# IC Compiler™
# Advanced Geometries
# User Guide

Version J-2014.09-SP2, December 2014

SYNOPSYS®

# Copyright Notice and Proprietary Information



## Destination Control Statement

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

## Disclaimer

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

## Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

## Third-Party Links

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

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

## Copyright Notice for the Command-Line Editing Feature



## Copyright Notice for the Line-Editing Library

# Contents

## Appendix A. Library Preparation

# Preface

This preface includes the following sections:

# About This User Guide

The Synopsys 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 features and recommended usage of the IC Compiler advanced geometries mode. For information about the IC Compiler implementation and integration flow, see the *IC Compiler Implementation User Guide*. For information about the design planning flow, see the *IC Compiler Design Planning User Guide*.

## Audience

This user guide is for design engineers who use IC Compiler to implement advanced-node 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 the IC Compiler tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

https://solvnet.synopsys.com/DocsOnWeb

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

- Milkyway Environment™
- IC Validator
- StarRC™

## Release Notes

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

To see the *IC Compiler Release Notes*,

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

   https://solvnet.synopsys.com/DownloadCenter

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

## Conventions

The following conventions are used in Synopsys documentation.

| Convention | Description |
|---|---|
| `Courier` | Indicates syntax, such as `write_file`. |
| *`Courier italic`* | Indicates a user-defined value in syntax, such as `write_file` *`design_list`*. |
| **`Courier bold`** | Indicates user input—text you type verbatim—in examples, such as<br>`prompt>` **`write_file top`** |
| [ ] | Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]` |
| ... | Indicates that arguments can be repeated as many times as needed, such as<br>*`pin1 pin2 ... pinN`* |
| \| | Indicates a choice among alternatives, such as<br>`low \| medium \| high` |
| Ctrl+C | Indicates a keyboard combination, such as holding down the Ctrl 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

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

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

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

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

## Contacting the Synopsys Technical Support Center

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

- Open a support case to your local support center online by signing in to the SolvNet site at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at

    http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within North America.
  - Find other local support center telephone numbers at

    http://www.synopsys.com/Support/GlobalSupportCenters/Pages

# 1

# Overview

The IC Compiler advanced geometry (IC Compiler-AG) mode supports advanced design rules and considers double-patterning requirements throughout the design planning and implementation flows for the 20-nm process node and below.

The commands used to perform the design planning and implementation tasks in advanced geometry mode are the same as those used in the other IC Compiler packages. For detailed information about these commands, see the *IC Compiler Design Planning User Guide* and the *IC Compiler Implementation User Guide*, as well as the command man pages.

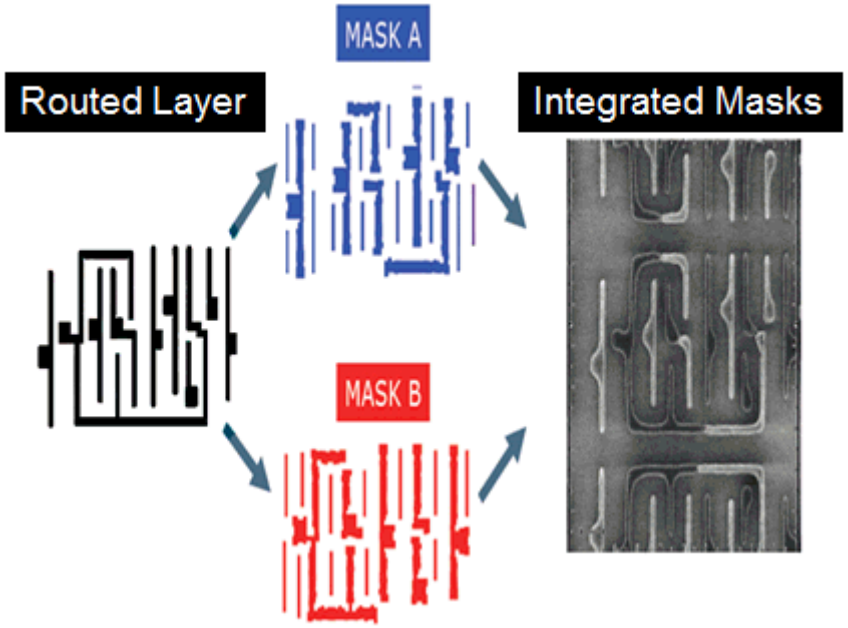
To learn about the advanced-node support enabled by the advanced geometry mode, see

- Double-Patterning Concepts
- Licensing Requirements
- Library Requirements
- Double-Patterning Mask Constraints
- Double-Patterning in the Design Flow
- Starting the IC Compiler Tool in Advanced Geometry Mode

# Double-Patterning Concepts

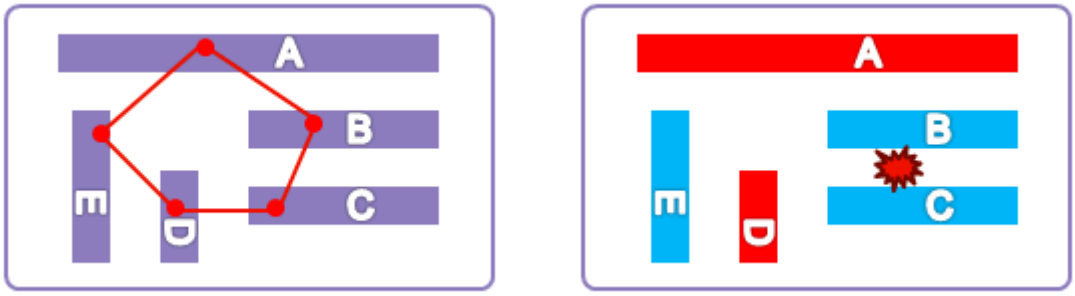
At the 20-nm process node and below, printing the required geometries is extremely difficult with the existing photolithography tools. To address this issue, a new technique, *double patterning*, is used to partition the layout mask into two separate masks, each of which has an increased manufacturing pitch to enable higher resolution and better printability, as shown in Figure 1-1.

*Figure 1-1 Double-Patterning Example*




To use double patterning, you must be able to decompose the layout into two masks, each of which meets the double-patterning spacing requirements. In the double-patterning design flow, the masks are identified by a color, as shown in Figure 1-1. A double-patterning violation occurs if your layout contains a region with an odd number of neighboring shapes where the distance between each pair of shapes is smaller than the double-patterning minimum spacing. This type of violation, which is called an *odd cycle*, is shown in Figure 1-2.

*Figure 1-2 Odd-Cycle Violation*




Layout with an odd cycle Spacing violation prevents decomposition

If the spacing between any pair in the loop is greater than the double-patterning minimum spacing, no violation occurs and the layout can be decomposed. For example, in Figure 1-3, if the spacing, *x*, between segments B and C is greater than the double-patterning minimum spacing, there is no odd cycle and the layout can be decomposed.

*Figure 1-3 No Odd-Cycle Violation*


Layout with no odd cycle Layout can be decomposed

The IC Compiler advanced geometry mode ensures that the generated layout is conducive to double patterning by considering the double-patterning spacing requirements during placement and routing and preventing odd cycles.

In general, double patterning is performed only on the bottom (lowest) metal layers, which are referred to as *double-patterning layers*. The metal shapes on the double-patterning layers must meet the double-patterning spacing requirements, whether they are routing shapes or metal within the standard cells and macros. The metal shapes on other layers do not need to meet the stricter double-patterning spacing requirements.

## Licensing Requirements

The IC Compiler advanced geometry mode requires the Galaxy-AdvRules license key. When you run multicore processes in advanced geometry mode, the tool requires one Galaxy-AdvRules license key for every four cores.

To perform In-Design double-patterning verification using IC Validator, you must also have the ICValidator2 license package for IC Validator.

## Library Requirements

The IC Compiler advanced geometry mode has the following library requirements (in addition to the standard IC Compiler library requirements):

- Advanced-node design rules in the technology file

  The technology file must contain the definitions for the routing rules that apply to your process, including double-patterning spacing and any other advanced-node design rules.

  Note:
  These rules are considered only during signal routing; the power and ground router does not recognize the double-patterning spacing rules or advanced-node design rules.

  The tool identifies the double-patterning layers by the existence of the following double-patterning minimum spacing attributes in the `Layer` section of the technology file:

  - `doublePatternEndToEndMinSpacing`
  - `doublePatternEndToSideMinSpacing` (or `doublePatternEndToSideMinSpacingTbl`)
  - `doublePatternSideToSideMinSpacing`
  - `doublePatternCornerMinSpacing`

  For information about defining the routing design rules in the technology file, see the *IC Compiler Technology File and Routing Rules Reference Manual*.

- Electromigration rules in the TLUPlus file

  The electromigration rules required for advanced nodes are defined in an electromigration TLUPlus file. To load the electromigration rules into the IC Compiler tool, use the `read_signal_em_constraints` command. This command reads only the electromigration rules from the TLUPlus file; it does not read the parasitic tables.

For example, to read the electromigration constraints from a TLUPlus file named em.tluplus, use the following command:

```
icc_shell> read_signal_em_constraints -tluplus em.tluplus
```

You can specify additional electromigration rules by using the `set_em_options` command. For information about using this command, see the signal integrity chapter in the *IC Compiler Implementation User Guide*.

- Double-patterning compliant standard cell libraries

  Two types of standard cell libraries are used for double-patterning:

  - Correct-by-construction

    In a correct-by-construction standard cell library, the cells have sufficient spacing to the cell boundaries to ensure that double-patterning violations do not occur during placement, and the libraries can be used as-is.

    When you use correct-by-construction libraries, you follow the uncolored design flow and the tool determines the appropriate mask settings for the pins and net shapes.

  - Precolored

    In a precolored standard cell library, the metal shapes inside the cells are assigned a double-patterning mask constraint, which is often referred to as a color. During placement and routing, the tool must consider these mask constraints to ensure that double-patterning violations do not occur.

    For information about setting double-patterning mask constraints on standard cells and hard macros, see Library Preparation.

    When you use precolored libraries, you follow the precolored design flow and the tool uses the double-patterning mask constraints to determine the appropriate mask settings for the pins and net shapes.

  Before starting place and route, you must determine which type of standard cell library you are using, as this affects the design flow.

### See Also

- Double-Patterning Mask Constraints
- Double-Patterning in the Design Flow

# Double-Patterning Mask Constraints

Double-patterning mask constraints indicate the mask requirements for the metal shapes of the physical pins and nets in your design. These mask requirements drive placement and routing to ensure that the resulting layout is double-patterning compliant.

