

SIEMENS EDA

Calibre® Multi-Patterning User's and Reference Manual

Software Version 2021.2
Document Revision 14

SIEMENS

Unpublished work. © 2021 Siemens

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software, Inc., its subsidiaries or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with Siemens. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This document is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made. Siemens disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' **End User License Agreement** may be viewed at: www.plm.automation.siemens.com/global/en/legal/online-terms/index.html.

No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

TRADEMARKS: The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: support.sw.siemens.com

Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Revision History

Revision	Changes	Status/ Date
14	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released April 2021
13	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released January 2021
12	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released October 2020
11	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released July 2020

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Siemens EDA documentation source. For specific topic authors, contact the Siemens Digital Industries Software documentation department.

Revision History: Released documents maintain a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation which are available on <https://support.sw.siemens.com/>.

Table of Contents

Revision History

Chapter 1		
Introduction to Calibre Multi-Patterning		15
Multi-Patterning Overview		15
Decomposition Workflows		16
Key Concepts		16
Product Requirements		22
Syntax Conventions		22
Chapter 2		
Multi-Patterning Methods		25
Purpose of Multi-Patterning.....		25
Litho-Etch-Litho-Etch (LELE) Process.....		27
Litho-Litho-Etch (LLE) Process		28
Litho-Freeze-Litho-Etch (LFLE) Process		29
Self-Aligned Double-Patterning (SADP) Process.....		30
Self-Aligned Quadruple-Patterning (SAQP) Process		32
Chapter 3		
Multi-Patterning Command Reference		35
Multi-Patterning SVRF Commands.....		37
DFM BALANCE		39
DFM CLUSTER.....		45
DFM DP		59
DFM DP MATCH		78
DFM DP OPTIMIZE		81
DFM DP OPTIMIZER_CONSTRAINTS		88
DFM MP.....		90
DFM MP SELECT CELLS		102
DFM MPSTITCH.....		108
DFM RET NMDPC EMULATION.....		113
DFM SPEC BALANCE.....		114
DFM SPEC MPSTITCH		120
INCLUDE TVF tp_model		123
INCLUDE TVF tp_separator		127
INCLUDE TVF tp_stitch_gen		130
INCLUDE TVF mp_stitch_gen		134
RET DPSTITCH.....		138
RET NMDPC		142
RET QP.....		147
RET TP		151

LITHO FILE Statement and Commands	156
LITHO FILE	157
action	160
anchor	164
collector	169
layer	170
options	171
resolve	173
seed	174
setlayer dpstitch	175
target	177
tilemicrons	178
setlayer dpstitch options Commands	180
dpFaceClass	182
dpFaceSet	186
dpMinStitchWidth	188
dpSpacing	190
dpStitchCandidates	196
dpStitchSize	203
minConvexCornerSpacing	205
intersectionStitch	207
minConcaveCornerOverlap	209
minIntersectionArmLength	211
dpStitchToViaSpacing	212
minConcaveCornerSpacing	214
version	216
Chapter 4	
Self-Aligned Double-Patterning Concepts and Command Reference	217
Self-Aligned Double-Patterning Concepts	218
Introduction to SID Cut-Fill Methodology	218
Cuts	235
Trim Mask	241
Derived Target	244
Error Handling	245
Operational Modes of Control	251
Self-Aligned Double-Patterning Command Reference	254
RET SIDCUTFILL	255
LITHO FILE	268
SADP-Specific Spacing Parameters	288
dpFaceClass	289
dpFaceSet	292
dpSpacing	294
Chapter 5	
Set Up and Run Calibre Multi-Patterning	301
calibre	302
Decomposing Whole Shapes	304

Table of Contents

Decomposing Stitched Shapes	306
Anchor Applications for Double-Patterning	309
Stitch Applications for Double-Patterning	311
Creating User-Drawn Stitches	312
Creating Pre-colored Stitches	312
Generating SVRF Custom Stitches	313
Auto-generating Stitches with DPSTITCH	314
Validation of Stitches	315
Control of Stitch Behavior with Priority and Mask Settings	320
Stitch and Priority Interaction	322
Using SVRF Variables within a LITHO FILE Statement	323
DP Debugging Methodology	324
Debugging DP Layout Errors	327
Fixing DP Layout Errors	333
MP Visualization and Error Resolution	335

Chapter 6

Example Files	341
Whole-Shape Decomposition	341
Stitch Shape Decomposition	342

Chapter 7

Calibre Multi-Patterning Best Practices	345
Edge Versus Polygon Separators	345
Guidelines for Deriving Good Separator and Action Layers	346
Tile Size Guidelines	349
Summary of Command Settings	349

Glossary

Index

Third-Party Information

Table of Contents

List of Figures

Figure 1-1. DFM DP Processing Flow	17
Figure 1-2. Whole and Stitched Decomposition Methods.....	18
Figure 1-3. Two Decomposition Flows.....	19
Figure 1-4. Whole Shape Decomposition	20
Figure 1-5. Combination Whole and Stitched Shape Decomposition.....	21
Figure 1-6. Primary Steps of Calibre Stitched Shape Decomposition.....	21
Figure 2-1. Double-Patterning Schematic	26
Figure 2-2. Triple Patterning Schematic	27
Figure 2-3. LELE Process Using Hardmask	28
Figure 2-4. Litho-Litho-Etch Process	29
Figure 2-5. Litho-Freeze-Litho-Etch Process	30
Figure 2-6. Self-Aligned Double-Patterning (SADP) Processes.....	31
Figure 2-7. SAQP Spacer and Etch Process	32
Figure 3-1. DFM BALANCE Recommended Flow	39
Figure 3-2. DFM BALANCE Stitch Solutions	42
Figure 3-3. Connected Component Model	47
Figure 3-4. Correct DFM CLUSTER Separators	48
Figure 3-5. SELECT_CELLS Usage.....	56
Figure 3-6. Self-Conflict With Opposite Separator Edge Pairs.....	66
Figure 3-7. Self-Conflict With Opposite and Same Separator Edges	66
Figure 3-8. SHORTEST_ANCHOR_LINE	67
Figure 3-9. Correct DFM DP Separators.....	72
Figure 3-10. Good and Bad Anchors.....	73
Figure 3-11. Good and Bad User Stitches	74
Figure 3-12. SELF_CONFLICT Error	76
Figure 3-13. SHORTEST_LOOP_LINE Output.....	77
Figure 3-14. Original Design.....	84
Figure 3-15. Design as Processed by DFM DP.....	85
Figure 3-16. Final Optimized Design After Applied Constraints	87
Figure 3-17. Correct DFM MP Separators	97
Figure 3-18. Good and Bad Anchors	98
Figure 3-19. Good and Bad Multi-Patterning User Stitches	100
Figure 3-20. BY CELL Output	105
Figure 3-21. BY LAYER Output.....	106
Figure 3-22. EXCLUDE_TOP Usage	107
Figure 3-23. Correct DFM DP Separators	109
Figure 3-24. Good and Bad Anchors	110
Figure 3-25. DFM BALANCE Use of OVERLAY	115
Figure 3-26. ring Warning Exclusion Scenario	145
Figure 3-27. ring_tie Appearance	145

Figure 3-28. self_conflict Error Layer	146
Figure 3-29. Crossing Separator Filtering	146
Figure 3-30. RET NMDPC LITHO FILE and Command	158
Figure 3-31. RET DPSTITCH LITHO FILE and Command	158
Figure 3-32. Generated Stitches and Separator Edge Pairs	161
Figure 3-33. User-Defined Stitch Candidate Example	162
Figure 3-34. Anchor Applications for Triple-Patterning	166
Figure 3-35. Anchor Applications for Quadruple-Patterning	168
Figure 3-36. Resolve Behavior	173
Figure 3-37. Face Class Constraint Keywords	183
Figure 3-38. Face Classes and Jogs	184
Figure 3-39. Use of maxAdjacentWidth With dpFaceClass	185
Figure 3-40. runLength and Conjunction	194
Figure 3-41. otherwise and minLength Usage	195
Figure 3-42. Using anchorx	197
Figure 3-43. dpStitchMode cycleAware Behavior	197
Figure 3-44. dpStitchMode targetNodePairs Behavior	198
Figure 3-45. Determination of Jogs	199
Figure 3-46. True (Default) Setting	199
Figure 3-47. False Setting	200
Figure 3-48. Polygon to Edge Conversion	202
Figure 3-49. dpStitchSize	203
Figure 3-50. minConvexCornerSpacing Usage	206
Figure 3-51. minConvexCornerSpacing with strict Option	206
Figure 3-52. minConcaveCornerOverlap Usage	210
Figure 3-53. minConcaveCornerOverlap with strict Argument	210
Figure 3-54. Basic dpStitchToViaSpacing Usage	213
Figure 3-55. dpStitchToViaSpacing with Two Via Enclosures	213
Figure 3-56. dpStitchToViaSpacing with a Bounded Via Enclosure	213
Figure 3-57. minConcaveCornerSpacing Usage	215
Figure 3-58. minConcaveCornerSpacing with strict Option	215
Figure 4-1. RET SIDCUTFILL Flow	219
Figure 4-2. Pattern Transfer Overview	220
Figure 4-3. Target Shapes Within a Boundary	221
Figure 4-4. Separate sadpMarker Layers	222
Figure 4-5. Mandrel and Non-mandrel Tracks	224
Figure 4-6. Track Layers	225
Figure 4-7. Generated Output and Error Layers	226
Figure 4-8. maxCutLength and the illegalCut Layer	227
Figure 4-9. Track Coloring	228
Figure 4-10. The cutMask Layer and sadpMarker Boundary Interaction	228
Figure 4-11. cutMask Shape Spacing Requirements	229
Figure 4-12. sadpMarker Containment	230
Figure 4-13. illegalTarget Error Layer	231
Figure 4-14. Insufficient Space for Terminal Mandrels	231

List of Figures

Figure 4-15. Filler Tracks	232
Figure 4-16. Odd-Track Spans.....	232
Figure 4-17. 2D Target Shape Placement	233
Figure 4-18. Jogs and Pads	233
Figure 4-19. Final Pattern with Jogs	234
Figure 4-20. trackTrim Parameter	234
Figure 4-21. Sliding Cut Implementation	235
Figure 4-22. Behavior of Sliding Cuts False	235
Figure 4-23. A User-Cut Implementation	236
Figure 4-24. Cuts in 2D Structures	238
Figure 4-25. Cuts Automatically Placed to Resolve Shorts.....	239
Figure 4-26. Pitch Alignment for Cuts	240
Figure 4-27. Trim Mask Implementation	242
Figure 4-28. Target- and Track-Related Spacing Errors	246
Figure 4-29. illegalTrackWidth Error	247
Figure 4-30. Poorly Placed Anchors Result in Anchor Conflict Error	247
Figure 4-31. Self-Conflict Anchor Error and Resulting Track Assignment	248
Figure 4-32. Spacing Errors.....	249
Figure 4-33. Two-Neighbor Spacing Criteria	250
Figure 4-34. Two-Neighbor Spacing Errors	250
Figure 4-35. Three-Neighbor Spacing Criteria	251
Figure 4-36. Three-Neighbor Spacing Errors	251
Figure 4-37. criticalTimingMarkers	257
Figure 4-38. trappedCuts Layer.....	262
Figure 4-39. maxDummyFillLength	279
Figure 4-40. Pitch Alignment Track Parameters	282
Figure 4-41. slidingJogCuts.....	283
Figure 4-42. terminationWidthSequence	284
Figure 4-43. trackWidthSequence	285
Figure 4-44. Trim Mask Artifacts and Parameters	286
Figure 4-45. RET SIDCUTFILL LITHO FILE and Commands.....	286
Figure 4-46. numNeighbors Condition	299
Figure 5-1. Seed Anchors	309
Figure 5-2. Identical Anchors	309
Figure 5-3. Floating Anchors.....	310
Figure 5-4. Multiple Anchors	310
Figure 5-5. Overlapping Anchors	310
Figure 5-6. User-defined Stitch Layer	316
Figure 5-7. Stitch Layer Must be Polygonal Rectangles	316
Figure 5-8. Stitch Shapes Removed from Target	317
Figure 5-9. Stitch Shapes Overlapping Separators	318
Figure 5-10. Design-based Stitch Placement Resolution.....	319
Figure 5-11. Stitches Must be DRC-Clean	320
Figure 5-12. Stitch Corner Rule Enforcement.....	320
Figure 5-13. Minimizing and Controlling Use of Stitches	321

Figure 5-14. Enforcing use of Stitches	321
Figure 5-15. Stitch and Priority Interactions	322
Figure 5-16. Spacing Checks for Opposite Mask Errors.....	325
Figure 5-17. Self-Conflict Shape Errors	325
Figure 5-18. Anchor-based Self-Conflict Errors	326
Figure 5-19. Ring and Warning Errors Highlighting Odd-Cycle Errors	326
Figure 5-20. Output Layout	328
Figure 5-21. Starting RVE	328
Figure 5-22. RVE Form	329
Figure 5-23. Selecting an Action Separator	329
Figure 5-24. Action Separators	330
Figure 5-25. Selecting a Loop Error	330
Figure 5-26. Loop Errors	331
Figure 5-27. Selecting an Error Type	331
Figure 5-28. Conflict Ring Errors	332
Figure 5-29. Selecting Ring and Warning Error Types.....	333
Figure 5-30. Conflict and Warning Ring Errors	333
Figure 5-31. First Attempted Edit and Error Results.....	334
Figure 5-32. Second Attempted Edit and Error Resolution.....	335
Figure 5-33. Anchor Path Examples	336
Figure 5-34. Self-Contained Violations.....	337
Figure 5-35. Self-Contained “Wheel” Violations	338
Figure 5-36. Equivalence Loop	339
Figure 5-37. Equivalence Loop Polygons	339
Figure 5-38. Equivalence Reduction Map	340
Figure 6-1. Whole-Shape Decomposition	341
Figure 6-2. Stitched Shape Decomposition	342
Figure 7-1. False Violations from Merged Separators	345
Figure 7-2. Converting Separators from Polygons to Edge Pairs	346
Figure 7-3. Singularity Results	347
Figure 7-4. Polygon Interaction with Less Than Two Edges	348
Figure 7-5. Polygon Interaction with Singularity	348

List of Tables

Table 1-1. Syntax Conventions	22
Table 3-1. Multi-Patterning SVRF Commands	37
Table 3-2. DFM BALANCE Layer Interaction	41
Table 3-3. Expressions	82
Table 3-4. Unary ~ and ! Operators	83
Table 3-5. Allowed Overlap of Different Layers	118
Table 3-6. LITHO FILE Statement and RET-Based Commands	156
Table 3-7. layer Command Layer Types	170
Table 3-8. Hardware Requirements and tilemicrons Settings	179
Table 3-9. setlayer dpstitch options Commands	180
Table 4-1. Cut Length Control Reference	251
Table 4-2. Cut Placement Control Reference	252
Table 4-3. Track Generation Control Reference	253
Table 4-4. SADP LITHO FILE Parameter Summary	269
Table 4-5. SADP-Specific LITHO FILE Statement and Parameters	288
Table 5-1. Command Compatibility of Processing Modes	303
Table 5-2. Stitch Methods and Validation	311
Table 7-1. Command Support of Singularity	347
Table 7-2. Command Behavior with Singularities	348
Table 7-3. Hardware Requirements and tilemicrons Settings	349
Table 7-4. Recommended Settings Summary	349

Chapter 1

Introduction to Calibre Multi-Patterning

Calibre® Multi-Patterning products facilitate fabrication of designs below the resolution limit of exposure wavelength by splitting a design into two or more masks that can independently be imaged in the scanner.

This chapter contains sections explaining how Calibre processes multi-patterning designs.

Multi-Patterning Overview	15
Decomposition Workflows	16
Key Concepts	16
Product Requirements	22
Syntax Conventions	22

Multi-Patterning Overview

Calibre Multi-Patterning decomposes layers into separate masks. Some layout topologies are not decomposable without adjustments to shapes by splitting them into sections and saving these sections to different masks (stitching).

Calibre DFM DP

Calibre® DFM DP decomposes the input shapes of a single layer into two mask layers with or without stitching the original shapes. Decomposing shapes without stitching avoids potential contour pinching and associated overlay problems. Conversely, using stitches can provide more flexibility achieving contour and target fidelity. Whole shape decomposition is typically used for double-patterning of contacts, vias, and conductors with regular spatial frequencies.

DFM BALANCE

Calibre® Multi-Patterning can post-process decomposed data for optimization of color assignments for density balancing across a design. Calibre DFM BALANCE is area- and window-based and supports these color mutations:

- Arbitrary Colorable Polygon (ACP) color mutation
- Biconnected Component (BCC) color mutation

Calibre DFM BALANCE provides the flexibility necessary to be applied at various points in your multi-patterning flow, execution performed after coloring or after fill insertion.

Decomposition Workflows

Calibre Multi-Patterning decomposition is performed using colored or uncolored flows, typically by design houses and foundries, respectively.

The colored flow is generally defined as designer-performed decomposition and uncolored as foundry-performed decomposition. In either case, Calibre Multi-Patterning processes are typically performed before OPC, LVS, and mask data preparation.

Although the details are different depending on whether you allow shapes to be stitched or not, in general the method is as follows:

1. Create a setup file. The setup file identifies layers and configures the settings.
2. Resolve conflicts. Provide settings that resolve most conflicts. This includes controlling priorities of action and anchor layers and stitching.
3. Decompose the input layer into two masks. Any conflicts that cannot be resolved are flagged in a separate layer.
4. Write the layers to a results database.

Related Topics

[Decomposing Whole Shapes](#)

[Decomposing Stitched Shapes](#)

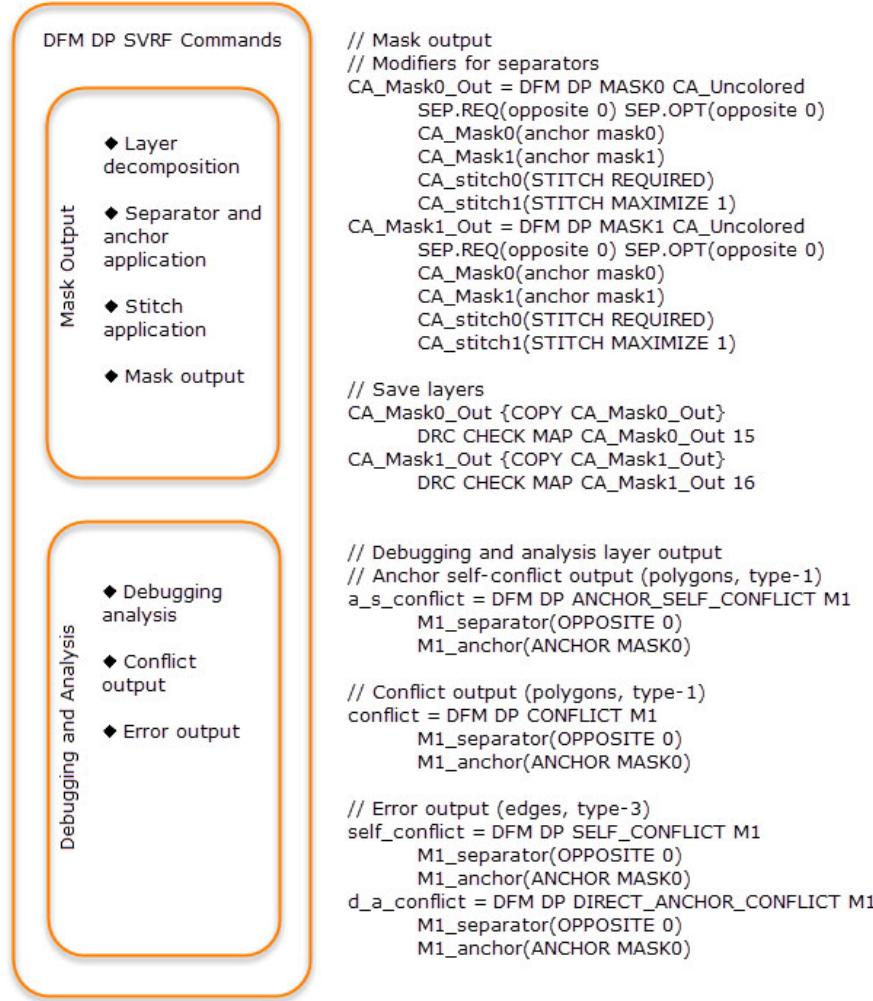
Key Concepts

Calibre Multi-Patterning decomposes designs by whole shape or stitching methods.

Calibre Double-Patterning Command Structure

The DFM DP command structure consists of two sections, best demonstrated through a functional block diagram ([Figure 1-1](#)). The abbreviated code snippets are represented on the right:

Figure 1-1. DFM DP Processing Flow



Decomposition Alternatives

Calibre performs double-patterning using two decomposition methods:

- [Whole Shape Decomposition](#) segregates whole shapes from the input layer into two masks or manages conductor layer stitching with stitch candidates. Calibre DFM DP can be used to decompose via and conductor layers by any foundry that does not permit stitching.

- **Stitched Shape Decomposition** using Calibre DFM DP identifies stitch areas and cuts shapes that would create conflicts during decomposition, then segregates both whole and cut shapes into two masks from the original layer.

Figure 1-2 depicts the differences between these two decomposition methods.

Figure 1-2. Whole and Stitched Decomposition Methods

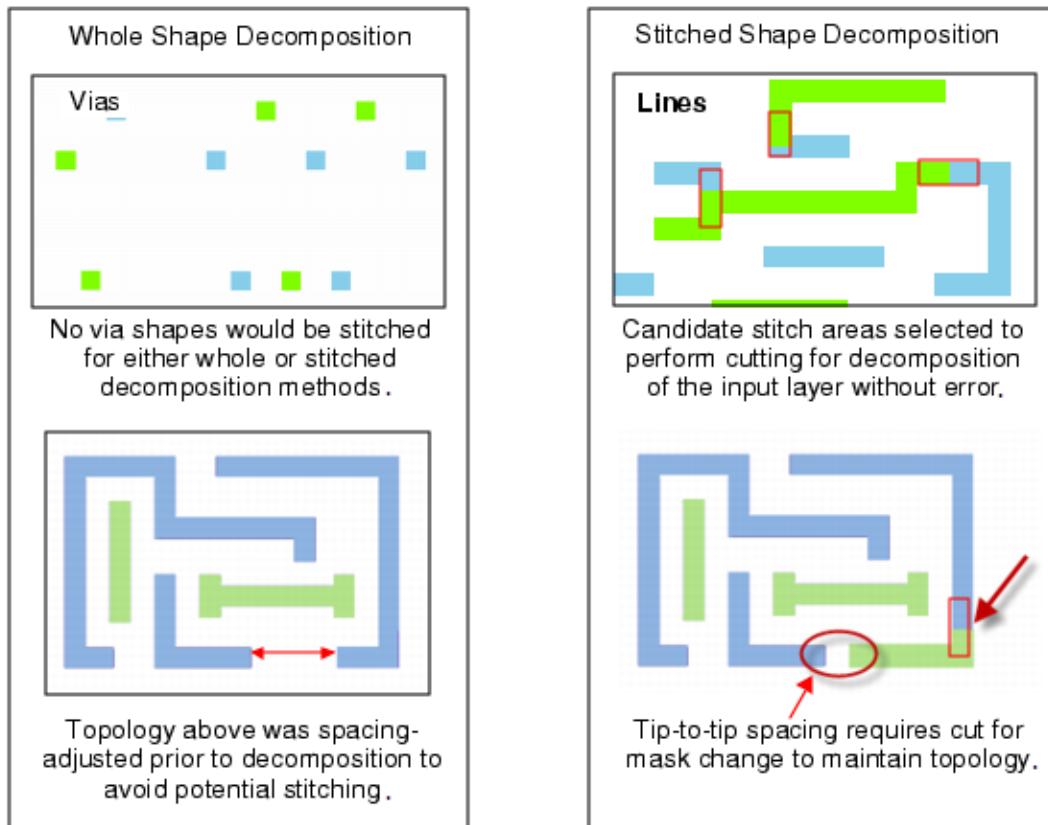
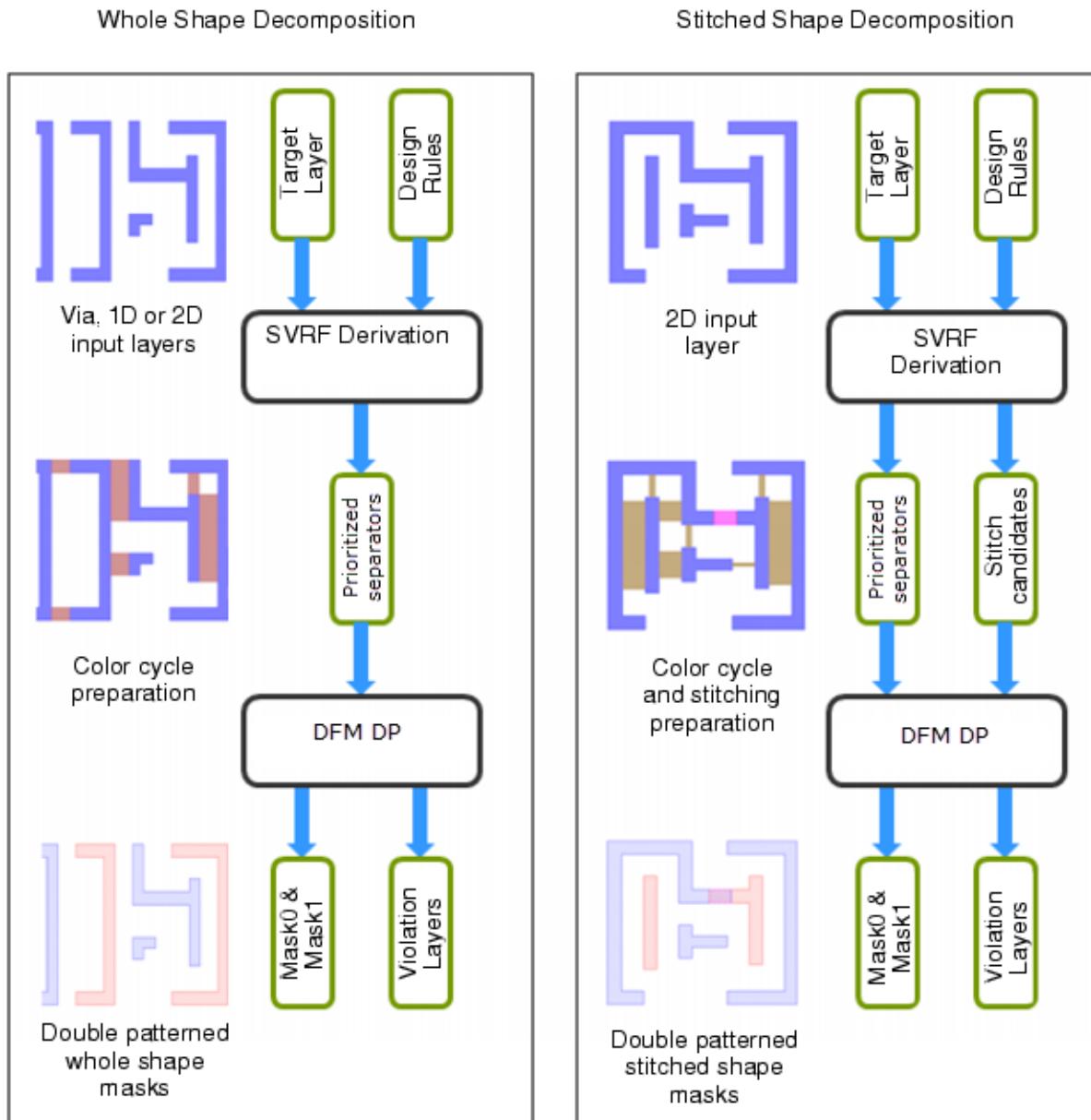


Figure 1-3 demonstrates whole shape and stitched shape decomposition flows.

Figure 1-3. Two Decomposition Flows



Whole Shape Decomposition

Whole shape decomposition can be performed using Calibre DFM DP. The tool decomposes input shapes of a single layer into two separate masks for subsequent exposure. This method of layer decomposition is typically used for double-patterning of contacts, vias, and conductors with regular spatial frequencies, for example, metal shapes with consistent pitch.

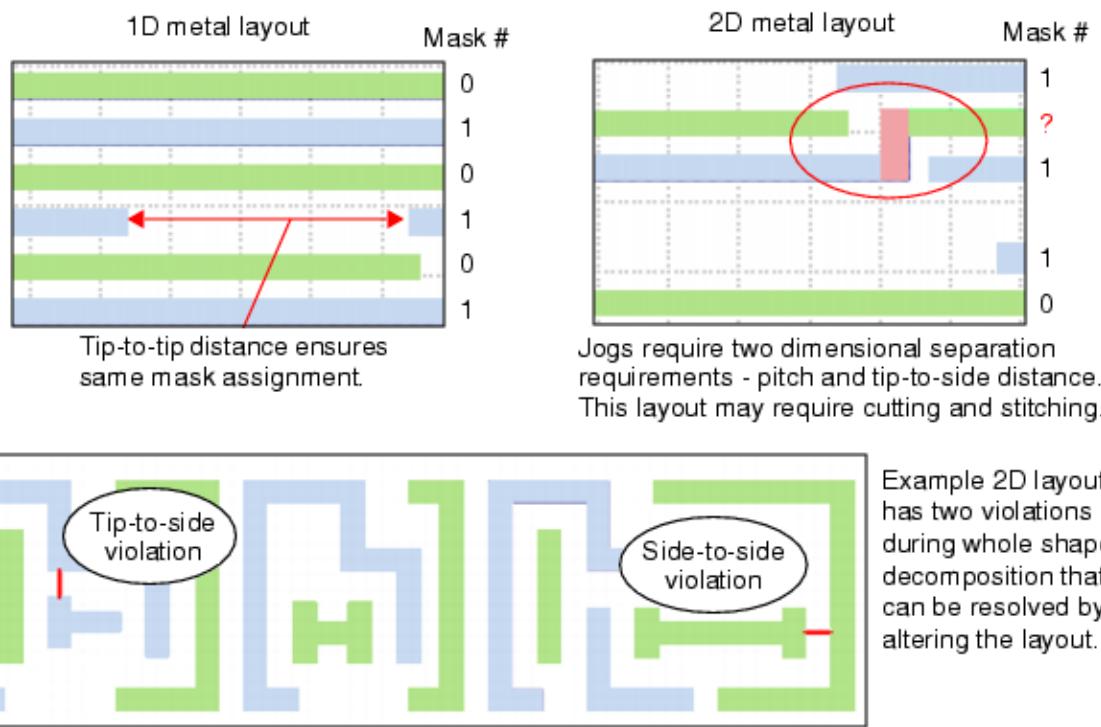
Decomposition of a layer can fail for a number of reasons but layers optimized for whole shape decomposition are easily segregated. Although critical layer conductors are drawn with greater

variation than contacts, when they are drawn with line pitch regularity, few or no line jogs, and optimized tip-to-feature spacing, they are good candidates for whole shape decomposition.

