight color also appears beside the net name in the Net names list.

   - To magnify the design and fit the nets in the view when the “Auto zoom fit” option is not selected, click the Zoom Fit button.

   - To select the nets, click the Select button.

     The selected nets appear in the selection color in the layout view. The default is white.

9. Select options to determine the appearance of the nets in active layout view.

- Auto color

  Automatically highlights the nets in the active layout view that correspond to the original net names in the display field. The default is off.

  When you click a feedthrough net, all net segments from the driver to the receiver, all pins connected to nets, and any in-place optimization buffers in between, are highlighted in the active layout view.

- Auto zoom fit

  Automatically zooms in on the displayed nets you highlight in the layout view. The default is on.

- Exclude child nets

  When you select this option, child nets are excluded; otherwise nets are drawn inside soft macros.

  If the net is a feedthrough net, it is drawn from the driver at the top level, through one or more soft macros, to the receiver. If the net ends at the boundary of a soft macro, the net inside or outside the soft macro is also drawn.

10. Select the “Explore Net Window” option.

    The Explore Net dialog box appears.

    When the Pin and Feedthrough Analysis dialog box is open and the names of the original top-level nets in the Nets list, located below the Analyze button are displayed, you can use the Explore Net dialog box to explore the physical paths for individual nets in the active layout view.

    The dialog box displays information about each net segment, including the name of the original net, the cell name, and the pin direction of the feedthrough path.

    You can click the plus sign or minus sign expansion button next to a net segment name to display or hide the names of its pin connection.

11. Click Close to close the Pin and Feedthrough Analysis dialog box.

Removing Feedthrough Ports and Nets From Blocks and Voltage Areas

You can remove the feedthrough ports and nets that exist in your design, including those created by other tools, from all soft macros and plan groups in the design or on a per net basis. You can also remove feedthrough ports from voltage areas and feedthroughs with buffers inserted inside hierarchical blocks (soft macros or plan groups) or voltage areas.

For pure feedthroughs, all ports and the child level net are removed. For regular feedthroughs, one port will remain as well as the child level net.

When a feedthrough port is removed, all the pins of that port are deleted and the nets (for which feedthroughs are removed at the parent level of the logic hierarchy) are reconnected.

To remove feedthrough ports and nets,

1. Choose Pin Assignment > Remove Feedthroughs.

   The Remove Feedthrough Ports and Nets dialog box appears.

   Alternatively, you can use the `remove_fp_feedthroughs` command.

2. Remove feedthrough ports and nets – Choose whether to remove route feedthrough ports and nets from all plan groups and soft macros in your design (the default), or from a specified plan group or soft macro.

   To remove feedthrough ports and nets from a specified plan group or soft macro, click in the text box and enter the name of the plan group or soft macro from which to remove the feedthrough ports.

3. Remove feedthrough ports from nets – Choose whether to remove feedthrough ports from all top-level nets in the design (the default), or from specified nets. To remove feedthrough ports from specified nets, click in the text box and enter the name of the net from which to remove the feedthrough ports.

4. Remove feedthrough ports from voltage areas – Choose whether to remove feedthrough nets and ports from all voltage areas or from specified voltage areas in your design. The default is all.

5. Remove original and buffered feedthroughs – When the original option is selected, the tool is instructed to remove all feedthrough ports and nets that are determined to be original as well as the added ports and nets. Feedthrough ports with an original property are defined as all feedthrough ports on blocks (plan groups and soft macros) that were generated originally from a Verilog definition. The original feedthrough ports and their connections to the top-level nets are removed.

   When the buffered option is selected, the tool is instructed to remove the buffered feedthrough ports that are determined to be added as well as the unbuffered feedthroughs. Feedthrough ports with an added property were automatically generated by pin assignment (`fp_place_pins` command) or by pin-cutting (`analyze_fp_routing` command), or were created manually. The inserted buffers are removed along with the buffered feedthrough ports.

   Note:

   You can select both options together to remove both original and buffered feedthrough ports and nets.

   If this option is disabled, the placement of original feedthrough nets and ports on blocks and voltage areas and their top-level connections are retained and the added buffered feedthrough nets and ports from all nets on all blocks and voltage areas are removed.

6. Click OK or Apply to remove feedthrough nets and ports from your design.

   The feedthrough ports are removed as they are processed. The feedthrough nets are removed after the last feedthrough port of a net is removed. The hierarchy preservation is updated on-the-fly both inside the blocks and at the top level.

Refining the Pin Assignment

You can analyze and evaluate the quality of the pin assignment results by checking the placement of soft macros pins in the design and the pin alignment. You can then refine the pin assignment based on the results.

This section includes the following topics:

- Checking the Pin Assignment
- Checking the Pin Alignment

Checking the Pin Assignment

You can check or verify the placement of soft macro pins in a design. After pins have been assigned, added, or moved, you can quickly verify if there are any spacing errors or missing pins. The relevant constraint values are read from the Pin Assignment Constraints dialog box (`set_fp_pin_constraints` command) to perform the pin placement checks.

To check the placement of soft macro pins,

1. Choose Pin Assignment > Check Pin Assignment.

   The Check Pin Assignment dialog box appears.

   Alternatively, you can use the `check_fp_pin_assignment` command.

2. Set the options as needed

   - Block Level Check - If you want to perform checks on top-level pins, select the “Block level” option. By default, checks are performed on soft macro pins.
   - Macro objects to check pins - Choose whether to perform pin checking within all soft macro pins or within only specified soft macro pins. The default is “All”. Checks are applied to pins on all soft macros in the design. If you select “Specified”, enter the list of macros to perform the check.
   - Macro types - Choose the types of macros where pin placement is checked. Select “All”, “Soft macro only” or “Hard macro only” based on your requirements.

- Nets to check pins - Choose whether to perform pin placement checks only on soft macro pins and relevant hard macro pins that are connected to the nets you specify. If you select the “Specified” option, enter a collection of net names in the text box. By default, checks are performed on all relevant soft macro pins and hard macro pins.
- Pin types - Specifies the types of pins to be checked: Signal, Power/Ground or both types.
- Spacing between signal pins - Checks if the spacing between any two signal pins is less than the number of wire tracks defined in the “Pin spacing” option (Pin Assignment Constraints dialog box) or the `-pin_spacing option` (`set_fp_pin_constraints` command).

  Spacing between preroute and pins - Checks if the spacing between pin and prerouted net objects is less than the number of wire tracks defined in the “Pin preroute spacing” option (Pin Assignment Constraints dialog box) or the `-pin_preroute_spacing` option (`set_fp_pin_constraints` command).

  IC Compiler reports an error for pins that are closer than the given number of wire tracks to a nearby preroute. The default is 3.

  If the design has wide (“fat”) wires, the tool checks the distance between the wire and pin on the same layer. It reports an error if the distance is smaller than the minimum distance found in the fat wire spacing table for the given layer of the wire and the width of the wire.
- Pins overlapped on same layers - Checks for shorts. A short occurs if two pins on the same layer touch or overlap each other.
- Pins on legal layers – Checks if all signal pins are placed within a predefined range on legal metal layers as specified in the “Allowed layers from” option (Pin Assignment Constraints dialog box) or the `-allow_layer` option (`set_fp_pin_constraints` command). You must specify a minimum and maximum range for the available metal layers. If a range is not specified, no checking is performed.
- Pins overlapped across layers - Reports the overlapping pins on different metal layers.
- Pins not centered on wire track - Checks if all pins are centered on a wire track.
- Missing signal pins - Checks for any missing signal pins.
- Off edge pins - Checks if part of a pin outline coincides with an edge. An error occurs if none of the pin edges coincide with the soft macro edge.
- Pins outside pin guide - Checks if the center of a pin is not inside the associated pin guide.
- Imperfect pin abutment/overlap - For pins on straddling or abutted edges, this option checks if the macro pins match.

- Non-routable pins on abutted edges - Checks for any unroutable pins on abutted edges. A macro pin is flagged as a single pin (unroutable) if any of one of the following scenarios occurs:
  - The macro pin is placed on a part of a macro edge that abuts another macro edge and the macro pin does not abut the other pin on the same metal layer to which it is connected.
  - The macro pin is placed on a part of a macro edge that coincides with the top-level cell boundary and the macro pin is connected to anything other than a terminal (top-level pin), unless the pin is connected solely to other pins on the same macro, and those pins are stacked on top of the macro pin.
  - The macro pin is placed on a part of a macro edge that coincides with the top-level cell boundary and the macro pin is connected solely to a terminal, but the terminal is not stacked on top of the macro pin on the same metal layer.

3. Select OK or Apply.

   If errors occur during the pin checking, first check your constraints to make sure you have captured them correctly and then check the log files to make sure you have applied them correctly. If the pin checks are good, but the pin placement still has errors, contact SolvNet online customer support or the Synopsys Technical Support Center for assistance.

Checking the Pin Alignment

You can check the pin alignment quality of results (QoR). You can view a list of nets with pins that are not optimally aligned by pin assignment. The list displays the total number of two connection nets among the specified nets that have at least one of their two connections inside a soft macro cell. It also reports the number and percentage of misaligned nets among the two connection nets.

You can also generate a detour report for pin feedthroughs.

To check the pin alignment,

1. Choose Pin Assignment > Check Pin Alignment.

   The Check Pin Alignment dialog box appears.

   Alternatively, you can use the `check_fp_pin_alignment` command.

2. Choose whether to check the pin alignment on all pins (the default) or on specified pins.

3. You can generate a detour report for pin feedthroughs. Choose the “Check pin feedthroughs detour” option.

   A pin detour report displays the net where the bounding box of the nonsoft macro pins on the net is smaller than the bounding box of all the pins on the net.

   You must specify a tolerance value for the detour report. The valid range is between 0 percent and 100 percent. The default value is 10 percent.

   If you do not specify the detour option, normal pin alignment checking is performed.

4. Select OK or Apply.

13

Performing Timing Budgeting

During the design planning stage, timing budgeting is an important step in achieving timing closure in a physically hierarchical design. The timing budgeting determines the corresponding timing boundary constraints for each top-level soft macro or plan group (block) in a design. If the timing boundary constraints for each block are met when they are implemented, the top-level timing constraints are satisfied.

Timing budgeting distributes positive and negative slack between blocks and then generates timing constraints in the Synopsys Design Constraints (SDC) format for block-level implementation.

Timing budgeting distributes the timing constraints from the top level to the block level, creating a new constraint set for each of the top-level blocks in the design. Budgeting also allocates slack between blocks for timing paths crossing the block boundaries. Timing budgeting propagates the timing constraints downward one hierarchical level at a time. Timing budgeting also propagates timing exceptions to block-level constraint sets.

You can use the timing budgeting feature in virtual flat mode to generate constraints for plan groups.

During timing budgeting, you can also create quick timing models for soft macros and plan groups and load them into your design.

This chapter includes the following sections:

Timing Budgeting Prerequisites

Before a cell can be budgeted, it has to meet the following requirements:

- It must be floorplanned and all cells must be placed and legalized. The closer the floorplan is to the final layout, the better the budgeting results will be.

  Note:
  A globally routed design, while not required, results in budgets that are more accurate.

- The top-level timing constraints must be loaded.
- The timing information available must be sufficient to successfully execute the timing analyzer on the cell to be budgeted. This requires complete timing views for all top-level soft and hard macros.

  You can run budgeting on a design that does not have timing models for all soft macros. If a soft macro does not have a timing model, the paths that interface with that macro are automatically marked as false paths.

- The timing information available must be sufficient to successfully execute the timing analyzer on the cell to be budgeted. This requires complete timing views for all top-level soft and hard macros.

  You can run budgeting on a design that does not have timing models for all soft macros. If a soft macro does not have a timing model, the paths that interface with that macro are automatically marked as false paths.

- Run a timing report on your top-level design before performing timing budgeting. If design errors are reported during the timing report, budgeting cannot be run successfully.
- Timing paths should have reasonably small slack values.

Budgeting is effective only when your design is close to meeting timing. It would be very difficult to meet timing if, for example, the worst negative slack was 10 nanoseconds and the clock period was also 10 nanoseconds. Timing is considered reasonable, however, if the worst negative slack is 10 percent or less but no more than 20 percent of the clock period. If it is more than 20 percent of the clock period, you should carefully review the results of the path timing report to determine the cause of the timing violations.

The most common cause of incomplete or missing constraints in the output timing boundary files is that no timing path passes through the port for which constraints are either incomplete or missing. This might be intended if false path specifications are loaded. However, it might be caused unintentionally, for example, by incomplete or incorrect timing views or by dangling nets.

Performing Pre-Budget Timing Analysis

You can perform pre-budget timing analysis on your design. The timing analysis is applied to blocks (plan group or soft macro) or the top-level design. Pre-budgeting timing analysis allows you to handle early checking of the timing rules and constraints set in the Synopsys Design Constraints (SDC) file before your design is submitted to the budgeting process.

By providing feedback on design and timing information, the timing checking analysis tool can help you determine the feasibility of your design and its timing constraints.

You can generate a report file that contains additional design information for validating the budgets assigned to each block in your design. You can use this report to analyze why certain budgets are assigned to each block port and to debug the budgets generated by the IC Compiler design planning timing budgeter.

The design information in the report file includes, but is not limited to, clocks in the design, timing constraints in the design, timing exceptions in the design, delay distribution on the worst timing path through the block ports, pins tied high or low, high negative slack on the timing path, and so on.

To generate a pre-budgeting timing analysis report file,

1. Choose Timing > Floorplan Timing Environment Check.

   The Timing Environment Analysis Options dialog box appears.

   Alternatively, you can use the `check_fp_timing_environment` command.
2. Select the Budgeting Analysis tab. This is the default.
3. Enter the hierarchical cell name or plan group name and the pin names on which to apply the budgeting analysis.

4. Select the options to use for performing budgeting analysis. The options you specify determine the type of information that is written to the output report file.

   You can show reports on:

   - Block pin statistics – For each block, prints the number of hierarchical pins that can and cannot be budgeted and the total number of pins on the block. A hierarchical pin is budgeted if timing constraints or exceptions are propagated to it. The default is on.
   - Unbudgetable pins – Reports why pins cannot be budgeted. The default is on. A hierarchical pin might not be budgeted for the following reasons:

     The pin is not connected to a net.

     The pin is not connected to a top-level net, but is connected to an internal net.

     The pin is not connected to an internal net, but is connected to a top-level net.

     Only fixed delay exists on one side of the pin. (There is nothing to budget but there will still be an input or output delay constraint on these pins.)

     Timing paths through pins are not constrained.

     The pin is constrained by a user-defined budget.

     The pin is connected to a power or ground net, or it has a `set_case_analysis` constraint on it.

     The pin is either an input or output clock port.
   - Unconstrained pins – For each block to be budgeted, lists the pin names that are not constrained. A pin is unconstrained if no timing path goes through it. The default is on.
   - Exception pins – For each block to be budgeted, lists the pin names that have timing exceptions propagated to it. The exceptions are `set_false_path`, `set_multicycle_path`, `set_min_delay`, or `set_max_delay`. The default is on.
   - Static logic pins – For each block to be budgeted, lists the pins that are set to a static logic state as a result of a case analysis statement. It reports if a block pin is tied high (1) or low (0) by a power or ground net in the logical netlist. It also reports if a block pin is set to a specific state by `set_case_analysis, set_logic_one, or set_logic_zero` constraints. The default is on.
   - Number of pin connections – Reports the number of pin connections from the startpoint and the endpoint of the timing paths going through one block to other blocks, through top-level standard cells, through pad cells, through top-level ports, and through ports on the plan group.

     Enter the number of pin connections to display for each type of net connection. The default is 1.

- Delay violating cells – Lists the top-level cells of interface paths that have cell delays greater than the specified percentage of the captured clock period. It traverses all the critical paths per clock domain through the block pin.

  The *percent* argument is required and must be greater than or equal to zero.

- Report to file – Prints the report to an output file in a single-line format. Enter the name of the output file. The default is off.

- Output in table format – Formats the report in a table. If the report is in a table, it is easier to read, but it might be difficult to parse using automated scripts. The default is off.

5. Select the Timer and Bottleneck Analysis tab to generate timing and bottleneck reports.

   You can show timing and bottleneck reports on:

   - Zero wire delay – Performs zero wire delay analysis and reports the negative-slack paths that have slack less than the negative percentage of the captured clock period. This will call timing analysis in a special mode, in which the wire delays are set to zero. The default is off.

     Specify a slack limit. The slack limit is the threshold percentage of the end clock period for each negative-slack path. A timing path is printed in the output report file if its slack is less than the negative percentage of the path's captured clock period.

   - Virtual IPO – Reports timing analysis results based on virtual in-place optimization. The default timing report from this option uses a slack threshold of zero. The default is off.

     Specify a slack limit for the timing paths in the virtual in-place optimization timing report. Only paths that have slack less than the slack percentage of the capture clock period are reported.

   - Bottleneck cells – Reports bottleneck cells. It lists the cells in the design that contribute to multiple timing violations. It lists the leaf cell, its reference, and the number of paths through the leaf cell that have timing violations. Based on this report, you can check the fanin and fanout logic to determine the possible cause of the timing bottleneck. The default is off.

   - Run Virtual IPO – Disable this option if you do not want virtual in-place optimization to run before reporting bottleneck cells. The default is to run virtual in-place optimization.

   - Slack limit – You can specify the slack limit for reporting bottleneck cells only. Only timing paths having slack less than the slack limit will be explored to locate bottleneck cells. The default is 0.

   - Maximum number of cells – You can specify the maximum number of bottleneck cells to report. This allows you to limit the number of bottleneck cells reported for a design. The default is 20.

- Maximum number of paths – You can optionally specify a maximum number of paths to report.

6. Click OK.

The following information is printed in the report file for the whole design:

- Negative-slack paths based on zero wire delay
- Bottleneck cells
- Timing report based on virtual in-place optimization

The remainder of the report is divided by soft macro or plan group. All the timing environment information related to one soft macro or plan group is printed together.

Running the Timing Budgeter

You can run the timing budgeter to perform proportional timing budgeting (the default) or advanced timing budgeting based on virtual in-place optimization for plan groups or soft macros in the design.

Proportional budgeting allocates positive and negative slack based on the actual physical circuits in each path. It also allocates slack on the path based on the delay on the path portions within individual blocks. Timing budgets are distributed proportionally to the worst or best case delay that is actually present for each path (per clock domain) inside the top-level soft macros. Portions of interface paths which are hierarchically in the top level are considered fixed delays just as hard macros are considered as fixed delays. This means that all slack for interface paths is budgeted solely to blocks.