Note:

Double-patterning mask constraints are used only for the precolored flow; they are not necessary in an uncolored flow.

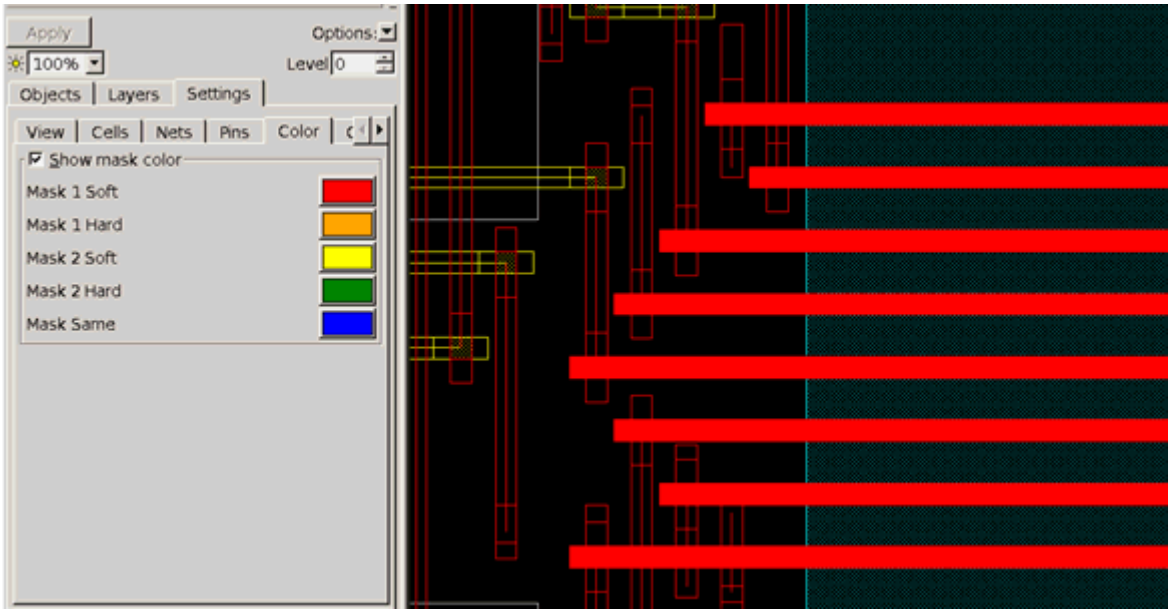
You can set double-patterning mask constraints on macro or standard cell pins (terminals and pin shapes), timing-critical nets (net shapes and route guides), and vias. For nets and pins, the double-patterning mask constraint is defined in the `double_pattern_mask_constraint` attribute. For vias, the double-patterning mask constraints are defined in the `lower_layer_double_pattern_mask_constraint`, `upper_layer_double_pattern_mask_constraint`, and `via_layer_double_pattern_mask_constraint` attributes. Table 1-1 shows the supported values for these attributes.

*Table 1-1 Double-Patterning Mask Constraint Values*

| Attribute value | Description |
|---|---|
| `same_mask` | This constraint means that the mask color is not yet been determined. Shapes with this attribute must be at least the double-patterning minimum spacing distance from any other colored metal shape. |
| `mask1_hard`<br>`mask1_soft` | This constraint means that the shape has the mask1 color. Shapes with this attribute must be at least the double-patterning minimum spacing distance from other mask1-colored metal shapes.<br>The `mask1_hard` attribute value is usually set on macro pins, while the `mask1_soft` attribute value is usually set on nets. |
| `mask2_hard`<br>`mask2_soft` | This constraint means that the shape has the mask2 color. Shapes with this attribute must be at least the double-patterning minimum spacing distance from other mask2-colored metal shapes.<br>The `mask2_hard` attribute value is usually set on macro pins, while the `mask2_soft` attribute value is usually set on nets. |
| `any_mask` | This constraint means that the shape is not colored. Shapes with this attribute must be at least the standard minimum spacing distance from other metal shapes; the double-patterning minimum spacing rules do not apply to these shapes. |

To show the double-patterning mask constraints (colors) when you view metal shapes in the GUI, select “Show mask color” in the Color tab of the View Settings panel, as shown in Figure 1-4.

*Figure 1-4 Displaying Mask Color in the GUI*




## Double-Patterning in the Design Flow

Double-patterning considerations affect all parts of the place and route flow.

Regardless of the type of library you are using, you must consider the additional resource requirements and design rule impacts throughout the flow. If you are not using a correct-by-construction library, you also need to ensure that the double-patterning mask constraints are properly set before starting place and route.

Most standard cell libraries used for double-patterning are correct-by-construction; therefore, in most cases, you will use the uncolored double-patterning design flow, which is shown in Figure 1-5. If you are using a precolored standard cell library, you must use the precolored double-patterning design flow, which is shown in Figure 1-6. The precolored flow is similar to the uncolored flow, but contains some additional steps, which are shown in red.

*Figure 1-5 Uncolored Double-Patterning Design Flow*

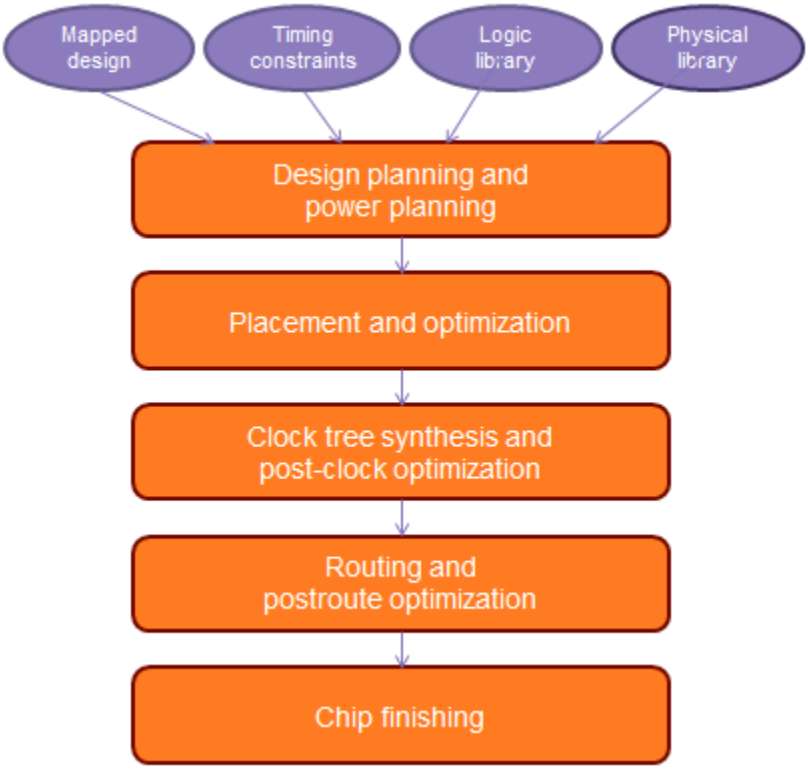
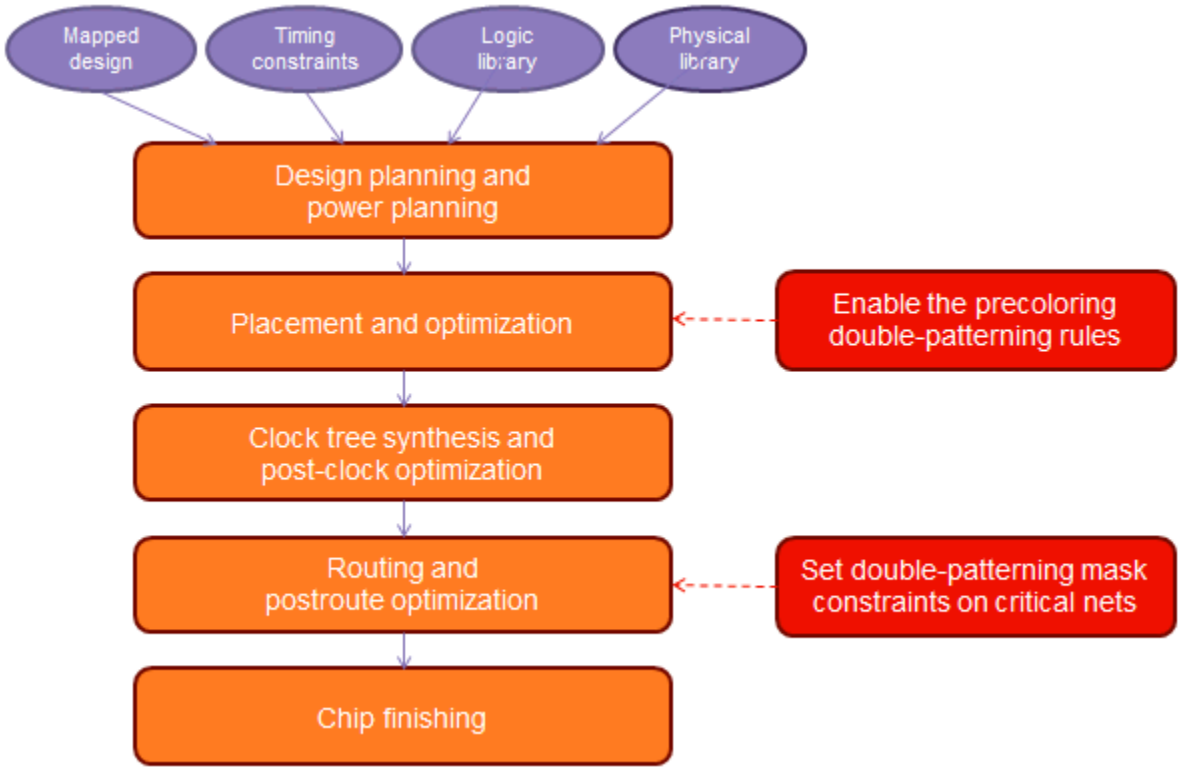


*Figure 1-6 Precolored Double-Patterning Design Flow*




To learn about using the IC Compiler tool with advanced-node designs, see

- Library Preparation (required only for the precolored flow)
- Design Planning and PG Routing
- Placement
- Routing
- Chip Finishing
- Streaming Out a Precolored Design

## Starting the IC Compiler Tool in Advanced Geometry Mode

To start the IC Compiler Tool in advanced geometry mode in the command-line interface, use the `-ag_mode` option when you run the `icc_shell` command.

```
% icc_shell -ag_mode
```

To start the tool in the graphical user interface (GUI), use the `-gui` option.

```
% icc_shell -ag_mode -gui
```

# 2

# Design Planning and PG Routing

During hierarchical design planning and power and ground (PG) routing, you must consider the double-patterning requirements to avoid odd-cycle violations or routing difficulties later in the design flow.