The best candidates are referred to as 1D patterns, or unidirectional patterns. They consist of nearly all lines running in the same direction. 2D or bidirectional patterns are also decomposed with additional separation constraints to avoid decomposition failure. 2D patterns contain lines routed in both X and Y axes.

[Figure 1-4](#) depicts 1D and 2D design facility and impediments, respectively, to whole shape decomposition.

Figure 1-4. Whole Shape Decomposition



To perform whole shape decomposition, you set up rules using action layers. Action layers contain shapes that abut two (and only two) target polygons. The shapes can be derived with SVRF prior to calling Calibre DFM DP.

The rules for action layers indicate whether the target shapes are assigned to the same mask or different masks in the output.

Stitched Shape Decomposition

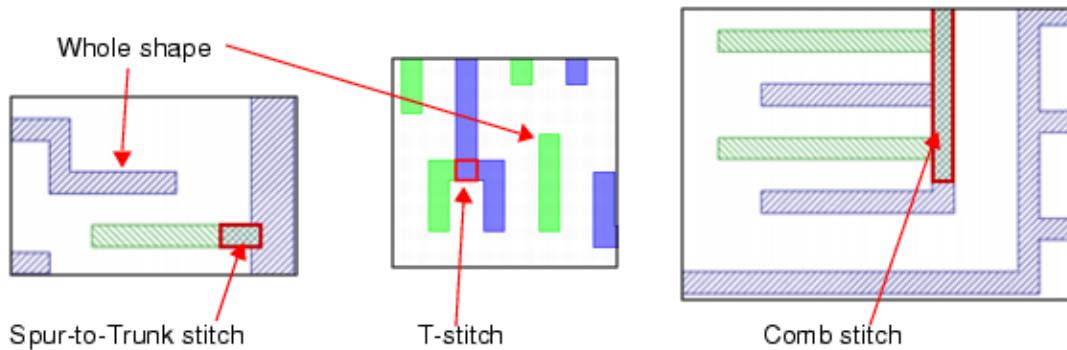
Stitched shape decomposition comprises two operations: stitch isolation and decomposition. The combination of these two functions adapts double-patterning to shapes forming bidirectional (2D) patterns and irregular spatial frequencies.

Most shapes during decomposition are retained as single shapes. Shape cutting and subsequent stitching is only used if necessary to avoid coloring conflicts.

If stitching is employed, DFM DP provides overlap regions where line cuts are made to separate portions of the input layer and place them on different masks. Both layer images are exposed separately and then etched to provide conductor continuity. Stitches provide decomposition alternatives keeping litho results within the process window without violations.

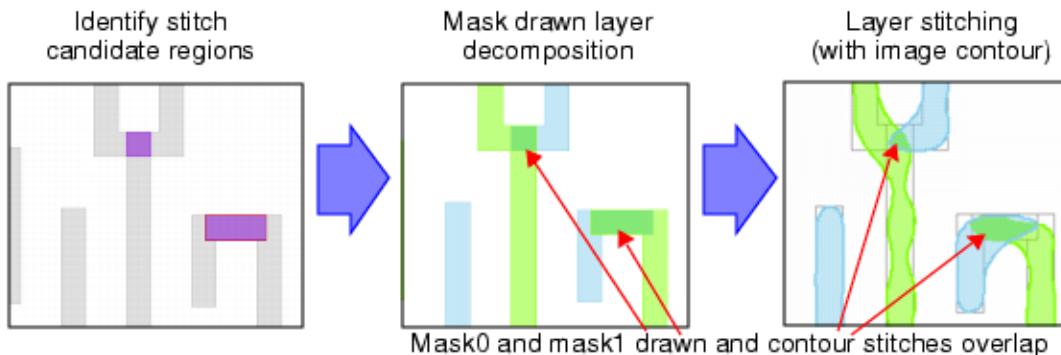
[Figure 1-5](#) depicts whole and stitched shape decomposition methods for a 2D design.

Figure 1-5. Combination Whole and Stitched Shape Decomposition



[Figure 1-6](#) summarizes the steps used to perform cutting and stitching with DFM DP.

Figure 1-6. Primary Steps of Calibre Stitched Shape Decomposition



Related Topics

[dpFaceClass](#)

[dpStitchCandidates](#)

Product Requirements

Familiarity with SVRF and TVF programming is recommended, but not required, for optimal use of Calibre Multi-Patterning.

To use the Calibre Multi-Patterning tools, you need the following:

- Calibre version 2014.1 or higher. For licensing information, refer to the *Calibre Administrator's Guide*. See the syntax and invocation for “`calibre`” on page 302.
- The original design layout in GDS or OASIS®¹ format.
- Specifications of edge classifications, their allowed spacing, and if applicable, allowed stiches. Edge classifications are defined within the LITHO FILE section of the SVRF rules.
- An SVRF file containing the verification rules (MRC checks) that are applied to your designs.

Syntax Conventions

The command descriptions use font properties and several metacharacters to document the command syntax.

Table 1-1. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	Underlining indicates either the default argument or the default value of an argument.
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.

1. OASIS® is a registered trademark of Thomas Grebinski and licensed for use to SEMI®, San Jose. SEMI® is a registered trademark of Semiconductor Equipment and Materials International.

Table 1-1. Syntax Conventions (cont.)

“ ”	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
or	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
...	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.

Example:

```
DEvice {element_name [‘(‘model_name‘)’]}
    device_layer {pin_layer [‘(‘pin_name‘)’] ...}
        [‘<‘auxiliary_layer‘> ...]
        [‘(‘swap_list‘) ...]
    [BY NET | BY SHAPE]
```


Chapter 2

Multi-Patterning Methods

The challenges of fabricating integrated circuits at sub-32 nm process nodes has necessitated several Multi-Patterning processing methods.

The methods described here are not foundry-specific.

Purpose of Multi-Patterning.....	25
Litho-Etch-Litho-Etch (LELE) Process.....	27
Litho-Litho-Etch (LLE) Process.....	28
Litho-Freeze-Litho-Etch (LFLE) Process	29
Self-Aligned Double-Patterning (SADP) Process	30
Self-Aligned Quadruple-Patterning (SAQP) Process.....	32

Purpose of Multi-Patterning

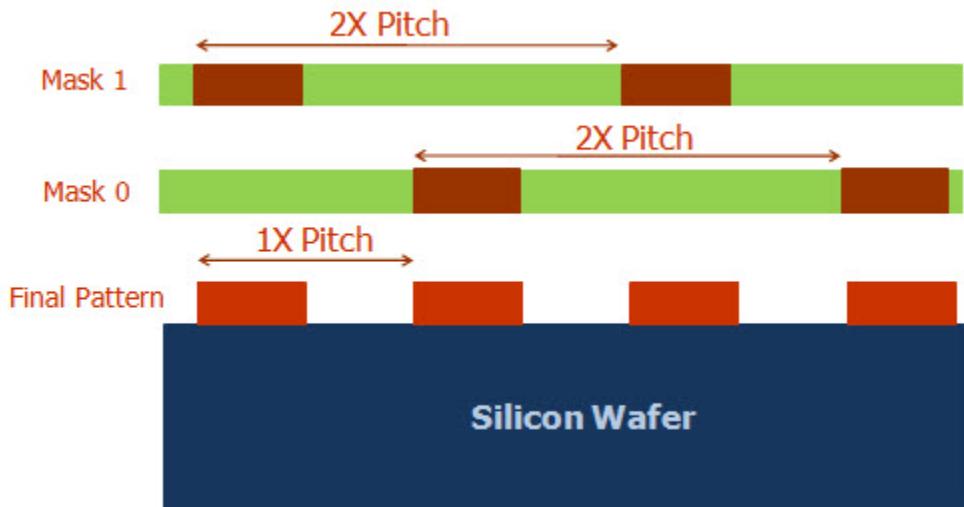
Multi-patterning is a resolution enhancement technique (RET) that decomposes original drawn shapes of a layout into two or more separate masks to be exposed independently.

One of several multiple exposure techniques is used to expose two or more separate patterns onto the same photoresist and then develop the photoresist, resolving the individually exposed patterns as one. This process increases the achievable process window, helps to resolves smaller feature density, and increases die yield across the wafer.

Double-Patterning

Double-patterning (DP) is an RET technique that enables lithographic printing below 20 nm. DP is based on pitch splitting, which requires decomposing the original shapes into two separate masks each with twice the designed pitch. The pitch is doubled so that half of the lines are printed on one mask and the other half on the second mask. Doubling the masks roughly doubles the processing steps necessary to complete the composite pattern. [Figure 2-1](#) is a schematic of the masks used relative to the final pattern intended.

Figure 2-1. Double-Patterning Schematic

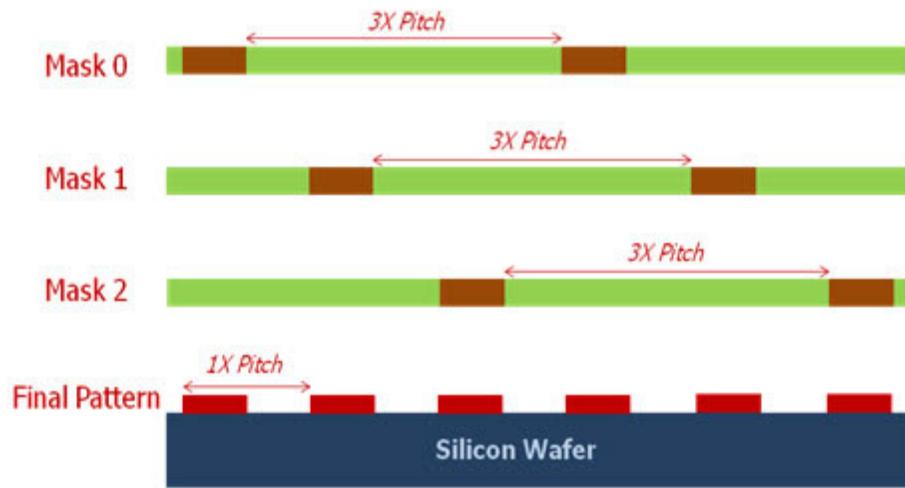


Typically, pitch splitting is performed in the fabrication facility by one of several processing techniques, described in the following sections:

- “[Litho-Etch-Litho-Etch \(LELE\) Process](#)” on page 27
- “[Litho-Litho-Etch \(LLE\) Process](#)” on page 28
- “[Litho-Freeze-Litho-Etch \(LFLE\) Process](#)” on page 29
- “[Self-Aligned Double-Patterning \(SADP\) Process](#)” on page 30

Triple- and Quadruple-Patterning

At 14 nm and below, double-patterning may not provide enough image fidelity. Triple-patterning (TP) and quadruple-patterning (QP) may be enlisted to handle increased spatial frequencies. Triple patterning uses three masks with three times the pitch; quadruple-patterning uses four masks. Because of the extra process steps, triple- and quadruple-patterning cost more and take longer to produce wafers. Triple- and quadruple patterning processes are similar to the [Litho-Etch-Litho-Etch \(LELE\) Process](#) with additional “LE” stages repeated. [Figure 2-2](#) is a schematic of the masks used, relative to the final pattern, for triple-patterning.

Figure 2-2. Triple Patterning Schematic

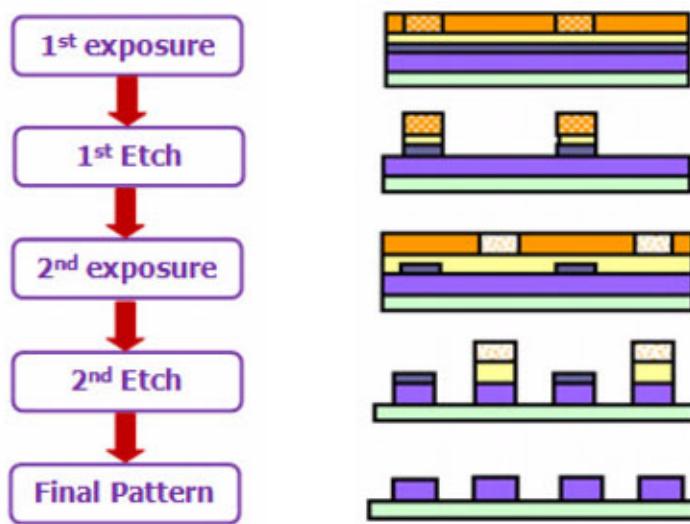
Litho-Etch-Litho-Etch (LELE) Process

LELE is the earliest double-patterning process.

After the first exposure, the resist is developed and the image is used to etch the pattern into a hardmask stack. The wafer is then recoated with photoresist and the second pattern is exposed. The resist is used as the mask for the second exposure, and the hardmask is used to transfer the pattern for the first exposure.

LELE, LLE, and LFLE processes are well suited to 2D design applications because of their ability to handle greater design variation. LELE generally takes more time, reducing throughput and costing more to process, than LLE or LFLE. [Figure 2-3](#) describes the LELE process.

Figure 2-3. LELE Process Using Hardmask



Related Topics

- [Litho-Litho-Etch \(LLE\) Process](#)
- [Litho-Freeze-Litho-Etch \(LFLE\) Process](#)
- [Self-Aligned Double-Patterning \(SADP\) Process](#)

Litho-Litho-Etch (LLE) Process

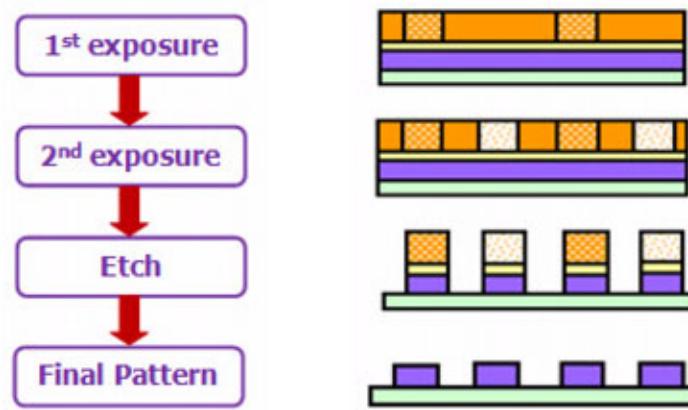
The LLE process consists of two exposures and a single development and etch.

The two exposures are performed sequentially using two masks that must be exchanged between exposures. After the two masks are exposed on the same resist layer, the resist is then developed and etched once, resolving the final pattern.

LLE faces the singular problem of distortion of the first exposed resist. This is due to the second optical exposure, thermal wear and solvent exposure. The resist sags, becomes misaligned, and falls out of critical dimension (CD).

Though less expensive and faster to manufacture than the other double-patterning processes, failure of the first resist requires extremely tight control to enable LLE as a viable process. LFLE addresses these issues. [Figure 2-4](#) depicts the fundamental aspects of the LLE process.

Figure 2-4. Litho-Litho-Etch Process



Related Topics

[Litho-Etch-Litho-Etch \(LELE\) Process](#)

[Litho-Freeze-Litho-Etch \(LFLE\) Process](#)

[Self-Aligned Double-Patterning \(SADP\) Process](#)

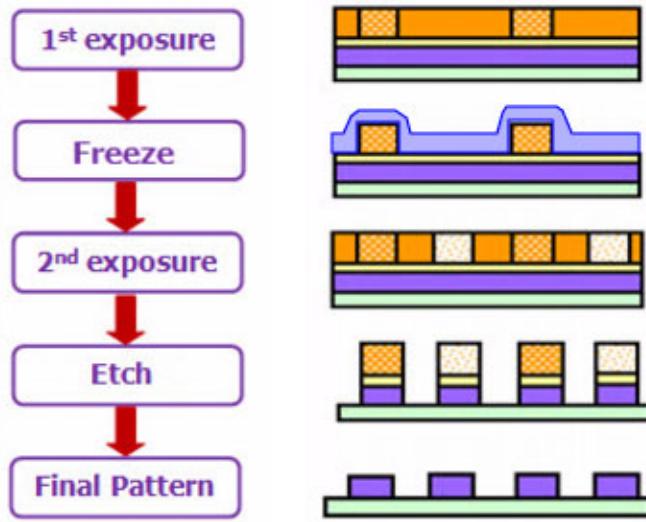
Litho-Freeze-Litho-Etch (LFLE) Process

The LFLE process is similar to LLE with the addition of a resist-freezing step.

LFLE consists of two exposures and a single development and etch. However, LFLE has an intermediate step between exposures that freezes, or stabilizes, the resist to withstand the debilitating effects of the second exposure and the following etch. The LFLE process provides better CD control than LLE and helps mitigate line-width roughness (LWR).

LFLE is less expensive than LELE and provides greater throughput in the fabrication facility. Figure 2-5 shows the added step of freezing the first exposed resist for maintained stability through the following process steps.

Figure 2-5. Litho-Freeze-Litho-Etch Process



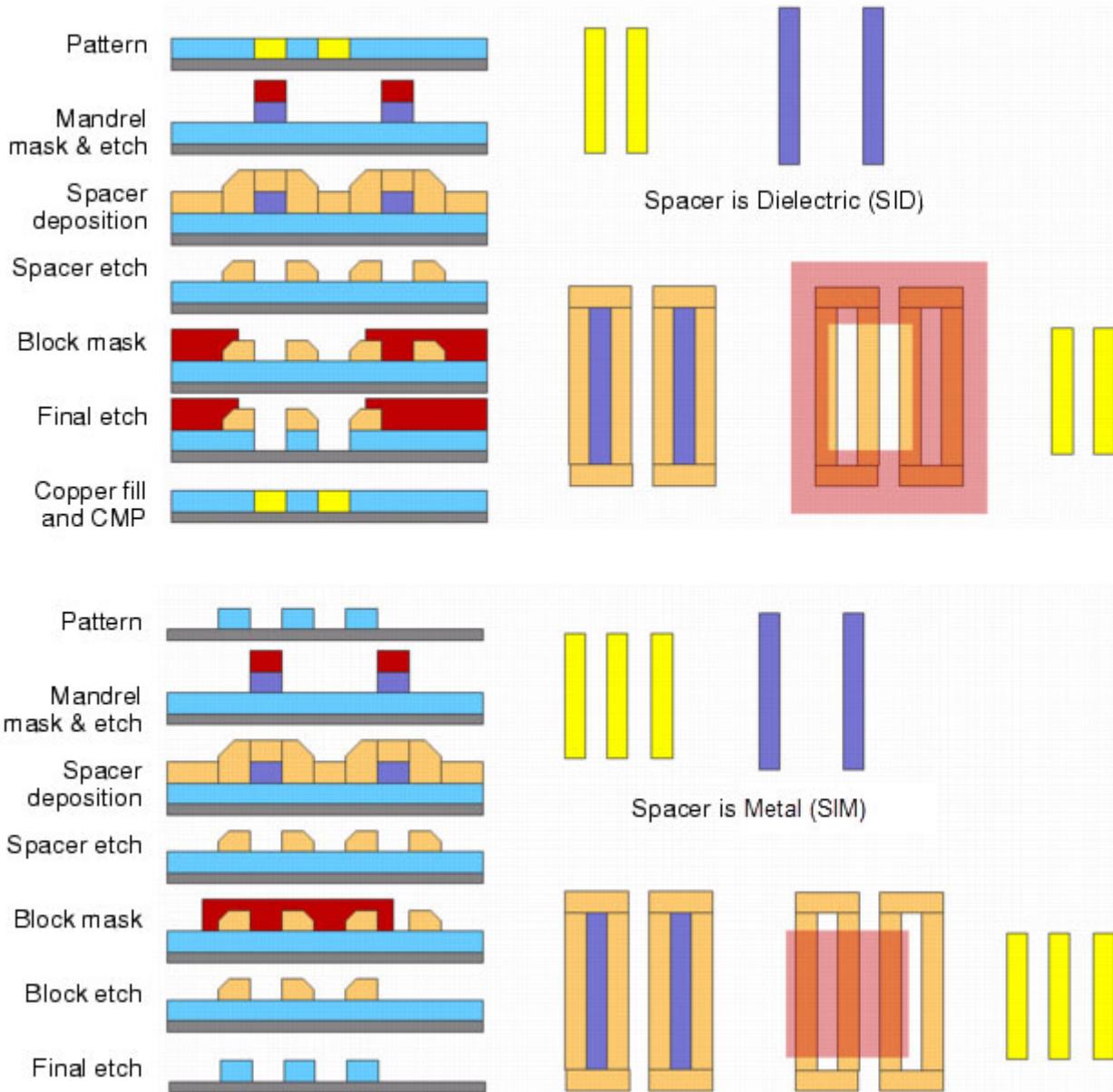
Related Topics

- [Litho-Etch-Litho-Etch \(LELE\) Process](#)
- [Litho-Litho-Etch \(LLE\) Process](#)
- [Self-Aligned Double-Patterning \(SADP\) Process](#)

Self-Aligned Double-Patterning (SADP) Process

SADP uses deposited spacers around mandrel shapes to print sub-resolution features. SADP is proficient in rendering 1D designs. SADP is not susceptible to overlay errors, because it uses only one exposure. Two forms of SADP are currently used: spacer-is-dielectric (SID) and spacer-is-metal (SIM). [Figure 2-6](#) shows the two processes.

Figure 2-6. Self-Aligned Double-Patterning (SADP) Processes



The first and second steps for SADP follow LELE, where the first exposure is performed followed by an etch. Next, standard materials used for sidewall spacers are deposited, followed by an anisotropic etch that removes all the film material on planar surfaces. Only the spacer material on the sidewalls remains.

For SID, after the sidewall etch-back, the gaps between spacers are filled and surfaced using chemical mechanical polishing (CMP). The spacers are etched, the fill is planarized, and the patterns remaining are the interconnect features.

For SIM, the original feature patterned by the first exposure (referred to as a mandrel) is removed by another etch step, leaving only the spacers. Because there are two spacers for every mandrel, the line density has now doubled. For SIM, The spacer defines the pattern. The sidewall spacer represents the final metal location.

Related Topics

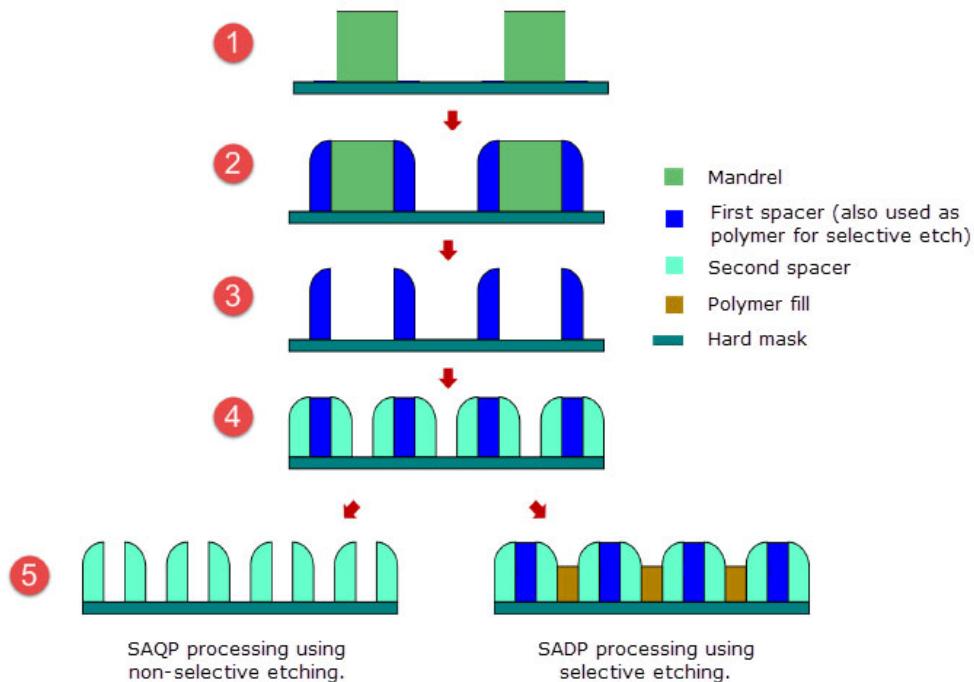
- [Litho-Etch-Litho-Etch \(LELE\) Process](#)
- [Litho-Litho-Etch \(LLE\) Process](#)
- [Litho-Freeze-Litho-Etch \(LFLE\) Process](#)

Self-Aligned Quadruple-Patterning (SAQP) Process

SAQP uses two-stage deposited spacers around mandrel shapes to print sub-resolution features.

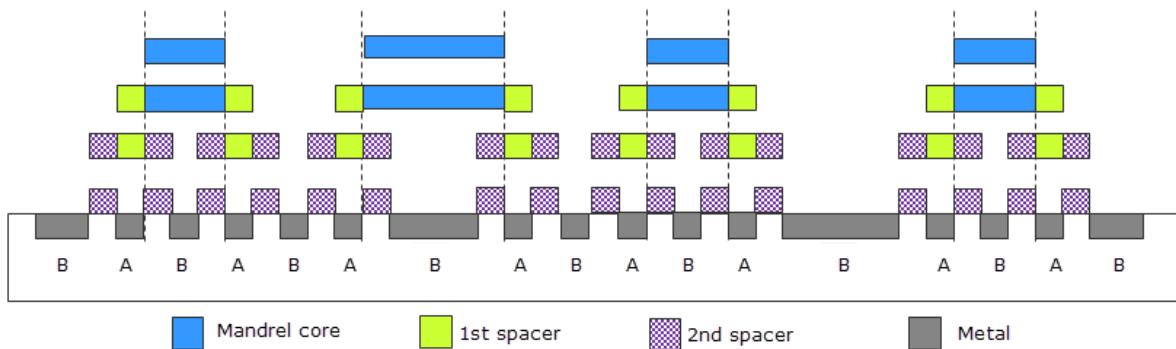
SAQP is proficient in rendering 1D designs. SAQP is not susceptible to overlay errors, because it uses only one exposure. Throughout this topic, the term mandrel core refers to the original mandrel generated by tracks implied by the location of target shapes. For the SAQP process, a secondary mandrel is formed (differentiated from the mandrel core) by the second spacers that are formed.

Figure 2-7. SAQP Spacer and Etch Process



1. The mandrel core is first printed.
2. The first spacer is then grown on both mandrel core sides.
3. The mandrel core is etched away, leaving the first spacers.
4. The second spacer is grown on both sides of the first spacer.
5. At this point, the process provides an option to proceed with either SAQP-based non-selective etching or SADP-targeted selective etching:
 - **SAQP with non-selective etching** —The first spacer is removed resulting in quadruple-pitch splitting tracks.
 - **SADP with selective etching** — The first spacer is washed away and filled with polymer (blue). Remaining gaps are also filled with polymer (brown). Subsequent mandrel cuts are applied to blue polymer tracks only, and non-mandrel cuts to brown polymer tracks only.

The following illustration examines the SAQP deposition process that produces quad-patterning. Theoretically, this approach could be extended to any patterning level.



For SAQP, the mandrel core may be expressed as:

$$\text{mandrel core} = B + (2 * \text{2nd spacer width})$$

The mandrel core, which is drawn, is differentiated from the resulting mandrel generated by the first spacer in the SAQP process. The metal terms require explanation:

- **A** — The width of the first sidewalls (spacers) grown. This width remains constant.
- **B** — The resulting center of the mandrel or the center of the space between mandrels. This width is equal to or larger than A, and has no upper width restriction.

Chapter 3

Multi-Patterning Command Reference

SVRF-based Calibre Multi-Patterning commands adhere to SVRF standards.

Multi-Patterning SVRF Commands	37
DFM BALANCE	39
DFM CLUSTER	45
DFM DP	59
DFM DP MATCH	78
DFM DP OPTIMIZE	81
DFM DP OPTIMIZER_CONSTRAINTS	88
DFM MP	90
DFM MP SELECT CELLS	102
DFM MPSTITCH	108
DFM RET NMDPC EMULATION	113
DFM SPEC BALANCE	114
DFM SPEC MPSTITCH	120
INCLUDE TVF tp_model	123
INCLUDE TVF tp_separator	127
INCLUDE TVF tp_stitch_gen	130
INCLUDE TVF mp_stitch_gen	134
RET DPSTITCH	138
RET NMDPC	142
RET QP	147
RET TP	151
LITHO FILE Statement and Commands	156
LITHO FILE	157
action	160
anchor	164
collector	169
layer	170
options	171
resolve	173
seed	174
setlayer dpstitch	175
target	177
tilemicrons	178
setlayer dpstitch options Commands	180
dpFaceClass	182
dpFaceSet	186
dpMinStitchWidth	188

dpSpacing	190
dpStitchCandidates	196
dpStitchSize	203
minConvexCornerSpacing	205
intersectionStitch	207
minConcaveCornerOverlap	209
minIntersectionArmLength	211
dpStitchToViaSpacing	212
minConcaveCornerSpacing	214
version	216

Multi-Patterning SVRF Commands

A typical SVRF rule file containing double-, triple-, and quadruple-patterning may also contain other SVRF commands that declare input layers, perform preliminary layer manipulation, and save output layers.

Many of these commands are shown in an exhaustive context in this manual, while some SVRF-related commands are documented in their entirety in the *Standard Verification Rule Format (SVRF) Manual*.

Table 3-1. Multi-Patterning SVRF Commands

Command	Description
DFM BALANCE	Calls the DFM SPEC BALANCE statement for each balanced mask to output.
DFM CLUSTER	Selects or generates separators or polygons based on geometric and graph-based properties of connected components.
DFM DP	Invokes Calibre double-patterning. Splits the target layer into two masks and generates error layers.
DFM DP MATCH	Provides rapid decomposition of similar polygon patterns.
DFM DP OPTIMIZE	Optimizes mask coloring through application of priority-based expressions applied to layers.
DFM DP OPTIMIZER_CONSTRAINTS	Generates constraint layers from DFM DP based on priority.
DFM MP	Invokes Calibre triple- or quadruple-patterning. Splits the target layer into three or four masks and generates error layers.
DFM MP SELECT CELLS	Generates a layer covering the area of selected cells based on various criteria.
DFM MPSTITCH	Generates multi-pattern stitches. References DFM SPEC MPSTITCH constraint specifications.
DFM RET NMDPC EMULATION	Emulates RET NMDPC operations using DFM DP algorithms.
DFM SPEC BALANCE	Specifies multi-pattern mask balancing criterion. Called by the DFM BALANCE command.
DFM SPEC MPSTITCH	Specifies multi-pattern stitching values and constraints. Called by DFM MPSTITCH.
INCLUDE TVF tp_model	Executes the specified encrypted <i>tp_model.tvf</i> file and its arguments.
INCLUDE TVF tp_separator	Executes the specified encrypted <i>tp_separator.tvf</i> file and its arguments.

Table 3-1. Multi-Patterning SVRF Commands (cont.)