After slack is allocated between blocks, the design is processed to generate a timing constraints file in the Synopsys Design Constraints (SDC) format for each block that has a timing model.

To run the timing budgeter,

1. Choose Timing > Allocate Budgets.

   The Allocate Budgets dialog box appears.

   Alternatively, you can use the `allocate_fp_budgets` command.

2. Set the options, depending on your requirements.

   - Hierarchical cells – Select this option to specify a list of hierarchical cell names for budgeting. By default, all plan groups and soft macros in the current design are budgeted.

- Fixed delay plan groups – Select this option if you want a list of cells to be treated as hard macros for budgeting purposes. This means that the budgets (input and output delays) calculated for cells designated as fixed delay cells will not have any slack allocated to them. This effectively allocates any time budget, which would normally be distributed to the fixed delay cell, to other soft macros which share in any interface timing paths associated with the fixed delay cell. The default is off.

  Enter the names of the cells you want to be treated as fixed delay cells in the text box.

- Fixed delay objects – Select this option to specify that certain objects that are implemented have fixed delays that will not change in subsequent optimizations. The list of objects includes plan groups, soft macro cells, hard macro cells, black boxes, hierarchical cells for plan groups, pins of soft macros with interface logic models, and pins of hierarchical cells for plan groups.

  The delay allocated to the timing paths passing through these pins during budgeting will be equal to the current delay.

- Black box cells – Select this option if you want a list of black box cells to be budgeted with black box timing models. The default is off.

  Enter the names of the black box cells in the text box. Black box cells are not budgeted unless they are included in this text box. Black box cells with timing models which are not included in the text box are treated by budgeting the same way as hard macros are treated.

- Output file name format specification – Enter the directory and naming style for the output SDC constraints file. The format string specifies the directory to write the SDC files and the type of names to output. If you specify *outputdir/i.sdc*, IC Compiler writes instance names. The string *outputdir/m.sdc* requests the tool to write reference names. One consolidated SDC file is written for designs containing multiply instantiated modules. See the man page for more information.

- Interblock logic – Select this option to perform top-level fixed delay budgeting. It is expected that the top-level delay will be optimized after the budgets are generated. The budgeted delay for the top-level path is exactly the same as the current delay at the top-level path segment. The default is on.

  When this option is deselected, top-level proportional budgeting is performed. The input and output delays for the ports on soft macros or plan groups are allocated, considering the top-level delay as proportional.

- Advanced budgeting based on Virtual IPO – Select this option to perform a virtual in-place optimization before allocating slack to predict possible timing improvements. The virtual in-place optimization imitates the physical optimization to improve timing by performing virtual driver sizing for the cells on critical nets by up-sizing gates to their highest drive strengths or by inserting repeaters on critical nets to fix timing and DRC violations. The improved timing can expose blocks that require additional synthesis.

This virtual in-place optimization does not perform any physical changes to the design; it is just held in memory for use with the budgeter and timing analyzer when what-if analysis is performed. The default is off.

Timing budgets are based on the estimated optimization.

- Incremental budgeting – Select this option to perform budgeting in incremental mode. During incremental budgeting, the timing budgeter retrieves block implementation information from designs with interface logic models (ILMs) and budgets the delays using this information. The block-level slack of the worst timing paths going through the block pins is taken into account. Pins associated with positive slack numbers will have their budgets tightened, and pins with negative slack numbers will have their budgets relaxed.

  Note:

  Use this option only on designs with interface logic models that were created by using the `create_ilm` or `create_ilm_models` commands. Designs created by these commands contain information about the block-level slack of the worst path through the hierarchical pins. If the design contains plan groups or if the ILMs in the design do not have pins with negative slack information, incremental budgeting will issue an error message and stop.

- Split long lines – Select this option if you want long SDC command lines in a budget file split into multiple lines by using the "\" Tcl continuation character at the end of the line. The default is on.

- Write constraints for partial netlists – The `allocate_fp_budgets` command does not create partial constraint files by default. Specify the `-print_partial_constraints` option to generate partial constraint files for blocks with a partial netlist. Partial netlists are commonly used in the on-demand-loading flow, the trace mode flow, and the interface logic model flow. The partial constraint file for blocks with a partial netlist is incomplete and contains only constraints for the portion of the block netlist at the top level.

- Enable QTM model generation – Select this option to enable quick timing model generation for the budgeted plan groups or soft macros. (This option is equivalent to the `create_qtm_model` command.) The delay arcs and the constraint arcs that are created in the quick timing models will represent the budgets on the plan groups or soft macros.

  You can load the quick timing models for the soft macros or plan groups after they are committed to soft macros. You can also generate a top-level timing report to check the QoR of your budgets and identify potential problems before running optimization on the individual blocks.

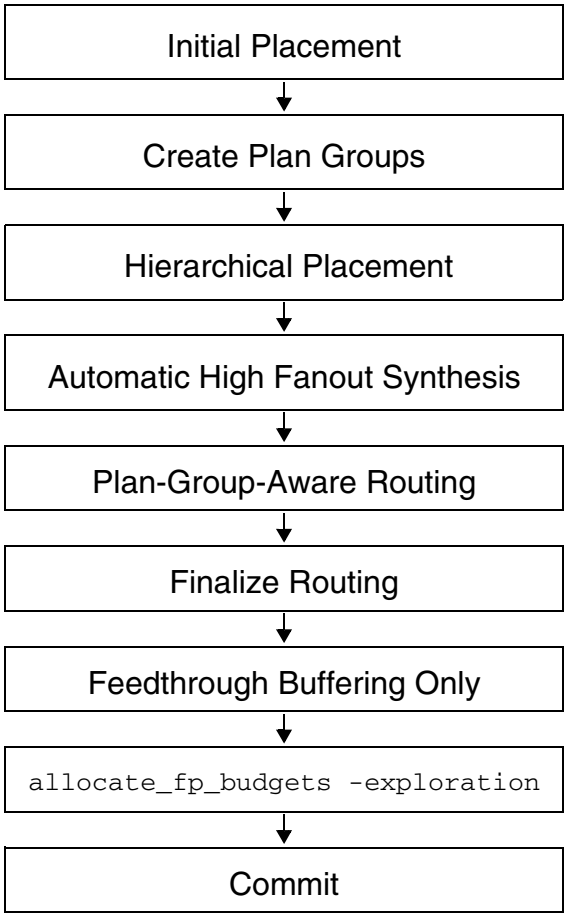
- Output QTM directory – Enter the name of the directory into which the quick timing model files are written after the timing budgeting is finished. The default is the current directory.

Performing Fast Time to Budget Analysis

You can perform timing budgeting on an early design version using the `-exploration` option to the `allocate_fp_budgets` command. Using this option, `allocate_fp_budgets` adjusts input and output delay and load values for block constraints. This option eliminates the need for an optimized netlist as a prerequisite for timing budget exploration.

Figure 13-1 shows a typical design planning flow for timing budget exploration.

Figure 13-1 Timing Budget Exploration Flow



Checking the QoR of the Timing Budgets

After you have loaded the quick timing models into your design, you can check the QoR of the timing budgets before running optimization for each individual block by using the `report_timing` command to create a top-level timing report to see if there are any paths with negative slack.

If the current budgets are either overconstrained or underconstrained, the timing report will show such paths as negative or positive slack. To check the results for any negative slack paths in the timing report, run a timing report for that path on a saved design with ILM models.

Zero slack budgeted paths are reported in the timing report with a slack of zero. Because the quick timing represents the timing of the blocks if they met the generated SDC budgets, the top-level timing should converge to a slack of zero.

Generating a Quick Timing Model From a CEL View

You can generate quick timing models for the current design or the soft macro cells inside the current design. The quick timing models are created, based on the clock information provided in the given SDC files, and the delay information in the quick timing models is based on the real delays in the design.

To generate quick timing models,

1. Choose Timing > Generate QTM Model.

   The Generated QTM Model dialog box appears.

   Alternatively, you can use the `generate_qtm_model` command.
2. Set the options, depending on your requirements.
   - Soft macro cells or current design – Select the soft macro cells, and specify a list of soft macro names from which to create quick timing models. By default, a quick timing model is created for the current design.
   - Cell name or SDC file name – Specify the list of cell names for the soft macros in the current design for which you want to create quick timing models.

     Specify the SDC files which contain clock information for the design or the soft macros. When creating quick timing models for the soft macros inside the current design, use this option together with the “soft macro cells” option and specify a list of cell names and SDC file names with this option to provide one SDC file for each soft macro cell.

- Output QTM directory – Specify the directory to where the quick timing model files will be written. By default, the files are written to the current working directory.

3. Click OK or Apply.

Generating a Quick Timing Model for the Partitions of Large Designs

Large designs that might not fit as comfortably as flat designs in IC Compiler are automatically partitioned and floorplanned. You can create a quick timing model for the partitions of large designs. The input to the quick timing model that is generated is the CEL view of the partition. The quick timing model, along with the top-level design, should fit comfortably in a design that is floorplanned in IC Compiler. The quick timing model timing generation for all CEL view partitions can occur in parallel.

The generated quick timing model output for the partitions of large designs will have

- Interface ports of the output quick timing model.

  The interface ports of the output quick timing model are based on the ports of the original CEL view. The direction of the quick timing model ports is the same as the direction of the ports in the Milkyway design.

- Sequential delay arcs (clock to output delay arcs)

  If the worst slack path through the output port does not start at the input port of the design, a sequential delay arc is created at the output port.

- Combinational delay arcs (input to output delay arcs)

  If the worst slack path through the output port starts at the input port of the design, a combinational delay arc is created.

- Setup and hold arcs (clock to input constraint arcs)

  The worst slack path for maximum and minimum delay mode are used to create setup and hold arcs.

- Load information about input pins and drive information about output pins

  The load value on the input port is the sum of all the pin and wire capacitances that are connected to the input port. The driving cell of the output port is the cell that directly drives the net connected to the output port.

Quick Timing Model Command Summary

Table 13-1 summarizes the commands in the quick timing model flow.

Table 13-1 IC Compiler Quick Timing Model Command Summary

To do this	Use this command
Begin defining a quick timing model.	`create_qtm_model`
Create a constraint arc.	`create_qtm_constraint_arc`
Create a delay arc for a quick timing model.	`create_qtm_delay_arc`
Create a drive type in a quick timing model description.	`create_qtm_drive_type`
Create a generated clock for the model.	`create_qtm_generated_clock`
Create a load type for a quick timing model description.	`create_qtm_load_type`
Create a path type in the quick timing model.	`create_qtm_path_type`
Create a quick timing model clock.	`create_qtm_clock`
Create a quick timing model port.	`create_qtm_port`
Create a quick timing model from a CEL view.	`generate_qtm_model`
Report model data.	`report_qtm_model`
Save the quick timing model.	`save_qtm_model`
Set a global parameter.	`set_qtm_global_parameter`
Set drive on a port.	`set_qtm_port_drive`
Set load on ports.	`set_qtm_port_load`
Set various technology parameters.	`set_qtm_technology`
Write out the quick timing model.	`write_qtm_model`

Performing Timing Budgeting On Plan Groups

If you have a design with plan groups, the top-level design will contain the complete design netlist. The `allocate_fp_budgets` command generates the SDC constraints and attributes for all the individual plan groups. These constraints and attributes are then transferred to the individual soft macro blocks when the hierarchy is committed. See "Converting Plan Groups to Soft Macros" in Chapter 14.

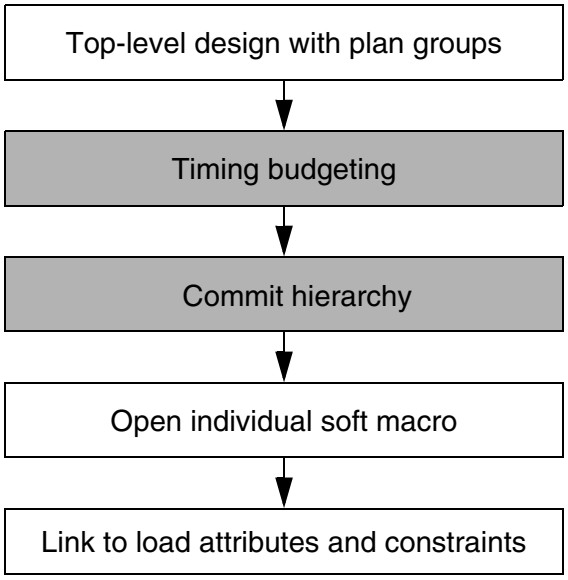
Note:

The automatic transfer of budgeted timing SDC constraints and attributes works on designs with single or multiple scenarios.

Timing budgeting on plan groups runs with full-chip timing. It honors the pin locations and the feedthrough nets assigned to the plan groups.

Figure 13-2 shows the flow for performing timing budgeting on plan groups.

Figure 13-2 Performing Timing Budgeting on Plan Groups



Note:

After committing the hierarchy using the `commit_fp_plan_groups` command, the created soft macros do not have timing models. At this point, you might encounter a few warning messages. These messages will not affect any physical design operations, such as refining the pin assignment.

If you want to perform an early timing check at the top level after you commit the hierarchy, do the following:

1. Save the soft macro CEL views by using the `save_mw_cel -hierarchy` command.
2. Close the soft macro CEL views by using the `close_mw_cel` command, which leaves open only the CEL view for the top-level design.
3. Generate interface logic models (ILMs) for the soft macros from the top level by using the `create_ilm_models` command.
4. Run a top-level timing report using the ILMs.

   This top-level timing report uses virtual-route based timing only, as there is no global route or parasitic information for the ILMs.

Budgeting with Multiply Instantiated Modules

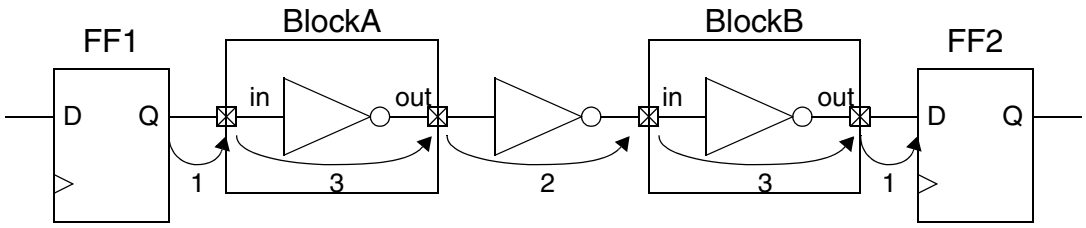
When multiply instantiated modules are present in the design, you must consider how constraints are treated for each module. When budgeting according to the master names of the multiply instantiated modules, you can generate a single SDC constraint file which contains a single set of constraints for each individual block of the multiply instantiated modules. A single SDC constraint file allows you to run a single optimization only once. You can also create multiple constraint files, one for each soft macro. Multiple constraint files are generated if you budget according to instance names.

To generate a single master constraints file, use the `allocate_fp_budgets -file_format_spec` *`output_dir`*`/m.sdc` command, which generates one master SDC file for each set of multiply instantiated modules. To generate a separate constraints file for each multiply instantiated module, use the `allocate_fp_budgets -file_format_spec` *`output_dir`*`/i.sdc` command.

Conflicts might occur when budgeting according to instance names. In this case, the strictest constraint is used if a conflict occurs.

Figure 13-3 shows an example design containing multiply instantiated modules.

Figure 13-3 Multiply Instantiated Module Design



In this example the clock period is 10 ns and the budgets have been allocated along the timing path as shown in the figure. If BlockA and BlockB are independent, the set of corresponding input and output constraints for each block are:

BlockA:

```
set_input_delay  1 [get_ports in]
set_output_delay 6 [get_ports out]
```

BlockB:

```
set_input_delay  6 [get_ports in]
set_output_delay 1 [get_ports out]
```

If BlockA and BlockB are multiply instantiated modules, the delays cannot be correctly combined by choosing the strictest value for each constraint. The combination of the delays would result in the following constraint which is impossible to meet for timing closure:

```
set_input_delay  6 [get_ports in]
set_output_delay 6 [get_ports out]
```

Instead, the budget values for the two multiply instantiated blocks are considered and the delay combination resulting in the strictest budget is kept. For the example design, either of the following constraint sets produces the correct result:

```
set_input_delay  1 [get_ports in]
set_output_delay 6 [get_ports out]
```

or

```
set_input_delay  6 [get_ports in]
set_output_delay 1 [get_ports out]
```

If the connectivity of the blocks gives a different budget for each block, the strictest budgeting constraint is created. Given this set of constraints:

BlockA:

```
set_input_delay  2 [get_ports in]
set_output_delay 6 [get_ports out]
```

BlockB:

```
set_input_delay  6 [get_ports in]
set_output_delay 1 [get_ports out]
```

The constraint defining the strictest budget is kept as shown in the following example.

```
set_input_delay  2 [get_ports in]
set_output_delay 6 [get_ports out]
```

This approach is used in designs with plan groups and interface logic models. For multicorner-multimode designs, constraints for each scenario are resolved independently.

Table 13-2 shows a list of conventions used when writing constraints to a combined SDC file.