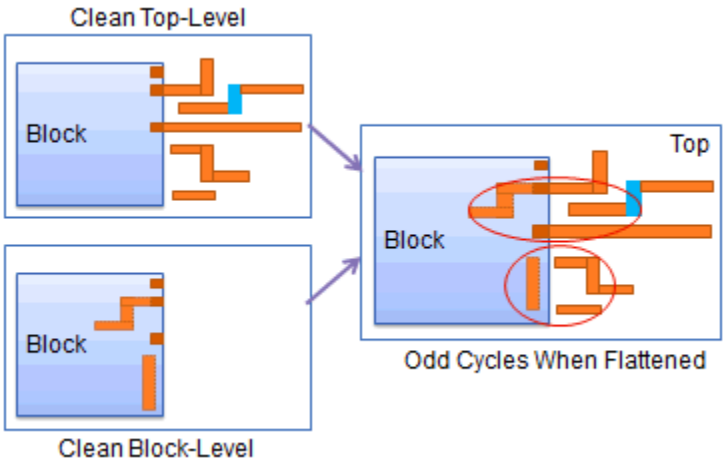
To learn about performing hierarchical design planning and PG routing on advanced-node designs, see

- Hierarchical Floorplanning
- Power and Ground Routing

# Hierarchical Floorplanning

Maintaining a double-patterning-compliant layout in a hierarchical flow can be difficult because you do not have complete visibility of the design throughout the flow. When working at the block level, the top-level shapes are not visible. When working at the top level, only the extracted pin shapes are visible. When the top level and blocks are integrated, odd-cycle violations can occur due to same-layer shapes within the double-patterning minimum spacing distance across the block boundary or same-layer shapes that connect to a block pin in the block and the top level. Figure 2-1 shows examples of these hierarchical odd-cycle violations.

*Figure 2-1 Hierarchical Odd-Cycle Violations*




To prevent these issues, follow these guidelines when designing a block:

- Place the block pins on non-double-patterning layers, if possible.

  If you must place block pins on a double-patterning layer,

  - Make the pins longer to enable via access.

    For blocks with low pin density, ensure that the length of the block pins is at least 10 times the layer pitch. For blocks with high pin density, ensure that the length of the block pins is at least 14 times the layer pitch.

  - Leave sufficient space between the block pins.

    If the block pins are uncolored, they must be separated by at least the double-patterning minimum spacing distance. If the block pins are precolored, ensure

that pins with the same mask color have at least the double-patterning minimum spacing distance between them.

In addition, for blocks with low pin density, you should skip at least one track for every five pins.

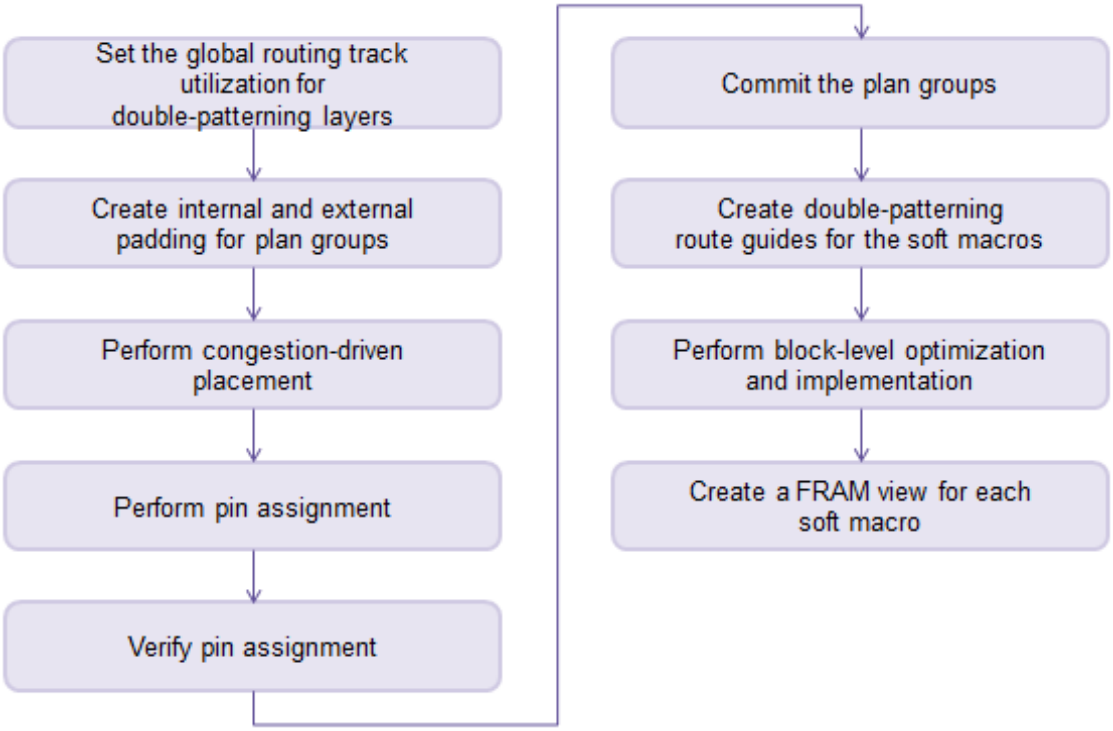
- Create placement blockages around the block pins.

- Create routing blockages on the double-patterning layers along the block boundary and around the block pins to prevent routing shapes in these areas.

## Implementing Double-Patterning-Compliant Blocks

Figure 2-2 shows the steps used to implement double-patterning-compliant blocks using the IC Compiler hierarchical design planning capabilities. Following this flow prevents double-patterning violations in the flattened design. Note that the same flow is used for both the uncolored and precolored flows.

*Figure 2-2 Design Planning Overview*



The following procedure provides additional information about these steps:

1. Set the global routing track utilization values for the double-patterning layers.

   For details, see "Setting the Global Routing Track Utilization" on page 4-4.

2. Create internal and external padding for the plan groups to prevent cells from being placed near the boundaries, where they might overlap with the longer pins used on double-patterning layers.

   To create the plan group padding, use the `create_fp_plan_group_padding` command.

   ```
   icc_shell> create_fp_plan_group_padding
   ```

   By default, the tool determines the internal padding size based on the average space between wire tracks on all the layers. You can increase the padding size by using the `-internal_widths` and `-external_widths` options with the `create_fp_plan_group_padding` command. However, you cannot decrease the padding size from the computed default.

3. Perform congestion-driven placement by using the `create_fp_placement` command.

   The `create_fp_placement` command considers the global routing track utilization values set in step 1 when performing the placement.

   ```
   icc_shell> create_fp_placement -congestion_driven
   ```

4. Perform plan-group-aware routing by using the `route_zrt_global` command. (This step is required only if you perform route-based pin assignment; it is not required if you perform flyline-based pin assignment.)

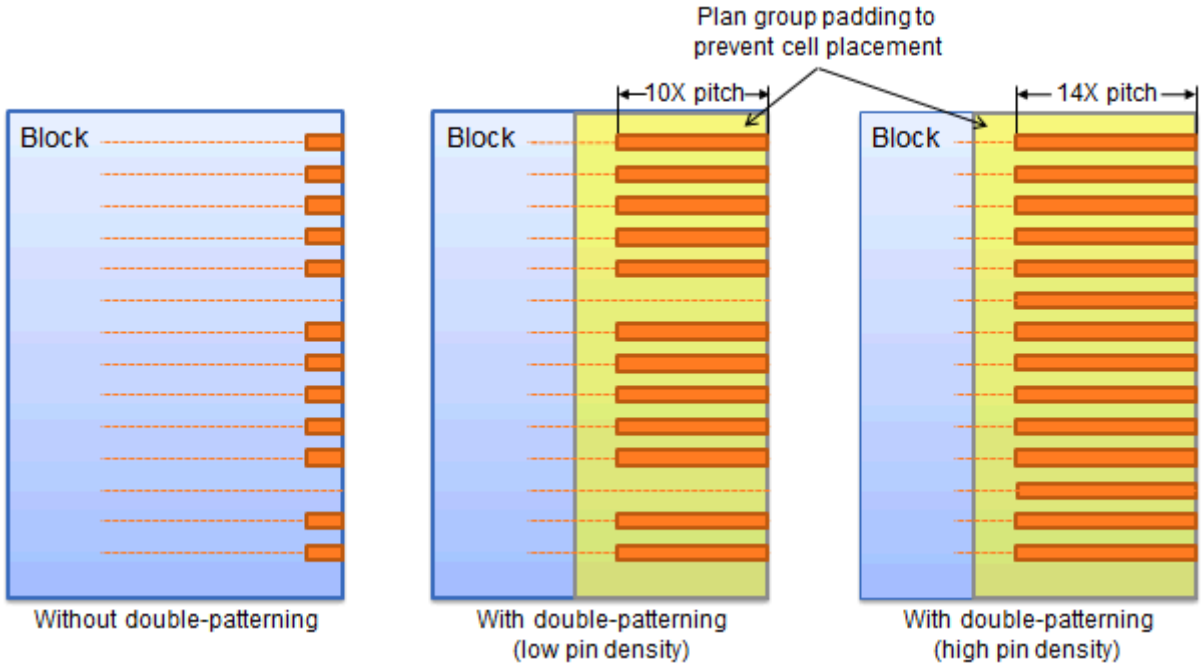
   ```
   icc_shell> set_route_zrt_common_options -plan_group_aware all_routing
   icc_shell> route_zrt_global
   ```

5. Perform pin assignment by using the `place_fp_pins` command.

   ```
   icc_shell> place_fp_pins -use_existing_routing [get_plan_groups]
   ```

   During pin assignment, the tool creates the block pins based on the global routing results (or on flylines). If possible, the tool places pins on non-double-patterning layers. If it must place a pin on a double-patterning layer, it automatically increases the pin depth (the length of the pin extending inside the block from the edge of the block). By default, the tool uses a double-patterning minimum pin depth of between 10 and 14 times the minimum track width, depending on the estimated pin density. Figure 2-3 shows an example of how the tool increases the pin depth for a double-patterning design.