Command	Description
INCLUDE TVF tp_stitch_gen	Executes the specified encrypted <i>tp_stitch_gen.tvf</i> file and its arguments.
INCLUDE TVF mp_stitch_gen	Executes the specified encrypted <i>mp_stitch_gen.tvf</i> file and its arguments.
RET DPSTITCH	Calls a LITHO FILE statement containing a setlayer dpstitch command that generates stitch candidate shapes. Used with RET NMDPC and DFM DP.
RET NMDPC	Invokes Calibre nmDPC and splits the target layer into two masks.
RET QP	Invokes Calibre quadruple-patterning using the RET QP command. Splits the target layer into four masks and generates error layers.
RET TP	Invokes Calibre triple-patterning using the RET TP command. Splits the target layer into three masks and generates error layers.

DFM BALANCE

Multi-Patterning SVRF Commands

Calls the DFM SPEC BALANCE statement for each balanced mask to output.

Usage

balanced_mask = DFM BALANCE name [[FULL] MASK color]

analysis_windows = DFM BALANCE name [UNMERGED [FAILED]]

Arguments

- ***name***

A required argument referencing the specifications named within the [DFM SPEC BALANCE](#) statement.

- **[FULL] MASK color**

An optional keyword that outputs all mask shapes with the specified color that have had their color changed or that have been promoted through hierarchy. If FULL is specified, the entirety of the mask in addition to those areas satisfying the density threshold are output.

- **UNMERGED [FAILED]**

An optional keyword specifying the type of layer output for analysis. UNMERGED outputs an unmerged layer with analysis windows updated with annotation properties. As an option to UNMERGED, FAILED outputs only those unmerged windows failing balancing requirements.

If neither keyword is specified, the output is a merged layer of all analysis windows that pass or fail balancing requirements.

Description

The DFM BALANCE command calls the DFM SPEC BALANCE statement within the same SVRF file. You must specify a DFM BALANCE command to output each balanced mask. In some cases, writing SVRF statements prior to DFM BALANCE prepares design layers for optimal balancing. See the examples for prebalancing considerations. For important exceptions regarding layer derivation support see “[Guidelines for Deriving Good Separator and Action Layers](#)” on page 346.

Recommended DFM BALANCE Flow

Although flexible in its implementation, specific DFM BALANCE flows provide optimum results. Siemens EDA recommends using DFM BALANCE as shown in [Figure 3-1](#).

Figure 3-1. DFM BALANCE Recommended Flow



1. Input Design
 - DFM BALANCE may be applied to IP, block or full chip designs.
 - Layouts may contain precolored or anchored polygons.
 - Layouts may contain fill shapes.
2. dpStitch
 - Stitching is provided for odd-cycle resolution.
 - Stitching is provided to facilitate density balancing.
3. nmDPC
 - Directed algorithms color an uncolored layout.
 - Employs a native tick-tock balancing algorithm and custom long-range separators for local density balancing.
 - Odd-cycle stitch minimization is provided.
 - Density balancing stitch maximization is provided.
4. DFM Fill
 - Arbitrary and differing-color fill is performed within open areas.
 - Differing-color fill is performed within large polygons.
5. DFM BALANCE
 - Pre-DFM balancing considerations may include:
 - Detect and isolate used stitches and replace them with edge-pair required separators.
 - Construct required balancing separators.
 - Consider colored polygon interaction with drawn anchors and anchor seed shapes while constructing mutable and anchors layers.
 - DFM BALANCE algorithmic provisions include:
 - Arbitrarily-colorable polygon (ACP) and connected component (CC) color alternation.
 - Window-based area-aware density balancing with change minimization.
 - Built-in error reporting of regions failing density balancing constraints.
6. Output Design
 - Layout is locally and globally density-optimized.

DFM BALANCE Layer Interactions

DFM BALANCE provides and restricts various layer interactions.

- ANCHOR and MUTABLE polygons of different colors are permitted to overlap but not abut.
- MUTABLE polygons touching separators may be colored differently to achieve density balancing.
- MUTABLE polygons connected with ANCHOR polygons by a SEPARATOR are considered as ANCHOR shapes, and are immutable (their color cannot be reassigned).
- ANCHOR polygons of different colors are permitted to overlap or abut.
- MUTABLE polygons of different color are not permitted to overlap.
- UNCOLORED polygons are not permitted to touch or overlap ANCHOR or MUTABLE polygons, or any SEPARATOR edge pair.

Table 3-2 charts all layer interactions.

Table 3-2. DFM BALANCE Layer Interaction

Layers:	Anchors _{mask1}	Separators	Mutable _{mask1}	Uncolored
Anchors _{mask0}	Allowed	Allowed	Overlap yes; abut no	Not allowed
Separators	Allowed	Allowed	Allowed	Not allowed
Mutable _{mask0}	Overlap yes; abut no	Allowed	Overlap yes; abut no	Not allowed
Uncolored	Not allowed	Not allowed	Not allowed	Allowed

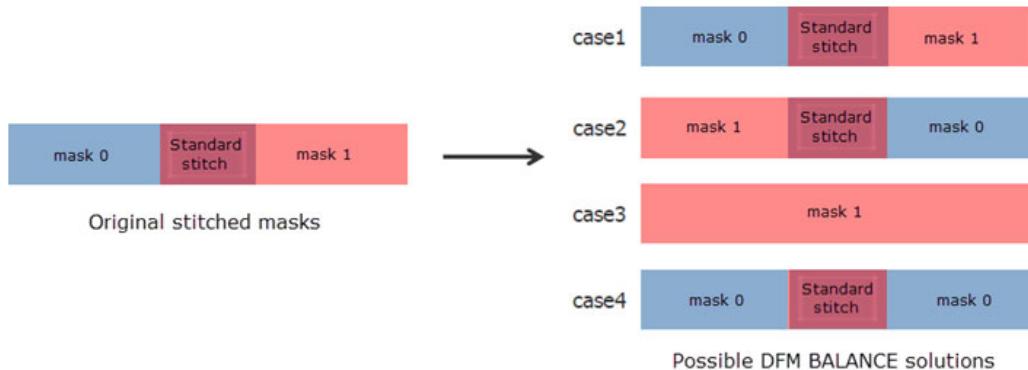
Note

 Prior to Calibre release 2016.3, ANCHOR-to-MUTABLE and MUTABLE-to-MUTABLE layer interactions were not allowed.

DFM BALANCE Stitch Implementation

DFM BALANCE can solve balancing issues for previously stitched masks. As depicted in Figure 3-2, multiple solutions are possible.

Figure 3-2. DFM BALANCE Stitch Solutions



DFM BALANCE Statistics

DFM BALANCE can print processing statistics. To invoke DFM BALANCE statistical output, you set an environment variable (Bourne shell shown):

```
CALIBRE_DFM_BALANCE_STATISTICS=1
export CALIBRE_DFM_BALANCE_STATISTICS
```

In the following example, the changes to shapes belonging to each mask are recorded.

Example of DFM BALANCE Statistical Output for Triple-Patterning

```
DFM Balance Statistics : -----
DFM Balance Statistics : SPEC - stat_test
DFM Balance Statistics : MASK 1 - FULL_MASK1, CHANGED_MASK1
DFM Balance Statistics :   Initial polygons: 1202 - mutable: 1202, anchored: 0
DFM Balance Statistics :   To other masks: 70 (5.8%) - MASK 2: 38, MASK 3: 32
DFM Balance Statistics :   From other masks: 48 (4.0%) - MASK 2: 25, MASK 3: 23
DFM Balance Statistics :   From OVERLAY: 5 (0.4%)
DFM Balance Statistics :   Final polygons: 1248
DFM Balance Statistics :   Promoted polygons: 16 (1.3%)
DFM Balance Statistics : MASK 2 - FULL_MASK2, CHANGED_MASK2
DFM Balance Statistics :   Initial polygons: 1134 - mutable: 1134, anchored: 0
DFM Balance Statistics :   To other masks: 66 (5.8%) - MASK 1: 25, MASK 3: 41
DFM Balance Statistics :   From other masks: 48 (4.2%) - MASK 1: 38, MASK 3: 10
DFM Balance Statistics :   From OVERLAY: 41 (3.6%)
DFM Balance Statistics :   Final polygons: 1280
DFM Balance Statistics :   Promoted polygons: 12 (1.1%)
DFM Balance Statistics : MASK 3 - FULL_MASK3, CHANGED_MASK3
DFM Balance Statistics :   Initial polygons: 1098 - mutable: 1098, anchored: 0
DFM Balance Statistics :   To other masks: 33 (3.0%) - MASK 1: 23, MASK 2: 10
DFM Balance Statistics :   From other masks: 73 (6.6%) - MASK 1: 32, MASK 2: 41
DFM Balance Statistics :   From OVERLAY: 42 (3.8%)
DFM Balance Statistics :   Final polygons: 1306
DFM Balance Statistics :   Promoted polygons: 0 (0.0%)
DFM Balance Statistics : -----
```

Examples

Example 1 — Command Variations

```
DFM SPEC BALANCE met1_balance 2 WITH DENSITY >= 0.45 ...

balanced_mask1 = DFM BALANCE met1_balance MASK 1
failed_windows_unmerged = DFM BALANCE met1_balance UNMERGED
failed_windows_merged = DFM BALANCE met1_balance
```

Example 2 — Commands as Checks

```
Balanced_mask1 {@ Balanced Mask1 polygons
    DFM BALANCE met1_balance MASK 1
}
Failed_windows_unmerged {
    DFM RDB (DFM BALANCE met1_balance UNMERGED FAILED)
    "density_results.rdb"
    CHECKNAME "Failed_windows_unmerged"
    COMMENT "windows that fail density balancing requirements"
}
Analysis_windows_merged {@ Merged passing and failing density windows
    DFM BALANCE met1_balance
}
```

Example 3 - Prebalancing Statements (enables optimal balancing)

```
// 1. Detect and isolate used stitches
used_stitches = M1_mask1 AND M1_mask0
used_stitches_separators =(used_stitches NOT COINCIDENT EDGE M1_target)
    INTERNAL <= 0.100

// 2. Construct balancing separators
balance_separators = DFM COPY used_stitch_separators M1_separators

// 3. Consider colored polygons interacting with anchors or
//     anchor seeds but not stitches
anchor_mask0_all = ((M1_mask0 NOT used_stitches) NOT anchor_mask1)
    INTERACT anchor_mask0
anchor_mask1_all = ((M1_mask1 NOT used_stitches) NOT anchor_mask0)
    INTERACT anchor_mask1

// 4. Remove stitches from mutable polygons
Mutable_mask0_without_stitching = (M1_mask0 NOT used_stitches)
    NOT INTERACT anchor_mask0_all
Mutable_mask1_without_stitching = (M1_mask1 NOT used_stitches)
    NOT INTERACT anchor_mask1_all

// 5. Derive final mutable masks
Mutable_mask0 = Mutable_mask0_without_stitching
    NOT INTERACT anchor_mask0_all
Mutable_mask1 = Mutable_mask1_without_stitching
    NOT INTERACT anchor_mask1_all

DFM SPEC BALANCE Mutable_mask0 2 WITH DENSITY >= 0.45 ...
balanced_mask0 = DFM BALANCE Mutable_mask0 MASK 1
failed_windows_unmerged = DFM BALANCE Mutable_mask0 UNMERGED
failed_windows_merged = DFM BALANCE Mutable_mask0
```

Example 4 - Double-Patterning

```
VARIABLE delta_den 0.4
VARIABLE window_size 0.2
VARIABLE window_step 0.1

Anchor_mask0_all = M1_DP_mask0 INTERACT Anchor_mask0
Anchor_mask1_all = M1_DP_mask1 INTERACT Anchor_mask1

Mutable_mask0 = M1_DP_mask0 NOT INTERACT Anchor_mask0_all
Mutable_mask1 = M1_DP_mask1 NOT INTERACT Anchor_mask1_all

DFM SPEC BALANCE "DP_spec" 2
    ANCHOR Anchor_mask0_all Anchor_mask1_all
    MUTABLE Mutable_mask0 Mutable_mask1
    SEPARATOR M1_Separators
    WITH DENSITY MIN_COLOR_OVER_TOTAL_COLOR >= delta_den
    BACKUP INSIDE OF EXTENT WINDOW window_size STEP window_step

DB_Balanced_mask0 = DFM BALANCE "DP_spec" FULL MASK 0
DB_Balanced_mask1 = DFM BALANCE "DP_spec" FULL MASK 1

DB_FailedWindows = DFM BALANCE "DP_spec" UNMERGED
```

Example 5 - Triple-Patterning

```
VARIABLE delta_den 0.3
VARIABLE window_size 0.2
VARIABLE window_step 0.1

Anchor_mask1_all = M1_TP_mask1 INTERACT Anchor_mask1
Anchor_mask2_all = M1_TP_mask2 INTERACT Anchor_mask2
Anchor_mask3_all = M1_TP_mask3 INTERACT Anchor_mask3

Mutable_mask1 = M1_TP_mask1 NOT INTERACT Anchor_mask1_all
Mutable_mask2 = M1_TP_mask2 NOT INTERACT Anchor_mask2_all
Mutable_mask3 = M1_TP_mask3 NOT INTERACT Anchor_mask3_all

DFM SPEC BALANCE "TP_spec" 3
    ANCHOR Anchor_mask1_all Anchor_mask2_all Anchor_mask3_all
    MUTABLE Mutable_mask1 Mutable_mask2 Mutable_mask3
    SEPARATOR balance_separators
    WITH DENSITY MIN_COLOR_OVER_TOTAL_COLOR >= delta_den
    BACKUP INSIDE OF EXTENT WINDOW window_size STEP window_step

DB_full_mask1 = DFM BALANCE "TP_spec" FULL MASK 1
DB_full_mask2 = DFM BALANCE "TP_spec" FULL MASK 2
DB_full_mask3 = DFM BALANCE "TP_spec" FULL MASK 3

DB_FailedWindows = DFM BALANCE "TP_spec" UNMERGED
```

DFM CLUSTER

Multi-Patterning SVRF Commands

Selects or generates separators or polygons based on geometric and graph-based properties of connected components.

Usage

Corner Keyword Usage

```
output_layer = DFM CLUSTER
  target_layer
  separator_layer ...
  corner_keyword [MARKER]
    {[D21_CHAINS] | [DEGREE {constraint | MINIMUM | MAXIMUM}]}
    [INSIDE OF LAYER inside_layer]
    [NEIGHBOR OF anchor_layer]
    [NOFLOAT]
    [select_keyword select_layer]
      | [{SELECT_CELLS | SELECT_CELLS_LOWEST} cell_area_layer]
```

COUNT Usage

```
output_layer = DFM CLUSTER
  target_layer
  separator_layer ...
  COUNT constraint
    [INSIDE OF LAYER inside_layer]
```

D21_CHAINS Usage

```
output_layer = DFM CLUSTER
  target_layer
  separator_layer ...
  D21_CHAINS
    [corner_keyword [INSIDE OF LAYER inside_layer] [MARKER]]
    [NOFLOAT]
    [select_keyword select_layer]
```

D21_CONSTRAINTS Usage

```
output_layer = DFM CLUSTER
  target_layer
  separator_layer ...
  D21_CONSTRAINTS
    [select_keyword select_layer]
```

DEGREE Constraint Usage

```
output_layer = DFM CLUSTER
  target_layer
  separator_layer ...
```

DEGREE constraint

[{DISTANCE | SPACING} *constraint* [*distance_spacing_options*]]
[*select_keyword* *select_layer*]

DEGREE MINIMUM and MAXIMUM Usage

output_layer = DFM CLUSTER

target_layer

separator_layer ...

DEGREE {MAXIMUM | MINIMUM}

[*corner_keyword* [INSIDE OF LAYER *inside_layer*] [MARKER]]
[NOFLOAT | {{DISTANCE | SPACING} *constraint* [*distance_spacing_options*]}}]
[{{SELECT_CELLS | SELECT_CELLS_LOWEST} *cell_area_layer*}
[*select_keyword* *select_layer*]]

DISTANCE and SPACING Usage

output_layer = DFM CLUSTER

target_layer

separator_layer ...

{DISTANCE | SPACING} *constraint* [*distance_spacing_options*]
[DEGREE {*constraint* | MAXIMUM | MINIMUM}]
[INSIDE OF LAYER *inside_layer*]
[*select_keyword* *select_layer*]

EXCLUDE SEPARATORS and SHORTEST Usage

output_layer = DFM CLUSTER

target_layer

separator_layer ...

{EXCLUDE SEPARATORS *separator_layer*...
| SHORTEST
| {EXCLUDE SEPARATORS *separator_layer*... SHORTEST}}

NEIGHBOR OF Usage

output_layer = DFM CLUSTER

target_layer

separator_layer ...

NEIGHBOR OF {*anchor_layer*}

[*corner_keyword* [INSIDE OF LAYER *inside_layer*] [MARKER]]
[NOFLOAT]
[{{SELECT_CELLS | SELECT_CELLS_LOWEST} *cell_area_layer*}]
| *constraint*

Select Keyword Usage

output_layer = DFM CLUSTER

target_layer

separator_layer ...

select_keyword* *select_layer

[*corner_keyword* [INSIDE OF LAYER *inside_layer*] [MARKER]]

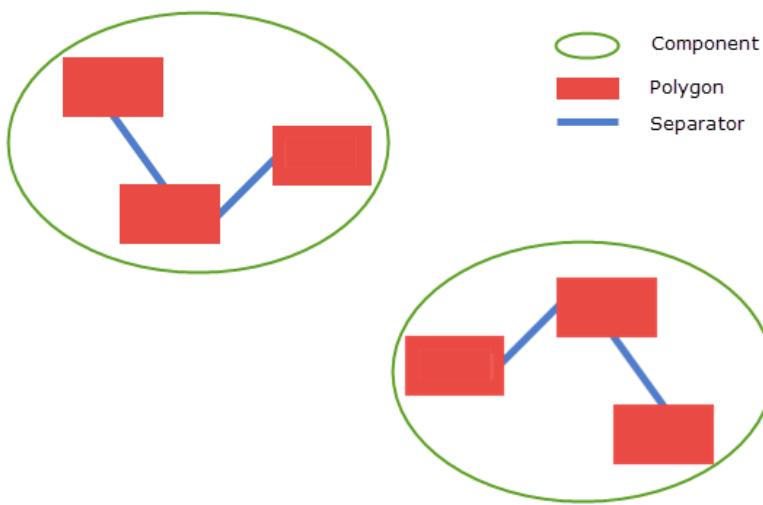
```
[D21_CHAINS | D21_CONSTRAINTS
  | DEGREE {constraint | MAXIMUM | MINIMUM}]
[NOFLOAT]
[{:SPACING | DISTANCE} constraint [distance_spacing options]]
```

Description

DFM CLUSTER selects or generates either separators or polygons, extracted from the target layer, based on geometric and graph-based properties of connected components. A connected component is a collection of polygons having an unbroken connection sequence by separators, where any polygon can be traced to any other polygon within the connected component by traversing a sequence of polygons and separators.

DFM CLUSTER requires one or more keywords to perform the specified operation. The DFM CLUSTER operations are classified into corner, graph, select, and DISTANCE and SPACING keyword groups. One of either a corner, graph, select, or SPACING or DISTANCE keyword must be specified for each DFM CLUSTER command. Some keywords may be used in combination with each other. For allowed keyword combinations, refer to their respective descriptions.

Figure 3-3. Connected Component Model



Arguments

Layers

- *output_layer*

A required layer used in further derivations or saved to an output database. The generated layer can be either separators or polygons, depending on the keyword used.

- *target_layer*

A required argument specifying the target design layer. The target_layer must be a polygon layer (layer-type 1).

- *separator_layer...*

A required argument specifying one or more separator layers. Each separator layer must be an edge pair (layer-type 3).

[Example 3-1](#) shows SVRF code that provides typical separator derivation, and [Figure 3-4](#) shows correctly formed separators.

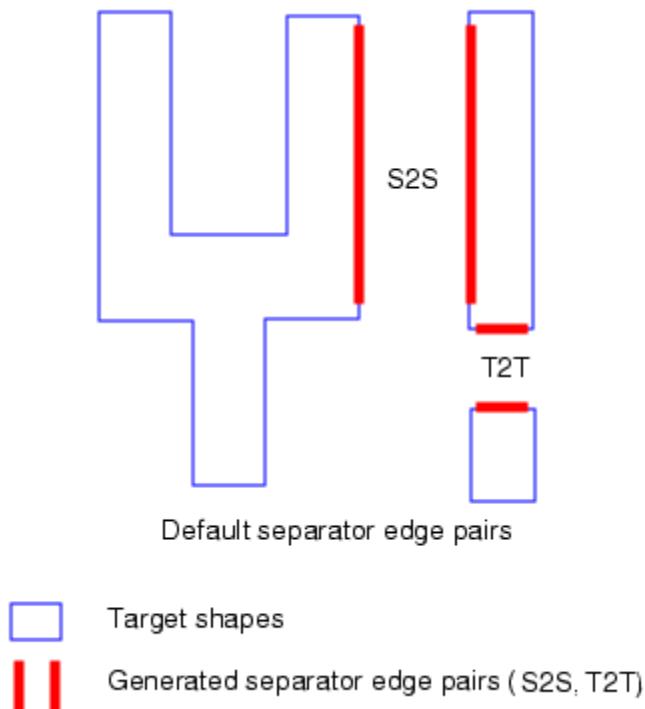
Example 3-1. Typical Separator Derivation

```
m1_tip = CONVEX EDGE m1 ANGLE1 == 90 LENGTH1 >= 0.03
          ANGLE2 == 90 LENGTH2 >= 0.03 WITH LENGTH <= 0.06
m1_side = m1 NOT COIN EDGE m1_tip

m1_S2S = EXT m1_side < 0.06 OPPOSITE PARALLEL ONLY PROJ > 0
m1_T2S = EXT m1_tip m1_side < 0.065 PARALLEL ONLY
m1_T2T = EXT m1_tip < 0.085 PARALLEL ONLY
```

The input separator layer must consist of edge pairs only (layer-type 3) and each side of the edge pair must touch exactly one drawn polygon. [Figure 3-4](#) demonstrates separators.

Figure 3-4. Correct DFM CLUSTER Separators



Keywords

- *corner_keyword*

An optional keyword that specifies a corner polygon of each component to be output. Only one corner_keyword may be specified for each DFM CLUSTER command. Corner keywords terms are ordered. For example, LEFT TOP returns the leftmost polygon and, if there are two or more polygons, subsequently returns the topmost one.

Corner keywords can be used in combination with select, D21_CHAINS, DEGREE MAXIMUM and DEGREE MINIMUM. NEIGHBOR OF can be used exclusively with corner keywords. Outputs a polygon (type-1) layer.

BOTTOM LEFT

An optional keyword that outputs a polygon layer containing the bottom-left polygon of each connected component. If there are two or more bottommost polygons, the leftmost polygon is selected.

BOTTOM LEFT

An optional keyword that outputs a polygon layer containing the bottom-left polygon of each connected component. If there are two or more bottommost polygons, the leftmost polygon is selected.

BOTTOM RIGHT

An optional keyword that outputs a polygon layer containing the bottom-right polygon of each connected component. If there are two or more bottommost polygons, the rightmost polygon is selected.

LEFT BOTTOM

An optional keyword that outputs a polygon layer containing the left-bottom polygon of each connected component. If there are two or more leftmost polygons, the bottommost polygon is selected.

LEFT TOP

An optional keyword that outputs a polygon layer containing the left-top polygon of each connected component. If there are two or more leftmost polygons, the topmost polygon is selected.

RIGHT BOTTOM

An optional keyword that outputs a polygon layer containing the right-bottom polygon of each connected component. If there are two or more rightmost polygons, the bottommost polygon is selected.

RIGHT TOP

An optional keyword that outputs a polygon layer containing the right-top polygon of each connected component. If there are two or more rightmost polygons, then the topmost polygon is selected.

TOP LEFT

An optional keyword that outputs a polygon layer containing the top-left polygon of each connected component. If there are two or more topmost polygons, the leftmost polygon is selected.

TOP RIGHT

An optional keyword that outputs a polygon layer containing the top-right polygon of each connected component. If there are two or more topmost polygons, the rightmost polygon is selected.

MARKER

An optional argument that modifies corner-selected output. Instead of outputting entire polygons, a small square representing each polygon is output. Siemens EDA recommends using the MARKER keyword to reduce DFM CLUSTER run time.

- **COUNT constraint**

An optional keyword and constraint that outputs all polygons from a component in which the flat polygon count satisfies the constraint.

COUNT cannot be combined with any other DFM CLUSTER keywords. Outputs a polygon (type-1) layer.

Example 3-2. COUNT Usage

```
// Outputs all components with exactly two polygons
D2 = DFM CLUSTER target sep COUNT ==2
// Outputs all isolated polygons (components with exactly one polygon)
D2 = DFM CLUSTER target sep COUNT ==1
// Outputs all components with no less than five polygons
D2 = DFM CLUSTER target sep COUNT >=5
```

- **D21_CHAINS**

An optional keyword that outputs polygons from the target layer that have a degree (defined as the number of polygons they are directly connected to) of 1 or 2, comprising a D21 chain. A D21 chain is defined as a connected component of degree 1 and 2 polygons. When NOFLOAT is specified, single degree 1 and degree 2 polygons are not output.

D21_CHAINS can only be used alone or in combination with select and corner keywords, and NOFLOAT. Outputs a polygon (type-1) layer.

- **D21_CONSTRAINTS**

An optional keyword that outputs a separator between every pair of polygons that meet two criteria:

- The polygons belong to the same D21 chain.
- The polygons are a distance of 2 separators from one another. The distance between two polygons is defined as the minimum number of separators that must be traversed in order to travel from one polygon to another.

D21_CONSTRAINTS can only be used alone or with select keywords. Outputs a separator (type-3) layer.

- DEGREE {*constraint* | MAXIMUM | MINIMUM}

An optional keyword and constraint pair which outputs all polygons that satisfy the specified constraint, or minimum or maximum degree. Outputs a polygon (type-1) layer, except when DISTANCE or SPACING is used in combination, which outputs a separator (type-3) layer.

- *constraint* — An SVRF constraint as described in the [Constraints](#) section of the *Standard Verification Rule Format (SVRF) Manual*, with the exception of the != constraint, which is not allowed. DEGREE constraint may be used alone or in combination only with selection keywords, and DISTANCE or SPACING.
- MAXIMUM | MINIMUM — Optional keywords that output the maximum or minimum degree polygon from each connected component, respectively. If there is more than one polygon in the same component with the MAXIMUM or MINIMUM degree of all nodes in the component, they are all returned. If NOFLOAT is not specified, then every floating polygon will be output because each floating polygon is considered its own connected component. DEGREE MAXIMUM or MINIMUM can be used alone or in combination only with select, corner, and DISTANCE or SPACING keywords, and NOFLOAT, SELECT_CELLS, and SELECT_CELLS_LOWEST options.

- {DISTANCE | SPACING} *constraint* [*distance_spacing_options*]

One of two optional keywords and constraint that generate separators that satisfy the specified spacing constraints. Only one SPACING or DISTANCE keyword may be specified per DFM CLUSTER command. For SPACING, the constraint is only applied between polygons within the same component. For DISTANCE, the constraint is only applied between polygons in different components. The SPACING keyword also measures polygon notches (outside edge to outside edge of the same polygon shape). The SPACING and DISTANCE keywords are a subset of the SVRF EXTERNAL operation.

DISTANCE and SPACING may be used alone or in combination with select keywords and DEGREE constraint. Outputs a separator (type-3) layer.

constraint

A required argument that specifies the spacing constraint. All constraint values must be non-negative integers. The various forms for the constraint (where *a* and *b* are lower and upper range bounds, respectively) are:

< *b* / <= *b* / == *a* / > *a* < *b* / >= *a* < *b* / > *a* <= *b* / >= *a* <= *b*

distance_spacing_options

CORNER TO CORNER

An optional argument that requires separators to hold a corner-to-corner connection to shapes. The shapes involved must be non-projecting.

EXCLUDE SHIELDED

An optional argument that requires edges to be projecting with no shape between their facing edges.

OPPOSITE

An optional argument that requires separators to be projecting on to one another.

OPPOSITE EXTENDED *extension_value*

An optional argument that requires separators to be projecting on to one another with the specified orthogonal extension.

RELATIVE

An optional argument that uses the longest separator length of a component to modify the constraint. The numerical values provided in the constraint are treated as coefficients and applied to the length of the longest separator on a per-component basis.

For example, the constraint $>= 0.5 <= 2$ results in separators for each component that are at least half the length of the longest separator of that component but no more than twice as long. When the RELATIVE argument is used, the constraint must specify a finite range of values and cannot be in the form $=a$.

SINGULAR

An optional argument that requires separators to be a point, where shapes are touching corner-to-corner. The shapes involved must be non-projecting.

- EXCLUDE SEPARATORS *separator_layer...*

An optional argument that outputs a layer containing separators from the initial separator layer set (those separator_layers specified by DFM CLUSTER) mapping to graph edges different than those mapped to by the second set of separator layers (those specified by EXCLUDE SEPARATORS). Those separators of the second set that do not map to the same graph edges of the first set are excluded from the output. Multiple separator_layers may be specified.

EXCLUDE SEPARATORS may be used alone or with the SHORTEST keyword, and is incompatible with all other DFM CLUSTER keywords. Outputs a separator (type-3) layer.

- NEIGHBOR OF *anchor_layer* [*corner_keyword* | *constraint*]

An optional keyword and polygon layer that selects polygons that are directly connected by a separator to polygons on the anchor_layer. Optionally, either a corner keyword or constraint may be specified. When corner keywords are specified, the selected polygons form new connected components that may differ from the connected components of the target_layer. Alternatively, a constraint may be specified to output shapes with the number of connections to the anchor_layer.

Only corner keywords may be used in conjunction with NEIGHBOR OF. When an optional corner keyword is specified, a corner polygon is returned for every new component (see the Corner Keywords section for more information). However, NEIGHBOR OF can be indirectly used in combination with non-corner keywords if the output of one DFM CLUSTER command provides input to another.

When used with SELECT_CELLS or SELECT_CELLS_LOWEST options, NEIGHBOR_OF nodes do not count when the node is found within a higher cell, or the connected separator is found within a higher cell. Outputs a polygon (type-1) layer.

- ***select_keyword select_layer***

An optional keyword that selects components that satisfy the specified topological operation with respect to the select_layer, and outputs polygons that belong to those components. Only one select_keyword may be specified per DFM CLUSTER command. A select_layer must be specified with each select_keyword. The select_keyword operations behave similarly to the standard SVRF topological operations, with the difference that all polygons in a connected component are processed as a single polygon with respect to the topological operation being performed.

Any select_keyword can only be used in combination with corner, D21_CONSTRAINTS, D21_CHAINS, DEGREE, and DISTANCE or SPACING keywords, and can only use the SELECT_CELLS and SELECT_CELLS_LOWEST options. DISTANCE or SPACING can only be used in combination with DEGREE. When D21_CONSTRAINTS, or DISTANCE or SPACING is used in combination, select keywords output a separator (type-3) layer. All other output is a polygon (type-1) layer.