Table 13-2 Combined SDC File Constraint Guidelines and Examples

Command	Description and examples	
create_clock	Clocks might connect to different pins in different instances. Also different clocks might connect to the same pin, in different instances. These conflicts are resolved by creating all the clocks that connect to a specific pin in any instance on that pin in the Master. This might result in multiple clock definitions on the same pin, but the timer can use all of them.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`create_clock -name clk [get_ports {clk}]`	`create_clock -name clk`	`create_clock -name clk [get_ports {clk}]`
`create_clock -name clk [get_ports {clk1}]`	`create_clock -name clk [get_ports {clk2}]`	`create_clock -name clk [get_ports {clk1 clk2}]`
`create_clock -name clk1 [get_ports {clk}]`	`create_clock -name clk2 [get_ports {clk}]`	`create_clock -name clk1 -add [get_ports {clk}]` `create_clock -name clk2 -add [get_ports {clk}]`
`create_clock -name clk [get_ports {clk}]`	`create_clock -name clk [get_pins {U1/A}]`	`create_clock -name clk list [[get_ports {clk}] [get_pins {U1/A}]]`

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
create_clock <exception_clock>	Virtual clocks are generated to model exceptions. If an exception affects one or more of the multiply instantiated modules, the corresponding virtual clock is created.	
create_generated_clock	Clocks generated in different instances that have unique names are treated the same as they are in individual SDC files. If generated clocks are created with the same name have any conflicting attributes (options `-source`, `-multiply_by`, `-divide_by`, `-invert`, `-master_clock`), a warning is issued and a clock is selected for output to the SDC file.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`create_generated_clock -name clk_g1 -source U1/A -multiply_by 2 U2/B`	`create_generated_clock -name clk_g2 -source U1/C -multiply_by 2 U2/D`	`create_generated_clock -name clk_g1 -source U1/A -multiply_by 2 U2/B` `create_generated_clock -name clk_g2 -source U1/C -multiply-by 2 U2/D`
`create_generated_clock -name clk_g -source U1/A -multiply_by 2 U2/B`	`create_generated_clock -name clk_g -source U1/C -multiply_by 2 U2/D`	`create_generated_clock -name clk_g -source U1/A -multiply_by 2 U2/B`
set_clock_uncertainty	The uncertainty set on clocks is valid for the entire design and is the same for all multiply instantiated modules. If uncertainty is defined on ports or pins, the worst-case uncertainty value is used in the merged SDC.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_clock_uncertainty 0.5 [get_clocks clk]`	`set_clock_uncertainty 0.5 [get_clocks clk]`	`set_clock_uncertainty 0.5 [get_clocks clk]`
`set_clock_uncertainty 0.5 [get_ports clk1]`	`set_clock_uncertainty 0.6 [get_ports clk1]`	`set_clock_uncertainty 0.6 [get_ports clk1]`
set_clock_latency	The largest value among all latency values in the different SDC files for the `-max` option for the same clock. Similarly, the smallest value is chosen for the `-min` option.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
`set_clock_latency -max 0.2 [get_clocks {clk}]`	`set_clock_latency -max 0.3 [get_clocks {clk}]`	`set_clock_latency -max 0.3 [get_clocks {clk}]`
`set_clock_latency -min 0.1 [get_clocks {clk}]`	`set_clock_latency -min 0.2 [get_clocks {clk}]`	`set_clock_latency -min 0.1 [get_clocks {clk}]`
`set_clock_transition`	The same command is written for all multiply instantiated modules in the combined SDC file.	
`set_propagated_clock`	The same command is written for all multiply instantiated modules in the combined SDC file.	
`set_input_delay`	Two `set_input_delay` commands are generated in the combined SDC file if the `set_input_delay` statement for the same pin refers to a given clock in one instance and a different clock in a another instance. If the delay values on the same port with respect to the given clock are different for different instances, the value defining the strictest budget is selected.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_input_delay 2.3 -clock [get_clocks {clk1}] -max [get_ports {in}]`	`set_input_delay 5.8 -clock [get_clocks {clk2}] -max [get_ports {in}]`	`set_input_delay 2.3 -clock [get_clocks {clk1}] -max [get_ports {in}] set_input_delay 5.8 -add_delay -clock [get_clocks {clk2}] -max [get_ports {in}]`
`set_output_delay`	Two `set_output_delay` commands are generated in the combined SDC file if the set_output_delay statement for the same pin refers to a given clock in one instance and a different clock in a another instance. If the delay values on the same port with respect to the given clock are different for different instances, the value defining the strictest budget is selected.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_output_delay 2.3 -clock [get_clocks {clk1}] -max [get_ports {in}]`	`set_output_delay 5.8 -clock [get_clocks {clk2}] -max [get_ports {in}]`	`set_output_delay 2.3 -clock [get_clocks {clk1}] -max [get_ports {in}] set_output_delay 5.8 -clock [get_clocks {clk2}] -add_delay -max [get_ports {in}]`

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples		
set_false_path	If the set_false_path exception refers only to clocks, pins or ports outside the multiply instantiated modules, it is modeled by using virtual clocks. If some of the pins are inside the multiply instantiated modules, a one-to-one correspondence is required for all blocks. Otherwise the set_false_path constraint is dropped from the combined SDC file.		
	Module1.sdc	*Module2.sdc*	*Combined.sdc*
	`set_false_path -from [get_clocks {clk}]`	`set_false_path -from [get_clocks {clk}]`	`set_false_path -from [get_clocks {clk}]`
	`set_false_path -from [get_clocks {clk_virtual1}]`	`set_false_path -from [get_clocks {clk_virtual2}]`	`set_false_path -from [get_clocks {clk_virtual1}]` `set_false_path -from [get_clocks {clk_virtual2}]`
	`set_false_path -from [get_pins {U1/A}]`	`set_false_path -from [get_pins {U1/A}]`	`set_false_path -from [get_pins {U1/A}]`
	`set_false_path -from [get_pins {U1/A}]`	`set_false_path -from [get_pins {U2/A}]`	no output for the exception
set_multicycle_path	If the set_multicycle_path exception refers only to clocks, pins, and ports outside the multiply instantiated modules, it is modeled by using virtual clocks. If all of the pins are inside multiply instantiated modules, a set_multicycle_path statement is output in the combined SDC file if there is a one-to-one correspondence for all the multiply instantiated modules.		
	Module1.sdc	*Module2.sdc*	*Combined.sdc*
	`set_multicycle_path -from [get_clocks {clk}]`	`set_multicycle_path -from [get_clocks {clk}]`	`set_multicycle_path -from [get_clocks {clk}]`
	`set_multicycle_path -from [get_clocks {clk_virtual1}]`	`set_multicycle_path -from [get_clocks {clk_virtual2}]`	`set_multicycle_path -from [get_clocks {clk_virtual1}]`

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
`set_multicycle_path -from [get_clocks {clk_virtual2}] set_multicycle_path -through [get_pins {U1/A}]`	`set_multicycle_path -through [get_pins {U1/A}]`	`set_multicycle_path -through [get_pins {U1/A}]`
`set_multicycle_path -through [get_pins {U1/A}]`	`set_multicycle_path -through [get_pins {U2/A}]`	No `set_multicycle_path` statement.
`set_min_delay`	If the `set_min_delay` exception refers only to clocks, pins, and ports outside the multiply instantiated modules, it is modeled by using virtual clocks. If all of the pins are inside multiply instantiated modules, a `set_min_delay` statement is output in the combined SDC file if there is a one-to-one correspondence for all the multiply instantiated modules.	
`set_max_delay`	If the `set_max_delay` exception refers only to clocks, pins, and ports outside the multiply instantiated modules, it is modeled by using virtual clocks. If all of the pins are inside multiply instantiated modules, a set_max_delay statement is output in the combined SDC file if there is a one-to-one correspondence for all the multiply instantiated modules.	
`set_drive`	If different `set_drive` values exist for the same port in different instances, largest resistance value is used for the `-max` option. The smallest resistance value is used for the `-min` option. The options `-rise` and `-fall` are treated separately.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_drive 2 -max [get_ports {in}]`	`set_drive 3 -max [get_ports {in}]`	`set_drive 3 -max [get_ports {in}]`
`set_drive -min [get_ports {in}]`	`set_drive 3 -min [get_ports {in}]`	`set_drive 2 -min [get_ports {in}]`
`set_driving_cell`	The library cell with the largest drive resistance to output is selected if there are different library cell and pin paths for a given port in different instances. If the `set_driving_cell` command for one port has the same library cell and pin for different instances, but the `input_transition_rise` or the `input_transition_fall` numbers are different, the largest number for -max and the smallest number for -min is output.	

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_driving_cell -lib_cell INVX2 -pin Z [get_ports {in}]`	`set_driving_cell -lib_cell INVX4 -pin Z [get_ports {in}]`	`set_driving_cell -lib_cell INVX2 -pin Z [get_ports {in}]`
`set_driving_cell -lib_cell INVX2 -pin Z -input_transition_rise 0.5 -max [get_ports {in}]`	`set_driving_cell -lib_cell INVX2 -pin Z -input_transition_rise 0.7 -max [get_ports {in}]`	`set_driving_cell -lib_cell INVX2 -pin Z -input_transition_rise 0.7 -max [get_ports {in}]`
`set_input_transition`	The largest value for `-max` conditions and the smallest value for `-min` conditions are chosen if different input transition numbers exist for the same port in different instances. The `-rise` and `-fall` options are treated separately.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_input_transition -max 0.5 [get_ports {in}]`	`set_input_transition -max 0.7 [get_ports {in}]`	`set_input_transition -max 0.7 [get_ports {in}]`
`set_input_transition -min 0.5 [get_ports {in}]`	`set_input_transition -min 0.7 [get_ports {in}]`	`set_input_transition -min 0.5 [get_ports {in}]`
`set_load -pin_load`	The largest value for `-max` and the smallest value for `-min` are chosen if the `set_load -pin_load` statements for the same port on different instances have different values.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_load -pin_load 10 -max [get_ports {out}]`	`set_load -pin_load 15 -max [get_ports {out}]`	`set_load -pin_load 15 -max [get_ports {out}]`
`set_load -pin_load 10 -min [get_ports {out}]`	`set_load -pin_load 15 -min [get_ports {out}]`	`set_load -pin_load 10 -min [get_ports {out}]`
`set_load -wire_load`	The largest value for `-max` and the smallest value for `-min` are chosen if the `set_load -wire_load` statements for the same port on different instances have different values.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
`set_load -wire_load 10 -max [get_ports {out}]`	`set_load -wire_load 15 -max [get_ports {out}]`	`set_load -wire_load 15 -max [get_ports {out}]`
`set_load -wire_load 10 -min [get_ports {out}]`	`set_load -wire_load 15 -min [get_ports {out}]`	`set_load -wire_load 10 -min [get_ports {out}]`
`set_port_fanout_number`	The largest value is output if the `set_port_fanout_number` for the same port on different instances have different values.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_port_fanout_number 10 [get_ports {out}]`	`set_port_fanout_number 20 [get_ports {out}]`	`set_port_fanout_number 20 [get_ports {out}]`
`set_logic_zero`	A one-to-one correspondence is required for all the blocks if some of the pins with `set_logic_zero` statements are inside the multiply instantiated modules. The `set_logic_zero` constraint is dropped from the combined SDC file otherwise. The same is true for `set_logic_zero` statements on ports.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_logic_zero [get_ports {in}]`	`set_logic_zero [get_ports {in}]`	`set_logic_zero [get_ports {in}]`
`set_logic_zero [get_ports {in}]`	`no set_logic_zero statement for port in`	no `set_logic_zero` statement for port in
`set_logic_one`	A one-to-one correspondence is required for all the blocks if some of the pins with `set_logic_one` statements are inside the multiply instantiated modules. The `set_logic_one` constraint is dropped from the combined SDC file otherwise. The same is true for `set_logic_one` statements on ports.	
`set_case_analysis`	A one-to-one correspondence is required for all blocks in the pin name and case analysis value if the pins are inside multiply instantiated modules. Otherwise the `set_case_analysis` constraint is dropped from the combined SDC file. The same is true for `set_case_analysis` statements on the ports.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_case_analysis 0 [get_ports {in}]`	`set_case_analysis 0 [get_ports {in}]`	`set_case_analysis 0 [get_ports {in}]`

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
`set_case_analysis 0 [get_ports {in}]`	`set_case_analysis 1 [get_ports {in}]`	No `set_case_analysis` statement generated
`set_case_analysis 0 [get_ports {in}]`	No `set_case_analysis` statement for port in	No `set_case_analysis` statement generated
`set_operating_conditions`	Operating conditions must match exactly for all multiply instantiated module instances for the generation of `set_operating_conditions` statements.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_operating_conditions -max BCCOM`	`set_operating_conditions -max BCCOM`	`set_operating_conditions -max BCCOM`
`set_operating_conditions -max BCCOM`	`set_operating_conditions -max WCCOM`	No `set_operating_conditions` statement generated.
`set_disable_timing`	The list of disabled cells, pins or ports in the current design match exactly for all multiply instantiated module instances for the generation of `set_disable_timing` statements.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_disable_timing -from A -to Z [get_cells {U1}]`	`set_disable_timing -from A -to Z [get_cells {U1}]`	`set_disable_timing -from A -to Z [get_cells {U1}]`
`set_disable_timing -from A -to Z [get_cells {U1}]`	`set_disable_timing -from B -to Z [get_cells {U1}]`	No `set_disable_timing` statement generated
`set_disable_timing -from A -to Z [get_cells {U1}]`	No corresponding `set_disable_timing` statement.	No `set_disable_timing` statement generated
`set_clock_gating_check`	A `set_clock_gating_check` statement is written to the combined SDC file if it is present in at least one of the SDC files generated for individual instances. If two different `set_clock_gating_check` statements are applied to equivalent objects in the two SDC files, the strictest constraint is written.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
`set_clock_gating_check -setup 0.5 [get_cells {U1}]`	`set_clock_gating_check -setup 0.5 [get_cells {U1}]`	`set_clock_gating_check -setup 0.5 [get_cells {U1}]`
`set_clock_gating_check -setup 0.5 [get_cells {U1}]`	`set_clock_gating_check -setup 0.7 [get_cells {U2}]`	`set_clock_gating_check -setup 0.5 [get_cells {U1}]` `set_clock_gating_check -setup 0.7 [get_cells {U2}]`
`set_clock_gating_check -setup 0.5 [get_cells {U1}]`	`set_clock_gating_check -setup 0.7 [get_cells {U1}]`	`set_clock_gating_check -setup 0.7 [get_cells {U1}]`
`set_max_time_borrow`	The worst case max time borrow value is written to the merged SDC file.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_max_time_borrow 0.5 {latch1}`		`set_max_time_borrow 0.5 {latch1}`
`set_max_time_borrow 0.5 {latch1}`	`set_max_time_borrow 0.6 {latch1}`	`set_max_time_borrow 0.5 {latch1}`
`set_max_transition`	The worst case max transition time value is written to the merged SDC file.	
`set_max_capacitance`	The worst case `set_max_capacitance` value is written to the merged SDC file.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_max_capacitance 1.0 [get_ports port1]`	`set_max_capacitance 1.5 [get_ports port1]`	`set_max_capacitance 1.0 [get_ports port1]`
`set_min_capacitance`	The worst case `set_min_capacitance` value is written to the merged SDC file.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_min_capacitance 0.3 [get_ports port1]`	`set_min_capacitance 0.4 [get_ports port1]`	`set_min_capacitance 0.4 [get_ports port1]`

Table 13-2 Combined SDC File Constraint Guidelines and Examples (Continued)

Command	Description and examples	
set_max_fanout	The worst case `set_max_fanout` value is written to the merged SDC file.	
Module1.sdc	*Module2.sdc*	*Combined.sdc*
`set_max_fanout 10 [get_ports port1]`	`set_max_fanout 15 [get_ports port1]`	`set_max_fanout 10 [get_ports port1]`

Performing Hierarchical Signal Integrity Budgeting

Timing budgeting can consider crosstalk effects across soft macro boundaries when generating SDC constraints. Hierarchical signal integrity budgeting can extract effective drive strength for input pins and coupling capacitance for interface nets. The timing budgeter can retrieve crosstalk analysis information from a hierarchical signal integrity module and then write out this information to block pin attributes. Bringing the top-level crosstalk information into the block level during budgeting makes it easier for block-level crosstalk analysis to consider the crosstalk effects at the top level.

Before running timing budgeting, set the `enable_hier_si` variable to true to enable signal integrity budgeting and take the crosstalk timing effects across soft macro boundaries into account.

```
icc_shell> set enable_hier_si true
```

Run the timing budgeting command.

```
icc_shell> allocate_fp_budgets
```

IC Compiler will process the hierarchical signal integrity information during budgeting and will write out this information about the block's pin attributes.

The total effective drive strength is written out on block pins to represent the top-level aggressor drive strength, and the total coupling capacitance is written out on block pins to represent the total coupling capacitance in the block CEL view.

Block-Level Hierarchical Signal Integrity Flow

To run the block-level hierarchical signal integrity flow, do the following:

1. Enable the hierarchical signal integrity flow.

```
set enable_hier_si true
```

2. Define the signal integrity options used for analysis or optimization.

   ```
   set_si_options -static_noise true -delta_delay true
   ```

   The -static_noise true option reduces static noise.

   The `-delta_delay true` option optimizes the timing with crosstalk delay.

3. Perform routing and postroute optimization.

   ```
   route_opt -xtalk_reduction
   ```

   IC compiler performs crosstalk prevention during global routing and track assignment, and crosstalk analysis and optimization during the postroute optimization phase.

4. Create a Milkyway signal-integrity-aware interface logic model (ILM).

   ```
   create_ilm -include_xtalk
   ```

   The `-include_xtalk` option includes crosstalk information, such as boundary net aggressor data, effective net resistance, total capacitance, and coupling capacitance.

Top-Level Hierarchical Signal Integrity Flow

To run the top-level hierarchical signal integrity flow, do the following:

1. Enable the hierarchical signal integrity flow.

   ```
   set enable_hier_si true
   ```

2. Define the signal integrity options used for analysis or optimization.

   ```
   set_si_options -static_noise true -delta_delay true
   ```

   The -static_noise true option reduces static noise.

   The `-delta_delay true` option optimizes the timing with crosstalk delay.

3. Perform routing and postroute optimization.

   ```
   route_opt -xtalk_reduction
   ```

   IC Compiler performs crosstalk prevention during global routing and track assignment, and crosstalk analysis and optimization during the postroute optimization phase.

Performing Post-Budget Timing Analysis

After budgeting a design, you can perform post-budget timing analysis to generate a report containing budgeted and actual delays through a hierarchical block.

Note:
This command must be run during the same IC Compiler timing budgeting session in which the budgets were created.

To perform post-budget timing analysis,

1. Choose Timing > Floorplan Budgeting Report.

   The Budgeting Report dialog box appears.