*Figure 2-3 Double-Patterning Pin Assignment Results*




If the default double-patterning minimum pin depth is not sufficient to enable pin access for your design, set a pin-depth constraint by using the `-depth` option with the `set_pin_physical_constraints` command. This option specifies the minimum pin length in microns.

```
icc_shell> set_pin_physical_constraints [get_pins *] \
   -layers {dpt_layers} -depth min_length
icc_shell> set_fp_pin_constraints -use_physical_constraints on
icc_shell> place_fp_pins -use_existing_routing [get_plan_groups]
```

6. Verify the pin assignment by using the `check_fp_pin_assignment` command.

```
icc_shell> check_fp_pin_assignment -pin_size
```

When you run the `check_fp_pin_assignment` command with the `-pin_size` option, it verifies that all pins on double-patterning layers have the minimum depth and issues errors for pins shorter than this minimum depth.

If you are using the precolored flow, use the `-pin_double_pattern_mask_constraint` option with the `check_fp_pin_assignment` command to verify the compatibility of abutted pins. The X's in Table 2-1 indicate the incompatible attribute combinations.

*Table 2-1 Incompatible Abutted Pin Combinations*

| | same_mask | mask1_hard | mask1_soft | mask2_hard | mask2_soft | any_mask |
|---|---|---|---|---|---|---|
| same_mask | | | | | | X |
| mask1_hard | | | | X | X | X |
| mask1_soft | | | | X | X | X |
| mask2_hard | | X | X | | | X |
| mask2_soft | | X | X | | | X |
| any_mask | X | X | X | X | X | |

7. Perform incremental pin placement or editing, if needed.
8. Commit the plan groups into soft macros.

   ```
   icc_shell> commit_fp_plan_groups
   ```

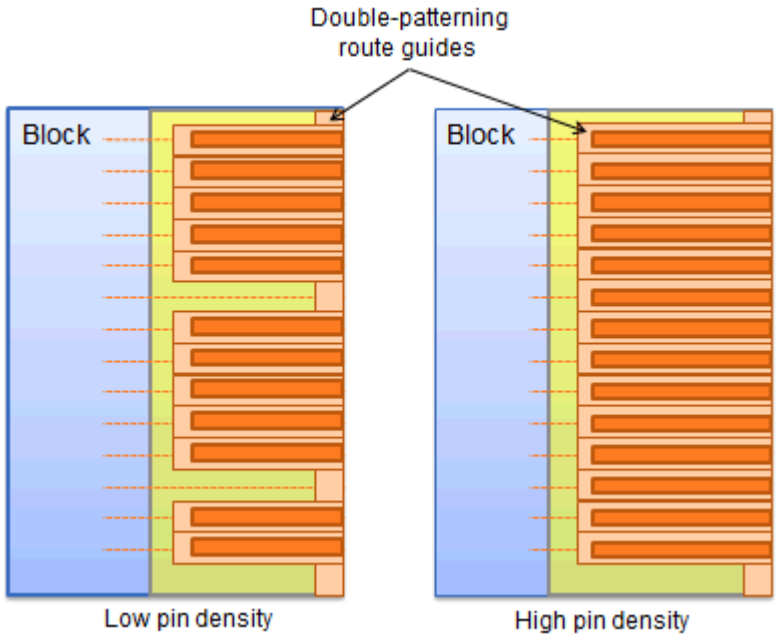
9. Create mask constraint route guides around the pins and boundaries of each soft macro.

   A mask constraint route guide is a zero-spacing route guide that prevents the router from creating any net shapes within its boundary and guides the router to create routes without creating hierarchical mask constraint violations.

   ```
   icc_shell> create_mask_constraint_route_guides
   ```

   Figure 2-4 shows the mask constraint route guides created by this command. The figure on the left illustrates the route guides for a design with low pin density, while the figure on the right illustrates the route guides for a design with high pin density.

*Figure 2-4 Mask Constraint Route Guides*




The route guides created by this command around pins use the naming style __DPT_ROUTE_GUIDE_PIN_*x*_*y*, where *x* is the mask number and *y* is a nonnegative integer. The route guides created by this command around the boundary use the naming style __DPT_ROUTE_GUIDE_BDRY_*x*_*y*, where *x* is the mask number and *y* is a nonnegative integer. You can query, view, and remove these route guides just like any other route guides.

Note:

If you modify the pin shapes or pin placement, you must regenerate the mask constraint route guides. The `create_mask_constraint_route_guides` command removes any existing mask constraint route guides in the design and then creates new ones.

10. Perform block-level optimization and implementation.

11. Create a FRAM view for each soft macro that includes mask constraint route guides.

    By default, the `create_macro_fram` command creates mask constraint route guides on all layers for which double-patterning spacing rules are defined in the technology file.

    ```
    icc_shell> create_macro_fram
    ```

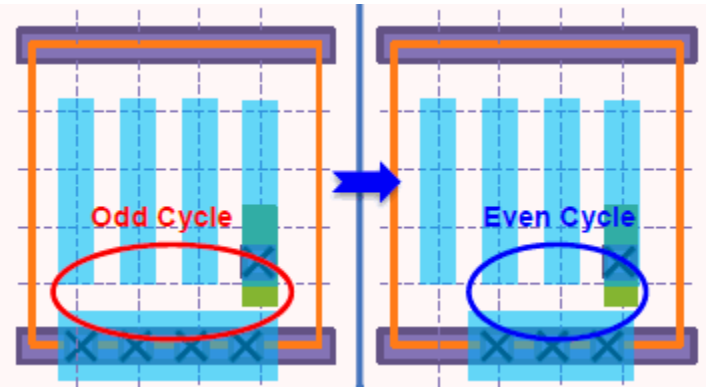
## Power and Ground Routing

Although the power and ground router does not consider double-patterning requirements, you can help to prevent odd cycles with fixed shapes by

- Avoiding jogs that occupy or block an odd number of pitches (odd-pitch jogs)
- Avoiding odd-pitch nonpreferred direction shapes
- Avoiding off-grid shapes that block even tracks
- Avoiding via arrays that have an even number of vias

  When a via array has an even number of vias, it can interact with fixed shapes to create an odd cycle. Using a via array with an odd number of vias reduces the likelihood of generating odd cycles, as shown in Figure 2-5. To specify the via array size, use the `-size_by_array_dimensions` option with the `set_preroute_advanced_via_rule` command.

*Figure 2-5 Odd-Cycle Violation Due to Even-Sized Via Array*



# 3

# Placement

During placement, you need to ensure that the pins of the placed cells do not create double-patterning violations and that the additional resources required by double-patterning do not cause congestion issues.

To learn about performing placement on advanced-node designs, see

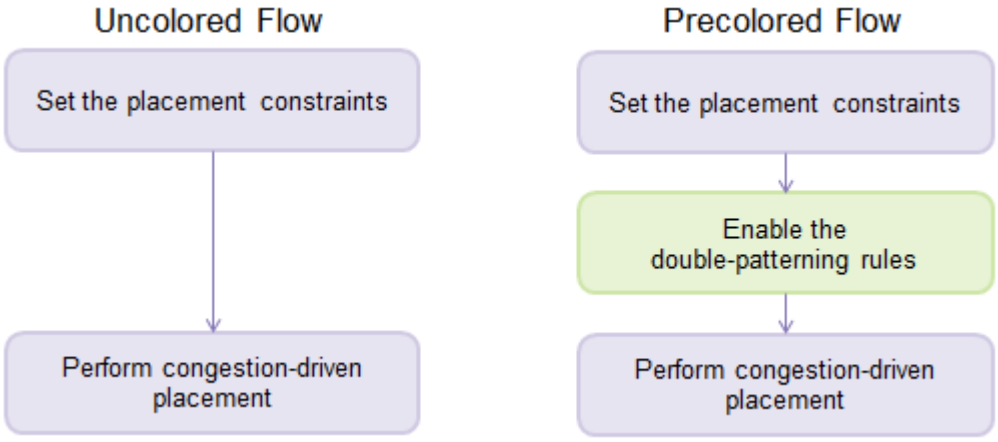
- Placement Overview
- Setting the Placement Constraints
- Performing Placement in the Precolored Flow
- Performing Congestion-Driven Placement

For general information about performing placement with the IC Compiler tool, see the “Placement and Optimization” chapter in the *IC Compiler Implementation User Guide*.

# Placement Overview

Figure 3-1 shows the steps used to perform placement in the double-patterning design flow. The left side shows the steps in the uncolored flow, while the right side shows the steps in the precolored flow, which contains an additional step (shown in red).

*Figure 3-1 Placement Tasks*




# Setting the Placement Constraints

If you are using the uncolored flow, no special settings are needed for the placer to ensure double-patterning compliance.

However, setting certain placement constraints helps the placer to generate a good placement for an advanced-node design. To learn about these constraints, see

- Defining Placement Blockages for the Power Network
- Increasing the Cell Spacing
- Enabling the Threshold Voltage Implant Layer Rules

## Defining Placement Blockages for the Power Network

For advanced-node designs, the power network is very dense. To prevent placement violations between placed cells and the power network, while still allowing some flexibility in placing cells that overlap the power network, consider the power network as a partial placement blockage by using the following command:

```
icc_shell> set_pnet_options -partial
```

## Increasing the Cell Spacing

To improve the routability of your design, you might need to increase the spacing between cells. In this case, use the `set_lib_cell_spacing_label` and `set_spacing_label_rule` commands to control the cell spacing.

For information about using these commands, see the command man pages or the "Defining Spacing Requirements" section in the placement chapter of the *IC Compiler Implementation User Guide*.

## Enabling the Threshold Voltage Implant Layer Rules

By default, placement does not consider the threshold voltage implant layer spacing rules. If your standard cell library has minimum width or minimum spacing rules for multiple threshold voltage cells, you must enable these rules before performing placement by setting the `legalizer_consider_vth_spacing` variable to `true`.

```
icc_shell> set_app_var legalizer_consider_vth_spacing true
```

Note:

In addition to setting this variable, threshold voltage implant layer rules must be defined in the technology file and the implant width property must be attached to the FRAM views of the multiple threshold voltage cells. For information about these library preparation requirements, see "Defining Implant Layer Rules in the Technology File" on page A-2.