SELECT_CUT *select_layer*

An optional keyword and layer pair which selects any component that contains polygons sharing some, but not all, of their area with polygons from the select_layer.

SELECT_NOT_CUT *select_layer*

An optional keyword and layer pair which selects any component containing polygons sharing all or none of their area with polygons from the select_layer. SELECT_NOT_CUT is the complement of SELECT_CUT.

SELECT_ENCLOSE *select_layer*

An optional keyword and layer pair which selects any component containing polygons that completely enclose any polygon from the select_layer.

SELECT_NOT_ENCLOSE *select_layer*

An optional keyword and layer pair which selects any component containing polygons that do not completely enclose any polygon from the select_layer. SELECT_NOT_ENCLOSE is the complement of SELECT_ENCLOSE.

SELECT_INSIDE *select_layer*

An optional keyword and layer pair which selects any component containing polygons that share all of their area with polygons from the select_layer.

SELECT_NOT_INSIDE *select_layer*

An optional keyword and layer pair which selects any component containing polygons that do not share all of their area with polygons from the select_layer. SELECT_NOT_INSIDE is the complement of SELECT_INSIDE.

SELECT_INTERACT *select_layer*

An optional keyword and layer pair which selects any component containing polygons that share area or have a coincident edge with a polygon from the select_layer.

SELECT_NOT_INTERACT *select_layer*

An optional keyword and layer pair which selects any component containing polygons that do not share area or have a coincident edge with a polygon from the select_layer. SELECT_NOT_INTERACT is the complement of SELECT_INTERACT.

SELECT_OUTSIDE *select_layer*

An optional keyword and layer pair which selects any component containing polygons that share none of their area with polygons from the select_layer. Coincident edges do not share area.

SELECT_NOT_OUTSIDE *select_layer*

An optional keyword and layer pair which selects any component containing polygons that share some or all of their area with polygons from the select_layer. SELECT_NOT_OUTSIDE is the complement of SELECT_OUTSIDE.

SELECT_TOUCH *select_layer*

An optional keyword and layer pair which selects any component containing polygons that share none of their area, but have a coincident edge with polygons from the select_layer.

SELECT_NOT_TOUCH *select_layer*

An optional keyword and layer pair which selects any component containing polygons that either share area, or do not have a coincident edge with polygons from the select_layer. SELECT_NOT_TOUCH is the complement of SELECT_TOUCH.

- **SHORTEST**

An optional argument that outputs the shortest separators interacting with each target polygon.

SHORTEST may be used alone or with the EXCLUDE SEPARATORS keyword, and is incompatible with all other DFM CLUSTER keywords. Outputs a separator (type-3) layer.

Options

Options can only be used in conjunction with DFM CLUSTER keywords.

- **ANNOTATE, ANNOTATED**

DFM CLUSTER keywords combined with the ANNOTATE and ANNOTATED options generate output layers with component information in two steps:

- a. Save the output layer with component information by specifying ANNOTATE.
- b. Specify the output layer with ANNOTATED in parenthesis.

ANNOTATE and ANNOTATED do not apply to separator layers because they already have component information attached, and shapes output by corner keywords are not

compatible with ANNOTATE or ANNOTATED. The following example demonstrates ANNOTATE and ANNOTATED usage by selecting components consisting of three polygons of a degree equal to 2.

```
// Select degree-2 polygons:  
dg2 = DFM CLUSTER target sep DEGREE == 2 ANNOTATE  
  
// Output anchors that have three polygons:  
three_dg2 = DFM CLUSTER dg2(ANNOTATED) COUNT == 3  
  
// Select components with three polygons with degree = 2:  
comp_3_dg2 = DFM CLUSTER target sep SELECT_INTERACT three_dg2
```

- **INSIDE OF LAYER** *inside_layer*

An optional argument that requires keyword output to be contained within the specified *inside_layer*.

INSIDE OF LAYER can only be used with corner, COUNT, and DISTANCE or SPACING keywords. INSIDE OF LAYER cannot be used in combination with SELECT_CELLS or SELECT_CELLS_LOWEST options.

- **NOFLOAT**

An optional argument that prevents the processing of floating polygons by any keywords. Floating polygons are those not connected by separators to any polygon. If NOFLOAT is not specified, each floating polygon is processed as an individual component. NOFLOAT is always applied before the selection criteria of a keyword.

NOFLOAT can be used in combination with all keywords with the exception of D21_CONSTRAINTS, and DISTANCE or SPACING keywords.

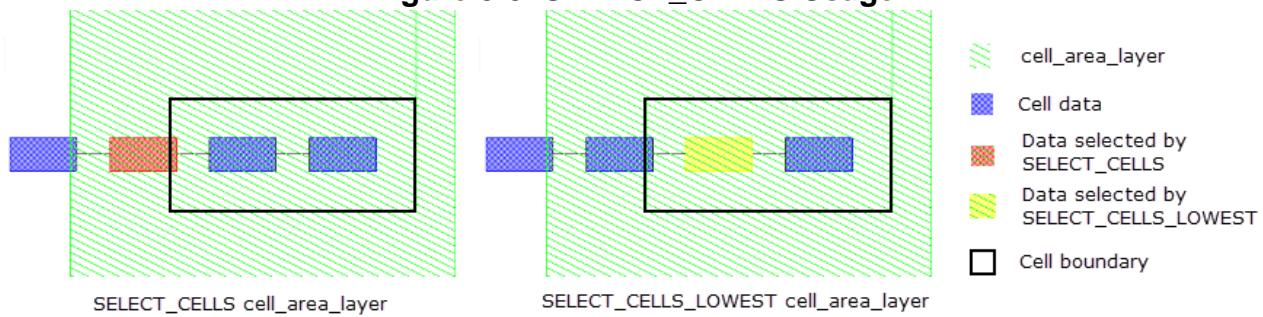
- {SELECT_CELLS | SELECT_CELLS_LOWEST} *cell_area_layer*

One of two mutually-exclusive optional arguments that selects data based on hierarchical cell location. SELECT_CELLS selects the data highest in hierarchical context within the specified *cell_area_layer* region. The *cell_area_layer* must be generated by DFM MP SELECT CELLS. SELECT_CELLS_LOWEST selects the data lowest in hierarchical context within the specified *cell_area_layer* region. When used in conjunction with DEGREE MAXIMUM or MINIMUM, only the separators in the selected cells and children of selected cells are counted when calculating the degree.

SELECT_CELLS and SELECT_CELLS_LOWEST can only be used in combination with NEIGHBOR_OF, DEGREE MAXIMUM or MINIMUM, and corner keywords, and cannot be used in combination with the INSIDE OF LAYER option.

The following figure demonstrates SELECT_CELLS and SELECT_CELLS_LOWEST behavior.

Figure 3-5. SELECT_CELLS Usage



Examples

Example 1 — Corner Keyword Usage

```
// Output left bottom polygon of each component:  
left_bottom = DFM CLUSTER target sep LEFT BOTTOM  
  
// Output markers on the left bottom polygon of each component:  
left_bottom_marker = DFM CLUSTER target sep LEFT BOTTOM MARKER  
  
// Output markers on the left bottom polygon of each component  
// excluding isolated polygons:  
left_bottom_nofloat = DFM CLUSTER target sep LEFT BOTTOM MARKER NOFLOAT
```

Example 2 — Select Keyword Usage

```
// Outputs components which interact with the block layer:  
interact = DFM CLUSTER target sep SELECT_INTERACT block  
  
// Outputs components which are not outside the block layer:  
not_outside = DFM CLUSTER target sep SELECT_NOT_OUTSIDE block  
  
// Outputs left bottom polygons of a component interacting with  
// a polygon on the block layer:  
corner_int = DFM CLUSTER target sep SELECT_INTERACT block LEFT BOTTOM  
  
// Outputs polygons where degree is equal to 1 or 2 on components  
// interacting with block layer:  
d21_int = DFM CLUSTER target sep SELECT_INTERACT block D21_CHAINS  
  
// Outputs left bottom polygon of D21 chain components that  
// interact with block layer:  
corner_d21_int = DFM CLUSTER target sep SELECT_INTERACT block  
D21_CHAINS LEFT BOTTOM
```

Example 3 — Graph Keyword Usage

```
// Output polygons which have a degree equal to 1 or 2:  
d21_chain = DFM CLUSTER target sep D21_CHAINS  
  
// Output the left bottom polygon which has a degree equal to 1 or 2  
// on each component:  
d21_chain_left_bottom = DFM CLUSTER target sep D21_CHAINS LEFT BOTTOM
```

```
// Output a separator layer of components with a degree of 1 or 2:  

chain_constraints = DFM CLUSTER target sep D21_CONSTRAINTS

// Output degree of 2 (neighbor count = 2) polygon to layer D2:  

D2 = DFM CLUSTER target sep DEGREE ==2

// Output degree greater than 1 and less equal than 3 to layer D23:  

D23 = DFM CLUSTER target sep DEGREE >1 <=3

// Output the maximum degree of any polygon within the component:  

dmax = DFM CLUSTER target sep DEGREE MAXIMUM

// Output maximum degree on components interacting with block layer:  

dmax_int = DFM CLUSTER target sep SELECT_INTERACT block DEGREE MAXIMUM

// Output the corner of minimum degree on each component:  

dmin_TR = DFM CLUSTER target sep DEGREE MINIMUM TOP RIGHT

// Output the corner of minimum degree on components interacting with  

// block layer:  

dmin_TR_int = DFM CLUSTER target sep SELECT_INTERACT block  

DEGREE MINIMUM TOP RIGHT
```

Example 4 — SPACING Usage

```
// Output a separator layer where polygons are the specified distance  

// from components and only have facing separators:  

sep_c = DFM CLUSTER target sep SPACING <0.2 OPPOSITE

// Output a separator layer where polygons are the specified distance  

// from components having a DEGREE less than or equal to 2:  

D2_sep = DFM CLUSTER target sep SPACING >0.1<=0.15 DEGREE <=2

// Output a separator layer where polygons have a distance less than  

// two times the longest existing separator within a component:  

sep_r = DFM CLUSTER target sep SPACING <2 RELATIVE OPPOSITE
```

Example 5 — NEIGHBOR OF Usage

```
// Output polygons connected to polygons in layer anchor:  

neighbors = DFM CLUSTER target sep NEIGHBOR OF anchor

// Output left bottom corner of components formed by NEIGHBOR OF.  

// Nodes that are selected by NEIGHBOR OF form new connected  

// components based on their separator-driven connectivity:  

neighbors = DFM CLUSTER target sep NEIGHBOR OF anchor LEFT BOTTOM

// Output markers of the left bottom corner of components formed by  

// neighbors of degree 4 nodes. Nodes that are selected by NEIGHBOR OF  

// form new connected components based on their separator-driven  

// connectivity:  

d_4 = DFM CLUSTER target sep DEGREE == 4  

neighbors = DFM CLUSTER target sep NEIGHBOR OF d_4 LEFT BOTTOM MARKER
```

```
// Output markers of the left bottom corner of components formed by
// neighbors of degree 4 nodes. If two neighbor nodes belong to different
// anchors that are overlapped, their neighbor group will be merged:
degree_4 = DFM CLUSTER target sep DEGREE ==4
neighbors = DFM CLUSTER target sep NEIGHBOR OF degree_4
LEFT BOTTOM MARKER

// Output polygons that have exactly two connections to the marker layer
// (degree 4 nodes).
degree_4 = DFM CLUSTER target sep DEGREE ==4
neighbors = DFM CLUSTER target sep NEIGHBOR OF degree_4 ==2
```

DFM DP

Multi-Patterning SVRF Commands

Invokes Calibre double-patterning. Splits the target layer into two masks and generates error layers.

Usage

```
output_layer = DFM DP
operation_keyword
target_layer
{separator_layer [‘({OPPOSITE | SAME} [priority])’]} ...
{anchor_layer ‘(ANCHOR {MASK0 | MASK1} [priority])’} ...
{stitch_layer ‘(STITCH
    {REQUIRED | USER | MINIMIZE min_priority | MAXIMIZE max_priority}’)’} ...
[PREFER {MASK0 | MASK1}]
[UNFILTERED]
[waiver_layer‘(WAIVER’)] ...
```

Description

Performs coloring for double-patterning designs. Outputs decomposed mask layers, or associated error or analysis layers. Each DFM DP command must specify at least one layer referencing a separator, anchor, or stitch, and may specify any combination, in any order, and any number of the three layers. For important exceptions regarding layer derivation support see “[Guidelines for Deriving Good Separator and Action Layers](#)” on page 346.

Arguments

- *output_layer*

A required output layer that can be saved to the output database and used for final decomposed mask output or error visualization analysis.

- *operation_keyword*

A required keyword directing DFM DP to perform its stated operation. You must specify one *operation_keyword* for each DFM DP command. All *operation_keywords* are classified into Mask Output and Error Visualization Keyword groups. The Error Visualization Keywords are sub-divided into two groups, those enabled or disabled by the specification of the [UNFILTERED](#) keyword.

Mask Output Keywords

MASK0 — A required type that outputs a polygon layer containing decomposed MASK0 geometries.

MASK1 — A required type that outputs a polygon layer containing decomposed MASK1 geometries.

Error Visualization Keywords

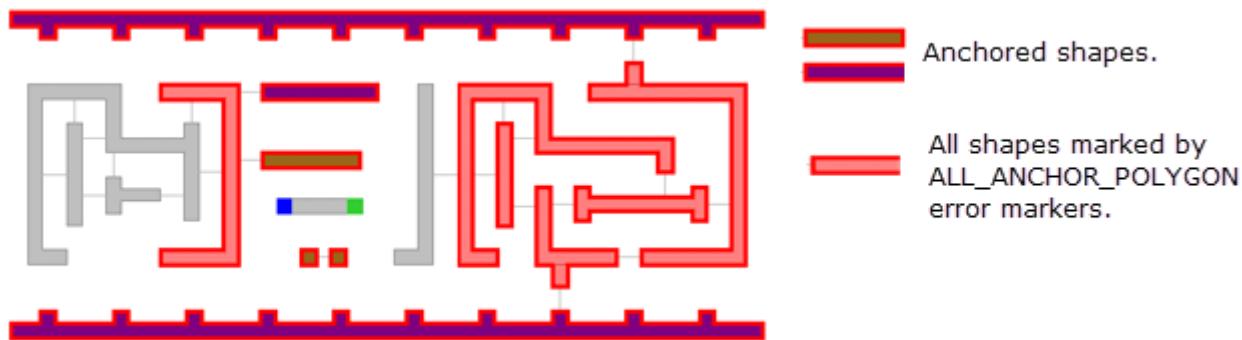
Many of the error layer keywords perform analysis on graphed components in the design. Components can be defined as chained polygons and separators that have no anchors breaking their connection sequence.

For error visualization output, the output type must be specified on a separate DFM DP command line. The error layer keywords are grouped in two categories: those enabled by specifying the UNFILTERED keyword, and those enabled by not specifying the UNFILTERED keyword.

All type-3 output layers (edge pairs) must be saved to the output database with a DFM COPY command. Type-1 and type-2 error layers (polygons and paths, respectively) can be saved with a COPY command. For a general discussion regarding error visualization and the implementation of these keywords, see the section entitled “[MP Visualization and Error Resolution](#)” on page 335.

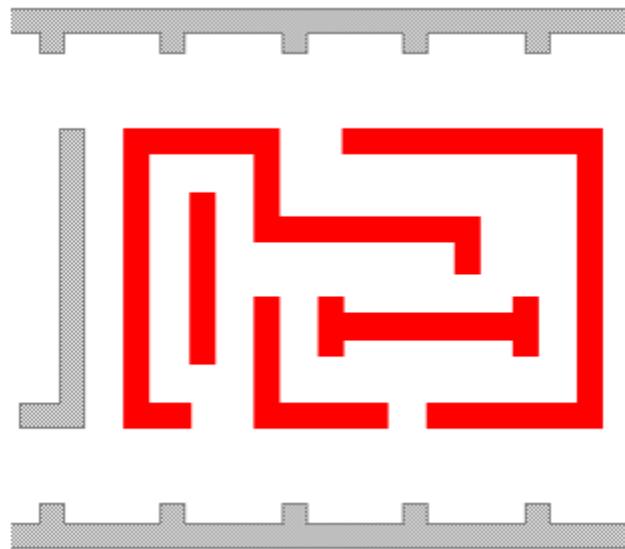
Keywords Enabled When UNFILTERED Is Not Specified

ALL_ANCHOR_POLYGONS — Outputs mask shapes that show all anchor path violations in each component, including components that are associated with biconnected components (BCCs) of violation loops, whether involved in violation loops or not. This is a type-1 layer. This output can be used for layer processing and visualizing spacing constraints. An example output:

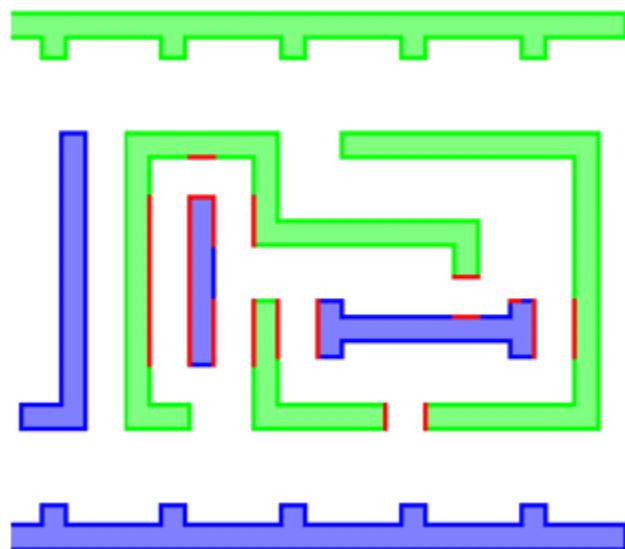


ALL_ANCHOR_SEPARATORS — Outputs all separators in every path between violated anchors, including all separators in all BCCs that these paths may intersect. This is a type-3 layer.

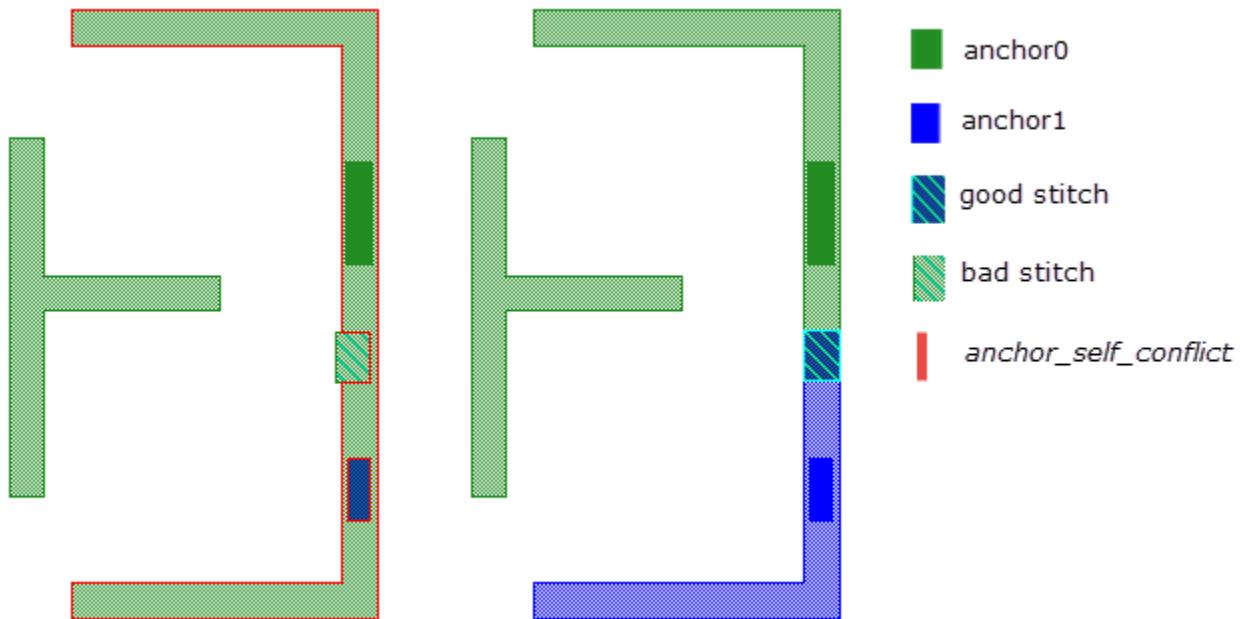
ALL_LOOP_POLYGONS — Outputs all mask shapes involved in all biconnected components (BCCs) that contain an odd cycle. This is a type-1 layer. An example output:



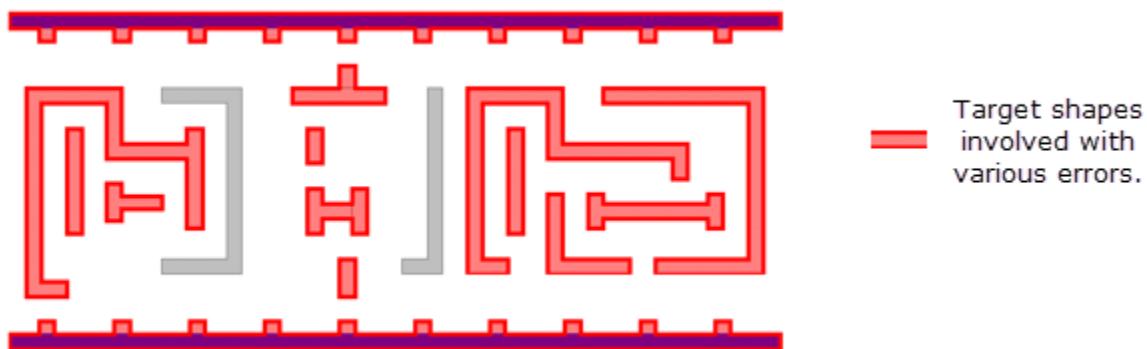
ALL_LOOP_SEPARATORS — Outputs all separators involved in all biconnected components (BCCs) that contain an odd cycle. This is a type-3 layer. An example output:



ANCHOR_SELF_CONFLICT_POLYGONS — Outputs a single target shape interacting with multiple MASK0 and MASK1 anchors that have no valid stitches between the anchors. This is a type-1 layer. An example output:

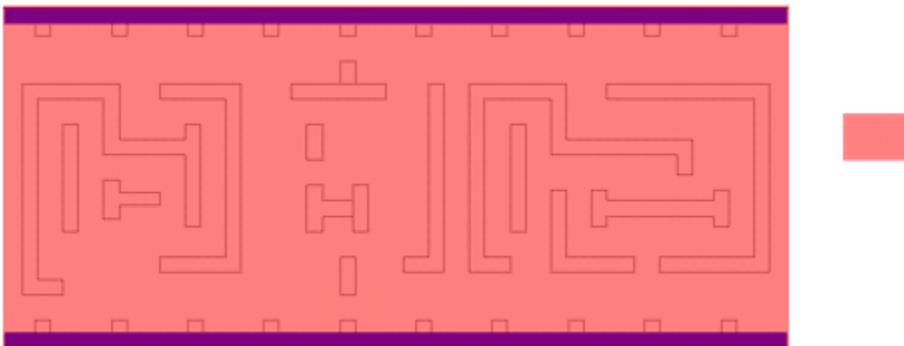


CONFLICT — Outputs a layer containing all target geometries in violation with any visualization error. The layer is formed by these violations and expands from these components to all polygons with connected separators. For errors stemming from a violation loop, all polygons in the biconnected component containing the loop are included. For errors stemming from anchor paths, all polygons in all biconnected components that connect two or more anchors in conflict are included. It stops expanding at anchors. This is a type-1 layer. An example output:



CONFLICT_MARKER — Outputs a layer containing a single shape covering the design extent if any conflict was found during DFM DP processing. This provides an

efficient means of determining at a glance whether the design contains errors or not. This is a type-1 layer. An example output:

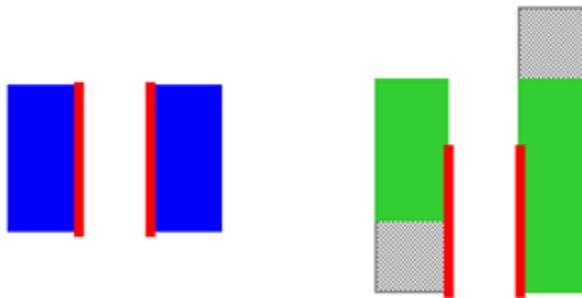


A single *conflict_marker* rectangle over the extent of the design if any error is found.

DIRECT_ANCHOR_CONFLICT_SEPARATORS — Outputs required separators that cause direct anchor violations. A direct anchor violation results from two causes:

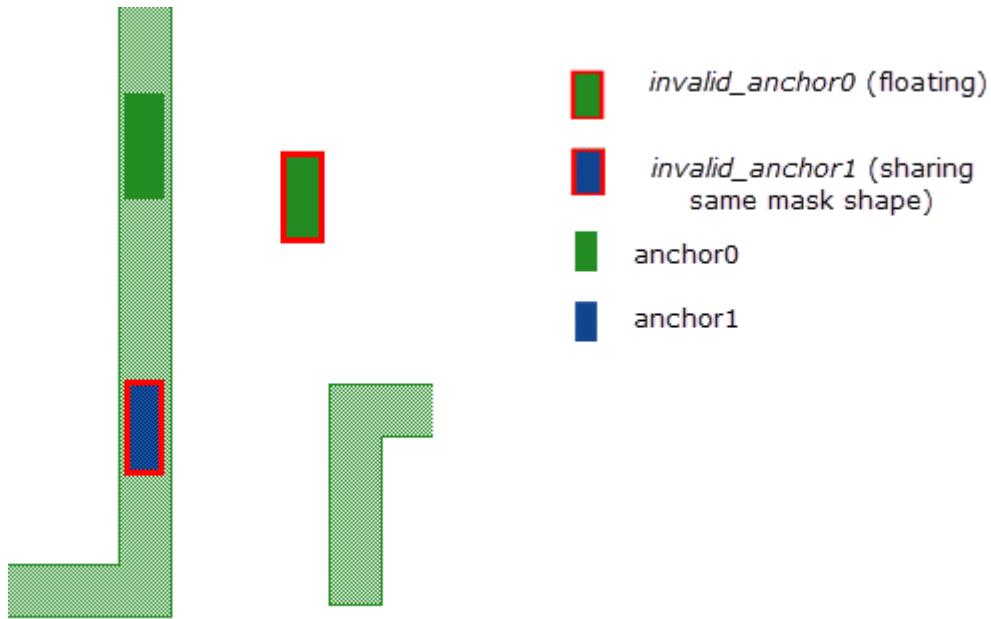
- A separator marked SAME found between two polygons that are anchored to different masks.
- A separator marked OPPOSITE found between two polygons that are anchored to the same mask.

If the separator also causes a self-conflict (a single polygon connecting both ends of an opposite separator), then it is reported as a self-conflict violation and not a direct anchor violation. This is a type-3 layer.



INVALID_ANCHOR0, INVALID_ANCHOR1 — Outputs a layer containing all MASK0 or MASK1 anchors not interacting with any target shape, or MASK1 anchors when MASK0 and MASK1 anchors are mutually interacting with the same target shapes (MASK0 anchors are arbitrarily accepted). MASK0 and MASK1

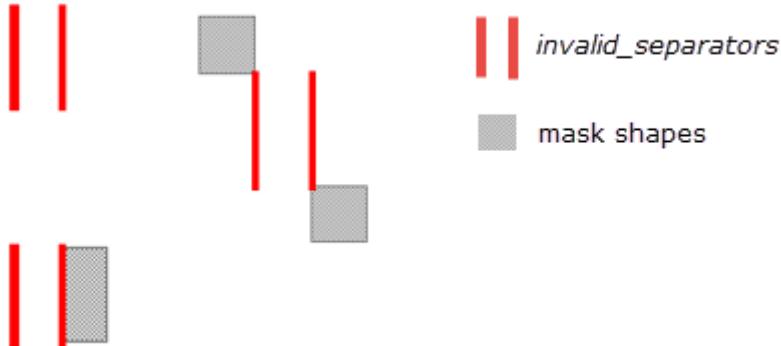
anchors that touch or overlap with stitches are also invalid. This is a type-1 layer. An example output of both error layers:



INVALID_SEPARATORS — Outputs all invalid separators from required or optional input separator layers. Each edge of a valid separator edge pair must touch exactly one target shape. See [Figure 3-9](#) on page 72 for more information. This is a type-3 layer. Invalid separators include:

- Separators not interacting with any target shapes.
- Separators interacting with more than two target shapes, or have an edge that interacts with more than one target shape.
- Separators having one edge not interacting with a target shape.
- Separators having one or more edges not interacting with any target shapes after stitches are removed.

Example separators include:



INVALID_STITCHES — Outputs all invalid stitches (either required or optional) from any stitch layer. Each stitch must cut a target shape in exactly two polygons (where the stitch touches only two shapes). For examples of good and bad stitches, see [Figure 3-11](#) on page 74. This is a type-1 layer. Invalid stitches may be categorized as:

- Stitches that touch other stitches from different stitch layers.
- Stitches specified as REQUIRED that touch separators with a priority of 0 or lower.
- Stitches that touch anchors.
- Stitches that extend outside the target layer or are entirely outside the target layer.
- Stitches not touching or overlapping with at least two target shapes (they fail to split a target shape into two parts).
- Stitches that split a target polygon into two pieces and touch one or more separators.
- REQUIRED or MAXIMIZE stitches that touch or overlap with more target shapes than there are colors (target shapes are counted after stitches are removed from them).

RING — Outputs a ring polygon similar to the SHORTEST_LOOP_LINE error layer, but adds rings for self-conflict violations and includes anchors for more extensive analysis. Ring detection keeps a complete ring intact within the lowest level of the design hierarchy. This is a type-1 layer. The RING output is also valid when UNFILTERED is specified.

Note

 When very large rings are found, they are ordered and then generated using path tracing to ensure that at least one ring is output. A warning message is generated before this process is initiated, because the computations may be time intensive.

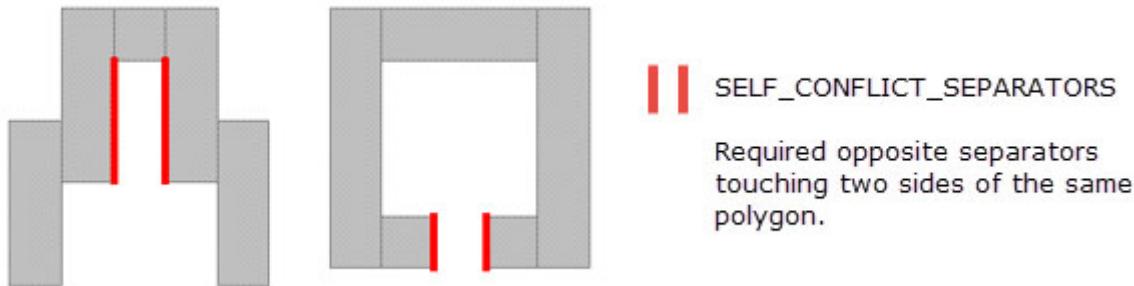
SELF_CONFLICT_POLYGONS — Outputs a layer containing the target shapes involved with self-conflict separators as determined by **SELF_CONFLICT_SEPARATORS**. This is a type-1 layer. Self-conflict polygons take various forms:



SELF_CONFLICT_SEPARATORS — Outputs all required separators that form a self-conflict. This is a type-3 layer. There are two types of self-conflicts:

- An OPPOSITE separator that touches the same polygon on two sides.