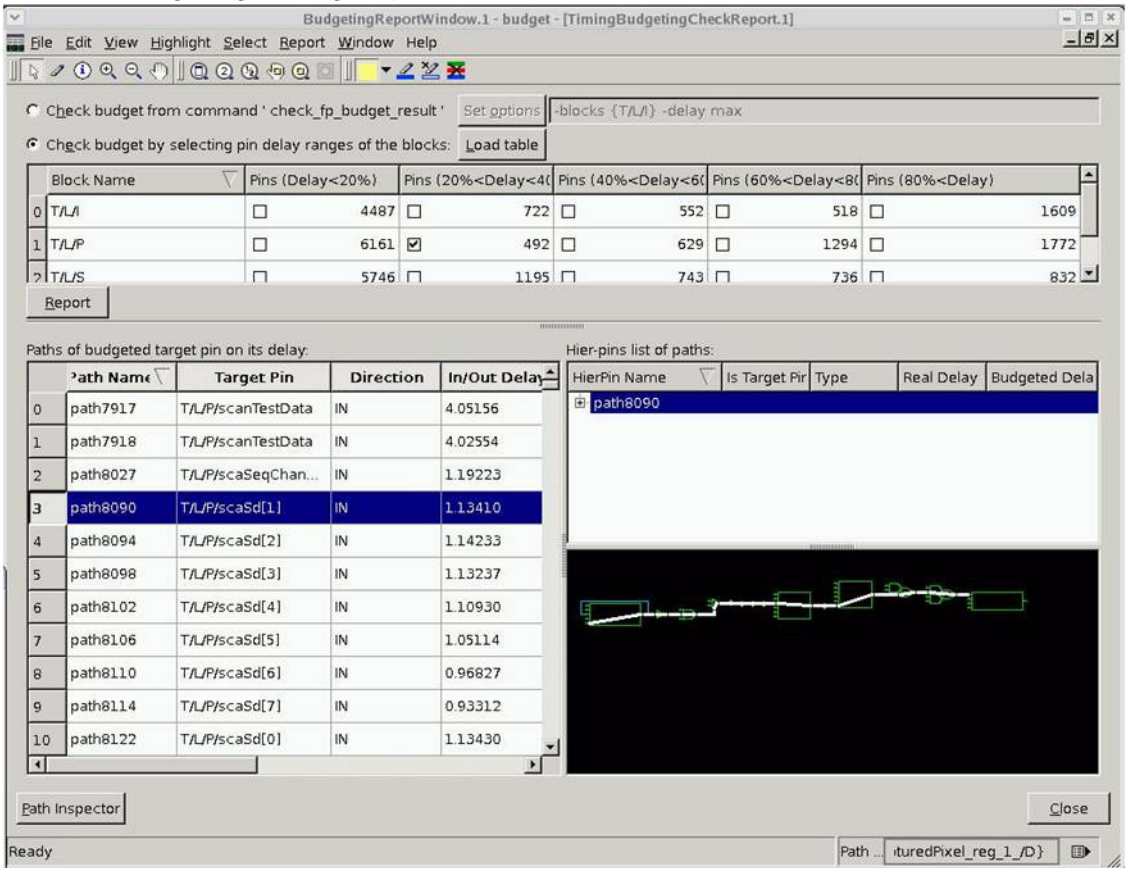
   Alternatively, you can use the `check_fp_budget_result` command.

2. Select “Check budget from command ‘check_fp_budget_result’” and enter the `check_fp_budget_result` command options `-blocks`, `-pins`, or `-file_name` based on your requirements. You can also select command options by clicking the “set option” button and entering the options –blocks, -pins and file name in the pop-up dialog. A list of paths is displayed according to the options provided.

   Alternately, select “Check budget by selecting pin delay ranges of the blocks” and click Load Table. The table is populated with Block Name and a count of pins within each delay category. Select the check box to display the list of paths within the category.

3. Select a path from the list and click Path Inspector. A graphical representation of the path is displayed.

Figure 13-4 Budgeting Timing Report Window



A timing path starts at a startpoint and ends at an endpoint. Timing paths are divided into segments. For each timing path segment, the report prints the actual and budgeted delay or the fixed delay values of the segment. You can use this information to analyze the process of generating budgets for your design.

You can also generate a post-budget timing analysis report using the `check_fp_budget_result` command. For each pin on a block, the report lists the worst case timing paths per endpoint clock domain that go through the pin. For each path segment on the timing path, the report lists the budgeted and actual delays for the path segment.

The `check_fp_budget_result` command uses the following syntax:

```
check_fp_budget_result
 [-blocks block]
 [-pins pin]
 -file_name output_design_report
```

Note:

If no option is specified, information for the whole design is written out.

Table 13-3 check_fp_budget_result Command Options

Command option	Description
`-blocks` *block*	Specifies the name of a plan group in the design. It reports the timing analysis information related to that particular plan group in the design. You can specify the value of block by using a collection generated by the `get_cells` command. By default, a budgeting analysis report is generated for all the plan groups in the design.
`-pins` *pin*	Specifies the name of a pin on a plan group in the design. It reports the timing analysis information related to that particular pin in the design. You can specify the value of pin by using a collection generated by the `get_pins` command. By default, a budgeting analysis report is generated for all the pins in all the plan groups.
`-file_name` *output_design_report*	Specifies the name of the output design report file. The design information is written to this file. This option is mandatory.

Performing Clock Latency Budgeting

You can perform clock latency budgeting. This budgeting feature uses information from files built by clock planning to include clock latencies associated with real clock tree branches that are external to the block being budgeted. Clock planning can create clock budgets for each plan group inside the design to specify the source and network latencies whether or not the clock has a sink inside the plan group.

The `allocate_fp_budgets` command creates virtual clocks to capture these latencies from clock planning. This budgeting feature also generates real latency values in budgets for physical clock ports created by clock planning on a block. The clock ports are created by the clock planning features. These latencies are associated with real physical ports.

Creating Budgets to Reflect Real Clock Latencies

After clock planning, you can create budgets to reflect real clock latencies. This consists of:

- Creating real clock latency values on clock ports for the block.

- Creating input and output virtual clocks for interface path endpoints that are partially outside the block.

  The format for defining virtual clocks created for latencies outside the block is

  *clock_name*_v_in

  and

  *clock_name*_v_out

Clocks launching (capturing) flip-flops on the top level have source latencies only.

Clocks launching (capturing) flip-flops in blocks have both source and network latencies.

Clock latency budgeting is on by default if timing budgeting (`allocate_fp_budgets` command) is run after clock planning.

You can enable (turn on) clock planning based latency values by setting the following variable in the icc_shell to `true` (the default) before performing timing budgeting:

```
set virtual_clocks_from_cp true
```

To disable (turn off) clock planning based latency values, set the following variable in the icc_shell to `false`.

```
set virtual_clocks_from_cp false
```

For a design which has synthesized clock trees in it which were not created by clock planning, you can have latencies for these existing clock trees included in budgeting by setting the following variable to a value of `true` before running the timing budgeter. The default is `false`.

```
set synthesized_clocks true
```

To disable (turn off) clock tree synthesis-based latency values, set the `synthesized_clocks` variable to `false`.

```
icc_shell> set synthesized_clocks false
```

Things to Look For

After you create budgets to reflect real clock latencies, check for the following:

- Ensure every real input clock port created by clock planning for a block budget has calculated source and network latency statements in the block budget.
- Ensure latency statements in the budgets reflect the actual clock network delays in the design.
- Ensure every input port `set_input_delay` and `set_output_delay` is referenced to a virtual clock that has the latency for the launch or capture clock associated with the path through the port.
- Check latencies in budgets against clock planning latency files.

  Note:

  Clock planning creates a source latency file and a network latency file for each clock port in the tcp_output directory.
- Check that latencies in budgets are only for clock ports defined in the block budget file.
- Check latencies in clock planning latency files against the clock skew report.

14

Committing the Physical Hierarchy

This chapter describes how to commit the physical hierarchy after finalizing the floorplan by converting plan groups into soft macros. Committing the hierarchy creates a new level of physical hierarchy in the virtual flat design by creating CEL views for selected plan groups. After committing the physical hierarchy, you can also “uncommit” the physical hierarchy by converting the soft macros back into plan groups.

In addition, this section also describes how to propagate top-level preroutes into soft macros, recover all pushed-down objects in child cells to the top-level, and uncommit the physical hierarchy by converting soft macros back into plan groups. It also describes how to commit plan groups with associated UPF power domains.

This chapter includes the following sections:

Converting Plan Groups to Soft Macros

You can convert selected plan groups or all plan groups with exclusive placement constraints into soft macro CEL views. The soft macros are created in the CEL view directory.

Prerequisites

Before converting plan groups to soft macros, your design should meet the following requirements:

- Exclusive plan groups, standard cells, and hard macros are legally placed.
- Traditional pin assignment or pin cutting flow is completed.
- In-place optimization, timing analysis, and timing budgeting are completed.

To convert plan groups to soft macros,

1. Choose Partition > Commit Plan Groups.

   The Commit Plan Groups dialog box appears.

   Alternatively, you can use the `commit_fp_plan_groups` command.
2. Set the options, depending on your requirements.
   - Specify whether to commit all plan groups or specified plan groups. The default is all.
   - Push down power and ground straps into the child cell – Select the option if you want power and ground straps copied (pushed) down into the newly created soft macro CEL views. The power and ground straps are removed from the top level. The default is off.
   - Commit to a new top cell – Select this option to commit the plan groups to a new top cell. A new top cell is created in your design where the top level changes occur. The default is off.
   - Top cell name – Enter the name of the new top-level cell.
3. Click Apply or OK.

Committing the physical hierarchy does the following:

- Converts the virtual flat design to a two-level hierarchical design.
- Removes standard cells and hard macros belonging to a plan group from the top cell and copies them down to the soft macro CEL which is created for the plan group.
- Creates nets in the soft macro CEL views based on the Hierarchy Preservation data.
- Pushes placement blockages and route guides down to the soft macro CEL views.

- (Optional) Pushes power and ground straps and standard cell preroutes down into each soft macro CEL view.

After the conversion from plan groups to soft macros is complete, all the standard cell instances of the plan groups become child cell instances of the corresponding converted soft macro cells and are deleted from the top cell. New corresponding cell instances of the new soft macro cells are created in the top cell and reside in the same location as the old plan groups.

Utilization of the new CEL views is the same as the utilization of the corresponding plan groups prior to running the `commit_fp_plan_groups` command. The default utilization is based on the sum of the area of standard cells and hard macros in the CEL view divided by the area in the core area bounding box of the CEL.

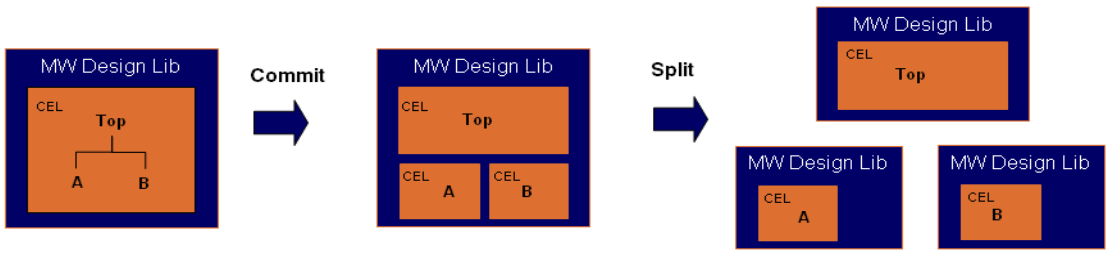
Splitting the Soft Macros

When you commit the soft macros, the IC Compiler tool creates a CEL view for each of the blocks in the same Milkyway design library. To allow different designers to work on different blocks without interference, you can use the `split_mw_lib` command to split the top-level Milkyway design into new Milkyway design libraries for each block and the top-level design.

After splitting, the new Milkyway design libraries inherit all the information, including the technology file and the reference library path, from the original Milkyway design.

Figure 14-1 shows the splitting of soft macros to create separate design libraries.

Figure 14-1 Splitting of Soft Macros



After block-level implementation, you can link the CEL views for the blocks in different design libraries to the top-level design by using the `set_mw_lib_reference` command.

Pushing Physical Objects Down to the Soft Macro Level

Physical objects (routing, route guides, blockages, cells, and cell rows) can be transferred (pushed down) from the top level to the soft macro level. You can control several options during the push down operation, including the layers to be copied, whether objects are moved or copied, if overlap is to be checked, whether new pins are created, whether blockages should be created, if a boundary region should be created, and other options.

To push physical objects down to the soft macro level,

1. Choose Partition > Push Down Objects.

   The Push Down Objects dialog box appears.

   Alternatively, you can use the `push_down_fp_objects` command.

2. Enable the “Push down objects” option. This is the default.

3. Set the remaining options, depending on your requirements.

   - Target soft macros – Select “All soft macros” (the default) or “Specified soft macros”. If you select the “Specified soft macros” option, enter the names of the soft macros and optionally specify the objects to be pushed down into the specified macros.
   - Types of objects to be pushed down – Select the types of objects to be pushed down. The default is Routing.

     Routing – This object type includes wires, paths, and vias. If a net is connected to these objects at the top level, but it does not exist in the soft macro, the net is created inside the soft macro.

     Route guides – This object type includes route guides and their corresponding attributes.

     Blockages – This object type includes metal layer blockages and soft or hard placement blockages.

     Cells – This object type includes all standard cells within the boundary of the soft macros. If a net is connected to these cells at the top level, but it does not exist in the soft macro, the net is created inside the soft macro. Any floating nets in the top level and child levels are removed. By default, all routing between cells is also pushed down. However, if the remove routing option is selected, the routing is removed from the top level and not pushed down into the child cell.

     Rows – This object type includes all cell rows that touch the specified soft macros. For row objects, the copy down and cut down options to cut type have the same behavior as copy down.

Shapes - This object type includes all rectilinear metal segments on routing layers that touch the soft macro. Note that net shapes, wires, paths, vias, and pins are handled as routing objects.

- Net type – You can specify the types of nets that are processed. You can specify more than one type of net.

  Power and ground – Select this option to process power and ground nets. This is the default. When power and ground nets are pushed down, the associated via cell instances are pushed down as well.

  A power and ground wire or path passing through the soft macro area is removed from the parent cell and moved into the child cell. If the power and ground net is connected to an existing port, new pins are created. If the power and ground net is not connected to an existing port, this wire, path, or via is not pushed down.

  Clock – Select this option to process clock wires.

  Clock wires or paths passing through the soft macro area are removed from the parent cell and moved into the child cell. If the clock net is connected to an existing port, new pins are created. If the clock net is not connected to an existing port, one new port and new pins are created.

  Signal – Select this option to process non-bus signal wires or paths.

  Non-bus signal wires or paths passing through the soft macro area are removed from the parent cell and moved into the child cell. If the signal net is connected to an existing port, new pins are created. If the signal net is not connected to an existing port, one new port and new pins are created.

- Cut type – You can specify how the tool should process the pushed-down objects.

  Push down – Creates a copy of the object in the soft macro and then deletes the object from the top level. This method applies to routing, route guides, blockages, and cell object types. This is the default.

  Cut down – This option is only valid for routing objects. Pins are created where the routing object touches the soft macro boundary, and the object is deleted from the top level. No copy of the object is created in the soft macro. For power and ground nets, the tool generates a Tcl file named *childCellName*_PGStrap.tcl. This Tcl file is used to create power straps for the design. For route guides and blockages, the behavior of the push down operation is the same for both cut down and copy down options.

  Copy down – Creates a copy of the object in the soft macro, but retains the object at the top level. For routes, cells, route guides, blockages, and shapes, you can either push down or copy down a top-level object. By default, the tool pushes down objects by creating a copy of the object in the soft macro, deleting the object from the top level, and creating pins for routing objects where the routing object touches the soft macro

boundary. The copy down option creates a copy of the object in the soft macro while retaining the object at the top level, and doesn't create pins for the routing objects of power and ground nets.

- Overlap checking – When overlap checking is on, limited checking of pushed down objects is performed. Objects overlapping on the same metal layer with a wire, path, via, or a pin with a different net type are flagged. Note that you must still run DRC checks to prevent shorts and other DRC violations. The default is off, and route overlaps are ignored even if they cause a short.
- Connect objects to child net – If you select this option, wires, cells, and shapes with a net ID are copied down into the soft macro and connected to the parent net. The default is off.
- Freeze push down nets – If you select this option, the routing constraints contained in the pushed down nets are frozen. The default is off.
- Allow feedthroughs – If you select this option, the command creates new ports for a feedthrough route and creates a new net at the top level for the split net. Otherwise it does not create new ports; if the top-level net has a connection with the soft macro, it creates pins at each crossing point.
- Allow partially overlap soft macros – If you select this option, cells which only partially overlap the soft macro boundary are pushed down.
- Horizontal offset and Vertical offset – You can specify the horizontal offset and vertical offset between the soft macro core and the cell rows. The default is 0.
- Create pins - If you select this option, pins are created when routes are copied down. This option only creates pins when the copy down option is selected, and the type of object pushed down is routing. The default is off, no pins are created.
- Remove routing - If you select this option and push down cells, routing will be removed from the top level and not be pushed down to the child cell.
- Partial overlap - By default, only cells that are completely within the soft macro boundary are pushed down. Using the partial_overlap option, cells that partially overlap the soft macro boundary are also pushed down. If a cell is partially overlapped with the block boundary and the partial_overlap option is not specified, routing blockages are created for the overlapped pins.
- Row offset - For row objects, this option implements a horizontal or vertical offset in microns between the soft macro core and the row object.
- Include shapes - This option controls the type of shapes that will be considered for push down. To process only shapes with a net connection, specify with_net_id. To process only nets not connected to any net, use without_net_id. By default, both types of shapes are processed.

- Object list - You can specify a list of specific objects to push down by using the object list option. The list can contain any supported object, such as nets, cells, wires, paths, vias, via arrays, route guides, blockages, and physical shapes. Note that partial buffer chains cannot be pushed down, you must select the entire buffer chain.
- Margin - If you select this option, a margin is created around the perimeter of the soft macro. Any object that overlaps with the margin boundary is pushed down and a blockage created inside the child cell to represent the top-level object.

4. Click OK or Apply.

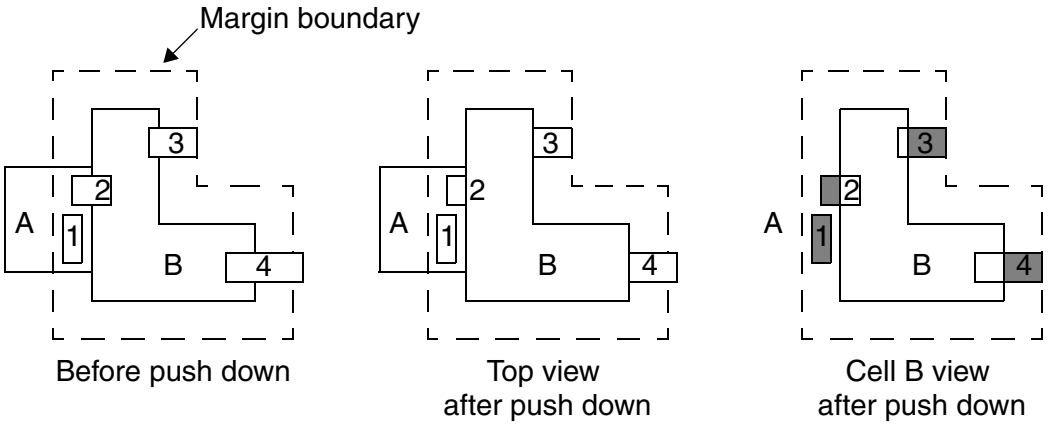
Using the Margin Option for Pushing Down Objects

Figure 14-2 shows the usage of the margin option to the push down objects based on offset from cell perimeter. The following `push_down_fp_objects` command treats routing objects based on the overlap with the cell B boundary and margin boundary.

```
push_down_fp_objects -object_type {routing} \
                     -margin {10, 10, 10, 10} [get_cells B]
```

Routing objects that overlap the boundary of cell B are pushed into cell B as routing, and objects within the margin are pushed down as route guides.