When you enable these rules,

- The legalizer considers the minimum width and minimum spacing rules of the threshold voltage implant layers and adjusts the cell placements to meet these rules. The following message in the log file indicates that these rules are enabled during legalization:

  ```
  INFO: Legalizer/Checker to consider Vth-based spacing rules
  ```

- The `check_legality` command reports violations of these rules. However, the violations are reported as general spacing rule violations and not specifically as threshold voltage implant layer spacing violations.

  To report the cells that have spacing violations, use the `-verbose` option with the `check_legality` command.

  After running the `check_legality` command, you can use the error browser to view the violations.

To remove the threshold voltage implant layer spacing rules, use the `remove_all_spacing_rules` command with the `legalizer_consider_vth_spacing` variable set to `true`.

# Performing Placement in the Precolored Flow

If you are using the precolored flow, the placer must ensure that cell placement does not cause double-patterning violations. To learn how to avoid double-patterning violations during placement, see the following topics:

- Enabling the Double-Patterning Rules
- Using Cell-Level Mask Swapping to Fix Double-Patterning Violations

## Enabling the Double-Patterning Rules

If you are using the precolored flow, you must enable the double-patterning rules by running the `enable_double_patterning_rules` command before performing placement. This command uses the double-patterning spacing rules defined in the technology file to generate cell-spacing rules for each standard cell in the reference libraries; these generated cell-spacing rules are attached to the Milkyway design library. To output a detailed report of the generated cell-spacing rules, use the `-verbose` option when you run this command.

When the double-patterning rules are enabled, the placer considers the double-patterning spacing requirements and avoids violations by increasing the cell spacing, changing the cell orientation, or moving the cell. In some cases, the tool can avoid violations without changing the cell placement by using a technique called *cell-level mask swapping*, which is described in the following section, "Using Cell-Level Mask Swapping to Fix Double-Patterning Violations."

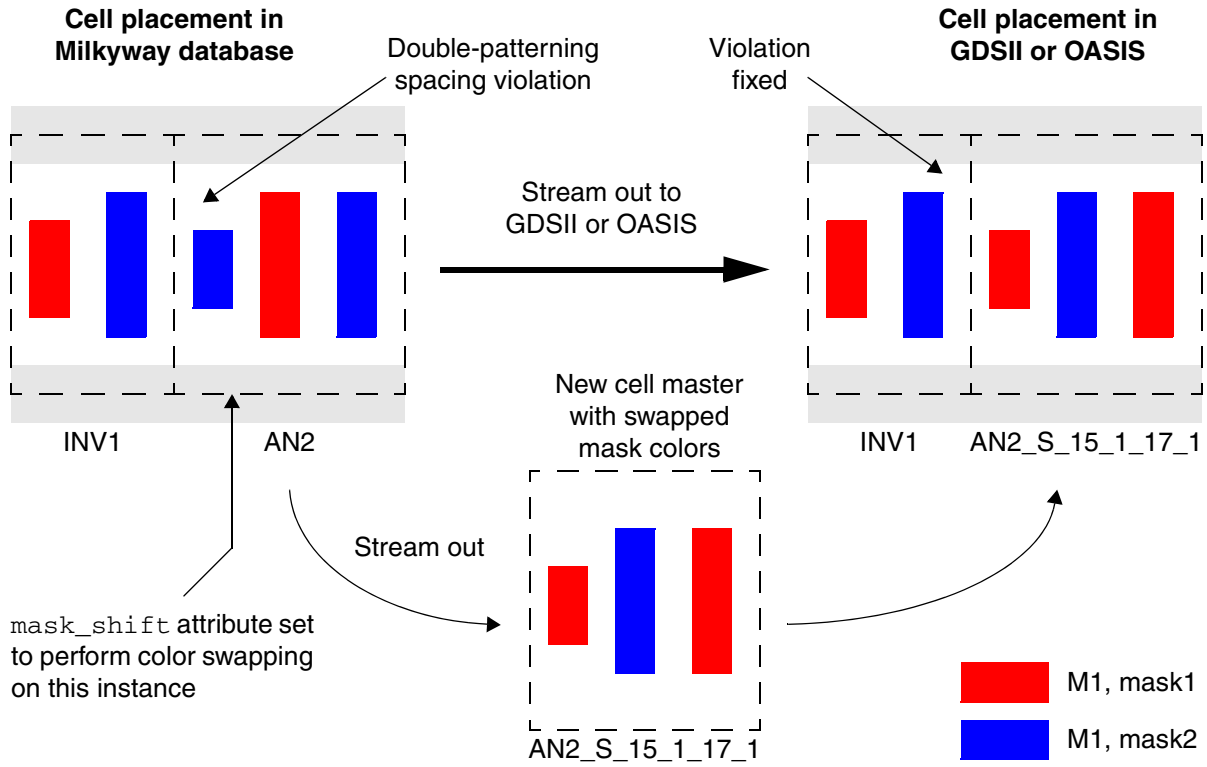
To remove the double-patterning cell-spacing rules from the Milkyway design library, run the `disable_double_patterning_rules` command.

To determine whether the double-patterning rules are enabled for placement, use the `is_double_patterning_enabled` command.

## Using Cell-Level Mask Swapping to Fix Double-Patterning Violations

When cell-level mask swapping is enabled, the tool can fix double-patterning violations by swapping the mask constraints in a cell instance instead of moving or flipping the cell, as shown in Figure 3-2. In this example, the red and blue rectangles are colored shapes in metal layer M1. There is a double-patterning violation between the blue shapes in the instances of the INV1 and AN2 cells. The violation is fixed upon stream-out by swapping the mask constraints in the AN2 instance.

*Figure 3-2 Double-Patterning Cell-Level Mask Swap*




To perform this swap, the placer sets the `mask_shift` attribute on the cell instance. This attribute is a text string that specifies whether to swap the mask constraints within each layer. For example, when the attribute is set to the following string, the tool swaps the mask constraints of the shapes in the M1 layer, but not in the VIA1 and M2 layers:

```
{{M1 1} {VIA1 0} {M2 0}}
```

The `write_stream` command uses the `mask_shift` attribute setting to modify the mask constraints of the shapes within each cell during stream-out to GDSII or OASIS format. In this example, the `write_stream` command swaps the mask constraints in layer M1. For that layer only, it writes out the shapes with a mask constraint of mask1 to the mask2 mask, and conversely, writes out the shapes a mask constraint of mask2 to the mask1 mask.

### See Also

- Enabling Cell-Level Mask Swapping
- Cell-Level Mask Swapping During Stream-Out

## Performing Congestion-Driven Placement

To get the best placement results on advanced-node designs, use the Zroute global router to perform congestion-driven placement by using the commands shown in Example 3-1.

*Example 3-1 Performing Congestion-Driven Placement Using the Zroute Global Router*

```
set_app_var placer_congestion_effort medium
set_app_var placer_show_zroutegr_output true
place_opt -congestion
```

# 4

# Routing

When routing advanced-node designs, Zroute avoids and fixes odd-cycle violations as well as the other routing design rule violations.

To learn about performing routing on advanced-node designs, see

- Routing Overview
- Preparing for Routing
- Routing Advanced-Node Designs
- Verifying the Routing
- Extraction for Advanced-Node Designs

For general information about performing routing with Zroute, see the “Routing Using Zroute” chapter in the *IC Compiler Implementation User Guide*.

## Routing Overview

Figure 4-1 shows the steps used to perform routing in the uncolored double-patterning design flow. Note that the step shown in darker purple is required only in the uncolored flow.

*Figure 4-1 Routing Tasks in the Uncolored Flow*

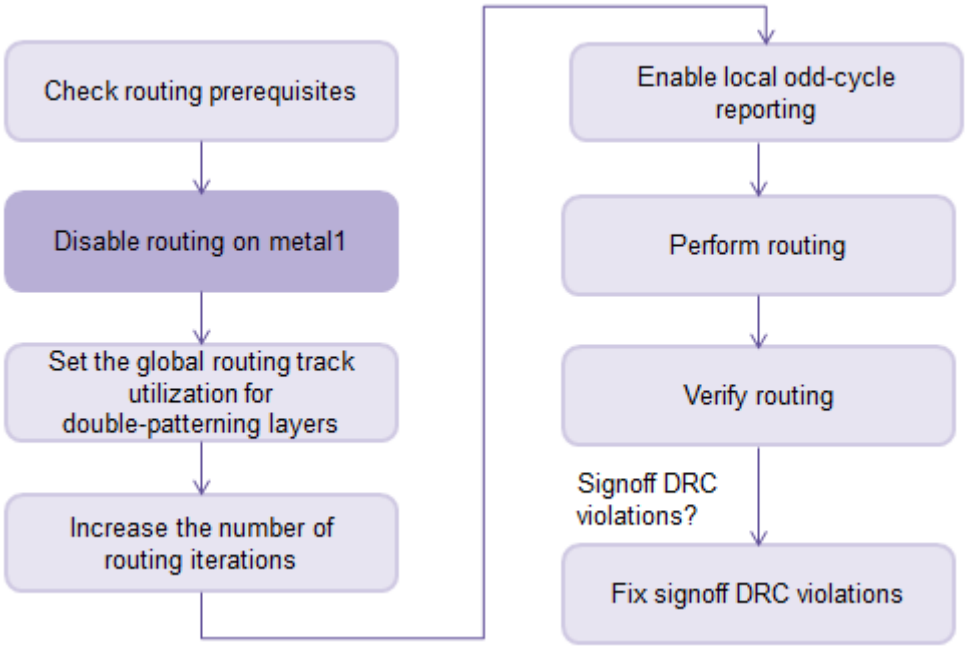
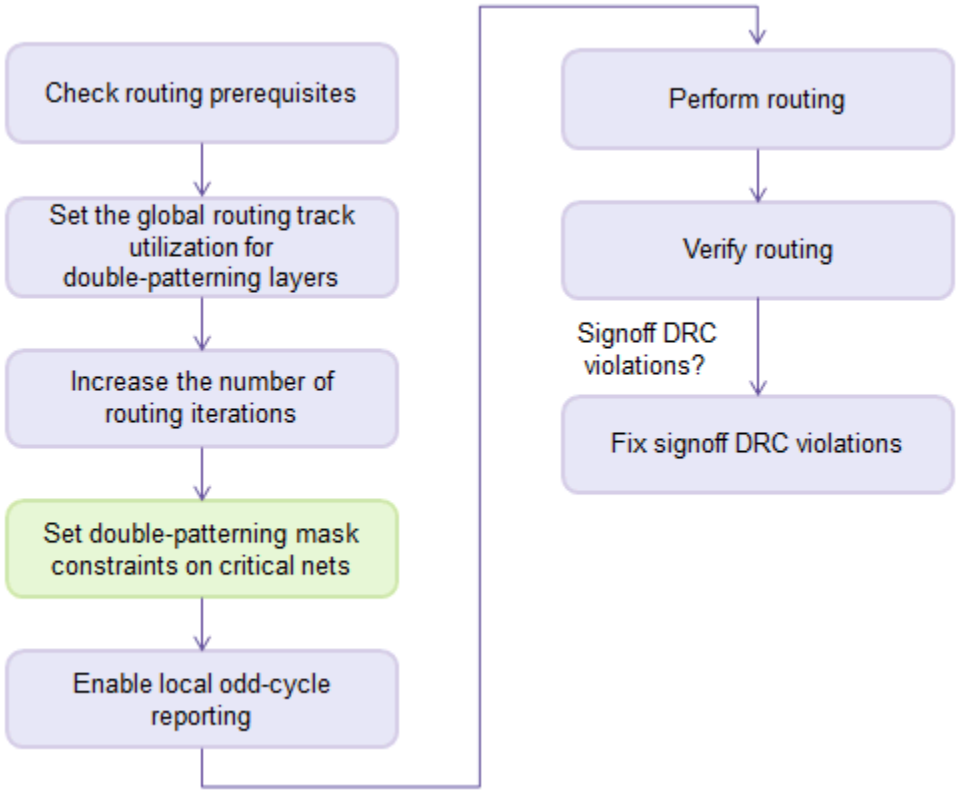