Figure 3-6. Self-Conflict With Opposite Separator Edge Pairs



- An OPPOSITE separator and a SAME separator that touch the same two polygons.

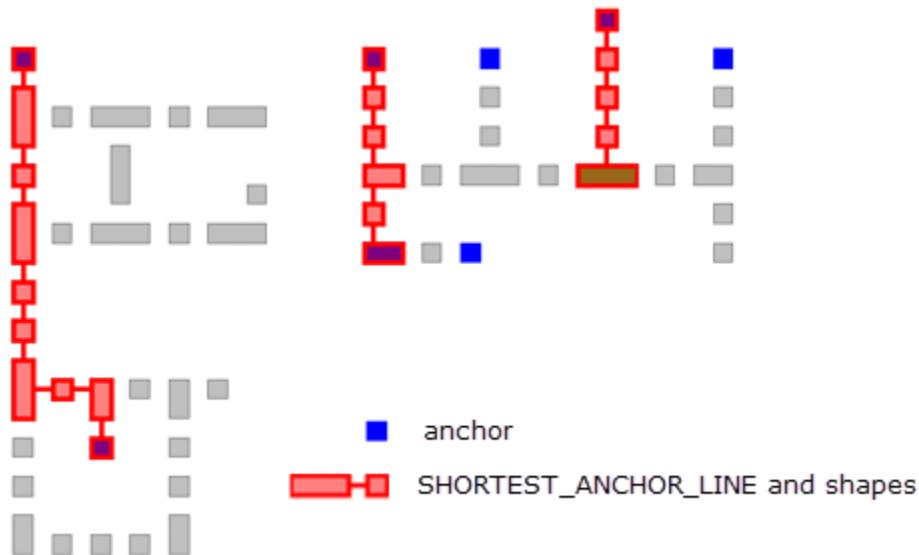
Figure 3-7. Self-Conflict With Opposite and Same Separator Edges



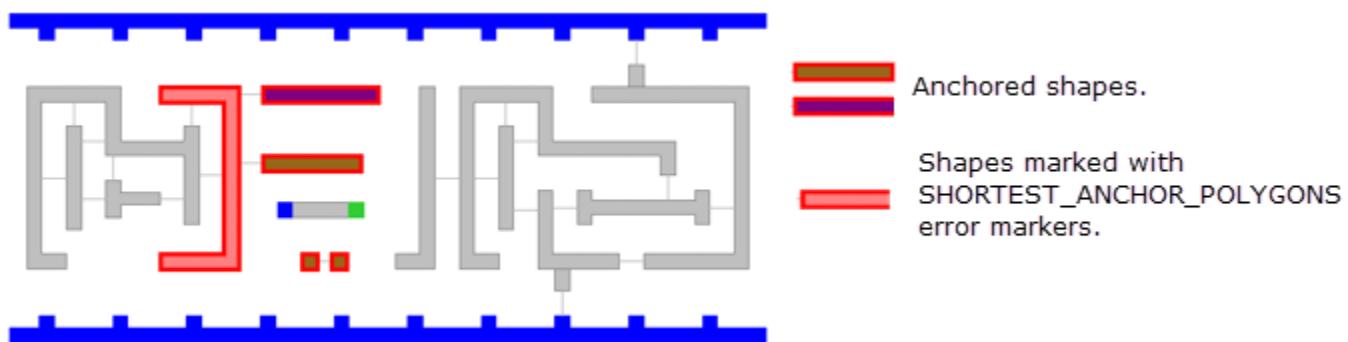
SHORTEST_ANCHOR_LINE — Paths between two anchors that form a violation. The length of these paths is minimized. Paths running through BCCs that have violation

loops are not included in this error layer. This is a type-1 layer. Shortest anchor lines highlight portions of any given short path:

Figure 3-8. SHORTEST_ANCHOR_LINE



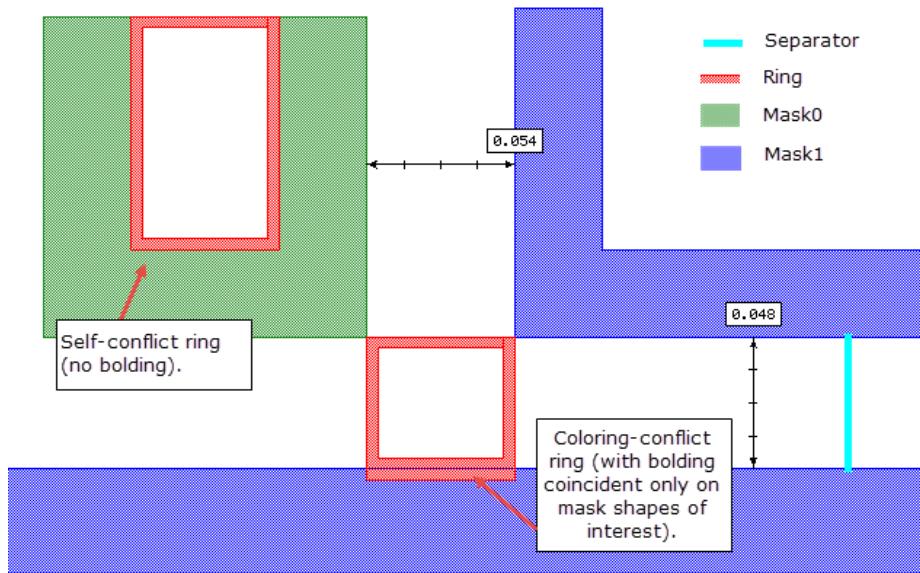
SHORTEST_ANCHOR_POLYGONS — Outputs mask shapes that are in the shortest anchor path. This is a type-1 layer. The SHORTEST_ANCHOR_POLYGONS error output contains a chain of one or more polygons connecting two or more anchored shapes:



SHORTEST_ANCHOR_SEPARATORS — Outputs edges representing the separators in the shortest anchor path (a type-3 layer).

SHORTEST_LOOP_LINE — Outputs an error visualization layer represented as a ring polygon. The shapes which SHORTEST_LOOP_LINES touch represent those polygons involved in the odd-cycle conflict. Where SHORTEST_LOOP_LINES rings appear extra wide and overlap involved shapes indicate those polygons that, after modification, would result in resolution of the odd-cycle conflict. Increasing the space between shapes at these locations within the SHORTEST_LOOP_LINES allows the adjacent polygons to be on the same mask, which removes the odd cycle. See [Figure 3-13](#) on page 77 for details on SHORTEST_LOOP_LINES exclusion areas.

This illustration shows two simple SHORTEST_LOOP_LINES, one resulting from a self-conflict error and one from an odd-cycle error. The self-conflict SHORTEST_LOOP_LINE has a normal ring width. The odd-cycle conflict SHORTEST_LOOP_LINE is wider where it overlaps a shape, indicating conflict resolution if it were to be modified:



```
m1_side2side = EXT m1_side < 0.06 OPPOSITE PARALLEL ONLY PROJ > 0
...
ring = DFM DP SHORTEST_LOOP_LINE M1 m1_side2side ...
ring {DFM COPY ring}
DRC CHECK MAP ring OASIS 10 out.oas MAXIMUM RESULTS ALL USER
```

SHORTEST_LOOP_LINE_UNMERGED — Outputs a layer comprising unmerged rings from different loops. DFM DP may be used to output unmerged loops as results database (RDB) format. Some loops output by SHORTEST_LOOP_LINE_UNMERGED may be disjointed; some may be detected in groups and reported as a single shape.

SHORTEST_LOOP_POLYGONS — Outputs a layer comprising of the mask shapes involved in an odd-cycle conflict.

SHORTEST_LOOP_SEPARATORS — Outputs a layer comprising of the edge pair separators involved in an odd-cycle conflict.

VIOLATED_OPTIONAL_SEPARATORS — Outputs a layer containing optional violated separators (those with a priority index greater than 0, a lower priority). Separators specifying the OPPOSITE keyword (see page 71) are considered violated if they are formed between two polygons assigned to the same mask, or if both ends of the separator touch the same polygon. Separators specifying the SAME keyword are considered violated if they are formed between two polygons with different mask assignments. Optional separators formed by anchor- and self-conflicts are included in this output. This output is a type-3 layer.

VIOLATED_OPTIONAL_STITCHES — Outputs a layer containing optional violated MINIMIZE and MAXIMIZE stitches (those with a priority index greater than 0, a lower priority). A MINIMIZE stitch is considered violated when used and a MAXIMIZE stitch is considered violated when not used. This output is a type-1 layer.

VIOLATED_SEPARATORS — Outputs a layer containing violated required separators (those with a priority index equal to 0, the highest priority). Errors generated by SELF_CONFLICT_SEPARATORS are included in this output. Errors generated by DIRECT_ANCHOR_CONFLICT_SEPARATORS are not included in this output. This output is a type-3 layer.

VIOLATED_STITCHES — Outputs the polygons of stitches specified with the REQUIRED argument of the stitch_layer keyword that are in violation. This output is a type-1 layer.

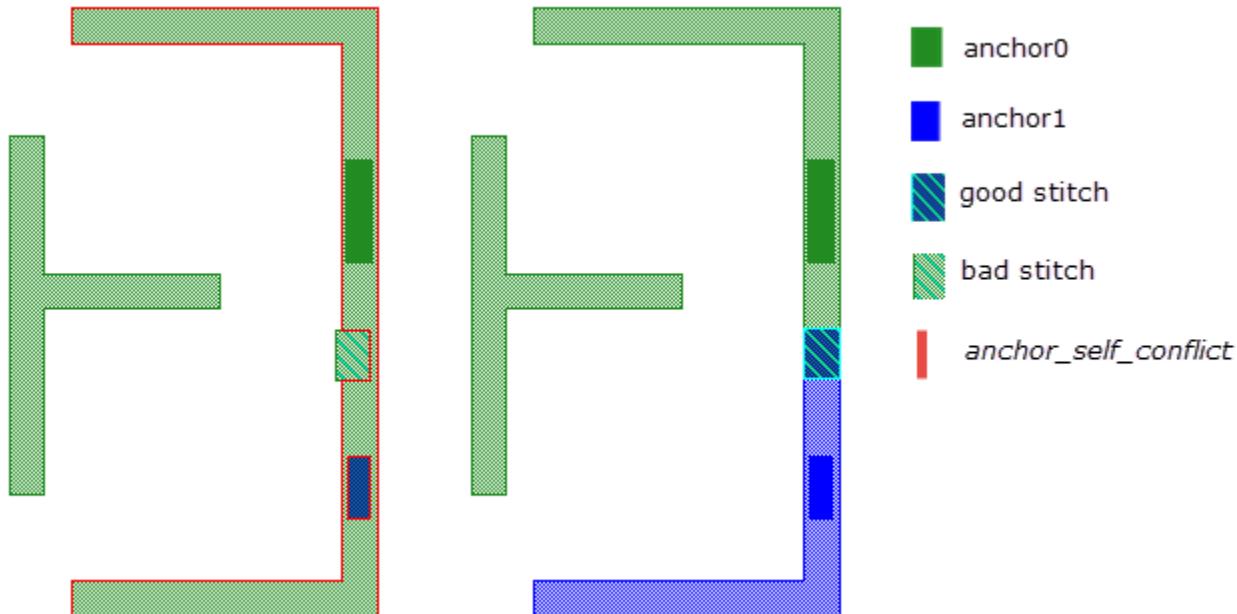
WARNING_LOOP_LINE — Outputs lines (visual rings) that comprise even cycles, containing opposite separators joining SHORTEST_LOOP-type error layers. This is a type-1 layer.

Keywords Enabled When UNFILTERED Is Specified

ANCHOR_CONFLICT — Outputs all shapes that are a subset of ALL_ANCHOR_POLYGONS and target shapes involved in direct anchor conflicts. This is a type-1 layer.

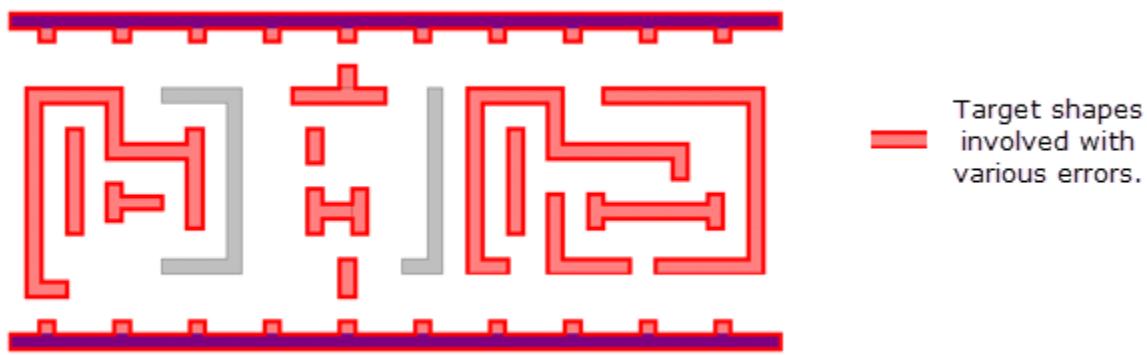
ANCHOR_PATH — Outputs thin polygon markers associated with conflicts between anchored target shapes. Each marker floats near or touches one side of all target shapes and separators associated with a color alternation conflict between two anchored target shapes. The output appears as a path similar to SHORTEST_ANCHOR_LINE, but includes direct anchor-to-anchor conflicts, and analyzes biconnected components that contain rings.

ANCHOR_SELF_CONFLICT — Outputs all target shapes interacting with multiple MASK0 and MASK1 anchors that have no valid stitches between the anchors. This is a type-1 layer. An example output:



ANCHOR_TREE — Outputs polygons that are found in any shortest anchor path error, including single-separator direct anchor violations.

CONFLICT — Outputs a layer containing all target geometries in violation with any visualization error. The layer is formed by these violations and expands from these components to all polygons with connected separators. For errors stemming from a violation loop, all polygons in the biconnected component containing the loop are included. For errors stemming from anchor paths, all polygons in all biconnected components that connect two or more anchors in conflict are included. It stops expanding at anchors. This is a type-1 layer. An example output:



LOOP — Outputs a layer with all polygons found within the set of shortest loops, and polygons having self-conflict violations. This is a type-1 layer.

RING — Outputs a ring polygon similar to the SHORTEST_LOOP_LINE error layer, but adds rings for self-conflict violations and includes anchors for more extensive analysis. Ring detection keeps a complete ring intact within the lowest level of the design hierarchy. This is a type-1 layer. The RING output is also valid when UNFILTERED is not specified.

Note

 When very large rings are found, they are ordered and then generated using path tracing to ensure that at least one ring is output. A warning message is generated before this process is initiated, because the computations may be time intensive.

SELF_CONFLICT — Outputs a layer containing the target shapes involved with self-conflict separators as determined by SELF_CONFLICT_SEPARATORS. This is a type-1 layer.

WARNING — Outputs a layer containing even cycles (non-violating loops in which only opposite separators exist) that share a separator contained in a SHORTEST_LOOP_LINE. This is a type-1 layer.

- ***target_layer***

A required argument specifying the layer to be decomposed. The target_layer must be a polygon layer.

- ***separator_layer* [‘{OPPOSITE | SAME} [*priority*]’]**

A required argument specifying a separator_layer. If no separator_layer is specified, you must specify an anchor_layer or stitch_layer. The separator_layer must be an edge pair (layer type-3). One or more separator layers may be specified.

OPPOSITE

Indicates polygons associated with the separator must be assigned to different mask layers.

SAME

Indicates polygons associated with the separator can both be assigned to the same mask layer.

[*priority*]

Separators can be specified as either required or optional during processing through the use of priority. The higher the priority value, the lower the priority of the separator. The priority value can be either 0 or a positive number. A separator of priority 0 enforces a required separator of highest priority. A separator specified with a priority value of 1 or greater is considered optional. The default priority value is 0.

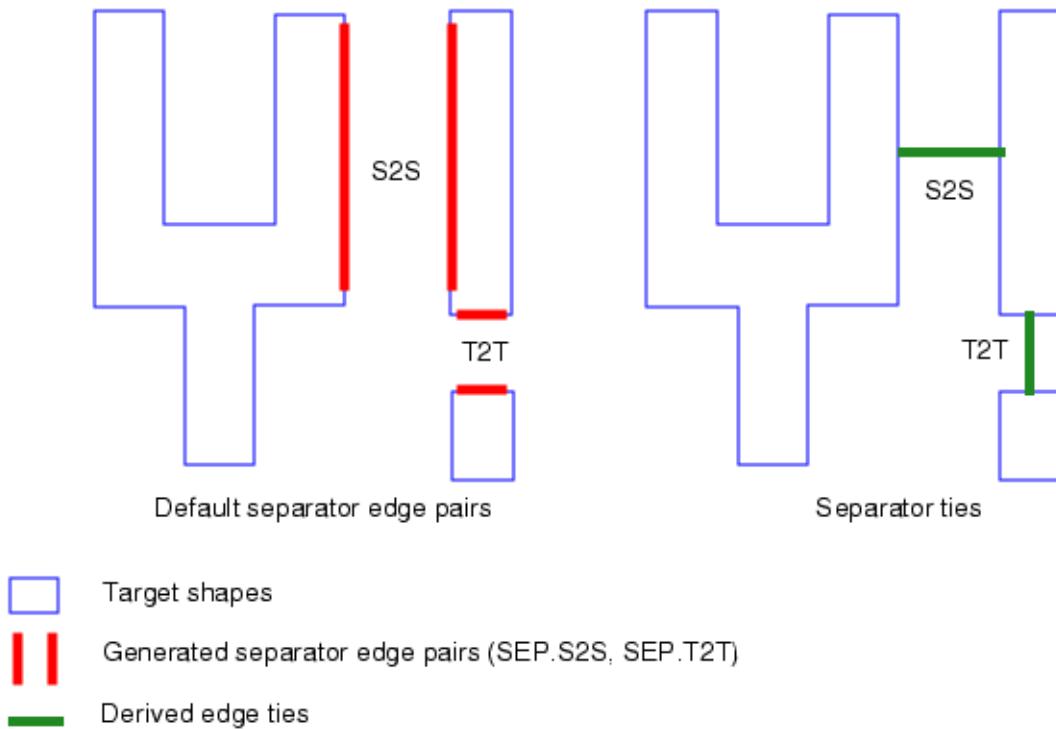
This SVRF code provides typical separator derivation, and [Figure 3-9](#) shows correctly formed separators.

```
m1_tip = CONVEX EDGE m1 ANGLE1 == 90 LENGTH1 >= 0.03
        ANGLE2 == 90 LENGTH2 >= 0.03 WITH LENGTH <= 0.06
m1_side = m1 NOT COIN EDGE m1_tip

m1_S2S = EXT m1_side < 0.06 OPPOSITE PARALLEL ONLY PROJ > 0
m1_T2S = EXT m1_tip m1_side < 0.065 PARALLEL ONLY
m1_T2T = EXT m1_tip < 0.085 PARALLEL ONLY
```

The input separator layer must consist of edge pairs only (layer type-3) and each separator of the edge pair must touch exactly one drawn shape. [Figure 3-9](#) demonstrates both edge- and tie-based separators. Combination polygon-separator error keywords always output tie-based separators.

Figure 3-9. Correct DFM DP Separators



- ***anchor_layer* ‘(ANCHOR {MASK0 | MASK1} [priority])’**

A required argument specifying one or more anchor_layers. If no anchor_layer is specified, you must specify a separator_layer or stitch_layer. Anchor layers must comprise either polygons or edges (layer types 1 or 2, respectively). Any number of anchor layers may be specified. Optionally, a priority may be assigned to the anchor. The value specified must be 1 or higher. 0 is the default value. The higher the priority value, the lower the priority of the anchor.

Anchor layers are identified by the keyword ANCHOR specified inside the parentheses. If no parentheses are specified, the argument is interpreted as a separator layer. Mask values for anchor layer are identified by the keywords MASK0 or MASK1 specified within the parentheses.

[Figure 3-10](#) demonstrates correct and incorrect anchors.

Figure 3-10. Good and Bad Anchors



Good anchors must be either polygons or paths and may touch, intersect, enclose, or be enclosed by target shapes.



Bad anchors have no interaction with target shapes.

- ***stitch_layer* ‘(‘**STITCH** {REQUIRED | USER | MINIMIZE *priority* | MAXIMIZE *priority*} ‘)’**

A required argument specifying one or more stitch_layers. If no stitch_layer is specified, you must specify a separator_layer or anchor_layer. Each of the stitch layers specified must be a polygon (layer type-1). The entirety of the stitch must be contained within the target shape and coincident with any target shape edge. A stitch is processed to result like a separator; after the stitch is subtracted from the drawn layer, the remaining space between target shapes determines whether the stitch is valid. For more information regarding invalid stitch criteria, see the error layer section entitled INVALID_STITCHES.

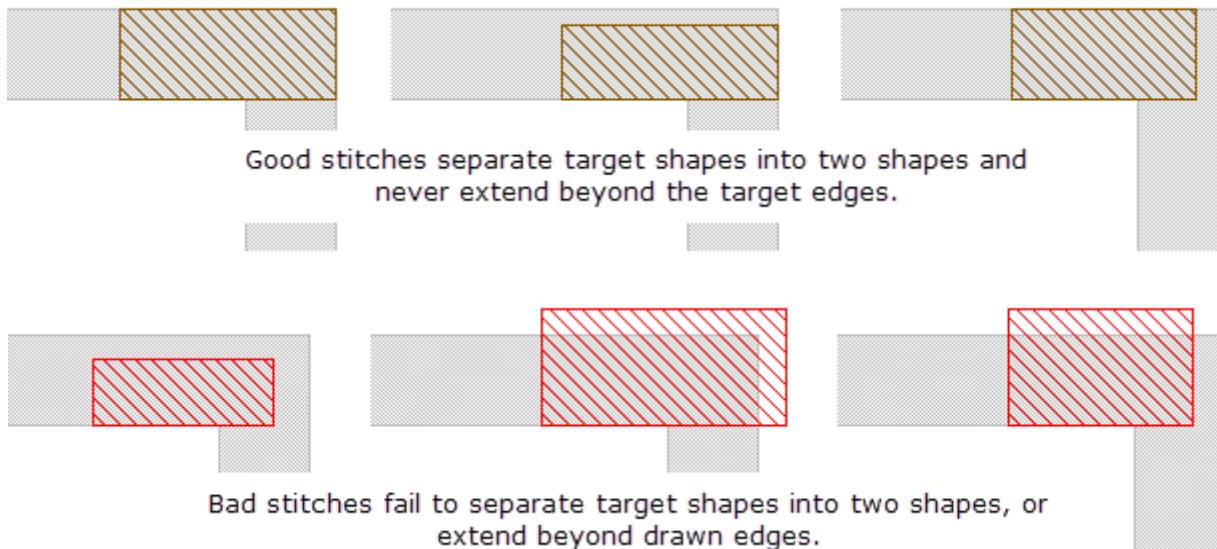
The REQUIRED and MAXIMIZE arguments indicate opposite-mask separators. The MAXIMIZE argument indicates stitches that are required or optional based on the value of its priority. Specifying MAXIMIZE priority as 0 is equivalent to specifying the REQUIRED argument. The MINIMIZE argument indicates optional stitches (those with potential placement on shapes without same-mask separators), and its priority must be specified as greater than 0.

No processing preferences exist between stitches of the same priority. If two stitches with the same priority are mutually exclusive, one of these two stitches will be arbitrarily chosen.

Stitches specified as USER stitches do not generate any separators. Separators for USER stitches should be specified using the separator_layer.

Figure 3-11 demonstrates differences between correct and incorrectly drawn stitches.

Figure 3-11. Good and Bad User Stitches



- **PREFER {MASK0 | MASK1}**

An optional keyword that assigns a majority of the polygons in a component after considering all priorities (required and optional) to the preferred mask. All isolated polygons are also placed on the preferred mask.

- **UNFILTERED**

An optional keyword that enables or disables output keywords supported by DFM DP. Specifying UNFILTERED allows use of output keywords that are compatible with NMDPC, and those including more types of violations combined within a smaller set of error layers. Not specifying UNFILTERED enables output keywords that provide better separation of errors by type to provide for easier debugging.

Note

 The set of layers output by specifying UNFILTERED are all generated concurrently; the separate set of layers output by not specifying UNFILTERED are also all generated concurrently. However, the two sets are not generated concurrently with each other, and may generate different MASK layers as well as inconsistent error layers. For this reason, Siemens EDA recommends against mixing output keywords from different sets.

- *waiver_layer*('WAIVER') ...

Optionally specifies a layer masking areas of the input layer within which rings are waived during error processing for DFM DP. Any number of waiver_layers may be specified.

Examples

Example 1 — Mask Output

```
CA_Mask0_Out = DFM DP MASK0 CA_Uncolored
    SEP.REQ SEP.OPT(SAME) CA_Mask0(anchor mask0) CA_Mask1(anchor mask1)

CA_Mask1_Out = DFM DP MASK1 CA_Uncolored
    SEP.REQ SEP.OPT(SAME) CA_Mask0(anchor mask0) CA_Mask1(anchor mask1)

CA_Mask0_Out {COPY CA_Mask0_Out} DRC CHECK MAP CA_Mask0_Out 15
CA_Mask1_Out {COPY CA_Mask1_Out} DRC CHECK MAP CA_Mask1_Out 16
```

Example 2 — Modifiers for Separators

```
// If separator type is not specified, default is opposite 0
CA_Mask0_Out = DFM DP MASK0 CA_Uncolored
    SEP.REQ(opposite 0) SEP.OPT(opposite 0)
    CA_Mask0(anchor mask0) CA_Mask1(anchor mask1)

CA_Mask1_Out = DFM DP MASK1 CA_Uncolored
    SEP.REQ(opposite 0) SEP.OPT(opposite 0)
    CA_Mask0(anchor mask0) CA_Mask1(anchor mask1)
```

Example 3 — Debugging and Analysis Layer Output

```
// Anchor self-conflict output (Type-1)
anchor_self_conflict = DFM DP ANCHOR_SELF_CONFLICT M1
    M1_separator(OPPOSITE 0)
    M1_anchor(ANCHOR MASK0)

// Conflict output (Type-1)
conflict = DFM DP CONFLICT M1
    M1_separator(OPPOSITE 0)
    M1_anchor(ANCHOR MASK0)

// Error output (Type-3)
self_conflict = DFM DP SELF_CONFLICT M1
    M1_separator(OPPOSITE 0)
    M1_anchor(ANCHOR MASK0)

direct_anchor_conflict = DFM DP DIRECT_ANCHOR_CONFLICT M1
    M1_separator(OPPOSITE 0)
    M1_anchor(ANCHOR MASK0)
```

Example 4 — Self-Conflicts

Results and derivation of the SELF_CONFLICT error layer:

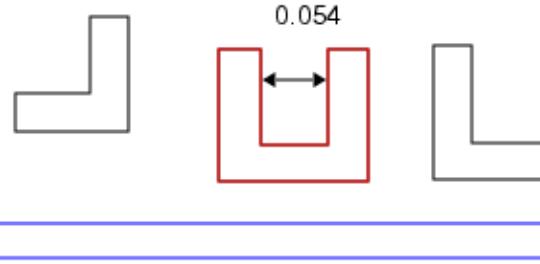
Figure 3-12. SELF_CONFLICT Error

Self conflict of a single shape from side-to-side separator derivation

 M1 mask0

 M1 mask1

 self_conflict error



```
m1_side2side = EXT m1_side < 0.06 OPPOSITE PARALLEL ONLY PROJ > 0
```

Depending on your rules, self_conflict may also generate errors similar to these:



Example 5 — Migrating from RET NMDPC to DFM DP

The following example shows the difference between RET NMDPC and DFM DP syntaxes and the simplification of DFM DP, in part because of no LITHO FILE block requirement:

RET NMDPC

```
LITHO FILE action_file [
    action action_layer 0 opposite
    action optional_action_layer 1 same
    anchor anchor_layer 0 mask0
]
mask0 = RET NMDPC action_layer target_layer optional_action_layer
        anchor_layer FILE action_file MAP MASK0
```

DFM DP

```
mask0 = DFM DP MASK0 target_layer action_layer(OPOSITIVE 0)
        optional_action_layer(SAME 1) anchor_layer(ANCHOR MASK0)
```

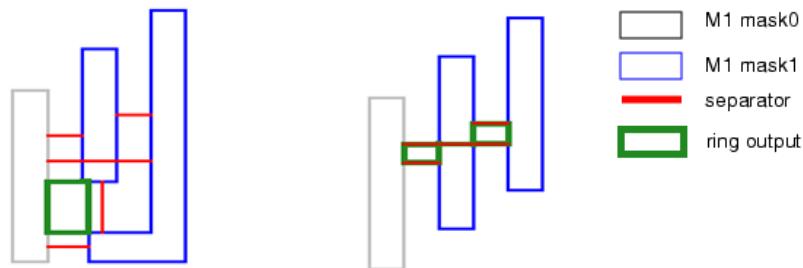
In addition, the following commands are rendered superfluous with DFM DP and are not supported:

- collector
- resolve
- seed

Example 6

This illustration shows that SHORTEST_LOOP_LINE errors are output where odd cycles are formed.

Figure 3-13. SHORTEST_LOOP_LINE Output



Ring warnings are output wherever odd cycles appear.

DFM DP MATCH

Multi-Patterning SVRF Commands

Provides rapid decomposition of similar polygon patterns.

Usage

```
output_layer = DFM DP MATCH
  {MASK0 | MASK1 | SYMMETRIC}
  input_layer
  marker_layer('MARKER size_value')
  [separator_layer [separator_layer...]]
```

Arguments

- *output_layer*

A required layer containing the output DFM DP MATCH. The *output_layer* is a polygon layer (layer-type 1).

- **MASK0 | MASK1 | SYMMETRIC**

One of three required arguments specifying DFM DP MATCH requirements:

- **MASK0, MASK1** — One of two keywords outputting anchors generated by detection of identical patterns around markers specified within the same command. For each marker, one anchor on either MASK0 or MASK1 is generated and saved to each group of input polygons connected by separators. The anchor is always generated from the input polygons. Multiple anchors may be generated for the same connected polygons if pattern detection areas of the multiple markers overlap. This is a type-1 layer.

- **SYMMETRIC** — A keyword outputting pattern detection areas for all patterns where anchor selection during decomposition would be ambiguous. This keyword reveals those patterns that DFM DP MATCH considers fully symmetric and is not able to select polygons for mask assignment without ambiguity. The output may be reviewed to determine whether patterns are fully symmetric or not, appearing identical in several orientations, or whether the metrics used to classify the polygons in the pattern fail to detect its asymmetry. This is a type-1 layer.