Figure 14-2 Using Margin to Control Routing Push Down



Pushing Down Routing in Multiply Instantiated Modules

Routes are pushed down differently in designs containing multiply instantiated modules. Only routes that overlap master instances when processing power and ground nets, or clock and signal nets without ambiguous connection, are pushed down. The tool checks the routing alignment between the master instance and non-master instances and issues a

warning message if alignment is not maintained. For power and ground nets, gaps are placed between the top-level power and ground straps and the PG pins for multiply instantiated module instances that are not aligned with the master instance and are not surrounded by power and ground rings. These gaps allow power and ground routing to connect the PG pins at a later time.

Pushing Physical Objects Up to the Top Level

Physical objects that were previously pushed down into a soft macro can be pushed back up to the top level. Note that when performing a consecutive push down and push up of objects, the objects will be pushed up to the top level, not their original location in the hierarchy.

To push up physical objects from the soft macros in the design to the top level,

1. Choose Partition > Push Up Objects.

   The Push Up Objects dialog box appears.

   Alternatively, you can use the `push_up_fp_objects` command.
2. Enable the “Push up objects” option. This is the default.
3. Set the remaining options, depending on your requirements.

   - Target soft macros – Select “All soft macros” (the default) or “Specified soft macros”. If you select the “Specified soft macros” option, enter the names of the soft macros from which to push up objects. By default, objects are pushed up from all soft macros in the current design.
   - Types of objects to be pushed up – Select the types of objects (routing, route guides, blockages, cells, rows, and shapes) to be pushed up to the top level. The default is Routing.

     Routing – This object type includes wires, paths, vias, and pins.

     Cells – This object type includes all standard cells within the boundary of the specified soft macros. Both cells and routing connected to the cells are pushed up, and floating nets are removed. If the remove routing option is selected, routing information is removed at the child level and is not pushed up. Note that partial buffer chains cannot be pushed up, you must select the entire buffer chain. No port punching occurs when pushing up selected child cells.

     Route guides –This object type includes route guides and their corresponding attributes.

     Rows – This object type includes all cell rows that touch the specified soft macros. The rows are not pushed up to the top level because the top level already contains the row objects; the rows are just removed from the soft macros.

Blockages – This object type includes all blockages on metal blockage layers, and soft or hard placement blockages. Attributes are also pushed up.

Shapes – This object type includes all rectangular and rectilinear metal segments that touch the specified soft macros on routing layers. You can control the type of nets that are pushed up by using the include shapes option. By default, unconnected wires, paths, and vias are also pushed up in this operation.

- Push up type – This option selects how objects should be treated during the push up process. Select push up to create a copy of the object at the top level and delete the object from the child level. Select copy up to create a copy at the top level and leave the child-level object intact.
- Selective object push up – You can enter a list of objects to push to the top level. If you select objects from a multiply instantiated module, all instances of the multiply instantiated module will be processed.
- Types of objects after push up from soft macros – Use this option with pushed up soft or hard macro pins (select the object type “Pins”, which includes top-level pins”) or with routing (select the object type “Routing”, which includes wires, paths, and vias”) only. You can specify the top-level object types after they are pushed up from the soft macro level. You can push up child level routing to top-level routing or to top-level pins if there are top-level ports on connected top-level nets. If the child level routing is pushed up to top-level pins and the routes are rectilinear, the pushed up routes are broken into rectangular electrically equivalent pins.

  Alternatively, you can push up soft or hard macro pins from the soft macro level back up to the top level as wire routing or as top-level pins.
- Layers of routings/pin objects – If you select this option, you can limit the objects that are pushed up to specific layers. You should use this option with routing and pin objects only.
- Top level nets connected to soft macros – If you select this option, you can specify a list of top-level interface nets for which routing, pin, cells, and shape object types will be pushed up. Only child objects with logical connections to those top-level nets are pushed up to the top level.
- Pushed down objects only – If you select this option, you can define the objects to be pushed up to the top level. By default, only the object types that were previously created during push down are pushed up to the top level. If you select “all”, the object types that are created solely inside soft macros are also pushed up to the top level.
- Child Objects – You can specify the child-level objects to be pushed up. The objects include nets, routing objects (wires, paths, vias, via arrays, and pins), route guides, blockages, cells, and shapes. By default, all object types specified in the objects type option section are pushed up.

4. Click OK or Apply.

Handling 45-Degree Redistribution Layer Routing

The `push_up_fp_objects` and `push_down_fp_objects` commands can support 45-degree redistribution layer routing. During push-down, the 45-degree routings are split at the intersecting points between the soft macro boundaries and where the 45-degree routings are added. Square-shaped pins are created at the intersection points during push-down; during push-up, these pins are deleted at the soft macro boundaries.

During push-up, the 45-degree routings are restored to the original state prior to push-down. (Wires are merged at the intersection points between the soft macro boundaries and routings.)

If a net has no ambiguous connection to a soft macro and a feedthrough is not allowed, routing that is collinear with a block boundary is pushed down.

Committing the Hierarchy of Plan Groups With Unified Power Format (UPF) Power Domains

The `commit_fp_plan_groups` command supports plan groups which have a unified power format (UPF) power domain associated with them. The power domain must be defined completely within the hierarchical cell instance of the plan group that is being converted to a soft macro. The `create_power_domain` command defines a power supply distribution network at the current scope (hierarchical level) or at the scope of a specified hierarchical instance. The `-scope` option specifies the name of an instance in which to create the power domain; it defines the domain boundary within the logic design. In a hierarchical UPF context, a “scope” is a hierarchical cell instance within which the power domain is completely contained.

During the commit hierarchy process, the top-level UPF power domains associated with the plan groups are pushed down into a new child cell view. This also makes it possible for the top-level design’s UPF constraints to be properly transferred and maintained. Embedded voltage areas and power domains associated with those voltage areas are also pushed into the child cell.

Near the end of the commit process, after the top-level hierarchical cell instance is replaced with a top-level child cell instance, all corresponding power domains are removed from the top-level cell, along with any voltage areas associated with the pushed down power domains. This is done to avoid the duplication of power domains across the hierarchy. Any top-level voltage area which was coincident with the plan group boundary, is also removed from the top cell and treated as DEFAULT_VA within the child cell instance.

The supply set handle specified by using the `-supply` option must be in a higher scope than the specified power domain. If extra supplies are defined by using the `create_power_domain -supply {extra_supplies}` command, only those supply sets can be used. In this case, no extra domain-independent nets can be used, unless they are bound to a supply set that is allowed or they are directly specified in the domain strategies. If no extra supply is defined, then all supply sets and domain independent nets within and above the power domain scope are available for the specified power domain.

Note:

If UPF power domains are not completely defined within the hierarchical cell instance, in other words they are larger than the plan groups associated with them, it means they might contain UPF elements that are outside the plan group and therefore, they cannot be committed.

Converting Soft Macros to Plan Groups

After plan groups are committed into soft macros, you can "uncommit" the physical hierarchy by converting the soft macros back into plan groups. The physical hierarchy is flattened into the parent.

Physical objects from the soft macro are "pushed up" to their relative positions in the top CEL view. Any physical objects that are connected to power and ground nets will be connected to their appropriate nets in the top CEL view, and any objects that have properties set in the soft macro will have the same properties set in the top CEL view.

To uncommit the physical hierarchy,

1. Choose Partition > Uncommit Soft Macros.

   The Uncommit Soft Macros dialog box appears.

   Alternatively, you can use the `uncommit_fp_soft_macros` command.
2. Choose whether to convert all soft macros or selected soft macros to plan groups.
3. Set the remaining options, depending on your requirements.
   - Preroutes to push up – Select the "All" option (the default) to push up all routing, including both power and ground, and detail routing. Select the "Power and Ground" option if you want only power and ground straps pushed up to the top CEL view.
   - Remove feedthrough ports on soft macros – Select this option to remove the feedthroughs from selected soft macros. All feedthrough nets and ports with name suffixes of *_pft and *_rft are removed.
4. Click OK or Apply.

When you click the OK or Apply button in the dialog box, the following physical objects are pushed up to the top CEL view from soft macros:

- Standard cells
- Straps on power and ground nets
- Vias on power and ground nets
- Placement blockages, including soft placement blockages and the internal padding around the boundary and cell rows
- Route guides

Keep the following points in mind when you uncommit the hierarchy:

- The uncommit process does not remove the soft macros that were created in the CEL views.
- Any routing done on the soft macros is deleted after you uncommit the hierarchy (except for the power and ground preroutes).

Propagating Unified Power Format Constraints From the Soft Macro to the Top Cell

During the uncommit process when the soft macros are converted back into plan groups, the `uncommit_fp_soft_macros` command automatically propagates the child level Unified Power Format (UPF) constraints and the Milkyway and Design Compiler information from the soft macro to the top level of the design.

During the uncommit hierarchy process, the `uncommit_fp_soft_macros` command causes the following actions when it propagates the UPF constraints to the top level.

- Locates all UPF power domains and their corresponding voltage area geometries and instantiates them in the top cell, according to how they existed in the soft macro. In the case of the top-most UPF power domain in the soft macro, which has a DEFAULT_VA, the voltage area is instantiated in the top cell with the same name as its corresponding power domain and with a geometry that is coincident with the plan group boundary.
- Reconnects the newly instantiated top-cell UPF Milkyway power domains to the existing UPF Milkyway objects.
- Transfers the Design Compiler UPF constraints from the soft macro NID cell attach files to the top-cell level NID plan group attach file.

A

Using the Flip-Chip Flow

This appendix describes how to use the flip-chip flow in IC Compiler. You can use hierarchical or virtual flat implementation flows for designs with flip-chip structures. The IC Compiler flip-chip interface can create both 45-degree and 90-degree redistribution layer routing.

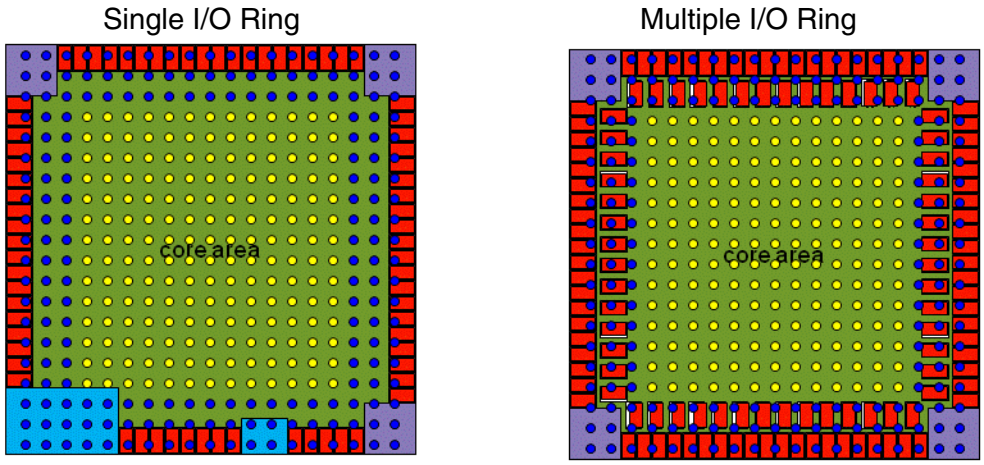
The following two flip-chip design flows are supported in IC Compiler design.

- Package-driven, also referred to as a bump-driven flow, where flip-chip I/O driver locations are optimally determined by package defined bump cell locations
- Die-driven, also referred to as an IC-driven flow, where individual bump cell locations are optimally determined by the I/O driver placement

The IC Compiler flip-chip interface supports two design styles: the single-ring perimeter I/O driver and the multiple ring perimeter I/O driver. In a perimeter ring I/O style, the flip-chip drivers are placed in a ring formation along the periphery of the chip.

Figure A-1 on page A-2 shows a peripheral single-ring I/O style and a multiple ring I/O style.

Figure A-1 Single-Ring and Multi-Ring I/O Styles



In a flip-chip design, a bump is a special cell representing a piece of metal, also called a solder bump, that forms an electrical contact to the chip's packaging. Similar to processing silicon wafers for wire bonding, openings are made on the wafer to expose the metal contact points. Solder bumps are formed on these metal contact points. The chip's power, ground, and signal I/Os are available through the solder bumps that are formed on the top side of the wafer during the final processing step. The bumps on the chip make the physical connections to the package substrate. The chip is connected to external circuitry (a circuit board, another chip, or wafer) by "flipping" it over so that the solder bumps connect to metal pads to complete the interconnect. This is in contrast to wire bonding in which the chip is mounted upright and wires are used to connect the chip pads to external circuitry.

The flip-chip driver is the I/O circuitry that drives or receives signals from or to the chip through the bump. There are signal bump cells, which are connected to signal nets, and power and ground bump cells, which are connected to power and ground straps. The locations of bumps on the chip are determined by the placement of the bump cells. (The I/O driver cells are placed independently.) You can place bump cells on top of the I/O drivers or anywhere within the core area. You can also place standard cells under bump cells

This appendix includes the following sections:

Preparing the Library

IC Compiler recognizes flip-chip I/O drivers and bump cells only when they are marked as flip-chip type. This section describes the library preparation for the flip-chip I/O drivers and bump cells.

I/O Drivers

To prepare the I/O drivers, do the following:

- Set the cell type as “flip-chip driver” by using the `cmMarkCellType` command.

  Note:
  The `cmMarkCellType` command is a Milkyway Environment command.

```
cmMarkCellType
setFormField mark_cell_type library_name library name
setFormField mark_cell_type cell_name cell name
setFormField mark_cell_type cell_type "flip chip driver"
formOK mark_cell_type
```

- Set the PAD port as “`flipchip`” by using the `dbSetCellPortTypes` command.

```
dbSetCellPortTypes
 ("PAD" "inout" "flipchip")
```

- Set the power and ground pins as flip-chip power and ground by using the `dbSetCellPortTypes` command.

```
dbSetCellPortTypes
 ("VDD "inout" "Power")
 ("VSS" "inout" "Ground")
```

The power and ground I/O drivers that connect to power and ground bumps should also have a “flipchip” property defined for the pins.

```
dbSetCellPortTypes
 ("VDD "inout" "Power" "flipchip")
 ("VSS" "inout" "Ground")
```

Bump Cells

To prepare the bump cells, do the following:

- Mark the bump cell as a flip-chip bump by using the `cmMarkCellType` command.

```
cmMarkCellType
setFormField mark_cell_type library_name library name
setFormField mark_cell_type cell_name cell name
setFormField mark_cell_type cell_type "flip chip pad(bump)"
formOK mark_cell_type
```

You need signal bumps and power and ground bumps.

Signal bumps are used to connect signals to I/O drivers.

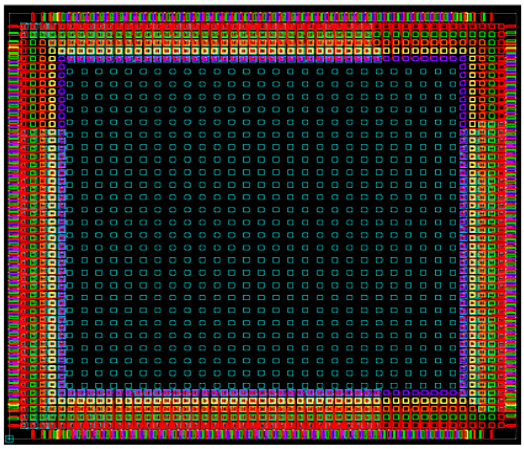
Power and ground bumps are arrayed over the core area with minimum spacing for power and ground delivery by using the `place_flip_chip_array` command.

For example:

```
place_flip_chip_array
 -physical_lib_cell {BUMP100_P}
 -prefix "VDD_CORE_"
 -start_point {1055 1090}
 -number 555
 -delta {270 540}
 -repeat {27 13)


place_flip_chip_array
 -physical_lib_cell {BUMP100_G}
 -prefix "VSS_CORE_"
 -start_point {1055 1360.426}
 -number 555
 -delta {270 540}
 -repeat {27 13)
```

Figure A-2 shows the power and ground bumps arrayed over the core area.

Figure A-2 Power and Ground Bumps Arrayed Over Core Area



- Set the port as “flipchip” by using the `dbSetCellPortTypes` command.
- Create bump terminals (pins) on a redistribution routing layer.

  To create a redistribution layer routing terminal,

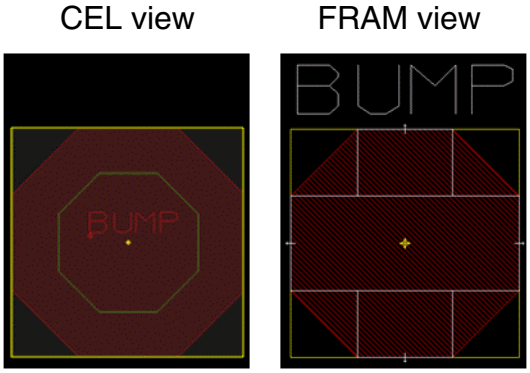
  - Create a .lib file with a terminal name and a corresponding .db file so that IC Compiler can create the bump cells and perform the logical connectivity. Without the .lib file, all bump cells are treated as physical-only cells.
  - Locate the layer for the octagonal-shaped passivation opening in the flip-chip bump cell.
  - Create a polygon in the CEL view by using the `geAddPolygon` command or choose Milkyway: Create > Polygon. Query the polygon to get a list of vertices.
  - Enter the name or number of the layer on which you want to create the polygon, select the L45 routing option, and then add all the vertices.

    You should now have an octagonal piece of metal in the bump CEL view.
  - Add text to the layer by using the `geAddText` command. Use the text layer that matches the newly created terminal.
  - Create a FRAM view by using the `auExtractBlockPinVia` command.