Figure 4-2 shows the steps used to perform placement in the precolored double-patterning design flow, which contains an additional step (shown in green).

*Figure 4-2 Routing Tasks in the Precolored Flow*




## Preparing for Routing

Before you run the router, you must ensure that the reference library and design meet certain prerequisites and that the routing setup is correct. To learn how to perform these tasks, see

- Checking the Routing Prerequisites
- Disabling Routing on the Metal1 Layer
- Setting the Global Routing Track Utilization
- Increasing the Number of Routing Iterations
- Setting Double-Patterning Mask Constraints on Nets
- Enabling Odd-Cycle Reporting

## Checking the Routing Prerequisites

Before you route your design, you must ensure that the physical library and design meet certain requirements. To learn about these requirements, see the “Prerequisites for Routing” section in the routing chapter of the *IC Compiler Implementation User Guide*.

Running the `check_zrt_routability` command to verify the routability of your design is particularly important for advanced-node designs. For detailed information about this command, see the man page or the “Checking Routability” section in the routing chapter of the *IC Compiler Implementation User Guide*.

You should also verify that there are no placement or routing congestion hotspots by generating and analyzing a congestion map, as described in the “Analyzing Congestion” section in the routing chapter of the *IC Compiler Implementation User Guide*.

## Disabling Routing on the Metal1 Layer

Note:

This task is not required if you are using the precolored flow.

If you are using the uncolored design flow, you must disable routing on the metal1 layer, except to connect to pins, to avoid double-patterning violations between the routes and metal in the standard cells. To disable routing on the metal1 layer, use the following commands:

```
icc_shell> set_ignored_layers -min_routing_layer M2
icc_shell> set_route_zrt_common_options \
   -global_min_layer_mode allow_pin_connection \
   -net_min_layer_mode allow_pin_connection \
   -connect_within_pins_by_layer_name {{M1 via_wire_standard_cell_pins}}
```

## Setting the Global Routing Track Utilization

In general, the track utilization values for the double-patterning layers should be lower than for the other routing layers to reserve space for the larger minimum spacing requirements. The default track utilization values for the double-patterning layers are 80 percent for metal1 and metal2, 85 percent for metal3, and 90 percent for any other double-patterning layers. The track utilization for non-double-patterning layers is 100 percent.

You can adjust these track utilization values by using the `set_route_zrt_global_options` command to set the `-double_pattern_utilization_by_layer_name` option. For example, to set the track utilization to 70 percent on the metal2 and metal3 layers, use the following command:

```
icc_shell> set_route_zrt_global_options \
   -double_pattern_utilization_by_layer_name {{M2 70.0} {M3 70.0}}
```

## Increasing the Number of Routing Iterations

Due to the increased complexity of the advanced-node design rules, as well as the double-patterning requirements, DRC convergence typically takes longer for advanced-node designs. By default, the `route_opt` command performs 10 iterations of detail routing during initial routing and 4 iterations during ECO routing. To improve DRC convergence, you should increase the number of routing iterations performed by the `route_opt` command.

For example, to increase the number of initial routing iterations to 40 and the number of ECO routing iterations to 20, use the following command:

```
icc_shell> set_route_opt_strategy -search_repair_loops 40 \
   -eco_route_search_repair_loops 20
```

## Setting Double-Patterning Mask Constraints on Nets

Note:

This task is not required if you are using the uncolored flow.

If you are using the precolored design flow, you can set double-patterning mask constraints on timing-critical nets, such as clock nets, by defining a precoloring routing rule and applying it to the net.

To define a precoloring routing rule, use the following syntax:

```
define_routing_rule rule_name
   -double_pattern_mask_constraints
      {layer1 constraint1 layer2 constraint2 ... layern constraintn}
```

where *constraint* is one of `same_mask`, `mask1_soft`, or `mask2_soft`.

For example, to define a precoloring routing rule that sets `mask1_soft` constraints on the M4 and M5 layers, use the following command:

```
icc_shell> define_routing_rule clock_mask1_soft \
   -double_pattern_mask_constraints {M4 mask1_soft M5 mask1_soft}
```

You apply the precoloring routing rule to a net in the same way as any other nondefault routing rule. For example, to apply the clock_mask1_soft precoloring routing rule defined in the previous example to the CLK clock, use the following command:

```
icc_shell> set_clock_tree_options -clock_trees CLK \
   -layer_list {M4 M5} -routing_rule clock_mask1_soft \
   -use_default_routing_for_sinks 1
```

To ensure DRC convergence, you should set double-patterning mask constraints only on a very few timing-critical nets.

### See Also

- Double-Patterning Mask Constraints
- "Using Nondefault Routing Rules" section in the routing chapter of the *IC Compiler Implementation User Guide*

## Enabling Odd-Cycle Reporting

By default, Zroute does not report odd-cycle violations during detail routing. To enable the reporting of odd-cycle violations that occur within a routing partition (local odd cycles), use the `set_route_zrt_common_options` command to set the `-report_local_double_pattern_odd_cycles` option to `true`.

```
icc_shell> set_route_zrt_common_options \
   -report_local_double_pattern_odd_cycles true
```

# Routing Advanced-Node Designs

To route an advanced-node design, use the `route_opt` command. When routing an advanced-node design, Zroute

- Uses the double-patterning spacing rules to avoid and fix local odd-cycle violations

  Unlike other routing design rules, all metal shapes need not meet the double-patterning spacing rules; the tool checks only for odd-cycle violations. Zroute automatically identifies metal shapes that can cause odd-cycle violations, such as those caused by odd-pitch jogs, off-grid wires, or routing in the nonpreferred direction, and applies the double-patterning spacing rules only when needed.

- Ensures that all metal shapes meet the other routing rules, including the advanced-node routing rules

### See Also

- Double-Patterning Concepts
- Routing Precolored Designs
- Routing Nets in the GUI

## Routing Precolored Designs

When routing a precolored design, Zroute assigns a mask color to net shapes based on the precoloring routing rules and by propagating the double-patterning mask constraints on precolored pins connected to same-layer shapes.

During the first detail routing iteration, Zroute propagates the color of a colored pin to the wire or same-layer via enclosure connected to the pin. The color is propagated until the wire changes to another layer or the propagation threshold distance of 500 nm is reached. If the same-layer wire extends beyond the propagation threshold distance, a DRC violation occurs.

During subsequent detail routing iterations, Zroute performs incremental color propagation for colored pins within the propagation threshold distance from any changed shapes.

Zroute determines the spacing requirement for a colored shape as follows:

- If a neighboring shape on the same layer is the same mask color or is an undetermined mask color (`same_mask` mask constraint), the shapes must meet the double-patterning minimum spacing requirements.
- If a neighboring shape on the same layer is a different mask color or is not colored, the shapes must meet the minimum spacing defined for the layer (or the nondefault routing rule, if one has been assigned to the net).

## Routing Nets in the GUI

When you use the Advanced Route tool to route nets in the GUI for an advanced-node design, you must enable double-patterning odd-cycle checking for the interactive DRC capability by using the following command:

```
icc_shell> gui_set_pref_value -category IdrcOptions \
   -key "DPTOddCycle" -value true
```

Figure 4-3 shows an odd-cycle violation detected by the interactive DRC capability.

*Figure 4-3 Interactive DRC Odd-Cycle Violation*


## Verifying the Routing

After detail routing is complete, you should verify the routing results. To learn about verifying the routing results, see

- Verifying the Routing in the IC Compiler Tool
- Performing Signoff Design Rule Checking
- Fixing Signoff DRC Violations

### Verifying the Routing in the IC Compiler Tool

To perform double-patterning compliance checking and design rule verification in the IC Compiler tool, run the `verify_zrt_route` command. Although Zroute detail routing performs local odd-cycle checks, the `verify_zrt_route` command might detect additional odd-cycle violations because it uses a larger partition size for design rule checking.

## Performing Signoff Design Rule Checking

To perform double-patterning compliance checking and design rule verification with signoff accuracy, use the `signoff_drc` command to perform In-Design physical verification using the IC Validator tool.

To run In-Design physical verification, set the physical signoff options as described in the "Setting The Physical Signoff Options" section in the routing chapter of the *IC Compiler Implementation User Guide* and then run the `signoff_drc` command as shown in the following example:

```
icc_shell> set_physical_signoff_options -exec_cmd icv \
   -drc_runset runset_file -mapfile mapping_file
icc_shell> signoff_drc \
   -read_cel_view \
   -ignore_child_cell_errors \
   -user_defined_options {-holding_cell} \
   -run_dir initial_icv_error_dir
```

Note:

The `signoff_drc` command uses the on-disk design information, not the design information in memory. You must save the current state of the design before running the `signoff_drc` command.