- *input_layer*

A required argument specifying the input design layer. The *input_layer* must be a polygon layer (layer-type 1).

- **marker_layer ('MARKER size_value')**

A required input layer. The areas of pattern detection are centered around each marker polygon, and extend by the *size_value* specified in the X- and Y-axis. If the marker polygon is a rectangle, the center of the pattern detection area is in the center of the rectangle. If the marker polygon is not rectangular, the pattern detection area is located at the center of

polygon mass. Irrespective of marker_layer shape, the pattern detection area always results in a square after applying the size_value. This is a type-1 layer.

- *separator_layer [separator_layer...]*

One or more optional separator layers. Separators specify same or different coloring constraints for input polygons. DFM DP MATCH uses separator layers to limit the matches generated to the output_layer. This is a type-3 layer.

Description

The DFM DP MATCH operation provides reproducible double-patterning decomposition for instances of identical patterns found within the input polygons. Each instance is indicated by a marker polygon. Input polygons within the pattern detection areas (those located near the marker polygons) are compared to determine whether they match or not. All matching patterns, regardless of orientation or hierarchy, are considered identical. Patterns are recognized heuristically, all detection areas around specified markers are analyzed and common patterns are then detected. Furthermore, identical patterns within different layouts result in the same suggested decomposition.

Hotspot Application

A typical application is to enforce identical decomposition around all hotspot locations specified by hotspot markers. DFM DP MATCH produces suggested anchor polygons that can be used in subsequent DFM DP operations to enforce the desired decomposition. For each hotspot marker, one anchor is produced for each group of input polygons connected by separators. In double patterning, anchoring just one polygon is sufficient to uniquely determine the colors of all polygons in a connected group. Only the separators connecting polygons within the pattern detection area are considered. Therefore, it is possible to generate multiple anchors for the same polygon group, if some of the separators forming the connected group are outside of the pattern detection area. Also, if the pattern detection areas of two different markers overlap, multiple anchors may be produced for the same group or even the same polygon. For these reasons, Siemens EDA recommends that anchors generated by DFM DP MATCH are used as optional anchors (mask assignment suggestions). If the anchors are treated as mandatory, the decomposition will fail unless all instances of the same pattern can be colored identically for all patterns.

Examples

Example 1 — DFM DP MATCH Usage

```
// Output mask0 anchors
match_anchor0 = DFM DP MATCH MASK0 V1
    via_marker(MARKER 0.5)

// Output mask1 anchors
match_anchor1 = DFM DP MATCH MASK1 V1
    via_marker(MARKER 0.5)

// Output marker shapes identifying symmetric patterns
symmetric_patterns = DFM DP MATCH SYMMETRIC V1
    via_marker(MARKER 0.5)

// Output marker shapes identifying symmetric patterns with the
// specified separator layer only
symmetric_patterns = DFM DP MATCH SYMMETRIC V1
    via_marker(MARKER 0.5) via_sep
```

DFM DP OPTIMIZE

Multi-Patterning SVRF Commands

Optimizes mask coloring through application of priority-based expressions applied to layers.

Usage

```
final_mask = DFM DP OPTIMIZE
  {MASK0 | MASK1}
  initial_mask_0('MASK0')
  initial_mask_1('MASK1')
  constraint_layer('CONSTRAINTS priority TARGET ['expression']')
  [constraint_layerN('CONSTRAINTS priority TARGET ['expression']') ...]
  expression_layer [expression_layerN ...]
```

Arguments

- **final_mask**

A required layer containing the final output layer of DFM DP OPTIMIZE. The final_layer is a polygon layer (layer-type 1).

- **MASK0 | MASK1**

A required argument specifying one of two mask keywords, indicating which mask to generate. There is no default mask keyword.

- **initial_mask_0 ('MASK0')** **initial_mask_1 ('MASK1')**

A required argument couplet specifying the mask layer that must be output by DFM DP, and its mapping mask keyword. The initial_mask layers must be polygons (layer-type 1).

- **constraint_layer ('CONSTRAINTS priority TARGET ['expression']' ...)**

A required layer containing the constraints of the target layer for the specified priority. This layer must be generated by [DFM DP OPTIMIZER_CONSTRAINTS](#). The constraint_layer must be a polygon layer (layer-type 1).

CONSTRAINTS priority

A required keyword and integer pair specifying a constraint. The priority is an integer inversely specifying the applied priority of the constraint; a lower number indicates a higher priority.

TARGET '['expression']'

A required keyword and argument specifying an numeric expression. Expressions provide customizable control over operational computations. Expressions define calculations using the specified target layers. An expression may contain numbers (including numeric variables), operators, parentheses and the functions listed in [Table 3-3](#).

The argument *x* (held by some expressions shown in the table) is a numeric argument, including numeric variables and other numeric-valued expressions. There is no compile-time or runtime exception checking on arguments of these functions and

unreasonable or undefined values may result from certain arguments. Division by zero does not satisfy any constraint. Parentheses have the highest precedence for the calculation of numeric values and may be used for grouping of terms.

Table 3-3. Expressions

Function	Definition
AREA('input_layer')	Area of input_layer in user units of length squared.
AREA_MASK0('input_layer')	Area of polygons that interact with mask0 target polygons on the input_layer in user units of length squared.
AREA_MASK1('input_layer')	Area of polygons that interact with mask1 target polygons on the input_layer in user units of length squared.
COUNT('input_layer')	Count of polygons on the input_layer.
COUNT_MASK0('input_layer')	Count of polygons that interact with mask0 target polygons on the input_layer.
COUNT_MASK1('input_layer')	Count of polygons that interact with mask1 target polygons on the input_layer.
PERIMETER('input_layer')	Total perimeter of polygons on the input_layer in user units.
PERIMETER_MASK0('input_layer')	Total perimeter of polygons that interact with mask0 target polygons on the input_layer in user units.
PERIMETER_MASK1('input_layer')	Total perimeter of polygons that interact with mask1 target polygons on the input_layer in user units.
SQRT('x')	The square root of x.
EXP('x')	The exponential (base e) of x.
LOG('x')	The natural logarithm of x
SIN('x')	The sine of x in radians.
COS('x')	The cosine of x in radians.
TAN('x')	The tangent of x in radians.
MIN('expression1, expression2')	Returns the lesser value of the two expressions. The expressions must be of the same form as the primary DFM DP OPTIMIZE expression, but without the square brackets.
MAX('expression1, expression2')	Returns the maximum value of the two expressions. The expressions must be of the same form as the primary DFM DP OPTIMIZE expression, but without the square brackets.

Unary and Binary Operators

Unary operators (+, -, !, ~) require one numeric argument, and unary operators have the same precedence. The + and - unary operators are positive and negative signs, respectively. The ! operator returns 0 (or false) if its argument is non-zero and 1 (or true) if its argument is 0. The ~ operator returns 0 (or false) if the argument is positive and 1 (or true) if the argument is non-positive. The Operators table shows some useful combinations of the ~ and ! operators.

Table 3-4. Unary ~ and ! Operators

x	!(x)	!!(x)	!(x-1)	!!(x-1)	~x	~~x	~(x-1)	~~(x-1)
3	0	1	0	1	0	1	0	1
2	0	1	0	1	0	1	0	1
1	0	1	1	0	0	1	1	0
0	1	0	0	1	1	0	1	0
-1	0	1	0	1	1	0	1	0
-2	0	1	0	1	1	0	1	0
-3	0	1	0	1	1	0	1	0

The binary operators ^, *, /, +, and - require two numeric arguments. The ^ operator is the same as the C language *pow()* function, where x^y means x raised to the y power. The *, /, +, and - operators are multiplication, division, addition, and subtraction, respectively, and have their customary precedence applied. The ^ operator has the same precedence as * and /. The binary operators // (OR) and && (AND) are the same as the C language operators of the same type, except they have the same precedence as the binary operators + and -. They must be specified with numeric-valued inputs. For example, AREA(via) && AREA(met) return 1 if both inputs are non-zero, and 0 otherwise. AREA(via) // AREA(met) return 1 if either input is non-zero, and 0 otherwise.

Description

DFM DP OPTIMIZE evaluates expressions enabling mask coloring. The results of mask output from DFM DP are based on calculations using additional polygonal layers. A component is defined as a group of polygons sharing separator edges between them. All polygons within a component are constrained based on priority and are subject to mutual color flipping. For a given non-anchored component, if the resulting value of the expression is greater than or equal to 0, then the color of the polygons in that component is not changed. If the resulting value is negative, coloring of polygons in the component is enabled. We can also change their coloring where the priority allows.

The DFM DP OPTIMIZE command is the culmination of a three-step process:

1. Generate the initial mask layers with **DFM DP**.
2. Obtain constraints of given priorities by executing
DFM DP OPTIMIZER_CONSTRAINTS. Priority values specified have an inverse

relationship to application; the lower a priority number is, the higher the priority application. The constraints identify components formed with a given priority (where separators of a priority are less than or equal to the specified priority) and also determine if a component is anchored. A component can be anchored if the anchor has a priority less than or equal to the specified priority.

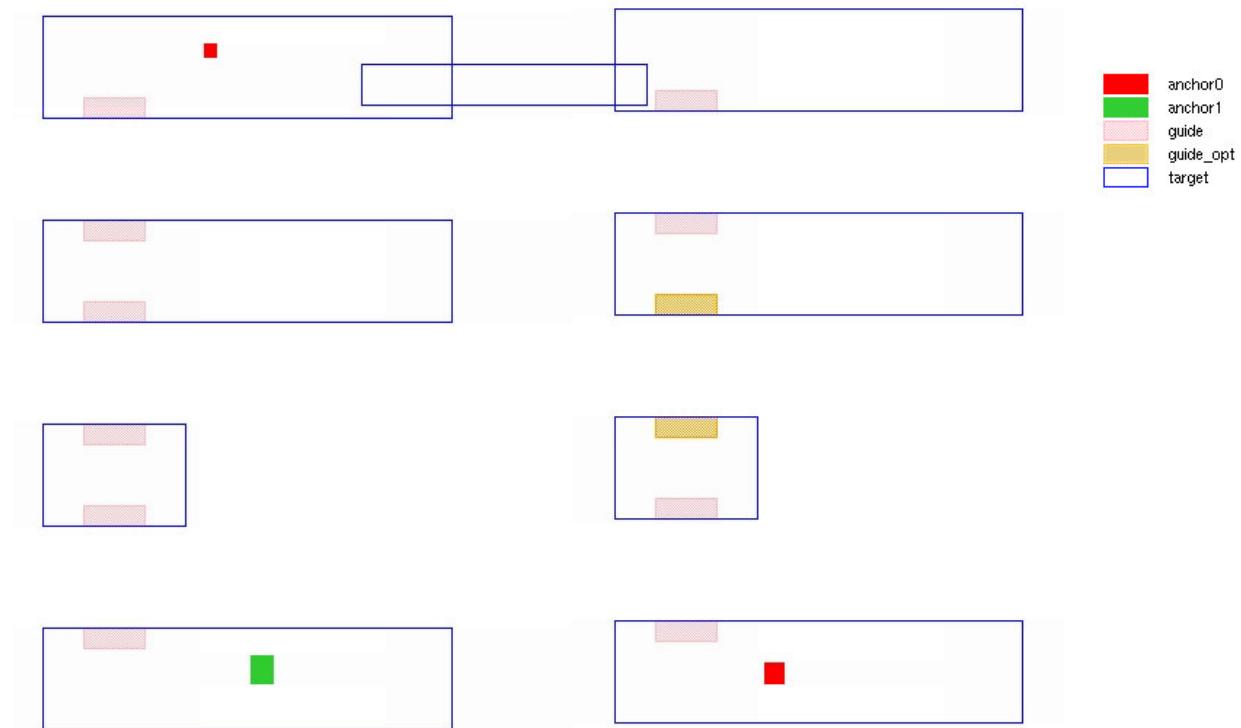
3. Execute the DFM DP OPTIMIZE operation using the mask output, constraints, and by evaluating a required expression. The result of the expression determines whether polygon color within a component should stay the same or be flipped.

Examples

A Sample Design to be Processed by DFM DP OPTIMIZE.

This design contains a target layer, separator guides and seeds. Seeds are polygons that guide the evaluation of expressions. Each of these layers are processed by DFM DP OPTIMIZE to obtain an optimal coloring solution. The constraints used in this example employ COUNT_MASK expressions. Where expressions evaluate to negative values, coloring is enabled.

Figure 3-14. Original Design



Original design with anchors and separator guides. Dark yellow guides are subsequently processed as optional. Pink guides are required. The red anchor in the lower right sets up a potential conflict resolved by DFM DP OPTIMIZE.

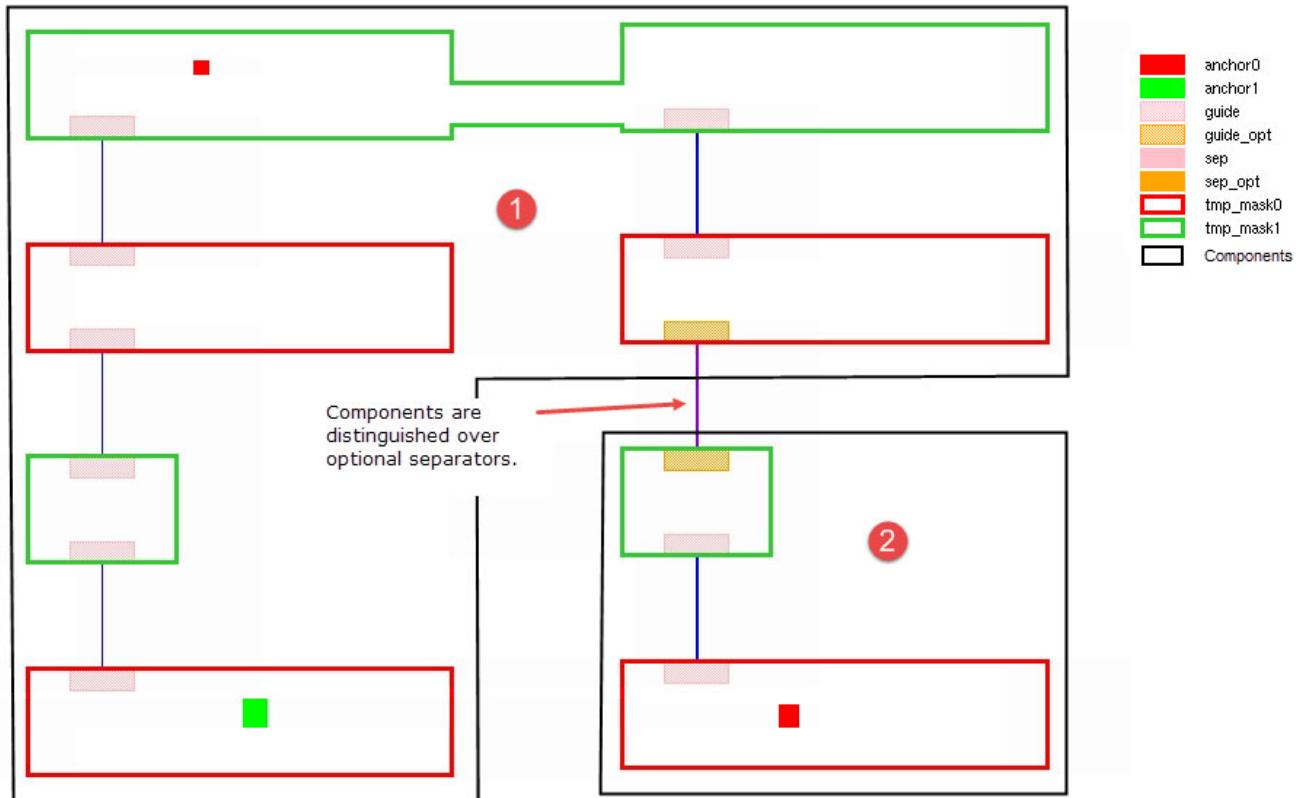
Step 1 — Separator Derivation and DFM DP Generation of Initial Mask Layers.

```
// Separator derivation
sep = EXT (guide COINCIDENT EDGE target) OPPOSITE <= 0.5
sep_opt = EXT (guide_opt COINCIDENT EDGE target) OPPOSITE <= 0.5

// Output initial layer for mask0 (highlighted in red)
tmp_mask0 = DFM DP MASK0 target
sep
sep_opt(opposite 1)

// Output initial layer for mask1 (highlighted in green)
tmp_mask1 = DFM DP MASK1 target
sep
sep_opt(opposite 1)
```

Figure 3-15. Design as Processed by DFM DP



The separators may be derived from guides that can be placed by design. The guides are processed to produce required or optional separators, in this case sep and sep_opt, respectively. The initial mask0 and mask1 output layers, tmp_mask0 and tmp_mask1, are then output by DFM DP using the MASK0 and MASK1 keywords, respectively. In two cases, the polygon colors do not match the anchors applied. The constraints are applied in the next step, correcting this problem.

Step 2 — DFM DP OPTIMIZER_CONSTRAINTS Generates Priority-Based Constraints.

As seen in [Figure 3-15](#), two independent chains of polygons connected by required separators are represented by constraint_0. For this example, the boundaries of the independent polygon chains are determined over optional separators.

```
// Output constraints by priority (highlighted in magenta)
// These constraint layers are not viewable.
// This constraint is priority 0.
// Constraints are formed using priority-0 edges only.
constraint_0 = DFM DP OPTIMIZER_CONSTRAINTS(0)
    target
    sep
    sep_opt(opposite 1)

// This constraint is priority 1.
// Constraints are formed using priority-0 and priority-1 edges.
constraint_1 = DFM DP OPTIMIZER_CONSTRAINTS(1)
    target
    sep
    sep_opt(opposite 1)
```

Step 3 — DFM DP OPTIMIZER Generates Final Masks Using Expressions Specified by Priority-Based Constraints.

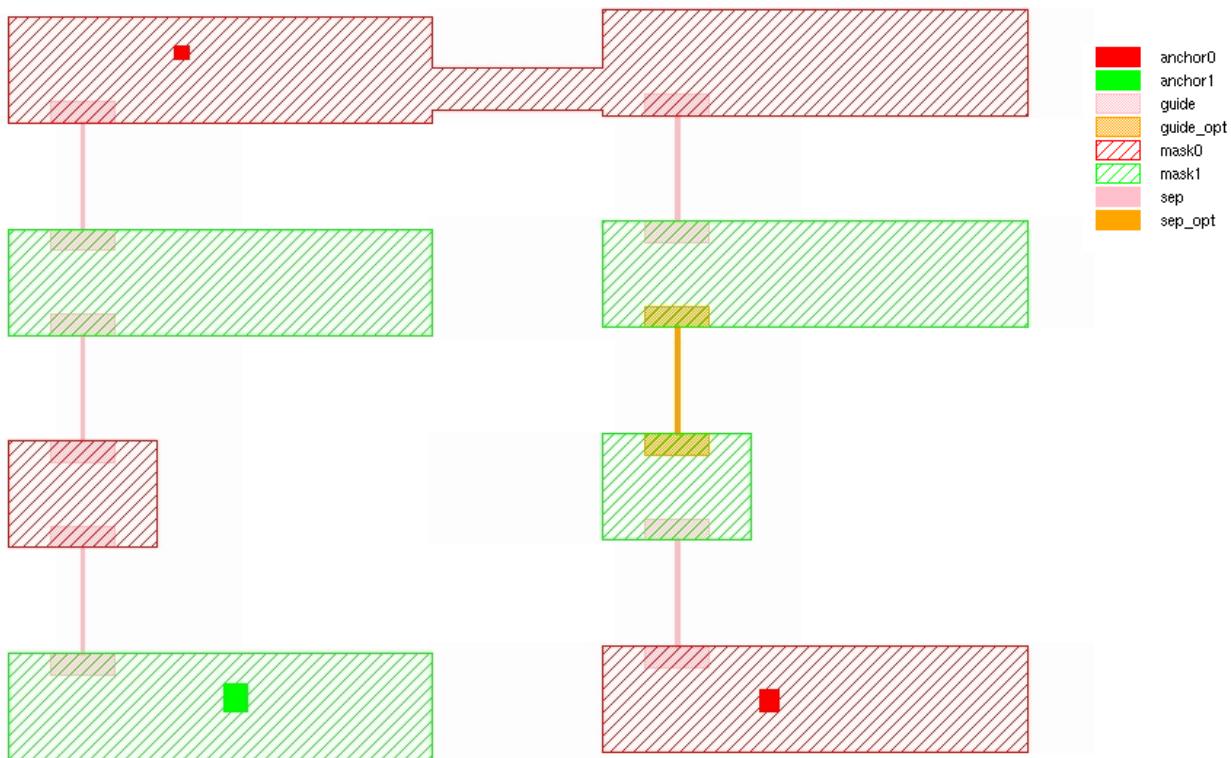
For constraint_0, component 1 as depicted in [Figure 3-15](#), the applied expression, $2 * COUNT_MASK0(anchor0) - COUNT_MASK0(anchor1)$ evaluates to $2 * 0 - 1 = -1$, a negative number enabling optimization-based color changes. For component 2, the same expression evaluates to $2 * 1 - 0 = 2$, a non-negative number resulting in no color changes.

For constraint_1, both components 1 and 2, as depicted in [Figure 3-15](#), the applied expression, $2 * COUNT_MASK0(anchor0) - COUNT_MASK0(anchor1)$ evaluates to $2 * 1 - 1 = 1$, a positive number disabling optimization-based color changes. However, this constraint is of a lower priority; no color changes take effect on this component.

```
// Output final mask0
mask0 = DFM DP OPTIMIZE MASK0
    tmp_mask0(MASK0)
    tmp_mask1(MASK1)
    constraint_0(CONSTRAINTS 0
        TARGET [ (2 * COUNT_MASK0(anchor0)) - COUNT_MASK0(anchor1) ]
    )
    constraint_1(CONSTRAINTS 1
        TARGET [ (2 * COUNT_MASK1(anchor1)) - COUNT_MASK0(anchor0) ]
    )
    anchor0 anchor1

// Output final mask1
mask1 = DFM DP OPTIMIZE MASK1
    tmp_mask0(MASK0)
    tmp_mask1(MASK1)
    constraint_0(CONSTRAINTS 0
        TARGET [ (2 * COUNT_MASK0(anchor0)) - COUNT_MASK0(anchor1) ]
    )
    constraint_1(CONSTRAINTS 1
        TARGET [ (2 * COUNT_MASK1(anchor1)) - COUNT_MASK0(anchor0) ]
    )
    anchor0 anchor1
```

Figure 3-16. Final Optimized Design After Applied Constraints



The optimized design result shows the lower right polygon with its color changed to mask0 from mask1 to adopt the anchor named anchor0 placed on it. To facilitate this color change, the two mask1 polygons on the right are allowed to be placed on the same mask because their connecting separator is optional.

DFM DP OPTIMIZER_CONSTRAINTS

Multi-Patterning SVRF Commands

Generates constraint layers from DFM DP based on priority.

Usage

```
constraint_layer = DFM DP OPTIMIZER_CONSTRAINTS('priority')
    target_layer
    {separator_layer ['('SAME | OPPOSITE')']} ...
    {anchor_layer ['('ANCHOR {MASK0 | MASK1}')']} ...
```

Arguments

- **constraint_layer**

A required output layer containing the constraints of the target layer for the specified priority. This layer is subsequently input by [DFM DP OPTIMIZE](#). The constraint_layer must be a polygon layer (layer-type 1).

Note

 Inputting the constraint_layer to any command other than DFM DP OPTIMIZE results in the following runtime error message:

```
litho.transcript:ERROR:
    Error OPR1 on line 111 of rules_file.svrf -
    Layers created by DFM DP OPTIMIZER_CONSTRAINTS
    cannot be used by this operation.
```

- **priority**

A required argument specifying the value of the priority. Priorities have an inverse value and application relationship; the lower the value, the higher the applied priority of the constraint. The target_layer must be a polygon layer (layer-type 1).

- **target_layer**

A required argument specifying the target design layer. The target_layer must be a polygon layer (layer-type 1).

- **separator_layer ['('SAME | OPPOSITE')']**

A required argument specifying one or more separator layers previously specified by DFM DP. Each separator and mask color association: SAME associates separators between shapes of the same mask; OPPOSITE associates separators between shapes of different masks. OPPOSITE is the default mask association. Any number of separator_layers may be specified. Separator layers must be an edge pair (layer-type 3).

- **anchor_layer ['('ANCHOR {MASK0 | MASK1}')']**

An optional argument specifying an anchor layer previously specified by DFM DP. Each anchor layer must be paired with the ANCHOR keyword and either the MASK0 or MASK1

arguments. Any number of anchor_layers may be specified. Anchor layers must be a polygon (layer-type 1).

Description

DFM DP OPTIMIZER_CONSTRAINTS specifies the constraints of given priorities. The constraints determine if a component is anchored and then apply the specified priority to the component (components may contain separators with priorities less than or equal to the specified priority). A component can be anchored if the anchor has a priority less than or equal to the specified priority. DFM DP OPTIMIZE then generates final mask layers using these constraints.

Examples

DFM DP OPTIMIZER_CONSTRAINTS Generates Priority-Based Constraints.

```
// Constraints output with priority of 1
constraints_1 = DFM DP OPTIMIZER_CONSTRAINTS(1)
    M1_target M1_sep
    M1_opt_sep1(OPPOSITE 1)
    M1_opt_sep2(OPPOSITE 2)
    M1A0 (ANCHOR MASK0) M1A1 (ANCHOR MASK1)

// Constraints output with priority of 2
constraints_2 = DFM DP OPTIMIZER_CONSTRAINTS(2)
    M1_target M1_sep
    M1_opt_sep1(OPPOSITE 1)
    M1_opt_sep2(OPPOSITE 2)
    M1A0 (ANCHOR MASK0) M1A1 (ANCHOR MASK1)
```

DFM MP

Multi-Patterning SVRF Commands

Invokes Calibre triple- or quadruple-patterning. Splits the target layer into three or four masks and generates error layers.

Usage

```
output_layer = DFM MP {3 | 4}
  operation_keyword
  target_layer
  separator_layer ['({ SAME | OPPOSITE } [priority])'] ...
  anchor_layer '({ ANCHOR { MASK1 | MASK2 | MASK3 | MASK4 } })' ...
  stitch_layer '({ STITCH
    REQUIRED | USER | MINIMIZE priority | MAXIMIZE priority })' ...
```

Arguments

- ***output_layer***
A required output layer that can be saved to the output database and used for final decomposed mask output or error visualization analysis.
- **3 | 4**
A required parameter that declares the number of masks that DFM MP must process. You specify 3 for triple-patterning and 4 for quadruple-patterning.
- ***operation_keyword***
Any of the following required keywords directing DFM MP to perform its stated operation. You must specify at least one *operation_keyword* per DFM MP command. Subsequent *operation_keyword* keywords are specified on subsequent DFM MP commands.
operation_keyword keywords are classified in three keyword groups, Mask Output, Error Visualization and Graph Reduction.

Mask Output Keywords

- MASK1** — A required keyword that outputs a layer containing MASK1 geometries.
- MASK2** — A required keyword that outputs a layer containing MASK2 geometries.
- MASK3** — A required keyword that outputs a layer containing MASK3 geometries.
- MASK4** — A required keyword that outputs a layer containing MASK4 geometries.
Only used for DFM MP 4.

Error Visualization Keywords

For error visualization output, the output type may be one of the following keywords, each specified on a separate DFM MP command line. For more information regarding error visualization and the implementation of these keywords, see the section entitled “[MP Visualization and Error Resolution](#)” on page 335.

ANCHOR_SELF_CONFLICT_POLYGONS — Outputs a layer containing a subset of the conflict layer containing any single target shape with a color conflict between anchors on that shape. This error occurs when a single shape is anchored for multiple masks such as MASK1 through MASK n . This is a type-1 layer.

CONFLICT — Outputs a layer containing all target geometries in violation with any visualization error. The layer is formed by coloring violations (where two polygons are on the same mask, therefore forcing a separator between them), then expands from these components to all polygons with connected separators. It stops expanding at anchors or edges that are removed by reduction. This is a type-1 layer.

DIRECT_ANCHOR_CONFLICT_SEPARATORS — Outputs a layer containing separators between two polygons that are both anchored to the same mask. This is a type-3 layer.

EQUIV_LOOP_COMBINED — Outputs a layer containing merged polygons and edge separators of an equivalence loop where two polygons have a forced equivalence path and a separator between them. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

EQUIV_LOOP_POLYGONS — Outputs a layer containing all the polygons of an equivalence loop. An equivalence loop violation is defined as two polygons having a forced equivalence path and a separator between them. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

EQUIV_LOOP_EQUIV_POLYGONS — Outputs a layer containing only those polygons of an equivalence loop that have a forced equivalence. An equivalence loop violation is defined as two polygons having a forced equivalence path and a separator between them. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

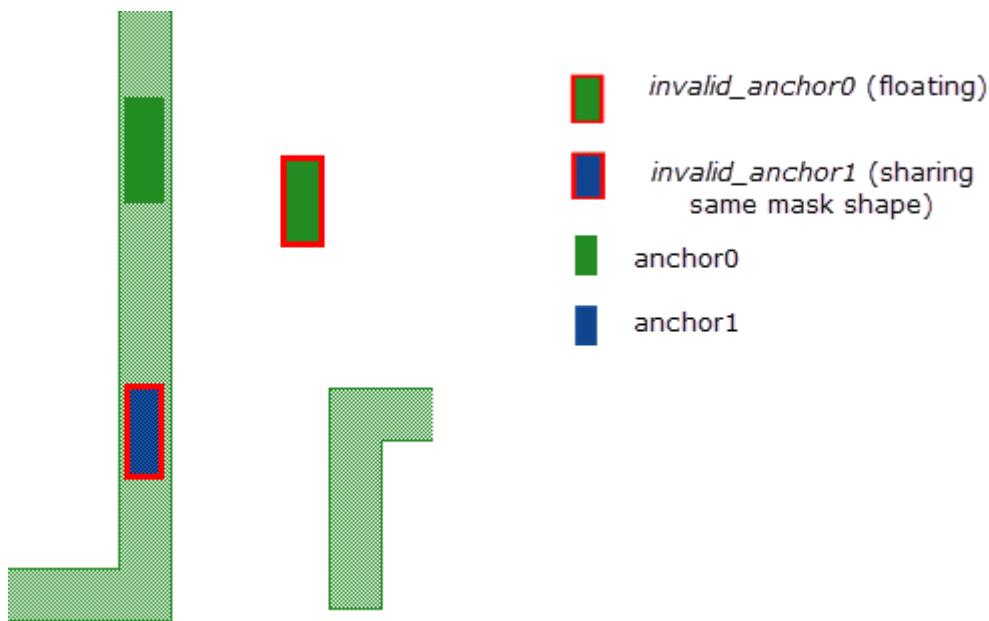
EQUIV_LOOP_SEPARATORS — Outputs a layer containing all separators of an equivalence loop. An equivalence loop violation is defined as two polygons having a forced equivalence path and a separator between them. This visualization keyword is only available for a mask number of 3. This is a type-3 layer.