    The CEL and FRAM view should match the terminal name that you created in the .lib file.

  Figure A-3 shows the CEL and FRAM views of a bump cell.

Figure A-3 Bump CEL and FRAM Views



Creating an Initial Die Size

Use the `initialize_floorplan` command to create an initial die size for the core area of your design, based on input control parameters such as target utilization, aspect ratio, core size, row number, chip boundary, and wire tracks.

Using a Die-Driven (IC-Driven) Driver and Bump Placement Flow

In a die-driven flow, the IC design dictates the flip-chip bump cell locations to the IC packages.

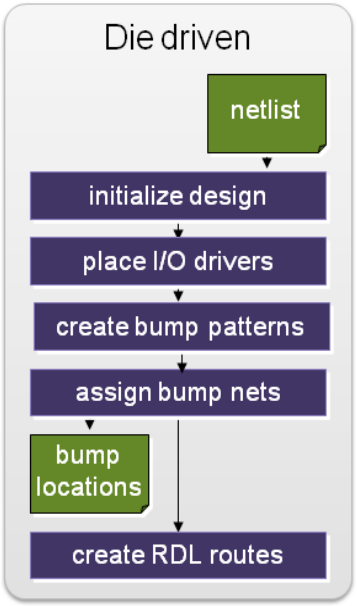
- The bump cells are not instantiated, nor are they connected to flip-chip I/O drivers in the Verilog netlist.
- Individual bump cell locations are optimally determined by the placement locations of the flip-chip I/O drivers and the bump pattern placement.
- The connectivity between the I/O ports and the I/O drivers can be output in the Verilog netlist.

The following topics are covered in this section:

- Placing the Flip-Chip I/O Drivers
- Creating a Pattern of Bump Cells in a Ring Configuration
- Creating a Pattern of Bump Cells in an Array Configuration
- Automatically Assigning Nets From Bumps to I/O Drivers
- Assigning Bump Pattern Personality Types

Figure A-4 shows the die-driven design flow style.

Figure A-4 Die-Driven Design Flow Style



Placing the Flip-Chip I/O Drivers

In most cases, designers are provided with guidelines on where to place the flip-chip I/O drivers. For example, some I/O drivers are placed on the left-side of the chip in a particular order, other I/O drivers are placed along the top of the chip in a particular order, and so forth.

Creating a Pattern of Bump Cells in a Ring Configuration

You can create a specified number of bump cells and place them in your design in a ring configuration by using the `place_flip_chip_ring` command. You can also specify the number of rings to be created and the spacing between adjacent rings.

The `place_flip_chip_ring` command uses the following syntax:

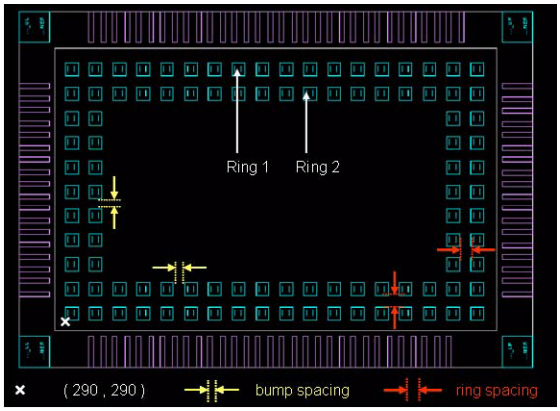
```
place_flip_chip_ring
 -physical_lib_cell cell_name
 -prefix prefix
 -number num_bump
 -bump_spacing spacing
 -ring_number ring_number
 -ring_spacing ring_spacing
 -boundary boundary_box
 [-left_orientation N | W | S | E | FN | FS | FW | FE]
 [-stagger_offset offset
 [-extra_spacing extra_spacing]
 [-num_extra_spacing num_extra]
```

In the following example,100 flip-chip bumps are created and placed in two rings. The spacing between the adjacent bump cells and the adjacent ring cells is 70 microns, and the outermost ring is set to the lower-left coordinates of (290 290) and the upper-right coordinates of (2940 1934).

```
place_flip_chip_ring -physical_lib_cell {PAD80B} -prefix "BUMP_"
-bump_spacing 70 -ring_number 2 -ring_spacing 70 -number 100 -boundary
"290 290 2940 1934"
```

Figure A-5 shows the bump cells placed in a ring configuration.

Figure A-5 Placing Bump Cells in a Ring Configuration



Creating a Pattern of Bump Cells in an Array Configuration

You can create a specified number of flip-chip bump cells and place them in your design in a two-dimensional array pattern by using the `place_flip_chip_array` command. You specify the size of the array starting at a specified coordinate and the distance between adjacent bumps in the array pattern.

The `place_flip_chip_array` command uses the following syntax:

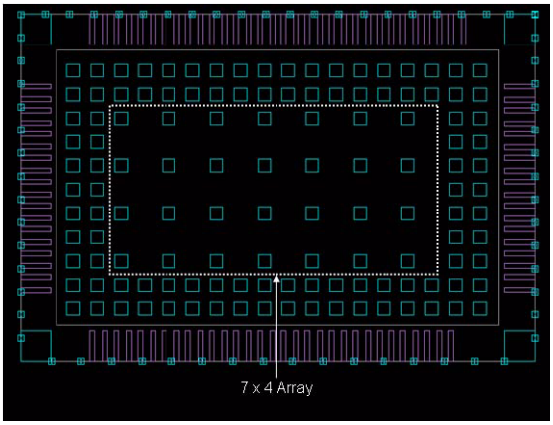
```
place_flip_chip_array
 -physical_lib_cell phys_lib_cels
 -prefix prefix
 -start_point start_point
 -number num_bump
 -delta {x y}
 -repeat {i j}
[-orientation N | W | S | E | FN | FE | FS | FW]
[-cell_origin lower_left | center]
```

In the following example, 28 flip-chip VDD_CORE_* bumps are created and placed in a 7 by 4 array pattern, starting at the lower-left x-coordinate of 590 and the lower-left y-coordinate of 590. There is an X-pitch and a Y-pitch of 300 microns between adjacent bump cells.

```
place_flip_chip_array -physical_lib_cell {PAD80B_P} -prefix "VDD_CORE_"
-start_point {590 590} -number 28 -delta {300 300} -repeat {7 4}
```

Figure A-6 on page A-10 shows the bump cells placed in a 7 by 4 array pattern.

Figure A-6 Placing Bump Cells in a 7 by 4 Array Pattern

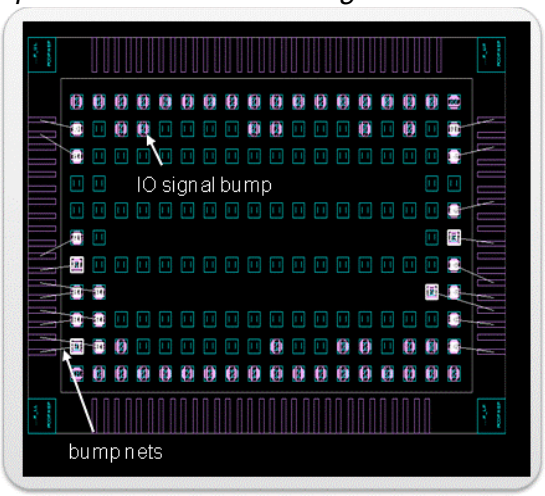


Automatically Assigning Nets From Bumps to I/O Drivers

Based on the current driver placement results, the `assign_flip_chip_nets` command locates the nearest unassigned flip-chip bumps, matches them to flip-chip I/O drivers using the shortest wire length method and then automatically assigns new logical nets or reconnect the nets between the flip-chip bumps and I/O drivers, as shown in Figure A-7. The I/O drivers are placed considering the connectivity to the core logic. The bump cell instances are connected to nearby I/O drivers automatically.

When you run the `assign_flip_chip_nets` command, it also optimizes the bump-to-driver interconnect wire length.

Figure A-7 Automatic Bump Net and I/O Driver Assignment



The `assign_flip_chip_nets` command uses the following syntax:

```
assign_flip_chip_nets
 [-personality_type personality_type_list]
 [-prefix prefix]
 [-uniquify num_to_uniquify]
 [-multiple_terminal_pins pin_name_or_collection]
 [-terminal_layers layer_name_or_collection]
 [-terminal_names terminal_name_list]
 [-eco]
```

Note:

The driver-to-bump matching is processed by personality types that were defined by the `set_flip_chip_type` command. Only drivers and bumps with identical personality types are matched.

For bumps with an existing logical connection, unless the `-eco` option is specified, the previous connection to the bump is removed, and a new bump net is assigned. The newly matched bump is reconnected to the matching flip-chip driver. The new bump net inherits the net name from the flip-chip driver pad to which the bump is now connected.

You can use the `-uniquify` *`num_to_uniquify`* option to specify whether to and how to uniquify the driver nets. This option takes effect only when there are multiple power drivers whose flip-chip ports are connected to the same net. The *`num_to_uniquify`* argument accepts integers from `-n` to `0` to `n`. The default is 1-to-1uniquify.

- `0` – Do not uniquify. Assign only one bump for the multiple flip-chip driver ports. The original driver net can be reused.
- `1` – 1-to-1 uniquify. Assign one bump for each flip-chip driver port. New nets are created with the original net name followed by "_#". A VDD net, for example can be uniquified to VDD, VDD_1, VDD_2, and so forth.
- `n` – n-to-1 uniquify. Assign one bump for every `n` flip-chip driver port.
- -n – 1-to-n connection. Assign `n` bumps for each flip-chip driver port.

You can specify the flip-chip driver pins that can have multiple terminals by using the `-multiple_terminal_pins` option. When you specify this option, one bump is assigned to each terminal of the pins you specified. Use the `get_pins` command to specify the pins or enter the pin names in a Tcl list.

You can specify the layers on which the multiple terminals are assigned a bump by using the `-terminal_layers` option. Use the `get_layers` command to specify the layers or enter the names of the layers in a Tcl list.

You can specify the names of terminals of flip-chip driver pins to be assigned by using the `-terminal_names` option. This option cannot be used with the `-multiple_terminal_pins` or `-terminal_layers` option. If you use this option, each terminal name that you specify is counted as an individual flip-chip driver pin.

When the `-eco` option is specified, existing logical connections are kept, and a new logical net is created only if the flip-chip driver pad has no existing net connection.

Note:

The `assign_flip_chip_nets` command can handle multiple PAD pins per I/O driver, with multiple bump cells connected to one single I/O driver with multiple pad pins.

You can also place I/O signal nets, I/O power and ground nets, and core power nets, as shown in Figure A-8 on page A-12.

- The I/O signal drivers are placed considering the connectivity to the core logic.
- The I/O power and ground drivers are placed based on guidelines from the I/O provider.
- The signal bumps are placed close to the I/O drivers.
- The core power bumps are arrayed over the core area.

Figure A-8 Placing I/O Signal Nets and I/O Power and Ground Nets

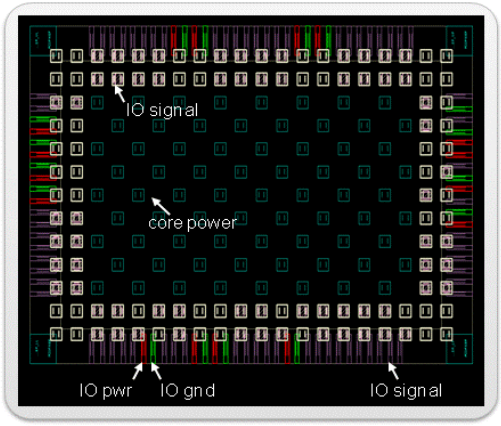
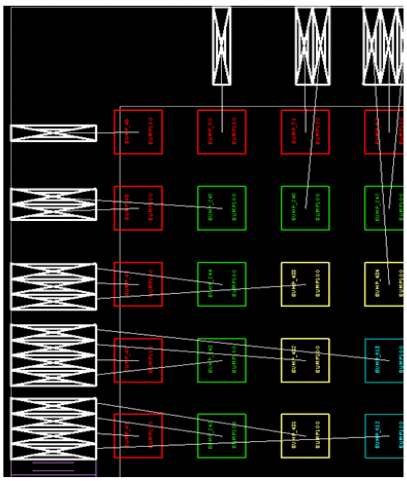


Figure A-9 shows an example of the connectivity flylines between bump nets and I/O drivers.

Figure A-9 Assigning Bump Nets: Connectivity Between Bump Nets and I/O Drivers



Note:

After a net is assigned between the flip-chip bump and I/O driver, the bump pin is visible in the GUI.

Merging Multiple Flip-Chip Nets Into One Net

You can merge the flip-chip nets that were uniquified by the `assign_flip_chip_nets` command into one net by using the `merge_flip_chip_nets` command.

The `merge_flip_chip_nets` command uses the following syntax:

```
merge_flip_chip_nets
 -from from_nets
 -to to_net
 [-update_routing]
```

The command disconnects all the components, including pins, I/O ports, and terminals, from a specified collection of flat nets and reconnects them to the newly merged net.

Note:

You should use the `-update_routing` option if these nets were partially routed by using the `route_flip_chip` command. This updates the owner attribute of the net shapes and vias of the specified nets in the collection.

Assigning Bump Pattern Personality Types

Before you can implement your flip-chip design, you can assign the flip-chip drivers and bump cells with the correct bump pattern “personality” type. This allows you to guide the net assignment process.

You use the `set_flip_chip_type` command to assign bump pattern personality types to the flip-chip bump cells, I/O drivers, and specified nets or cells. A personality type is a string that is associated with a flip-chip driver or a flip-chip bump cell. You can run this command multiple times; a new personality type setting overrides an existing personality type setting.

The `set_flip_chip_type` command uses the following syntax:

```
set_flip_chip_type -personality_type type [-pin] net_or_cell_list
```

The `-pin` option sets the personality type on pins. By default, personality is set on cells.

The following command, for example, selects all the signal bumps and I/O drivers and assigns the personality type “signal.”

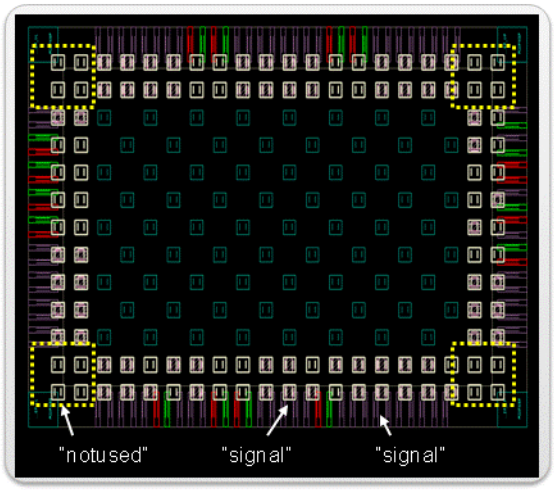
```
set_flip_chip_type -personality_type "signal" [get_cells "DRIVER_*"]
```

Another example is the following command, which selects the 4 x 4 bumps at each corner and assign the personality type “notused.”

```
set_flip_chip_type -personality_type "notused" [get_cells "BUMP_*"]
```

Figure A-10 shows a selection of “signal” and “unused” personality types.

Figure A-10 Using Personality Types to Guide Bump Net Assignment

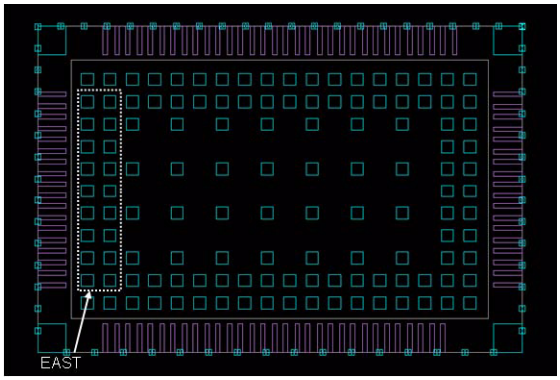


The following command sets the flip-chip bumps to the “east” personality type.

```
set_flip_chip_type -personality_type east [get selection]
```

Figure A-11 shows a selection of bump cells set to the “east” personality type.

Figure A-11 Selection of Bump Cells for Personality Type Assignment



Setting Personality for Flip-Chip Driver Pins

You can assign personality to driver pins by using the `-pin` option of the `set_flip_chip_type` command, and report the personality type of specified pins by using the `report_flip_chip_type` command. Setting a personality type on a driver pin groups the driver with an associated bump cell for better driver placement. The `set_flip_chip_type` command with the `-pin` option uses the following syntax:

```
set_flip_chip_type -pin -personality_type "signal" [get_selection]
```

You can clear the personality type assigned to a pin by using the `set_flip_chip_type -pin` command, with an empty string as the personality type. The `set_flip_chip_type` command to remove the personality setting uses the following syntax:

```
set_flip_chip_type -pin -personality_type "" [get_selection]
```

The `set_flip_chip_type -pin` command will set or remove the personality type depending on the type of object selected as shown in Table A-1.

Table A-1 Pin Personality Setting Based on Object Type

Selection type	Action
net	The personality type of the flip chip pin connected to the net is set or removed
cell	The personality type of the flip chip pin connected to the cell is set or removed
pin	The personality type of the flip chip pin is set or removed. A warning is issued if the pin is not a flip chip pin

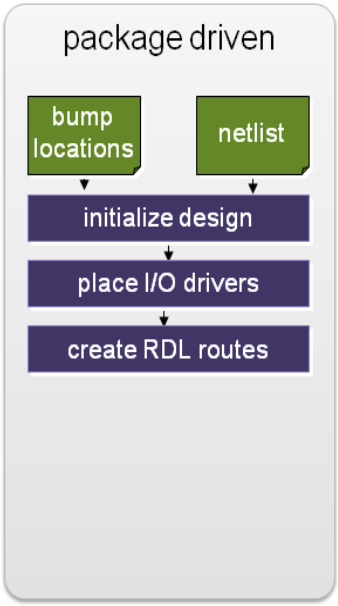
Using a Package-Driven (Bump-Driven) Bump and Driver Placement Flow

In a package-driven flow, the bump cells are imported and placed at physical locations predefined by the flip-chip package designer.

- The bump cells locations are defined in Design Exchange Format (DEF) or Advanced Input Format files.
- The bump cells instances, I/O driver instances, and the net connectivity between the bump cells and I/O drivers are defined in the Verilog netlist as input or output.
- The I/O drivers are placed as close as possible to their associated bump cells.
- After the bump cells and I/O drivers are placed and the connectivity is defined, the redistribution layer routes are formed to connect the bumps to the I/O drivers.