When you analyze the DRC violations reported by the `signoff_drc` command, you should not see local odd-cycle violations, as these are addressed by Zroute. If you see local odd-cycle violations, check the consistency between the technology file and the runset. If you see long-range odd-cycle violations (more than 5 microns in extent), use the `signoff_autofix_drc` command to fix these violations.

### See Also

- "Performing Signoff Design Rule Checking" section in the routing chapter of the *IC Compiler Implementation User Guide*

## Fixing Signoff DRC Violations

The In-Design automatic signoff DRC fixing feature addresses violations detected by the `signoff_drc` command, including

- Advanced-node design rule violations
- Double-patterning odd-cycle violations

Note that to fix odd-cycle violations, you must run a separate `signoff_autofix_drc` command that targets only these violations. When you run the `signoff_autofix_drc` command to fix odd-cycle violations, you must

- Use the `-config_file` option to specify the double-patterning configuration file

  The configuration file defines the layer-to-DRC-rule mapping in the following format:

  `"mask_layer_name" "full_IC_Validator_DRC_rule_comment"`

  The `signoff_autofix_drc` command processes only those rules that are specified in the configuration file. To determine the double-patterning rules for your technology, see the design rule manual (DRM) provided by your vendor.

- Use the `-custom_guidance dpt` option to enable a double-patterning-specific run
- Use the `-incremental_level off` option to disable incremental mode (odd-cycle fixing works only in non-incremental mode)

The following example shows the syntax used to perform automatic fixing of the odd-cycle violations detected by `signoff_drc`:

```
icc_shell> signoff_autofix_drc \
   -init_drc_error_db initial_icv_error_dir \
   -config_file DPT_config_file \
   -custom_guidance dpt
   -incremental_level off
```

**See Also**

- "Automatically Fixing Signoff DRC Violations" section in the routing chapter of the *IC Compiler Implementation User Guide*

## Extraction for Advanced-Node Designs

When the IC Compiler tool performs extraction for advanced-node designs, it considers the overlay variation due to the double patterning, as well as effects such as orientation-based width variation. The overlay variation is modeled by using the `ER_VS_SI_SPACING` tables. The orientation-based width variation is modeled by using the `ETCH_VS_WIDTH_AND_SPACING` tables. By default, the tool uses the reference direction specified in the TLUPlus file when applying the `ETCH_VS_WIDTH_AND_SPACING` tables. To explicitly specify the reference direction as vertical or horizontal, use the `set_extraction_options -reference_direction` command. For example, to explicitly set the reference direction as horizontal, use the following command:

```
icc_shell> set_extraction_options -reference_direction horizontal
```

If you run this command multiple times, the new setting overrides the existing setting; extraction always uses the latest setting.

For more information about the `ER_VS_SI_SPACING` and `ETCH_VS_WIDTH_AND_SPACING` tables, see the StarRC documentation.

# 5

# Chip Finishing

To learn about advanced-node considerations for chip-finishing tasks, see

- Redundant Via Insertion for Advanced-Node Designs
- Filler Cell Insertion for Advanced-Node Designs
- Boundary Cell Insertion for Advanced-Node Designs

## Redundant Via Insertion for Advanced-Node Designs

To perform redundant via insertion on an advanced-node design,

- Get a recommended redundant via list from your semiconductor vendor.

  Use the `define_zrt_redundant_vias` command to specify this list before running Zroute.

- Perform redundant via insertion as a standalone task after completing initial routing and again after completing all postroute optimization runs.

**See Also**

- "Inserting Redundant Vias" section in the chip finishing chapter of the *IC Compiler Implementation User Guide*

## Filler Cell Insertion for Advanced-Node Designs

By default, filler cell insertion using the `insert_stdcell_filler` command does not consider the threshold voltage implant layer spacing rules. If your standard cell library has minimum width or minimum spacing rules for the multiple threshold voltage cells, you must enable these rules before performing filler cell insertion by setting the `legalizer_consider_vth_spacing` variable to `true`.

```
icc_shell> set_app_var legalizer_consider_vth_spacing true
```

Note:

In addition to setting this variable, the threshold voltage implant layer rules must be defined in the technology file and the implant width property must be attached to the FRAM views of the multiple threshold voltage cells. For information about these library preparation requirements, see "Defining Implant Layer Rules in the Technology File" on page A-2.

If you enable both the threshold voltage implant layer spacing rules and the no-1X rule during filler cell insertion, the reference library must contain at least 2X and 3X filler cells; otherwise, filler cell insertion fails.

During filler cell insertion, the tool writes insertion details for the implant layers to the log file.

Note:

Consideration of the threshold voltage implant layer spacing rules during filler cell insertion is not the same as using the `-vt_filler` option with the `insert_stdcell_filler` command to specify the cells used as threshold voltage fillers.

After performing filler cell insertion, use the `signoff_drc` command to run DRC checking in the IC Validator tool to verify that there are no violations of threshold voltage implant layer spacing rules.

**See Also**

- "Inserting Filler Cells" section in the chip finishing chapter of the *IC Compiler Implementation User Guide*

## Boundary Cell Insertion for Advanced-Node Designs

Before placing the standard cells, you can add boundary cells to the design. Boundary cells consist of end-cap cells, which are added to the ends of the cell rows and around the boundaries of objects such as the core area, hard macros, blockages, and voltage areas, and corner cells, which fill the empty space between horizontal and vertical end-cap cells. End-cap cells are typically nonlogic cells that serve a certain purpose such as providing a decoupling capacitor for the power rail. Because the tool accepts any standard cell as an end-cap cell, ensure that you specify suitable end-cap cells.

To insert boundary cells, use the `insert_boundary_cell` command. You can specify different library cells for the end-cap cells for the left, right, top, and bottom boundaries, as well as for each inside and outside corner cell.

- For the end-cap cells for the left and right boundaries, you specify a single library cell by using the `-left_boundary_cell` and `-right_boundary_cell` options, respectively.
- For the end-cap cells for the top and bottom boundaries, you specify a list of library cells by using the `-top_boundary_cells` and `-bottom_boundary_cells` options, respectively. The command inserts the cells in the specified order. If the remaining space is smaller than the current cell, the command inserts the next cell in order that fits in the remaining space.

  To ensure that the end-cap cells inserted on the top and bottom boundary rows comply with the maximum diffusion-to-tap distance limit, you can also insert tap cells on these boundary rows. To specify the library cell to use for the tap cells, use the `-top_tap_cell` and `-bottom_tap_cell` options, respectively, and use the `-tap_distance` option to specify the distance in microns between the tap cells.

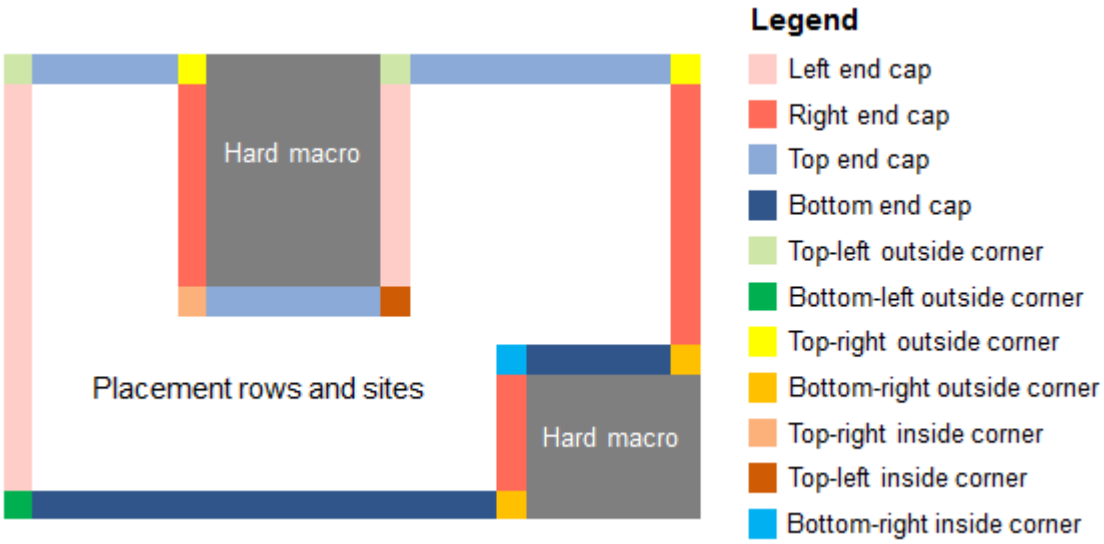
  For a vertical-row design, rows start at the bottom and end at the top, so the top boundary is along the left side of the design and the bottom boundary is along the right side of the design.

  For the flipped rows in a double-back design, the top boundary cells are used on the bottom boundaries and the bottom boundary cells are used on the top boundaries.

- For the outside corner cells, you specify a single library cell for each corner location by using the `-top_left_outside_corner_cell`, `-top_right_outside_corner_cell`, `-bottom_left_outside_corner_cell`, and `-bottom_right_outside_corner_cell` options.
- For the inside corner cells, you specify a list of library cells for each corner location by using the `-top_left_inside_corner_cells`, `-top_right_inside_corner_cells`, `-bottom_left_inside_corner_cells`, and `-bottom_right_inside_corner_cells` options. The command inserts the first corner cell that matches the size of the inside corner. If none matches exactly, it inserts the first cell that can be placed without violating any rules.

Figure 5-1 shows an example of the end-cap and corner cell locations for a horizontal-row design with two hard macros.

*Figure 5-1 Boundary Cell Locations*



By default, the command places the specified library cells in their default orientation around the core area, voltage areas, and hard macros without considering blockages or keepout margins. To change the default behavior, use the `-rules` option to specify one or more insertion rules.

- To flip the orientation of the boundary cells, specify one or more of the following rules: `mirror_left_boundary_cells`, `mirror_right_boundary_cells`, `mirror_left_outside_corner_cell`, `mirror_right_outside_corner_cell`, `mirror_left_inside_corner_cell`, and `mirror_right_inside_corner_cell`. You cannot flip the orientation of the top and bottom boundary cells.
- To prevent the command from placing boundary cells inside blockages or keepouts, specify one or more of the following rules: `respect_hard_blockage`, `respect_soft_blockage`, `respect_hard_macro_keepout`, and `respect_soft_macro_keepout`.
- To prevent boundary cell insertion when the row length equals two times the corner cell width plus one unit tile width, specify the `no_1x` rule. Note that if the row length equals two times the corner cell width, boundary cells are inserted.
- To swap the top and bottom inside corner cells on flipped rows, specify the `swap_top_bottom_inside_corner_cell` rule. When you specify this rule, the command uses the bottom inside corner cell on the top inside corner of flipped rows and the top inside corner cell on the bottom inside corner of flipped rows.