EQUIV_LOOP_VIOLATION_SEPARATORS — Outputs a layer containing the violation separators of an equivalence loop. An equivalence loop violation is defined as two polygons having a forced equivalence path and a separator between them. This visualization keyword is only available for a mask number of 3. This is a type-3 layer.

INVALID_ANCHOR n — Outputs a layer containing all invalid anchors assigned to mask n , where n must be less than or equal to the mask count. This is a type-1 layer. Invalid anchors may be categorized in the following ways:

- Anchors not interacting with any target shapes
- Anchors of different masks interacting with the same target shape

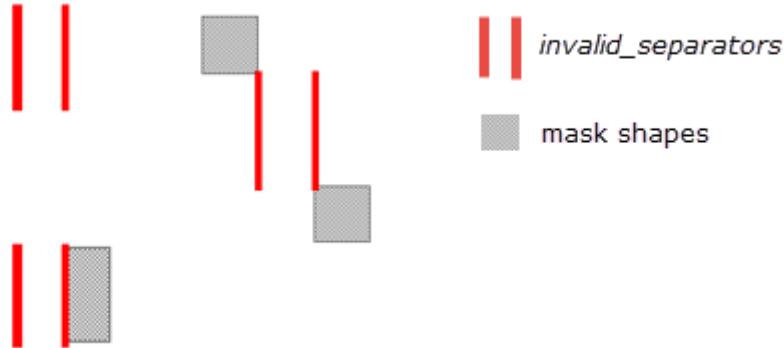
An example output of INVALID_ANCHOR error layer output:



INVALID_SEPARATORS — Outputs a layer containing invalid separators from required or optional input separator layers. Each edge of a valid separator edge pair must touch exactly one target shape. This is a type-3 layer. Invalid separators include:

- Separators not interacting with any target shapes.
- Separators interacting with more than two target shapes, or have an edge that interacts with more than one target shape.
- Separators having one edge not interacting with a target shape.
- Separators with one, three, or four edges, or that have a trivial edge.
- Separators touching a number of target shapes greater than or equal to the mask count after stitches are removed.
- Separators having one or more edges not interacting with a target shape after user stitches are removed.

Example separators include:



INVALID_STITCHES — Outputs a layer containing invalid stitches (either required or optional) that are derived from any separator layer. Each stitch must cut a target shape in exactly two polygons (where the stitch touches only two shapes). For examples of good and bad stitches, see [Figure 3-11](#) on page 74. This is a type-1 layer. Invalid stitches include:

- Stitches that touch or overlap other stitches from different stitch layers.
- Stitches that touch or overlap anchors.
- Stitches that extend outside or are entirely outside a target shape.
- Stitches that do not split a target shape into two separate parts after stitch removal.
- Non-user stitches that touch one or more separators.
- Stitches specified as REQUIRED or MAXIMIZE 0 that split a target shape into a number of shapes exceeding the mask count after stitches are removed.
- Stitches specified as REQUIRED or MAXIMIZE 0 that split a target shape into a number of shapes equal to or greater than the mask count, that also touch one or more required separators, after stitches are removed.

OPPOSITE_ANCHOR_EQUIV_POLYGONS — Outputs a layer containing polygons that must be on the same mask within an opposite-anchor path. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

OPPOSITE_ANCHOR_COMBINED — Outputs a layer containing merged polygons and separators within an opposite-anchor path. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

OPPOSITE_ANCHOR_POLYGONS — Outputs a layer containing polygons within an opposite-anchor path. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

OPPOSITE_ANCHOR_SEPARATORS — Outputs a layer containing separators within an opposite-anchor path. This visualization keyword is only available for a mask number of 3. This is a type-3 layer.

REDUCED_SEPARATORS_CLUSTER — Outputs a layer containing all remaining edges after reductions have been performed. This is a type 3 layer.

REDUCED_TARGET — Outputs a layer containing all target geometries remaining after reductions have been performed. This is a type 1 layer.

REDUCED_TARGET_POLYGONS — Outputs a layer containing all target geometries remaining after reductions have been performed. This is a type-1 layer.

REDUCED_TARGET_SEPARATORS — Outputs a layer containing all remaining edges after reductions have been performed. Each edge pair contains a property identifying which component it belongs to. If an edge pair belongs to multiple components, it is duplicated and each edge pair is given a unique component ID. This is a type-3 layer.

REMAINING_VIOLATIONS_COMBINED — Outputs a layer containing merged polygons and separators belonging to components from the CONFLICT layer which have no violations associated with them. This is a type-1 layer.

REMAINING_VIOLATIONS_POLYGONS — Outputs a layer containing polygons belonging to components from the CONFLICT layer which have no violations associated with them. This is a type-1 layer.

REMAINING_VIOLATIONS_SEPARATORS — Outputs a layer containing separators belonging to components from the CONFLICT layer which have no violations associated with them. This is a type-3 layer.

SAME_ANCHOR_COMBINED — Outputs a layer containing merged polygons, anchors and violation separators of a same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

SAME_ANCHOR_EQUIV_POLYGONS — Outputs a layer containing polygons that must be on the same layer within a same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

SAME_ANCHOR_POLYGONS — Outputs a layer containing polygons of a same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This visualization keyword is only available for a mask number of 3. This is a type-1 layer.

SAME_ANCHOR_SEPARATORS — Outputs a layer containing separators of the same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This visualization keyword is only available for a mask number of 3. This is a type-3 layer.

SAME_ANCHOR_VIOLATION_SEPARATORS — Outputs a layer containing separators participating in a same-anchor path violation (those found by

SAME_ANCHOR_POLYGONS) but are not part of the equivalence path (each same-anchor path violation contains exactly one separator in the equivalence path). This visualization keyword is only available for a mask number of 3. This is a type-3 layer.

SELF_CONFLICT_SEPARATORS — Outputs a layer containing the edges of separators marking a violation from one side of a polygon to another side of the same polygon. This is a type-3 layer.

SELF_CONTAINED_VIOLATION_POLYGONS — Outputs a layer containing polygons that form simple, self-contained structures that are inherently uncolorable. In triple-patterning, these structures are even wheels (wheel graphs with an even number of nodes) and K4s (four shapes that each have a separator touching the other three shapes). In quadruple-patterning, these structures are K5s (five shapes that each have a separator touching the other four shapes). This output is a type-1 layer.

SELF_CONTAINED_VIOLATION_SEPARATORS — Outputs a layer containing separators that form simple, self-contained structures that are inherently uncolorable. In triple-patterning, these structures are even wheels (wheel graphs with an even number of nodes) and K4s (four shapes that each have a separator touching the other three shapes). In quadruple-patterning, these structures are K5s (five shapes that each have a separator touching the other four shapes). This output is a type-3 layer.

SELF_CONTAINED_VIOLATION_COMBINED — Outputs a layer containing merged polygons and separators that form simple, self-contained structures that are inherently uncolorable. In triple-patterning, these structures are even wheels (wheel graphs with an even number of nodes) and K4s (four shapes that each have a separator touching the other three shapes). In quadruple-patterning, these structures are K5s (five shapes that each have a separator touching the other four shapes). This output is a type-1 layer.

VIOLATED_OPTIONAL_SEPARATORS — Outputs a layer containing optional violated separators (those with a priority index greater than 0, a lower priority). Separators specifying the OPPOSITE keyword (see page 71) are considered violated if they are formed between two polygons assigned to the same mask, or if both ends of the separator touch the same polygon. Separators specifying the SAME keyword are considered violated if they are formed between two polygons with different mask assignments. Anchor- and self-conflicts formed by optional separators are included in this output. This output is a type-3 layer.

VIOLATED_OPTIONAL_STITCHES — Outputs a layer containing optional violated MINIMIZE and MAXIMIZE stitches (those with a priority index greater than 0, a lower priority). A MINIMIZE stitch is considered violated when used and a MAXIMIZE stitch is considered violated when not used. This output is a type-1 layer.

VIOLATED_SEPARATORS — Outputs a layer containing required violated separators (those with a priority index less than or equal to 0, a higher priority). Errors generated by DIRECT_ANCHOR_CONFLICT_SEPARATORS or SELF_CONFLICT_SEPARATORS are not included in this output. This output applies only to required separators. This output is a type-3 layer.

VIOLATED_STITCHES — Outputs a layer containing polygons of stitches specified with the REQUIRED argument of the stitch_layer keyword that are in violation. This output is a type-1 layer.

Graph Reduction Keyword

For graph reduction, the output type may be this keyword, specified on its own DFM MP command line:

REDUCED_SEPARATORS_LDR — Reports the edges remaining in the graph after Low Degree Node Removal is performed. This is a type-3 layer.

- ***target_layer***

A required argument specifying the layer to be decomposed. The target_layer must be a polygon layer.

- ***separator_layer*** [‘(SAME | OPPOSITE [*priority*])’]...

A required argument specifying a separator_layer. If no separator_layer is specified, you must specify an anchor_layer or stitch_layer. The separator layer must be an edge pair (layer type-3). One or more separator layers may be specified. The SAME keyword indicates the polygons associated with the separator can both be assigned to the same mask layer. The keyword OPPOSITE indicates the polygons associated with the separator must be assigned to different mask layers. If no parentheses are specified, the separator behavior defaults to OPPOSITE with a priority of 0.

Separators can be specified as either required or optional during processing through the use of priority. The higher the priority value, the lower the priority of the separator. The priority value can be either 0 or a positive number. The default priority value is 0. A separator with the default priority of 0 enforces a required separator of highest priority. A separator specified with a priority of 1 (or greater) is considered lower priority.

Note

 Where any user-defined separators are invalid, all output layers, with the exception of the invalid separators layer, are potentially incorrect or inconsistent. In this specific case, Siemens EDA makes no guarantee these separators are correctly evaluated between connected polygons.

[Example 3-3](#) shows SVRF code that provides typical separator derivation, and [Figure 3-9](#) shows correctly formed separators.

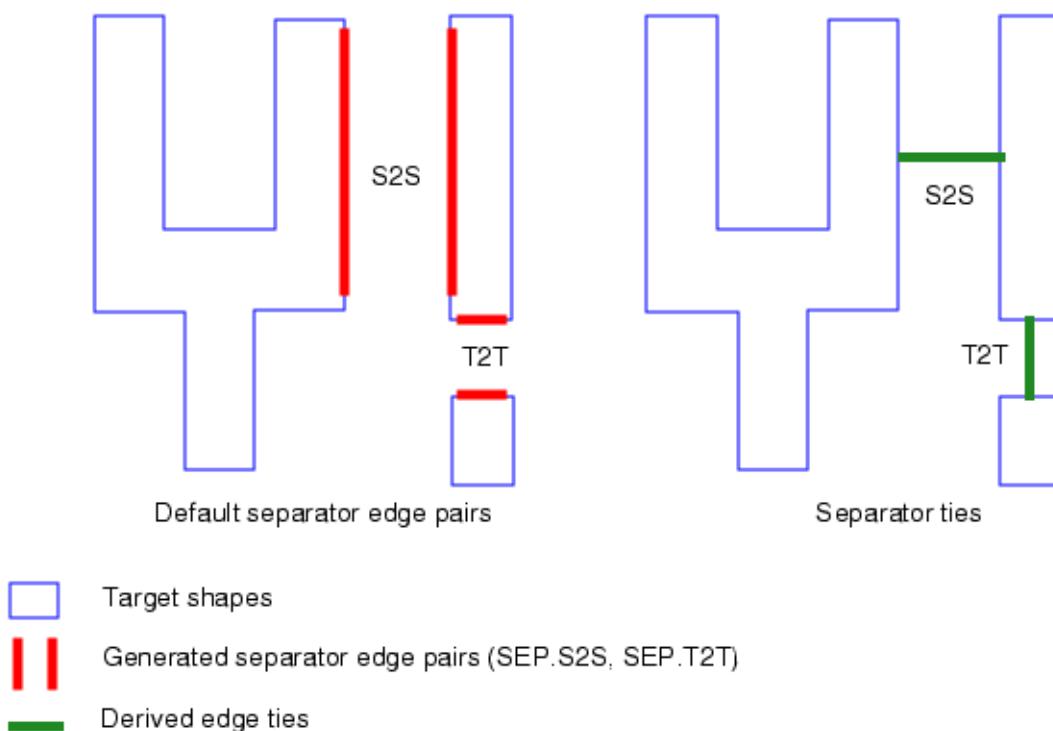
Example 3-3. Typical Separator Derivation

```
m1_tip = CONVEX EDGE m1 ANGLE1 == 90 LENGTH1 >= 0.03
          ANGLE2 == 90 LENGTH2 >= 0.03 WITH LENGTH <= 0.06
m1_side = m1 NOT COIN EDGE m1_tip

m1_S2S = EXT m1_side < 0.06 OPPOSITE PARALLEL ONLY PROJ > 0
m1_T2S = EXT m1_tip m1_side < 0.065 PARALLEL ONLY
m1_T2T = EXT m1_tip < 0.085 PARALLEL ONLY
```

The input separator layer must consist of edge pairs only (layer type-3) and each separator of the edge pair must touch exactly one drawn shape. [Figure 3-9](#) demonstrates both edge- and tie-based separators. Combination polygon-separator error keywords always output tie-based separators.

Figure 3-17. Correct DFM MP Separators



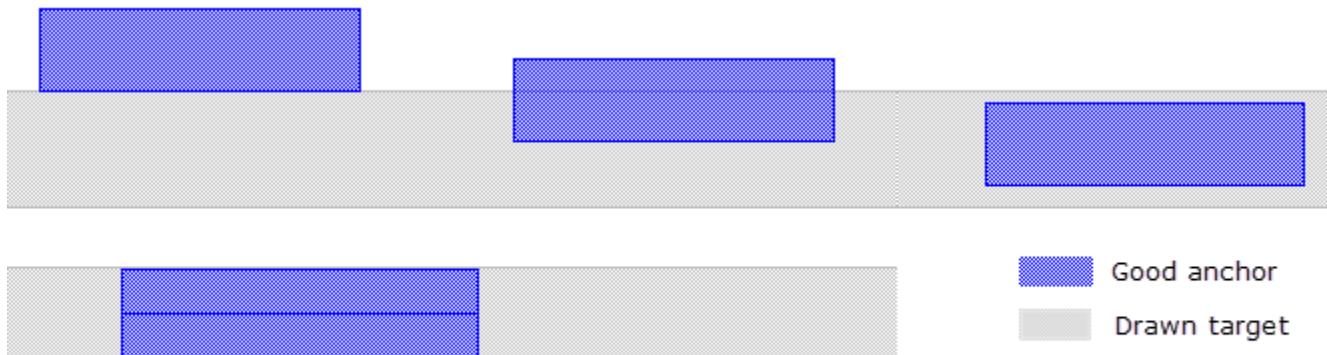
- ***anchor_layer* '('ANCHOR {MASK1 | MASK2 | MASK3 | MASK4} ')' ...**

A required argument specifying one or more anchor_layers. If no anchor_layer is specified, you must specify a separator_layer or stitch_layer. Anchor layers must comprise either polygons or paths (layer types 1 or 2, respectively). You may specify as many anchor_layers as allowed by the specified mask number, either 3 or 4; you cannot specify an anchor layer for a mask whose number is larger than the mask count.

Anchor layers are identified by the keyword ANCHOR specified inside the parentheses. If no parentheses are specified, the argument is interpreted as a separator layer. Mask values for anchor layer are identified by the keywords MASK1, MASK2, MASK3, and MASK4 (for quadruple patterning only) specified within the parentheses.

[Figure 3-10](#) demonstrates correct and incorrectly drawn anchors.

Figure 3-18. Good and Bad Anchors



Good anchor
Drawn target

Good anchors must be either polygons or paths and may touch, intersect, enclose, or be enclosed by target shapes.



Bad anchor
Drawn target

Bad anchors have no interaction with target shapes.

- ***stitch_layer* ‘(‘**STITCH** {MAXIMIZE *priority* | MINIMIZE *priority* | REQUIRED | USER}’)’...**

A required argument specifying one or more stitch_layers. If no stitch_layer is specified, you must specify a separator_layer or anchor_layer. Each of the stitch layers specified must be a polygon (layer type-1). The entirety of the stitch must be contained within the target shape and coincident with any target shape edge. A stitch is processed to result like a separator; after the stitch is subtracted from the drawn layer, the remaining space between target shapes determines whether the stitch is valid. Stitches that do not separate a target shape into at least two polygons are considered invalid; every stitch shape must touch at least two target shapes, not exceeding the number of masks. Stitches not touching any target shapes, those that touch other stitches, and those that touch anchors are invalid. A stitch that touches a separator is converted into multiple separators, one for each of the polygons resulting from the stitch removal.

Regarding non-user stitches, DFM MP allows non-user stitches to touch separators where the mask count is greater than 2. In this case, non-user stitches touching separators are converted into multiple separators, one for each polygon resulting from removing the stitch.

For example, if a separator exists between drawn target shapes A and B, and that separator touches a stitch on polygon A, and removing that stitch breaks A into three distinct polygons, A1, A2, and A3, then the following separators are formed:

A1 — B, A2 — B, A3 — B

If a stitch also splits shape B into two distinct polygons, B1 and B2, and the same separator touches that stitch, then the following separators are formed:

A1 — B1, A1 — B2, A2 — B1, A2 — B2, A3 — B1, A3 — B2

MAXIMIZE priority

Specified with a priority of 0, MAXIMIZE forces all stitches to be implemented. MAXIMIZE 0 is equivalent to REQUIRED. The higher the priority value, the lower the relative stitch priority. Enforced stitch priority is determined relative to priorities assigned to neighboring stitch_layers, and coloring constraints of separator_layers by their respective priorities.

MINIMIZE priority

Minimizes the number of stitches implemented by DFM MP. Minimizing stitch usage cannot be made mandatory, therefore MINIMIZE must specify a priority greater than 0.

REQUIRED

Implements all stitches, and any stitch which cannot be used (due to excessive coloring constraints) is reported by the VIOLATED_STITCHES error layer.

USER

Uses user-defined separator_layers to enforce coloring constraints between opposite sides of stitched polygons. User stitches must be user-generated and provided to DFM MP as an input separator layer. User-defined separators provide these capabilities and differences between non-user stitches:

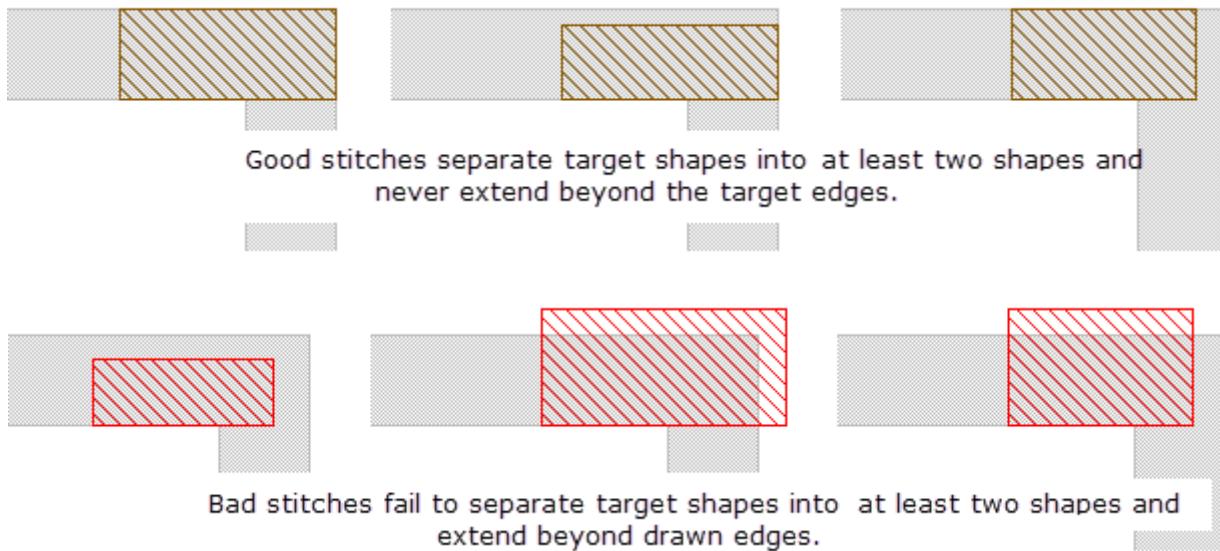
- Enabling opposite-color mask constraints between two sides of a stitch, making that stitch required.
- Providing an optional, same-color mask constraint between two sides of a stitch minimizing its use.
- Separators are not generated between pairs of resulting user stitches.
- Separators touching user stitches are not converted into multiple separators.

Note

 Where any user-defined separators are invalid, all output layers, with the exception of the invalid separators layer, are potentially incorrect or inconsistent. In this specific case, Siemens EDA makes no guarantee these separators are correctly evaluated between connected polygons.

Figure 3-19 demonstrates differences between correct and incorrectly drawn stitches.

Figure 3-19. Good and Bad Multi-Patterning User Stitches



Description

Performs coloring for triple- and quadruple-patterning designs. Outputs all decomposed, colored mask layers and their associated error and analysis layers. Each DFM MP command must specify at least one layer referencing a separator, anchor, or stitch, and may specify any combination, in any order, and any number of the three layers. For important exceptions regarding layer derivation support see “[Guidelines for Deriving Good Separator and Action Layers](#)” on page 346.

Examples

Example 1 - Mask Output

```
CA_Mask1_Out = DFM MP 3 MASK1 CA_Uncolored SEP.S2S SEP.T2T
    CA_Mask1(anchor mask1) CA_Mask2(anchor mask2) CA_Mask3(anchor mask3)

CA_Mask2_Out = DFM MP 3 MASK2 CA_Uncolored SEP.S2S SEP.T2T
    CA_Mask1(anchor mask1) CA_Mask2(anchor mask2) CA_Mask3(anchor mask3)

CA_Mask3_Out = DFM MP 3 MASK3 CA_Uncolored SEP.S2S SEP.T2T
    CA_Mask1(anchor mask1) CA_Mask2(anchor mask2) CA_Mask3(anchor mask3)

CA_Mask1_Out {COPY CA_Mask1_Out} DRC CHECK MAP CA_Mask1_Out 15
CA_Mask2_Out {COPY CA_Mask2_Out} DRC CHECK MAP CA_Mask2_Out 16
CA_Mask3_Out {COPY CA_Mask3_Out} DRC CHECK MAP CA_Mask3_Out 17
```

Example 2 - Modifiers for Separators

```
// If separator type is not specified, default is opposite 0

DFM MP 3 MASK1 CA_Uncolored SEP.S2S(opposite 0) SEP.T2T(opposite 0)
    CA_Mask1(anchor mask1) CA_Mask2(anchor mask2) CA_Mask3(anchor mask3)
```

```
DFM MP 3 MASK1 CA_Uncolored SEP.S2S(opposite 0) SEP.T2T(opposite 0)
CA_Mask1(anchor mask1) CA_Mask2(anchor mask2) CA_Mask3(anchor mask3)
```

Example 3 - Debugging and Analysis Layer Output

```
DFM MP 3 CONFLICT CA_Uncolored SEP.S2S(opposite 0)
SEP.T2T(opposite 0) CA_Mask1(anchor mask1) CA_Mask2(anchor mask2)
CA_Mask3(anchor mask3)
```

DFM MP SELECT CELLS

Multi-Patterning SVRF Commands

Generates a layer covering the area of selected cells based on various criteria.

Usage

BY LAYER Usage

output_layer = DFM MP SELECT CELLS

 BY LAYER *marker_layer*

 CELL_AREA

 [EXCLUDE_TOP]

BY CELL Usage

output_layer = DFM MP SELECT CELLS

 BY CELL

 CELL_AREA

 {MAX_OVERLAP *value*

 [MIN_AREA_RATIO *value*] [MIN_TOTAL_AREA_RATIO *value*]

 | MIN_AREA_RATIO *value*

 [MAX_OVERLAP *value*] [MIN_TOTAL_AREA_RATIO *value*]

 | MIN_TOTAL_AREA_RATIO *value*

 [MAX_OVERLAP *value*] [MIN_AREA_RATIO *value*]}
 [EXCLUDE_TOP]

 [MIN_PLACEMENT_COUNT *number*]

 [OFFSET *value*]

 [REQUIRED_AREA *value*]

 [REQUIRED_SIZE *value*]

Arguments

BY LAYER Usage

- *output_layer*

A required output layer covering a hierarchically merged area of selected cells based on bit cell layer specification.

- **BY LAYER *marker_layer***

A required keyword providing a cell identifying marker layer for DFM MP SELECT CELLS. Selected cells are merged by abutment or overlap of the marker_layers found in neighboring cells. The cells are then pushed up the design hierarchy.

- **CELL_AREA**

A required keyword that generates a marker covering all cells matching the specified criteria.

- EXCLUDE_TOP

An optional keyword that excludes the top cell from consideration. If cell selection takes place within the top cell and EXCLUDE_TOP is enabled, the top cell extent (all merged areas) is not considered for output.

BY CELL Usage

- *output_layer*

A required output layer covering a hierarchically merged area of SRAM cells based on bit cell spatial parameters.

- BY CELL

A required keyword enabling DFM MP SELECT CELLS to use cell-to-cell relationships.

- CELL_AREA

A required keyword that generates a marker covering all cells matching the specified criteria.

- MAX_OVERLAP *value*

A required keyword that specifies the maximum cell-to-cell overlap ratio relative to cell area. One or more of MAX_OVERLAP, MIN_AREA_RATIO or MIN_TOTAL_AREA_RATIO must be specified.

- MIN_AREA_RATIO *value*

A required keyword that specifies the minimum ratio of cell area over the chip area expressed as *cell_area/chip_area*. One or more of MAX_OVERLAP, MIN_AREA_RATIO or MIN_TOTAL_AREA_RATIO must be specified.

- MIN_TOTAL_AREA_RATIO *value*

A required keyword that specifies the minimum ratio of total cell area over chip area expressed as *total cell_area/chip_area*. One or more of MAX_OVERLAP, MIN_AREA_RATIO or MIN_TOTAL_AREA_RATIO must be specified.

- EXCLUDE_TOP

An optional keyword that excludes the top cell from consideration. If cell selection takes place within the top cell and EXCLUDE_TOP is enabled, the top cell extent (all merged areas) is not considered for output.

- MIN_PLACEMENT_COUNT *number*

An optional keyword that specifies the minimum cell placement count.

- OFFSET *value*

An optional keyword that specifies the pullback distance around cell borders.

- REQUIRED_AREA *value*

An optional keyword that specifies the minimum cell area required for selection (the cell extent).

- REQUIRED_SIZE *value*

An optional keyword that specifies the minimum cell dimension within the cell border extent. The value specified must be less than both X and Y extents.

Description

DFM MP SELECT CELLS generates a layer covering all selected cells based on specified criteria. The general application is to find bit cell patterns in SRAM designs using two approaches:

- **BY LAYER** — Where a specified marker layer placed within bit cells of the SRAM design is used to locate cell relationships of abutment or overlap.
- **BY CELL** — Where specified area overlap ratios, cell areas, cell boundary offsets, and placement counts of bit cells of the SRAM design are used to select cells.

In both cases, the selected cells are merged and then pushed up the hierarchy of the design. When no remaining cells can be merged into a higher level, the merged area is output to the output_layer. The DFM CLUSTER SELECT_CELLS cell_area_layer keyword and layer pair is required to use the output_layer from DFM MP SELECT CELLS for selection of cell areas.