Figure A-12 shows the package-driven design flow style.

Figure A-12 Package-Driven Design Flow Style



The following topics are described in this section:

- Importing and Placing Bump Cells
- Saving the Flip-Chip Bump Cells
- Creating a Secondary Grid for Flip-Chip Driver Placement
- Defining Legal Locations for Placing Flip-Chip Drivers

- Setting Options for Flip-Chip Driver Placement
- Placing Flip-Chip I/O Drivers Concurrently With Standard Cells and Hard Macros
- Reporting Flip-Chip Driver to Bump Results

Importing and Placing Bump Cells

In a package-driven flow, bump cells are imported and placed in predefined physical locations as dictated by the flip-chip package house. The `read_flip_chip_bumps` command reads the bump locations from a text file in the Advanced Input Format file.

You can use the `-physical_lib_cell` option to specify a collection of physical library cells. The collection must contain one cell whose location serves as the reference point from which the flip-chip bump cells are placed in the die area. The default orientation of the bump cells is N (north).

The `read_flip_chip_bumps` command uses the following syntax:

```
read_flip_chip_bumps
-physical_lib_cell phys_lib_cels
[-orientation N | W | S | E | FN | FE | FS | FW]
file_name
```

IC Compiler only reads the bump locations from the [NETLIST] section of the AIF file.

Here is an example:

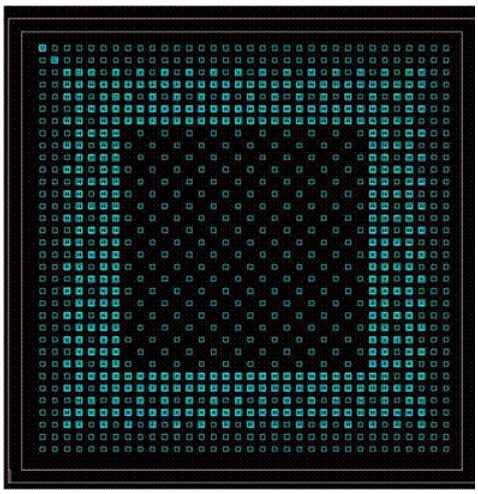
```
read_flip_chip_bumps -physical_lib_cell [get_physical_lib_cells "BUMP"]
BumpPattern.aif
```

Here is an example AIF file:

```
[NETLIST]
;NETNAMEDIE_PAD# DIE_PADSTACK_NAME DIE_PAD_X Die_PAD_Y
GND BUMP_103 BUMP 3795.0 -3105.0
VD33 BUMP_104 BUMP 3565.0 -3105.0
TE1_ON_4 BUMP_106 BUMP 3105.0 -3105.0
TEX_ON_4 BUMP_107 BUMP 2875.0 -3105.0
DATA_4[19] BUMP_116 BUMP 805.0 -3105.0
DATA_4[24] BUMP_117 BUMP 575.0 -3105.0
```

Figure A-13 on page A-18 shows the placement locations of the imported bump cells.

Figure A-13 Imported Bump Cells



Saving the Flip-Chip Bump Cells

You can write bump cell information, including the locations and net connections, to a text file in the Advanced Input format (AIF), by using the `write_flip_chip_bumps` command. You must specify the name of the output file.

The `write_flip_chip_bumps` command uses the following syntax:

```
write_flip_chip_bumps
  file_name
```

Creating a Secondary Grid for Flip-Chip Driver Placement

IC compiler supports the automatic placement of flip-chip drivers. To enable automatic placement, you must first create a secondary placement grid. Legal locations of the flip-chip drivers can be restricted to a secondary placement grid.

Typically, the legal locations of the flip-chip drivers are restricted to a secondary placement grid, which is different from the standard placement grid used for standard cell rows and columns.

You use the `set_flip_chip_grid` command to create a secondary placement grid with equally spaced grid points on which to place the flip-chip drivers. This secondary grid is the minimum matrix pitch defined for the actual flip-chip driver placement.

The `set_flip_chip_grid` command uses the following syntax:

```
set_flip_chip_grid
 -grid_origin {llx lly}
 -x_step x
 -y_step y
```

A flip-chip driver must be placed on the secondary placement grid, and all flip-chip driver legal location sites must also reside on this secondary grid.

In the following example, a secondary flip-chip placement grid is created starting at the grid origin of (0, 0) with spacing between the grid of 10 microns in both the X-direction and the Y-direction.

```
set_flip_chip_grid -grid_origin {0 0} -x_step 10 -y_step 10
```

Defining Legal Locations for Placing Flip-Chip Drivers

You can define the flip-chip cell sites that form the legal locations for placing the flip-chip drivers in an island style and you can define the constraints on the flip-chip driver placement by using the `set_flip_chip_driver_island` command. An island is an array of flip-chip driver locations that are abutted together. You can define an array of islands using a single `set_flip_chip_driver_island` command.

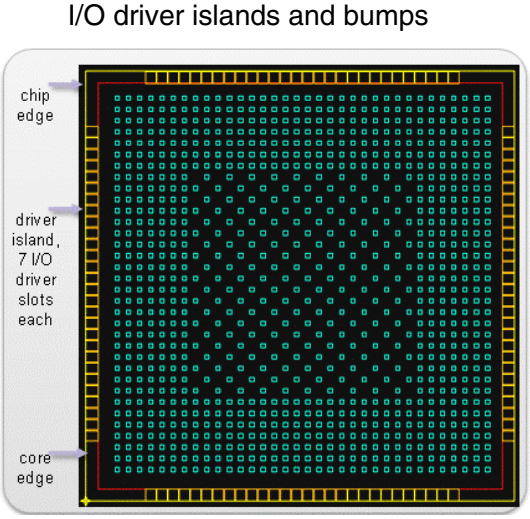
An island can contain a two-dimensional array of flip-chip driver slots. Each slot can hold a unit sized flip-chip driver. Slots within I/O driver islands can be constrained to allow the placement of specific types of drivers.

The `set_flip_chip_driver_island` command uses the following syntax:

```
set_flip_chip_driver_island
 -start_point start_point
 -repeat num_columns num_rows
 -spacing x_spacing y_spacing
 -island_size width height
 -num_driver num_x num_y
 [-filler cell_ref]
 [-personality_type_constraints personality_type max_num ...]
 [-default_orientation N | W | S | E | FN | FE | FS | FW]
 [-orient_by_row]
 [-compaction vertical | horizontal | none]
 [-center_packing]
 [-forced_orientation orientation col_index ...]
 [-reserved_for_cell cell_ref col_index row_index ...]
```

Figure A-14 shows the I/O driver islands and bumps.

Figure A-14 Creating I/O Driver Islands



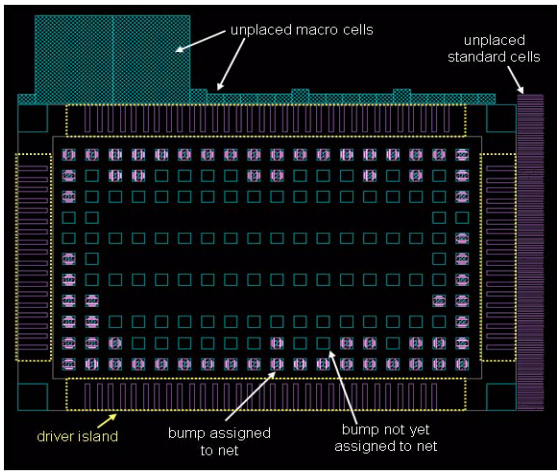
The following example shows a peripheral I/O style where 4 driver islands are needed: one for the left side, one for the right side, one for the top, and one for the bottom. This example creates a flip-chip driver island for the left side.

```
set_flip_chip_driver_island -start_point {0 465} -repeat {1 1} \
  -spacing {0 0} -island_size {190 1320} -num_driver {1 18} \
  -personality_type_constraints {{east 10} {VDD 2} {VSS 2}} \
  -default_orientation E
```

The flip-chip driver island starts at the coordinates (0 465). 18 flip-chip drivers are assigned to the island using different personality types: 10 of east personality, 2 of VDD and 2 of VSS. All flip-chip drivers instances will be placed in an E (east) orientation.

Figure A-15 shows a design prior to placement. At this stage in the design flow, the bump patterns exist, nets are assigned between the flip-chip bumps and the I/O drivers, and the driver islands are created.

Figure A-15 Design Prior to Placement



Setting Options for Flip-Chip Driver Placement

Prior to placing the flip-chip drivers and standard cells, you can set various flip-chip driver placement options by using the `set_flip_chip_options` command. You can enable a special optimization algorithm for placing flip-chip drivers based on fixed-bump location constraints by specifying the `-package_driven` option. When multiple width or height flip-chip drivers are placed, you can specify the `-multiple` option to allow the driver to consume more than one slot.

The `set_flip_chip_options` command uses the following syntax:

```
set_flip_chip_options
 [-disable_driver_placement]
 [-package_driven]
 [-one_softmacro_per_island]
 [-flip_chip_net_weight weight]
 [-softmacro_net_weight sm_weight]
 [-guardband_width {x y}]
 [-multiple width | height | both]
```

The following example sets a net weight value of 15 on the nets that connect the I/O drivers and the bump cells.

```
set_flip_chip_options -flip_chip_net_weight 15 -package_driven
```

Placing Flip-Chip I/O Drivers Concurrently With Standard Cells and Hard Macros

You can place the flip-chip I/O drivers concurrently with hard macros and standard cells in a virtual flat placement mode by using the `create_fp_placement` command. When you run this command, the virtual flat placement engine calls the driver placement under the flip-chip mode, places the flip-chip I/O drivers into defined legal sites, and snaps them to a secondary grid. To set the flip-chip mode, enter

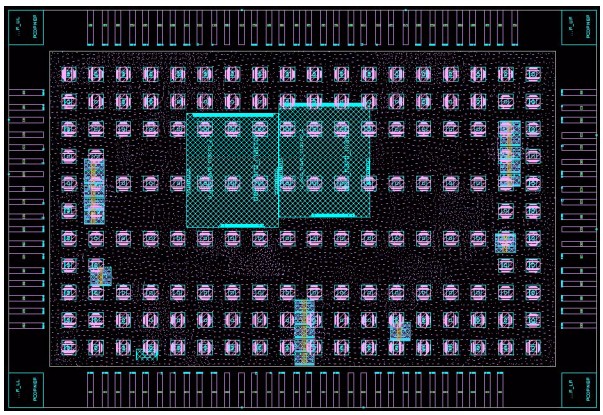
```
set_fp_placement_strategy -flip_chip off | on | honor_reserved
```

The `off` keyword specifies that the command does not do flip-chip driver placement. The `on` keyword directs the command to perform flip-chip driver placement. The `honor_reserved` keyword places the flip-chip drivers and also performs slot insertion and compression.

The objective of flip-chip driver placement is to enhance single-layer routability, while optimizing total wire length. You can further control the placement of flip-chip I/O driver cells by using the `set_flip_chip_driver_island -reserved_for_cell` command. This command reserves a specific flip-chip driver row and column location for a specified driver type.

Figure A-16 shows the simultaneous placement results of the flip-chip drivers, hard macros, and standard cells.

Figure A-16 Simultaneous Placement of Flip-Chip Drivers, Hard Macros, and Standard Cells



Reporting Flip-Chip Driver to Bump Results

You can issue a report in a table format of the flip-chip driver to bump matching results by using the `report_flip_chip_driver_bump` command.

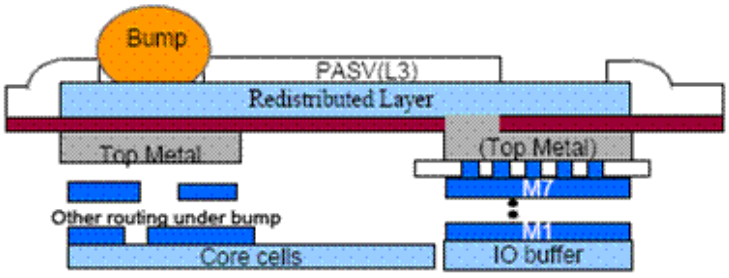
Each line in the table represents a flip-chip net connection. The first column specifies the bump cell; the second column lists the name of the net that connects the bump cell and the flip-chip I/O driver; and the remaining columns specify the driver pins.

Performing Automatic Bump Net Routing

Before you can perform regular signal detail routing, the bump nets must routed. Bump net routing, also referred to as redistribution layer routing, redistributes the wire-bonding pads to the bump pads without changing the placement of the I/O pads. Typically, bump nets are routed on the top metal layer of the die.

Figure A-17 shows a cross-section of redistribution layer routing.

Figure A-17 Cross-Section showing Redistribution Layer Routing



Note:

For peripheral I/O flip-chip design styles, single-layer routing on the top metal layer is preferred. In addition, single-layer routing on the top metal layer minus one level (dual-layer routing), is also allowed when necessary.

The following topics are described in this section.

- Using the Flip-Chip Routing Commands
- Routing the Flip-Chip Nets

Using the Flip-Chip Routing Commands

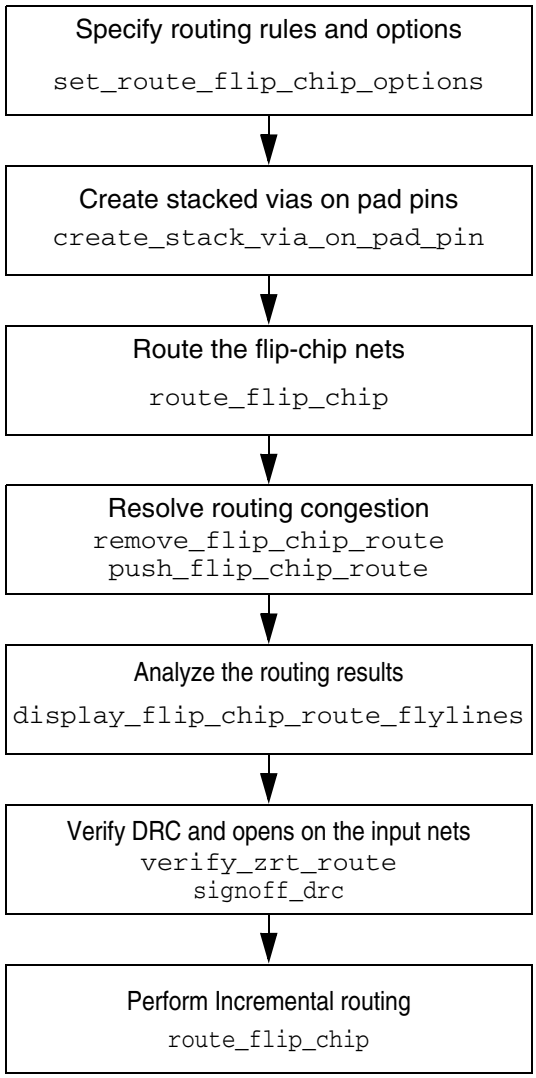
This section describes the usage of the following flip-chip routing commands.

- `set_route_flip_chip_options`
- `create_stack_via_on_pad_pin`
- `set_flip_chip_bump_attributes`

- `route_flip_chip`
- `remove_flip_chip_route`
- `push_flip_chip_route`
- `display_flip_chip_route_flylines`
- `verify_zrt_route`
- `signoff_drc`

You can use flip-chip routing commands to complete the flip-chip routing flow, shown in Figure A-18.

Figure A-18 Flip-Chip Routing Flow



Specifying Routing Rules and Options

Use the `set_route_flip_chip_options` command to set options for flip-chip routing. The options are stored in the design database for use by subsequent flip-chip routing steps.

The `set_route_flip_chip_options` command uses the following syntax:

```
set_route_flip_chip_options
 [-number_of_loops 1 | 2 | 3]
 [-route_with_soft_macros true | false]
 [-search_effort easy | normal | hard]
 [-layer_spacing {layer_spacing_pairs}]
 [-layer_width {layer_width_pairs}]
```

```
 [-rule_name name]
 [-nets {collection_of_nets} | -nets_in_file nets_file]
 [-output_unrouted_nets open_net_file]
 [-design_style area_IO | peripheral_without_corner_driver |
    peripheral_with_corner_driver | tsv]
 [-min_access_edge_length length]
```

Creating Stacked Vias on Pad Pins

Use the `create_stack_via_on_pad_pin` command to create stacked vias on the flip-chip I/O drivers from the pad pin to the top metal layer for flip-chip routing. Stacked vias must be created before you route the flip-chip nets. For more information, see "Adding stacked vias to driver pins to enable single layer routing" on page A-32.

The `create_stack_via_on_pad_pin` command uses the following syntax:

```
create_stack_via_on_pad_pin
 -from_metal layer  -to_metal layer  [-remove_existing_stack_via true |
false] [-route_type user_enter | signal_route] [-nets collection_of_nets
| -nets_in_file file_name] [-all_driver_pins true | false]
[-terminal_names {list_of_terminal_names}]
```

Controlling the Bump Access Direction During Flip-Chip Routing

Use the `set_flip_chip_bump_attributes` command to set attributes on flip-chip bump cells for flip-chip routing. The flip-chip router honors these attributes. You can specify the names of the bump cells on which to set the attributes and a collection of target edges of the flip-chip bump cells. The flip-chip router connects wires to the bump target edges you specify.

The `set_flip_chip_bump_attributes` command uses the following syntax:

```
set_flip_chip_bump_attributes
 -bump_cells collection_of_bump_cells
 -target_edges target_edges
```

Performing Redistribution Layer Routing

Use the `route_flip_chip` command to perform flip-chip redistribution layer routing on the flip-chip nets. For more information, See "Routing the Flip-Chip Nets" on page A-31.

The `route_flip_chip` command uses the following syntax:

```
route_flip_chip
  -routing_layer layer
 [-nets collection_of_nets | -nets_in_file nets_file]
 [-route_by_input_net_order]
 [-45_degree]
 [-terminal_names list_of_terminal_names]
```

During 45-degree redistribution layer routing, a turn too close to a bump cell or a pad pin might result in an acute angle error after net splitting occurs. You can control the wire length entering the bump cell or leaving the pad pin to reduce the chance of net splitting not being carried out by using the `set_route_flip_chip_options -min_access_edge_length` command.