To remove end-cap cells from your design, use the `remove_stdcell_filler -end_cap` command.

# 6

# Streaming Out a Precolored Design

To learn about streaming out a GDSII or OASIS file for a precolored design, see

- Defining a Precolor Layer Mapping File
- Controlling the Layers Written for Precolored Shapes
- Propagating Mask Constraints to Via Enclosures
- Cell-Level Mask Swapping During Stream-Out

## Defining a Precolor Layer Mapping File

In a GDSII file, the mask constraints are identified by data types. Table 6-1 shows the mapping between the GDSII data types and the mask constraints.

*Table 6-1 GDSIII Data Type to Mask Constraint Mapping*

| Data type | Mask constraint |
|---|---|
| 108 | `same_mask` |
| 120 | `mask1_hard` |
| 121 | `mask2_hard` |

When you stream out a GDSII file for a precolored design, you must use the precolor layer mapping file shown in Example 6-1 to map the mask constraints to the data types.

*Example 6-1 Precolor GDSII Layer Mapping File*

```
M  * *:108
M1 * *:120
M2 * *:121
```

If you have saved the precolor layer mapping file in the Milkyway design database by using the `set_stream_layer_map_file -format out` command, the `write_stream` command uses the saved layer mapping file. Otherwise, you must specify the precolor layer mapping file by using the `set_write_stream_options -map_layer` command before streaming out the GDSII file.

Note:

The precolor layer mapping file specified by the `set_write_stream_options -map_layer` command overrides the precolor layer mapping file saved in the Milkyway design database.

## Controlling the Layers Written for Precolored Shapes

By default, the `write_stream` command writes precolored shapes to their default layer and to an extra double-patterning layer that uses the mask constraint data types defined in Table 6-1. To write the shapes only to the double-patterning layer and not to the default layer, use the `set_write_stream_options -write_multiple_patterning_layer_only` command before running the `write_stream` command.

## Propagating Mask Constraints to Via Enclosures

If the `lower_layer_double_pattern_mask_constraint` attribute for a via connected to an M1 pin shape is set to `any_mask` (the default), the tool can automatically propagate the mask constraint from the pin shape to the M1 layer of the via enclosure when streaming-out the design data in GDSII or OASIS format using the `write_stream` command. The propagated mask constraint overrides the `any_mask` attribute setting. To do this, use the `set_write_stream_options -propagate_pin_mask_to_via_metal` command before running the `write_stream` command.

## Cell-Level Mask Swapping During Stream-Out

When cell-level mask swapping is enabled, the `write_stream` command performs the following tasks for cells that have been marked for mask swapping:

1. Copies the original cell
2. Modifies the mask assignments as specified by the `mask_shift` attribute
3. Creates a new cell master that has the switched mask assignments
4. Writes out one or more instances of the new cell

The `write_stream` command creates a name for the new cell master by joining the following items with underscore characters:

- Original cell master name
- Suffix string

  By default, the `write_stream` command uses SHIFT as the suffix string. To modify the suffix string, use the `set_write_stream_options -mask_shifted_cell_name_suffix` command before running the `write_stream` command.
- Layer number of first swapped layer
- The number 1, indicating a mask swap for the layer
- Layer number of next swapped layer
- The number 1, indicating a mask swap for the layer (and so on)

For example, suppose that the name of the original cell master is AN2, the default suffix string SHIFT is being used, and the `mask_shift` attribute of an AN2 cell instance is set as follows:

```
{{M1 1} {VIA1 0} {M2 1}}
```

In that case, the swap occurs in layer M1 (layer number 15) and in layer M2 (layer number 17), so the name of the newly generated cell master is AN2_SHIFT_15_1_17_1.

The IC Compiler tool does not add the new cell master to the Milkyway database. It only streams out the new cell master to the GDSII file.

### See Also

- Using Cell-Level Mask Swapping to Fix Double-Patterning Violations

# A

# Library Preparation

To learn about library preparation steps that might be required to run place and route on advanced-node designs, see

- Enabling Cell-Level Mask Swapping
- Defining Threshold Voltage Implant Layer Rules
- Setting Double-Patterning Mask Constraints on Pins

For general information about the IC Compiler library preparation process, see the *Library Data Preparation for IC Compiler User Guide*.

For recommendations about how to design standard cell libraries to improve the routability of advanced-node designs, see SolvNet article 035309, “Application Note for Physical Library Analysis for Optimal 28-nm and 20-nm Routing.”

## Enabling Cell-Level Mask Swapping

Cell-level mask swapping is controlled by the `fixedColor` attribute in the `Technology` section of the technology file. When this attribute is set to 0 (the default), cell-level mask swapping is enabled. When this attribute is set to 1, cell-level mask swapping is disabled.

Cell-level mask swapping is also controlled by the `FIXEDMASK` keyword in a LEF file. Mask swapping is disabled for all cells in a library created from a LEF file that has the `FIXEDMASK` keyword in the header section.

```
FIXEDMASK;
LAYER statement
...
```

Mask swapping is disabled for a specific cell in a LEF file if the `MACRO` definition contains the `FIXEDMASK` keyword.

```
MACRO macroName
   CLASS className subclassName ;
   FIXEDMASK ;
   ...
```

## Defining Threshold Voltage Implant Layer Rules

If your technology has minimum width and minimum spacing rules for the implant layers of the multiple threshold voltage cells, you must ensure that these rules are defined in the technology file. In addition, you must attach the implant width property to the FRAM views of the multiple threshold voltage cells before performing physical implementation.

### Defining Implant Layer Rules in the Technology File

For each implant layer, include a `Layer` section that defines the minimum width by setting the `minWidth` attribute and the minimum spacing by setting the `minSpacing` attribute. Example A-1 shows an example of P- and N-implant layer definitions for standard and low threshold voltages.

*Example A-1 Threshold Voltage Implant Layer Rule Definitions*

```
Layer "VTS_N" {
   layerNumber = 60
   maskName = "implant"
   isDefaultLayer = 1
   minWidth = 0.45;
   minSpacing = 0.45
}
```

```
Layer “VTS_P” {
   layerNumber = 61
   maskName = "implant"
   minWidth = 0.45;
   minSpacing = 0.45
}

Layer “VTL_N” {
   layerNumber = 62
   maskName = "implant"
   minWidth = 0.45;
   minSpacing = 0.45
}

Layer "VTL_P" {
   layerNumber = 63
   maskName = "implant"
   minWidth = 0.45;
   minSpacing = 0.45
}
```

## Updating FRAM Views With Implant Width Properties

The implant width property consists of the row-by-row implant layer widths that abut the left and right sides of multiple threshold voltage cells. If the following conditions are met, blockage, pin, and via (BPV) extraction stores the implant width property in the CEL view for these cells:

- The technology file defines the implant layers, as described in the previous section, “Defining Implant Layer Rules in the Technology File."
- The LEF or GDSII file defines the implant layer geometries for the cells

Physical implementation uses the FRAM views, rather than the CEL views, to check for spacing violations. Therefore, you must extract the implant width properties from the CEL views and attach them to the FRAM views before performing physical implementation. To do this, use the `extract_fram_property` command with the `-implant_width` option set to `true`.

```
icc_shell> extract_fram_property -lib myreflib -implant_width true
```

By default, the `extract_fram_property` command extracts the implant width property for all standard cells, standard filler cells, and tap cells in the reference library. To extract the implant width property for a specific cell, use the `-cell` option.

To report the extracted implant width properties, use the `report_fram_property` command. When you run this command, you must specify the reference library with the `-lib` option and specify the cell to report with the `-cell` option. You can report the extracted information either for a single cell by specifying the cell name or for all cells by using an

asterisk (*). For example, to report the extracted information for all cells in the myreflib reference library, use the following command:

```
icc_shell> report_fram_property -lib myreflib -cell *
```

# Setting Double-Patterning Mask Constraints on Pins

To set double-patterning mask constraints on the pins (terminals and pin shapes) of hard macros, hierarchical blocks, or standard cells, use the `set_attribute` command to set the `double_pattern_mask_constraint` attribute.

For hard macro pins, you can also set the double-patterning mask constraints by streaming in a precolored GDSII file.

After setting mask constraints on the pins, you must generate a FRAM view for the standard cell or macro, which is used by the router.

## Using the set_attribute Command To Set Mask Constraints

To set mask constraints on the terminals (physical pins) of hard macros, hierarchical blocks, or standard cells, use the `set_attribute` command. For example, to set a `mask1_hard` mask constraint on the DM[2] terminal of the MYRAM hard macro, use the following command:

```
icc_shell> set_attribute [get_terminals MYRAM/DM[2]] \
   double_pattern_mask_constraint mask1_hard
```

When you use the `extract_blockage_pin_via` command to convert the CEL view to a FRAM view, the mask constraints on the terminals are stored in the generated FRAM view.

## Importing a Precolored GDSII File

If the GDSII file for a hard macro contains precoloring information, the import (stream-in) process converts that information into `double_pattern_mask_constraint` attributes on the hard macro pins in the generated CEL view.

When you use the `create_macro_fram` command to convert the CEL view to a FRAM view, the mask constraints are stored for the macro pins.

In a GDSII file, precolored pins are identified by an extra layer; the data type indicates the mask constraint. Table A-1 shows the mapping between the GDSII data types and the mask constraints.

*Table A-1 GDSIII Data Type to Mask Constraint Mapping*

| Data type | Mask constraint |
|---|---|
| 108 | `same_mask` |
| 120 | `mask1_hard` |
| 121 | `mask2_hard` |

When you stream in the precolored GDSII file, you must use the layer mapping file shown in Example A-2 (which is the same as the layer mapping file used for stream out).

*Example A-2 Precolor GDSII Layer Mapping File*

```
M  * *:108
M1 * *:120
M2 * *:121
```

If you have saved the precolor layer mapping file in the Milkyway design database by using the `set_stream_layer_map_file -format in` command, the `read_stream` command uses the saved layer mapping file. Otherwise, you must specify the precolor layer mapping file by using the `set_read_stream_options -map_layer` command before streaming in the GDSII file.

Note:

The precolor layer mapping file specified by the `set_read_stream_options -map_layer` command overrides the precolor layer mapping file saved in the Milkyway design database.