In some flip-chip designs, I/O pads or flip-chip drivers can have electrically equivalent pins that must have separate routes to the same or different bump cells. Electrically equivalent pins are supported in redistribution layer routing in the flip-chip flow. This allows you to:

- Create separate routes to the same bump cell, also referred to as multiple-terminal net connections, or to different bump cells for specified flip-chip drivers or I/O pads with electrically equivalent pins
- Choose a specific electrically equivalent pin for routing
- Perform routing on multiple PG electrically equivalent pins to reduce IR drop

You can use the `-terminal_names` option to selectively route from the specified driver pins to the connected bump cells. The logic connections between the electrically equivalent pins and the bump cells must first be established by the `assign_flip_chip_nets` command.

Each electrically equivalent pin requires it own stacked via. Use the `create_stack_via_on_pad_pin -terminal_names` *`list`* command to create stacked vias on the flip-chip I/O drivers from the pad pin to the top metal layer for flip-chip routing.

Use the following syntax to specify the electrically equivalent pins. The first line defines a multi-terminal net, the second line defines an electrically equivalent pin.

```
{instance_name/pin_name}
{instance_name/port_name:pin_name}
```

Resolving the Routing Congestion

Use the `remove_flip_chip_route` command to resolve the routing congestion by removing specified flip-chip wires, paths, and contacts.

You can resolve routing congestion from routing by

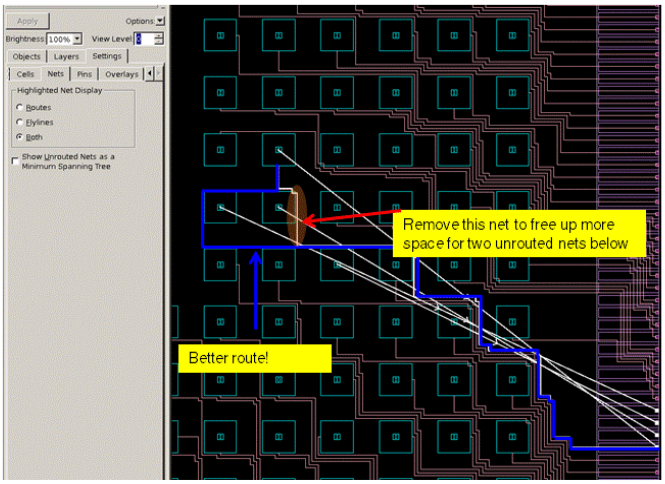
- Removing the nets that create the bottleneck net
- Pushing nets away from the congested area to create more routing resources

You can resolve routing congestion from placement by

- Rearranging the placement of the I/O pads or the bump cells or both
- Redoing the net assignment

Figure A-19 shows an example of removing a bottleneck net.

Figure A-19 Removing Nets that Create the Bottleneck



The `remove_flip_chip_route` command uses the following syntax:

```
remove_flip_chip_route [-nets {collection_of_nets} | -nets_in_file
net_file] [-width width] [-contact]
```

Use the `push_flip_chip_route` command to push the routed flip-chip nets that block the path of unrouted nets in a specified direction or away from the specified nets.

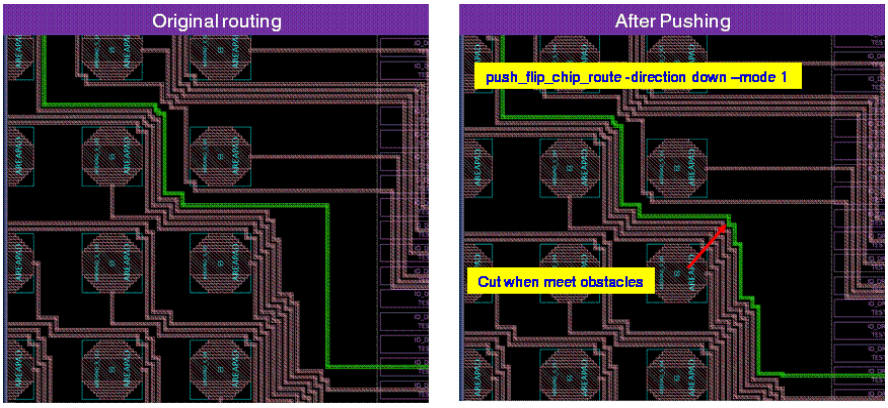
The `push_flip_chip_route` command uses the following syntax:

```
push_flip_chip_route -nets {collection_of_nets} | -nets_in_file net_file}
[-layer tech_layer_name] [-direction up | down | left | right] [-mode 1 |
2 | 3]  [-sweep_range sweep_range]
```

Both 90-degree and 45-degree routes can be pushed with the `push_flip_chip_route` command. Three modes of pushing the routed flip-chip nets are supported:

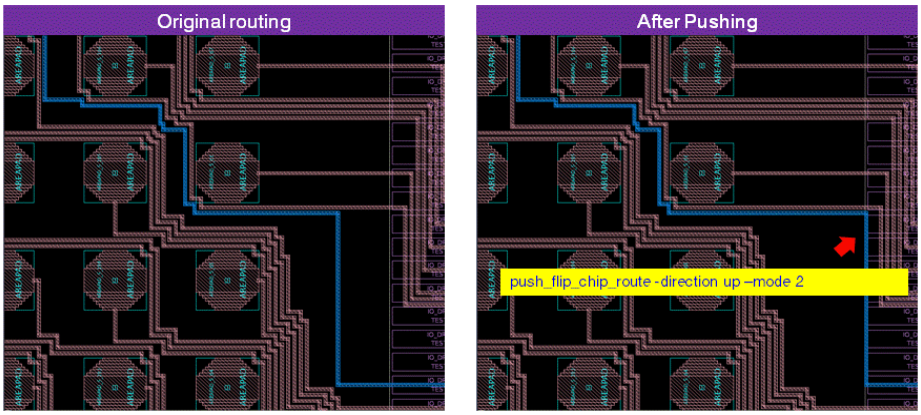
- Mode 1 – Pushes the routed nets toward a specified direction (up, down, left, or right) to free up more space for the unrouted nets. This is the default. Figure A-20 on page A-29 shows an example of push mode 1.

Figure A-20 Push Mode 1



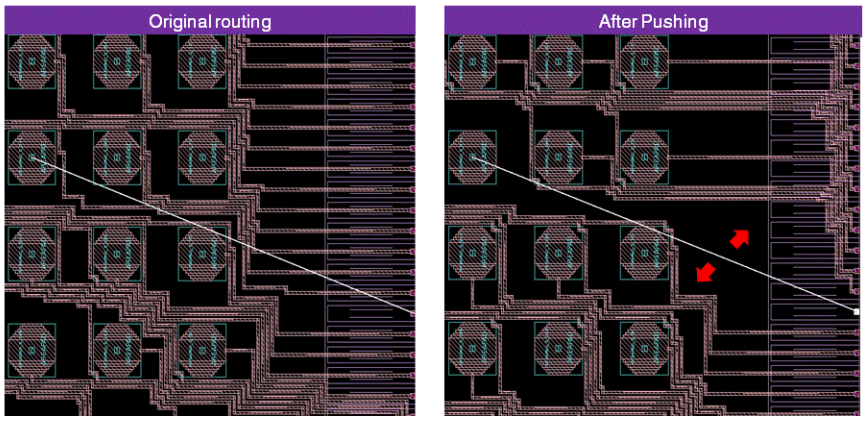
- Mode 2 – Pushes the target net along the edge of the flip-chip I/O drivers. Figure A-21 shows an example of push mode 2.

Figure A-21 Push Mode 2



- Mode 3 – Pushes neighboring wires away from the target net. Figure A-22 on page A-30 shows an example of push mode 3.

Figure A-22 Push Mode 3



Optimizing the Routing Pattern

Use the `optimize_flip_chip_route` command to improve and enhance the flip-chip routing patterns by running L-shape and Z-shape optimizations on the flip-chip nets. The command routes the flip-chip nets on a single metal layer.

The `optimize_flip_chip_route` command uses the following syntax:

```
optimize_flip_chip_route -layer tech_layer_name [-nets collection_of_nets
| -nets_in_file nets_file] [-change_route_type user_enter |
signal_route] [-split_net]
```

Analyzing the Routing Results

Use the `display_flip_chip_route_flylines` command to display the flylines of the specified flip-chip nets in the layout window of the IC Compiler GUI. You can view unrouted nets with flylines and identify any nets that are creating bottlenecks.

The `display_flip_chip_route_flylines` command uses the following syntax:

```
display_flip_chip_route_flylines [-nets collection_of_nets |
-nets_in_file nets_file] [-open_nets [-output_open_nets open_net_file]]
```

Reporting Crossover Nets

IC Compiler might create crossover nets while generating flip-chip nets. To report crossover nets, use the `report_flip_chip_flyline_cross` command. The `report_flip_chip_flyline_cross` command checks crossover net assignment and determines if the flyline between a driver cell and a bump cell crosses any other flylines between other driver cells and bump cells. If one net consists of multiple drivers and bump cells, each net segment is tested against other driver cell and bump cell flylines.

The report generated by the `report_flip_chip_flyline_cross` command contains a list of net pairs that cross. The following example shows a sample report.

```
icc_shell> report_flip_chip_flyline_cross
*****************************
Report : flyline cross
Design : chip1
Version: D-2010.03-ICC-SP1
*****************************

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

The following two nets cross each other!
net1             net2

REG_ADDR_1[5]    REG_ADDR_1[4]
TE1_ON_3         TE2_ON_3
DATA_2[28]       REG_DATA_2[29]
REG_DATA_3[30]   IDLE_3
REG_ADDR_3[1]    REG_ADDR_3[2]
REG_DATA_2[9]    REG_DATA_2[8]
R_TMU0_ON_2      R_TMU1_ON_2
PE_CLK_4         RESET_4
FE_CLK_4         BUSY_4

Total 9 crossing!

------------------------------------------------------------------
```

Verifying Design Rule Checking and Open Nets

Use the `verify_zrt_route` command to check and report design rule checking (DRC) violations and open nets on the input bump nets.

The `verify_zrt_route` command uses the following syntax:

```
verify_zrt_route [-nets collection_of_nets] [-open_net true | false]
[-report_all_open_nets true | false [-drc true | false] [-antenna true |
false] [-voltage_area true | false] [-check_from_user_shapes true |
false]
```

Use the `signoff_drc` command to detect 65 nm and below process design rule-checking violations with foundry runset.

Routing the Flip-Chip Nets

The flip-chip nets are routed by using the `route_flip_chip` command. Before running this command, however, it might be necessary to perform the following setup procedures.

- Specifying the names of the bump nets for routing

Use the `write_flip_chip_nets` command to write all the bump-to-driver net names into a text file. You can edit this file to specify a subset of the bump nets to be routed.

For example,

```
write_flip_chip_nets -file_name bump_net_list
```

- Defining special routing rules

  Use the `set_route_flip_chip_options` command to set various flip-chip routing options, and use the `define_routing_rule` command to save them as nondefault routing rules.

  For example,

  ```
  set_route_flip_chip_options -rule_name test -layer_spacing {m8 2.0} \
  -layer_width {m8 12}
  ```

- Adding stacked vias to driver pins to enable single layer routing

  Use the `create_stack_via_on_pad_pin` command to create stacked vias on the flip-chip I/O drivers from the pad pin to the top metal layer for flip-chip routing. The flip-chip router will then route from the stacked via on the redistribution layer to the bumps on the same layer.

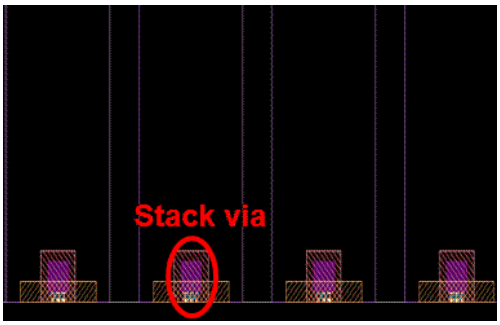
  The stacked vias are created based on the input nets specified in the `set_route_flip_chip_options` command. The size of the stacked vias is determined by the predefined nondefault routing rule and the shape of the pad pin.

  For example,

  ```
  create_stack_via_on_pad_pin -from_metal 36 -to_metal 38
  ```

  A stacked via pad is shown in Figure A-23.

Figure A-23 Stacked Via Pad



- Perform flip-chip redistribution layer routing on the flip-chip nets

  Use the `route_flip_chip` command to perform redistribution layer routing on the flip-chip nets. The command routes the flip-chip nets on a single metal layer.

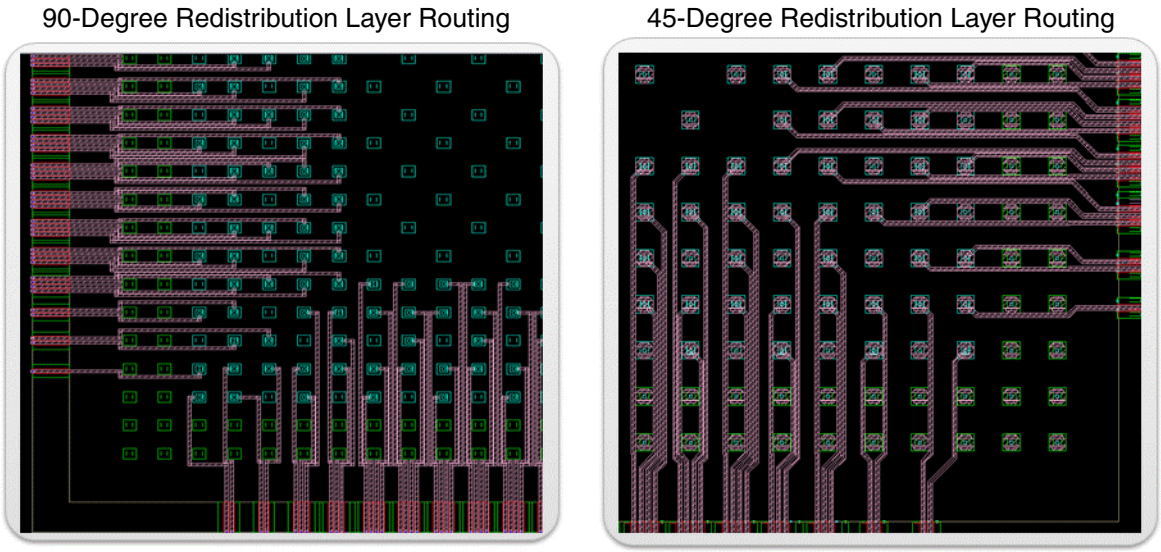
IC Compiler performs 45-degree routing and 90-degree Manhattan routing for flip-chip nets.

The following example shows how to set 45-degree redistribution layer routing.

```
route_flip_chip -nets_in_file bumpnet \
 -routing_layer top_layer \
 -45_degree
```

Figure A-24 shows an example of 45-degree and 90-degree redistribution layer routing.

Figure A-24 45-Degree and 90-Degree Redistribution Layer Routing



Using Flip-Chip Structures in Cover Macros

IC Compiler supports flip-chip bump and cover cells in the logical netlist. You can describe the logical connectivity between the flip-chip bump pads and cover cells and the I/O drivers in the Verilog netlist.

You can change the flip-chip bump and cover cells from physical-only cells to become part of the logical netlist by setting the following variable to `true`.

```
set_app_var disable_bump_cover_as_physical_only true
```

Setting this variable to a value of `false` will treat bump and cover cells as physical-only.

Summary of the Flip-Chip Commands

Table A-2 provides a summary of the flip-chip commands. For more information, see the appropriate man pages.

Table A-2 Summary of Flip-Chip Commands

Command	Description
`assign_flip_chip_nets`	Creates or reconnects nets between flip-chip I/O drivers to bumps.
`create_stack_via_on_pad_pin`	Creates stacked vias for the I/O drivers from the pad layer to the top metal layer for flip-chip routing.
`display_flip_chip_route_flylines`	Displays the flylines of the specified flip-chip nets in the layout window of the IC Compiler GUI. You can view unrouted nets with flylines and identify any nets that are creating bottlenecks.
`expand_flip_chip_cell_locations`	Proportionally expands or shrinks the relative locations of the selected flip-chip bump or driver cells.
`merge_flip_chip_nets`	Disconnects all the components, including pins, I/O ports, and terminals, from a specified collection of flat nets and reconnects them to the newly merged net.
`optimize_flip_chip_route`	Improves and enhances the flip-chip routing patterns by running L-shape and Z-shape optimizations on the flip-chip nets. The command routes the flip-chip nets on a single metal layer.
`place_flip_chip_array`	Creates the specified number of flip-chip bump cells and places them in a two-dimensional array pattern.
`place_flip_chip_ring`	Creates the specified number of flip-chip bump cells and places them in a ring configuration.
`push_flip_chip_route`	Pushes the routed flip-chip nets that block the path of unrouted nets in a specified direction or away from the specified nets.

Table A-2 Summary of Flip-Chip Commands (Continued)

Command	Description
`read_flip_chip_bumps`	Reads the flip-chip bump locations from an Advanced Input Format file.
`remove_flip_chip_route`	Helps resolve the routing congestion by removing specified flip-chip wires, paths, and contacts.
`report_flip_chip_driver_bump`	Prints a table of bump cells and corresponding nets and drivers.
`report_flip_chip_type`	Reports the personality types of specified nets, bumps, or I/O drivers.
`route_flip_chip`	Performs flip-chip redistribution layer routing to connect the flip-chip I/O driver and the bump cell.
`set_flip_chip_bump_attributes`	Sets attributes on flip-chip bump cells for flip-chip routing.
`set_flip_chip_cell_site`	Modifies flip-chip driver cell site properties to set different flip-chip driver legal location constraints.
`set_flip_chip_driver_array`	Defines the legal locations for flip-chip driver cells to be placed in an array configuration.
`set_flip_chip_driver_island`	Defines the flip-chip cell sites that form the legal locations for the flip-chip drivers to be placed in an island style.
`set_flip_chip_driver_ring`	Defines the legal locations for flip-chip driver cells to be placed in a ring configuration
`set_flip_chip_grid`	Creates equally-spaced grid points for flip-chip driver placement.
`set_flip_chip_options`	Sets general flip-chip driver placement options during virtual flat placement.
`set_flip_chip_type`	Assigns personality types to specified nets, flip-chip bumps, or flip-chip drivers.
`set_route_flip_chip_options`	Defines the flip-chip routing options.

Table A-2 Summary of Flip-Chip Commands (Continued)

Command	Description
`update_flip_chip_pin_locations`	Updates the flip-chip bump I/O pin locations for timing analysis.
`write_flip_chip_nets`	Writes all bump to pad net names into a file.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V