Examples

Example 1 — BY CELL Application

An output layer that covers cell relationships matching the criteria that excludes the top layer is mapped in red:

```
cell_area = DFM MP SELECT CELLS
    BY CELL
    CELL_AREA
    MAX_OVERLAP 0.01
    MIN_TOTAL_AREA_RATIO 0.001
    MIN_AREA_RATIO 0.01
    REQUIRED_SIZE 0.3
    REQUIRED_AREA 0.1
    EXCLUDE_TOP
```

The DFM MP SELECT CELLS output can then be processed by DFM CLUSTER. The DFM CLUSTER command shown uses the corner keyword LEFT BOTTOM. The output layer, left_bottom_high, contains the left bottom corner based on the highest hierarchical cell within the cell_area layer (the select keyword SELECT_CELLS selects the highest hierarchy):

```
left_bottom_high = DFM CLUSTER target separator
    LEFT BOTTOM
    NOFLOAT
    SELECT_CELLS cell_area
```

The output layer, left_bottom_low, contains the left bottom corner based on the lowest hierarchical cell within the cell_area layer (the select keyword SELECT_CELLS_LOWEST selects the lowest hierarchy):

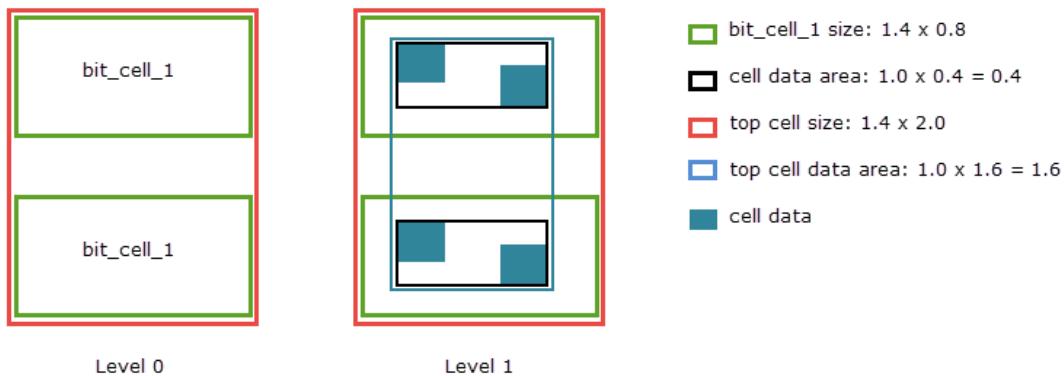
```
left_bottom_low = DFM CLUSTER target separator
    LEFT BOTTOM
    NOFLOAT
    SELECT_CELLS_LOWEST cell_area
```

Example 2 — BY CELL Output

In Figure 3-20, bit_cell_1 contains two polygons. The bit_cell_1 cells are arranged in two 1 by 2 arrays within the top cell (shown at left as Level 0). The red rectangle represents top cell size conforming to the bit_cell_1 placements. The green rectangles are the bit_cell_1 cells. Shown as Level 1 on the right, are dark blue data comprising bit_cell_1 cells, and the area of cell data extent represented by the black rectangle. The blue rectangle encompasses the total data extent at the top cell. The following code example demonstrates the minimum values required to select these data shown:

```
output_layer = DFM MP SELECT CELL BY CELL
    CELL_AREA
    MIN_TOTAL_AREA_RATIO 0.50 //cell data area*2/top cell data area=0.50
    MIN_AREA_RATIO 0.25 //cell data area/top cell data area=0.25
    REQUIRED_SIZE 0.40 //minimum cell data in X- or Y- axis considered
    REQUIRED_AREA 0.40 //minimum cell data area (X*Y)
    EXCLUDE_TOP
```

Figure 3-20. BY CELL Output



Example 3 — BY LAYER Application

The output layer, cell_area, selects the highest fully-merged cell area of layer1:

```
cell_area = DFM MP SELECT CELLS
    BY LAYER layer1
    CELL_AREA
```

The output layer, cell_area, selects the highest fully-merged cell area excluding the top cell of layer2:

```
cell_area = DFM MP SELECT CELLS
    BY LAYER layer2
    CELL_AREA
    EXCLUDE_TOP
```

The output layer, left_bottom_high, contains the left bottom corner based on highest hierarchical cell of the cell_area layer (the select keyword SELECT_CELLS selects the highest hierarchy):

```
left_bottom_high = DFM CLUSTER target separator
    LEFT BOTTOM
    NOFLOAT
    SELECT_CELLS cell_area
```

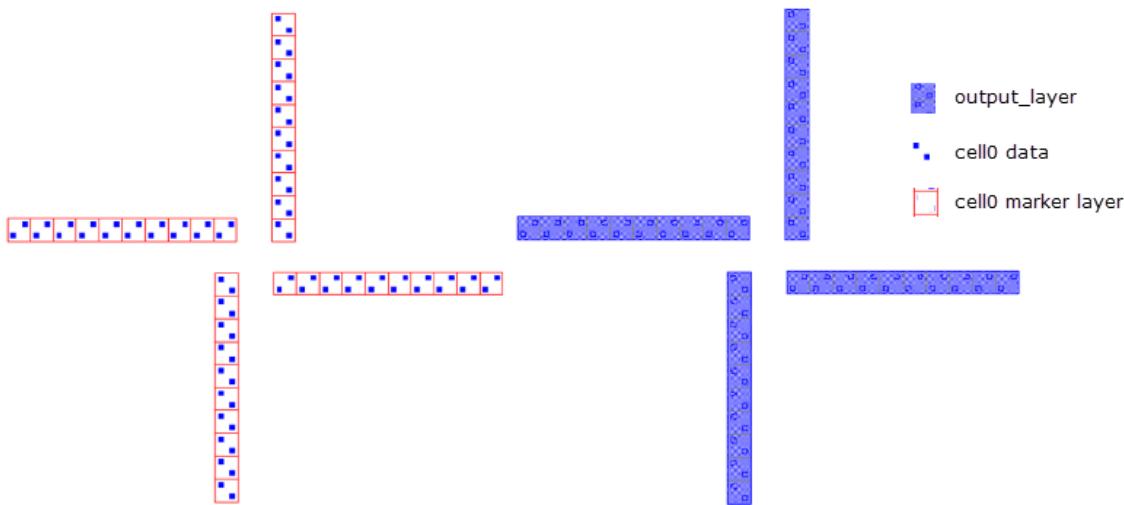
The output layer, left_bottom_low, contains the left bottom corner based on lowest hierarchical cell of the cell_area layer (the select keyword SELECT_CELLS_LOWEST selects the lowest hierarchy):

```
left_bottom_low = DFM CLUSTER target separator
    LEFT BOTTOM
    NOFLOAT
    SELECT_CELLS_LOWEST cell_area
```

Example 4 — BY LAYER Output

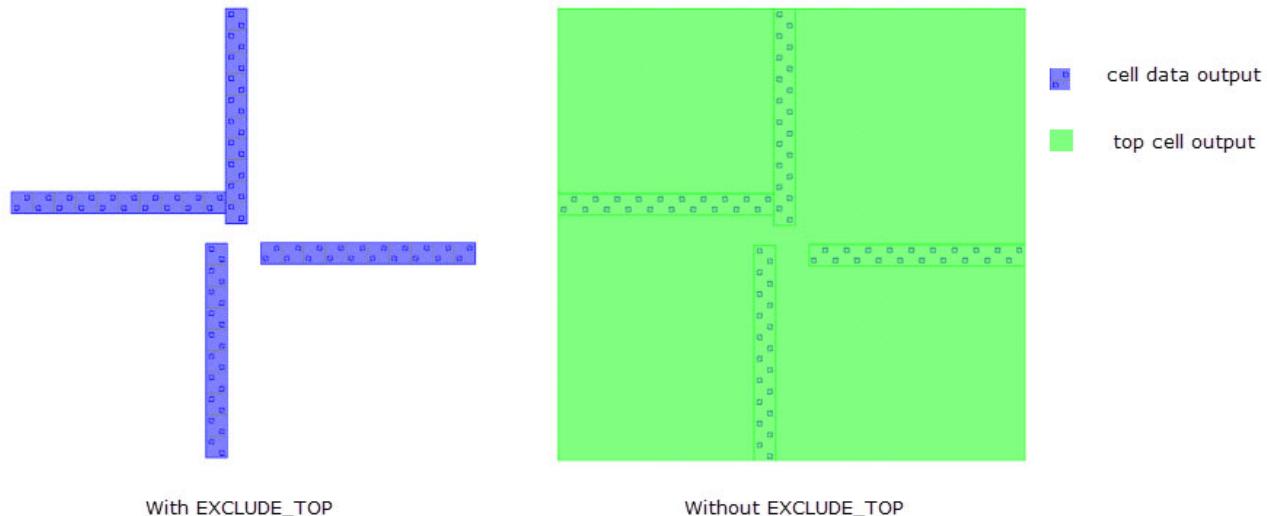
In [Figure 3-21](#), cell0 input cells have two polygons. The cells are arranged in four 1 by 10 arrays within the top cell. The red rectangles represent the marker layers inside cell0. As marker shapes within the cells touch, they are merged, resulting in four arrays. Full merging is completed below the top level, rendering EXCLUDE_TOP ineffective in this application.

Figure 3-21. BY LAYER Output



Example 5 — EXCLUDE_TOP Usage

In [Figure 3-22](#), merged, non-overlapping placements are completed within the cell level, so all placements of the cell are selected; also, any overlapping placements are merged within the top cell. If any areas are selected within the top cell, the output then covers the entire top cell area (shown in green). In this case, if EXCLUDE_TOP is specified, the output only contains cells under the top cell (shown in blue).

Figure 3-22. EXCLUDE_TOP Usage

DFM MPSTITCH

Multi-Patterning SVRF Commands

Generates multi-pattern stitches. References DFM SPEC MPSTITCH constraint specifications.

Usage

```
stitch_layer = DFM MPSTITCH
  spec_name
  target_layer
  [separator_layer]
  [ANCHOR anchor_layer1... anchor_layerN]
  [KEEPOUT keepout_layer]
  [VIOLATED_TARGET violated_layer]
```

Arguments

- **stitch_layer**
A required output layer to contain generated multi-pattern stitches.
- **spec_name**
A required argument referencing the specifications named in a [DFM SPEC MPSTITCH](#) statement. Multiple DFM MPSTITCH commands with different input layers can share a single DFM SPEC MPSTITCH statement.
- **target_layer**
A required argument specifying the input layer to be analyzed for stitch generation. The target_layer must be a polygon layer (layer type-1).
- **separator_layer**
An optional argument containing separators used to determine potential stitch sites. The separator layer must be an edge pair (layer type-3).

[Example 3-4](#) shows SVRF code that provides typical separator derivation, and [Figure 3-23](#) shows correctly formed separators.

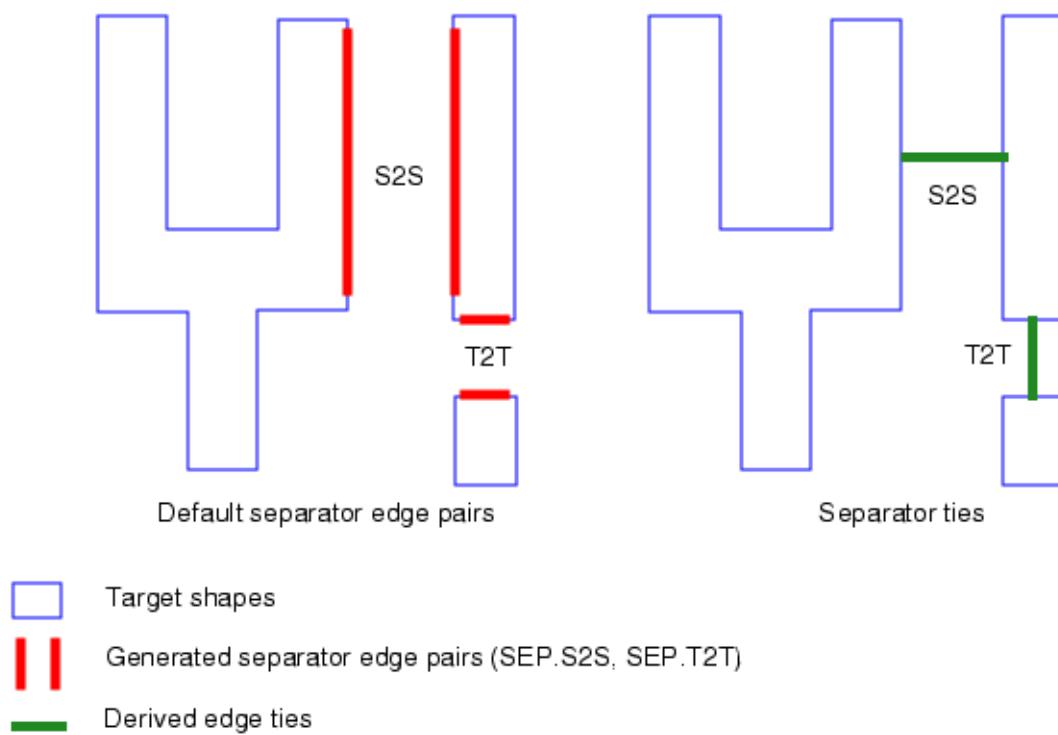
Example 3-4. Typical Separator Derivation

```
m1_tip = CONVEX EDGE m1 ANGLE1 == 90 LENGTH1 >= 0.03
          ANGLE2 == 90 LENGTH2 >= 0.03 WITH LENGTH <= 0.06
m1_side = m1 NOT COIN EDGE m1_tip

m1_S2S = EXT m1_side < 0.06 OPPOSITE PARALLEL ONLY PROJ > 0
m1_T2S = EXT m1_tip m1_side < 0.065 PARALLEL ONLY
m1_T2T = EXT m1_tip < 0.085 PARALLEL ONLY
```

The input separator layer must consist of edge pairs only (layer type-3) and each separator of the edge pair must touch exactly one drawn shape. [Figure 3-23](#) demonstrates both edge-and tie-based separators. Combination polygon-separator error keywords always output tie-based separators.

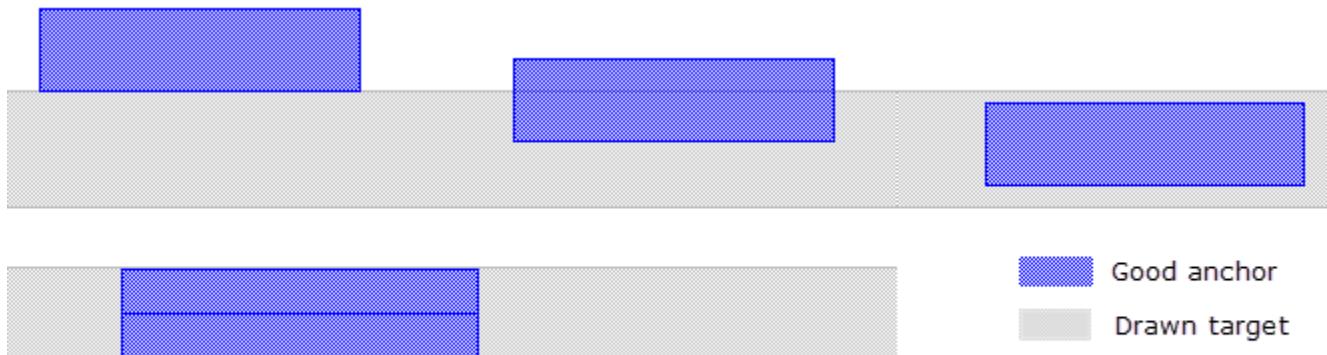
Figure 3-23. Correct DFM DP Separators



- ANCHOR *anchor_layer1...anchor_layerN*
An optional keyword with one or more user-specified layers used for anchor shapes to be applied to mask layers. The presence of an anchor_layer prohibits the placement of a stitch at that location. The same number of layers as the DFM SPEC MPSTITCH mask count must be specified. The anchor_layer must be a polygon (layer type-1).

[Figure 3-24](#) demonstrates correct and incorrect anchors.

Figure 3-24. Good and Bad Anchors



Good anchors must be either polygons or paths and may touch, intersect, enclose, or be enclosed by target shapes.



Bad anchors have no interaction with target shapes.

- **KEEPOUT *keepout_layer***

An optional user-derived layer that specifies areas where stitch placement is prohibited. The *keepout_layer* must be a polygon (layer type-1).

- **VIOLATED_TARGET *violated_layer***

An optional input polygon layer containing shapes that are a subset of the *target_layer*. There may be more stitch candidates found on the *violated_layer* than other parts of the *target_layer*. Typically, polygons generated by DFM MP with the SELF_CONTAINED_VIOLATION_POLYGONS and SELF_CONFLICT_SEPARATORS error layer keywords provide good stitch candidates. VIOLATED_TARGET may assist DFM MP to successfully place stitch candidates over the entire *target_layer*.

Description

Performs stitching for multi-patterning three- and four-mask designs. Generation of stitch candidates begins with the computation of corner proximities, forbidden regions, width keepout regions, and stitch keepout regions. The remaining target regions or anchor shapes may be used as potential stitch regions. DFM SPEC MPSTITCH must precede DFM MPSTITCH within the

SVRF file. DFM MPSTITCH generates legal stitch candidates for each potential region in this order:

1. The stitch is placed at the center of the potential stitch region.
2. Where the stitch touches more separators than allowed, the stitch is placed at the bottom-left or top-right of the region.

For important exceptions regarding layer derivation support see “[Guidelines for Deriving Good Separator and Action Layers](#)” on page 346.

Examples

Example 1 — DFM MPSTITCH Use for Three Masks

The DFM MPSTITCH command shown supports three mask layers. The DFM MPSTITCH command calls a DFM SPEC MPSTITCH statement that supports three mask layers. If used, the ANCHOR keyword is required to list that number of layers. The KEEPOUT layer, MB, is user-defined.

```
M1_stitches = DFM MPSTITCH m1_mpstitch3 M1
    ANCHOR M1_E1_anchor M1_E2_anchor M1_E3_anchor
    M1_Separator KEEPOUT M1KO
```

Example 2 — DFM MPSTITCH Use for Four Masks

The DFM MPSTITCH command shown supports four mask layers. The DFM MPSTITCH command calls a DFM SPEC MPSTITCH statement that supports four mask layers. The ANCHOR keyword now requires four layers.

```
M1_stitches = DFM MPSTITCH m1_mpstitch4 M1
    ANCHOR M1_E1_anchor M1_E2_anchor M1_E3_anchor M1_E4_anchor
    M1_Separator KEEPOUT M1KO
```

Example 3 — DFM MPSTITCH in Context

DFM MPSTITCH is shown here in context with its referenced DFM SPEC MPSTITCH statement. The DFM SPEC MPSTITCH command must precede the DFM MPSTITCH command.

```
VARIABLE MINIMUM_STITCH_WIDTH 0.022
VARIABLE MAXIMUM_STITCH_WIDTH 0.220
VARIABLE CONCAVE_CORNER_SPACING 0.050
VARIABLE CONVEX_CORNER_SPACING 0.026
VARIABLE MINIMUM_STITCH_LENGTH 0.066
VARIABLE MAXIMUM_STITCH_LENGTH 0.375
VARIABLE MINIMUM_MASK_AREA 0.00339
VARIABLE MASK_GRID_SPACING 0.001
VARIABLE DBU_PER_GRID_SPACING 1
```

```
DFM SPEC MPSTITCH m1_mpstitch3 3
  STITCH_LENGTH 0.06
  WIDTH 0.02 0.2
  CONCAVE_SPACING 0.05
  CONVEX_SPACING 0.025
  MIN_AREA 0.0035
  GRID_SPACING 0.001
  MEASURES > 0.022 < 0.220 opposite

M1_stitches = DFM MPSTITCH m1_mpstitch3
  M1_Target
  ANCHOR M1_Anchor1 M1_Anchor2 M1_Anchor3
  M1_Separator
  KEEPOUT M1KO

M1_Mask1 = DFM MP 3 MASK1
  M1_Target
  M1_stitches(STITCH REQUIRED)
  M1_Separator
  M1_Anchor1(ANCHOR MASK1)
  M1_Anchor2(ANCHOR MASK2)
  M1_Anchor3(ANCHOR MASK3)

M1_Mask2 = DFM MP 3 MASK2
  M1_Target
  M1_stitches(STITCH REQUIRED)
  M1_Separator
  M1_Anchor1(ANCHOR MASK1)
  M1_Anchor2(ANCHOR MASK2)
  M1_Anchor3(ANCHOR MASK3)

...
```

DFM RET NMDPC EMULATION

Multi-Patterning SVRF Commands

Emulates RET NMDPC operations using DFM DP algorithms.

Usage

DFM RET NMDPC EMULATION {YES | NO}

Arguments

- **YES**

Enables DFM DP emulation, performing all RET NMDPC operations using DFM DP. Some output may not match identically to that of DFM DP. Additionally, all emulated operations are performed with hyperflex where enabled.

- **NO**

Disables DFM DP emulation. This is the default argument.

Description

Emulates all RET NMDPC operations to be performed using DFM DP algorithms. Whether enabled through specification within an SVRF file or as an environment variable, RET_NMDPC is emulated by DFM DP, and hyperscaling is enabled if specified on the Linux®¹ command line.

Environmental Variables

Environment variables can also be used to invoke DFM DP emulation:

- **CALIBRE_DP_RET_EMULATION** — Set to 1 to emulate DFM DP using RET NMDPC. The default setting is 0.
- **CALIBRE_DP_RET_EMULATION_FALLBACK** — Set to 1 to enable execution of RET NMDPC if DFM DP emulation fails. When set to 0, if emulation of DFM DP fails, RET NMDPC terminates. The default setting is 0.
- **CALIBRE_NMDPC_HYPERSCALING_ENABLED** — If DFM DP emulation is disabled, this environment variable controls hyperscaling during RET NMDPC execution. Allowable values control hyperscaling:
 - 1 — Enables hyperscaling.
 - 0 — Disables hyperscaling.
 - -1 — Enables hyperscaling only if -hyper_remote is specified and a memory consumption heuristic is passable.

1. Linux® is a registered trademark of Linus Torvalds in the U.S. and other countries.

DFM SPEC BALANCE

Multi-Patterning SVRF Commands

Specifies multi-pattern mask balancing criterion. Called by the DFM BALANCE command.

Usage

DFM SPEC BALANCE *name num*

```
[ANCHOR anchor1 anchor2 ...]  
[MUTABLE layer1 layer2 ...]  
[OVERLAY layer]  
[PRECHECK layer]  
[SEPARATOR layer]  
[UNCOLORED layer]  
WITH DENSITY [expression constraint]  
[INSIDE OF EXTENT | INSIDE OF x1 y1 x2 y2 |  
INSIDE OF LAYER layer  
    [BY EXTENT [size_value] | BY POLYGON | BY RECTANGLE | CENTERED value]]  
[TRUNCATE | BACKUP | IGNORE | WRAP]  
[WINDOW [xy_value | x_value y_value] [STEP [xy_value | x_value y_value]]]
```

Arguments

- ***name***

The required name of the specifications set in the DFM SPEC BALANCE statement. This name is called by [DFM BALANCE](#). No DFM SPEC BALANCE statements may share the same name.

- ***num***

A required value representing the number of masks to process with DFM BALANCE. For double-, triple-, and quadruple-patterning, num must be 2, 3, and 4, respectively.

- **ANCHOR *anchor1 anchor2 ...***

An optional keyword and associated layer names specifying the anchor layers. Mask data on these layers remain immutable. The number of layers specified must match the num value; for double-, triple-, and quadruple-patterning, the number of layers specified must be 2, 3, and 4, respectively. ANCHOR and MUTABLE polygons of the same color cannot overlap or abut each other in any case. ANCHOR and MUTABLE polygons of different colors may overlap, but not touch. See [Table 3-5](#) on page 118 for allowed overlaps.

- **MUTABLE *layer1 layer2 ...***

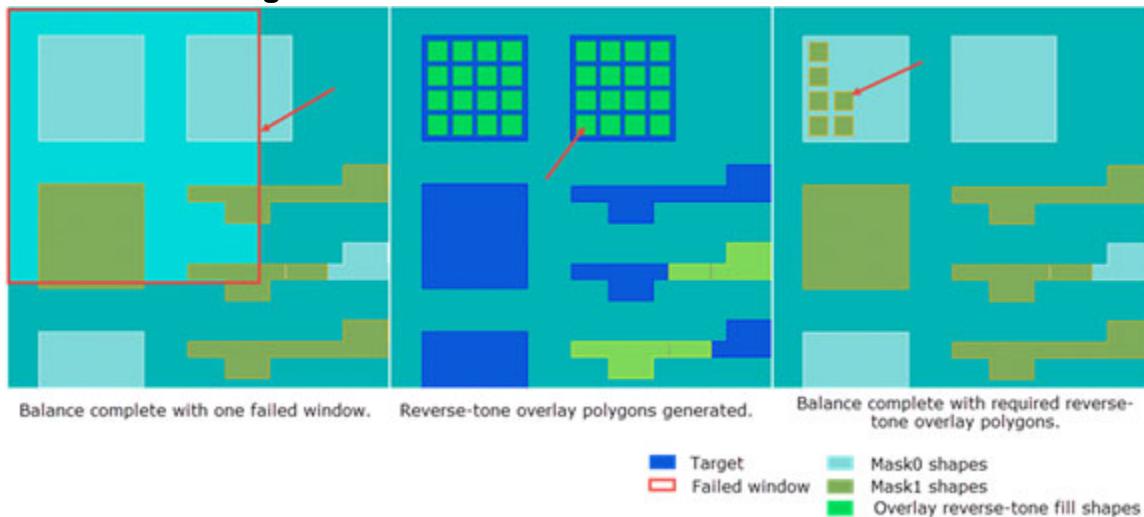
An optional keyword and associated layer names specifying layers allowed to be manipulated to achieve the specified density threshold. The number of layers specified must match the num value; for double-, triple-, and quadruple-patterning, the number of layers specified must be 2, 3, and 4, respectively. ANCHOR and MUTABLE polygons of the same color cannot overlap or abut each other in any case. ANCHOR and MUTABLE polygons of different colors may overlap, but not touch. However, MUTABLE polygons may touch

edge pair separators. MUTABLE polygons connected to ANCHOR polygons by a separator are considered as ANCHORS. See [Table 3-5](#) on page 118 for allowed overlaps.

- **OVERLAY layer**

An optional keyword and polygon layer specifying the layer used for reverse-tone overlay fill polygons. Polygons belonging to OVERLAY layers must not touch any SEPARATOR edge pair. The polygons of the OVERLAY layer may share areas with ANCHOR or MUTABLE polygons, and the areas outside of these polygons are ignored. Polygons of the OVERLAY layer are assigned to a mask only if they are required by density balancing to meet the density threshold.

Figure 3-25. DFM BALANCE Use of OVERLAY



- **PRECHECK layer**

This keyword and layer pair enables DFM BALANCE to use output from a previous Density operation using the same options as DFM SPEC BALANCE. If no data exists on the specified PRECHECK layer, DFM BALANCE writes the output layer. If density checking has already been performed and resulted in no failed windows, the DFM BALANCE run can be significantly shortened.

- **SEPARATOR layer**

An optional keyword and layer name specifying the separator layer. Shapes of MUTABLE layers touching a separator may be colored differently to achieve density balancing.

- **UNCOLORED layer**

An optional keyword and polygon layer name specifying a layer that has no previous coloring performed by NMDPC. Polygons assigned as UNCOLORED must not touch or overlap ANCHOR or MUTABLE polygons, or any SEPARATOR edge pair. Shapes on an UNCOLORED layer are randomly assigned colors to uncolored shapes as DFM BALANCE attempts minimization of errors across density windows.

- **WITH DENSITY** [*expression*] *constraint*

A required constraint pair with an optional expression keyword specifying the threshold and density method, respectively. All density windows qualified by the expression and constraint are output as balanced mask data. The WITH DENSITY keyword must appear after the keywords ANCHOR, MUTABLE, SEPARATOR, and UNCOLORED. The default expression keyword is MIN_COLOR_OVER_TOTAL_COLOR, however, a constraint must be specified. The expression keywords are:

MIN COLOR OVER TOTAL COLOR — A ratio of the minimum threshold area divided by the total area of the current mask. The associated constraint must have an evaluator that is greater than or equal to (\geq) paired with a numeric value less than or equal to 1/number of colors. This is the default expression.

MIN_COLOR_OVER_WINDOW_AREA — A ratio of the minimum threshold area divided by the total area of the current window. The associated constraint must have an evaluator that is greater than or equal to (\geq) paired with a numeric value less than or equal to 1/number of colors.

MAX_DELTA_OVER_TOTAL_COLOR — A ratio of the maximum variation divided by the total area of the current mask. The associated constraint must have an evaluator which is less than or equal to (\leq) paired with a positive numeric value.

MAX_DELTA_OVER_WINDOW_AREA — A ratio of the maximum variation divided by the total area of the current window. The associated constraint must have an evaluator that is less than or equal to (\leq) paired with a positive numeric value.

- **INSIDE OF EXTENT**

An optional keyword specifying the density checking boundary, where the boundary is automatically determined as the outer-most rectangle of all shapes on the target layer. The INSIDE OF EXTENT keyword must appear after WITH DENSITY and is mutually exclusive with any other INSIDE OF keyword.

- **INSIDE OF** *x1 y1 x2 y2*

An optional keyword specifying the density checking boundary, where the boundary is determined with the four static X- and Y-axis coordinates you specify. If any of the specified coordinates are negative, they must be individually enclosed by parentheses. The INSIDE OF *x1 y1 x2 y2* keyword must appear after WITH DENSITY and is mutually exclusive with any other INSIDE OF keyword.

- **INSIDE OF LAYER** *layer* [BY EXTENT [*size_value*] | BY POLYGON | BY RECTANGLE | CENTERED *value*]

An optional keyword specifying the density checking boundary, where the density checking boundary is determined by the outer-most extent of shapes on the layer specified. Using this option is computationally expensive.

Optional keywords provide four methods to control how INSIDE OF LAYER handles the specified layer.

BY EXTENT [size_value] — All shapes on the layer you specify are used as the density checking boundary. The rectilinear extent for any polygon is used. Optionally, you specify a sizing value that enlarges the size of the extents on the specified layer.

BY POLYGON — Shapes on the layer specified by the layer parameter are ANDed with the *layer* specified to INSIDE OF LAYER and then used as individual density checking boundaries.

BY RECTANGLE — Input shapes are vertically fractured into trapezoids, then these trapezoidal shapes are used as density checking boundaries.

CENTERED value — A four-coordinate extent is computed by the center point of the polygonal shape and the *value* you specify. The CENTERED option then computes the boundary in the same fashion as INSIDE OF *x1 y1 x2 y2*.

The INSIDE OF LAYER keyword must appear after WITH DENSITY and is mutually exclusive with INSIDE OF EXTENT and INSIDE OF *x1 y1 x2 y2*. For detailed information regarding these options and their behavior, see “[Density](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

- **TRUNCATE** | BACKUP | IGNORE | WRAP

Optional keywords providing four methods specifying how WINDOW handles areas near the data extent determined by INSIDE OF. For detailed information, see “[Density](#)” in the *Standard Verification Rule Format (SVRF) Manual*. These keywords must appear after WITH DENSITY.

TRUNCATE — The WINDOW does not exceed the data extent. This is the default method.

BACKUP — The WINDOW is made equal to the data extent.

IGNORE — The WINDOW is discarded if it ever exceeds the data extent.

WRAP — The WINDOW is placed and density calculated as if the data extent were duplicated.

- **WINDOW [xy_value | x_value y_value] [STEP [xy_value | x_value y_value]]**

An optional keyword and associated values specifying the size of the analysis window. You may optionally specify a step distance each window makes. The analysis window moves from left to right, bottom to top, across your design. You can specify a single value or separate values for X- and Y-axis directions for both WINDOW and STEP. The WINDOW keyword must appear after WITH DENSITY.

Description

The DFM SPEC BALANCE statement called by the DFM BALANCE command is specified within the same file. The DFM SPEC BALANCE statement can be called by multiple DFM BALANCE commands. Exercise care that layers adhere to the polygon interaction constraints listed with each layer assignment.

DFM BALANCE does not propagate color assignment based on graph analysis or anchors like NMDPC. Because there must be no layer interaction between UNCOLORED, ANCHOR, and

SEPARATOR edge pairs, all shapes on the UNCOLORED layer should be derived through the exclusion of ANCHOR and SEPARATOR layers.

The polygons of ANCHOR and MUTABLE layers with different colors can overlap with each other, but are not allowed to abut. Table 3-5 shows allowed overlaps between these two layer types of different colors:

Table 3-5. Allowed Overlap of Different Layers

This layer:	May overlap these layers:		
ANCHOR1	ANCHOR2	ANCHOR3	ANCHOR4
MUTABLE1	MUTABLE2	MUTABLE3	MUTABLE4
ANCHOR1	MUTABLE2	MUTABLE3	MUTABLE4
MUTABLE1	ANCHOR2	ANCHOR3	ANCHOR4

You can use variables for any names or numbers in the DFM SPEC BALANCE statement, as described in “[Using SVRF Variables within a LITHO FILE Statement](#)” on page 323.

Examples

Example 1 — Simple Example

```
VARIABLE delta_den 0.4
VARIABLE window_size 0.2
VARIABLE window_step 0.1

DFM SPEC BALANCE m1_balance 2
    ANCHOR mask0_anchors mask1_anchors
    UNCOLORED m1_uncolored
    SEPARATOR m1_separators

    WITH DENSITY > delta_den
    WINDOW window_size STEP window_step
    BACKUP
    INSIDE OF (-235.0) (-345.0) 235.0 345.0

balanced_mask0 = DFM BALANCE m1_balance MASK 1
balanced_mask1 = DFM BALANCE m1_balance MASK 2
balanced_mask = DFM BALANCE m1_balance FULL MASK 2
```

Example 2 — Includes Layer Processing Steps

```
// These input layers have anchors partially covering mutable polygons,
// and must be pre-processed prior to inputting them to DFM BALANCE.
LAYER M1A 1          // MUTABLE input 1
LAYER M1B 2          // MUTABLE input 2
LAYER M1A_Anchor 11  // ANCHOR input 1
LAYER M1B_Anchor 12  // ANCHOR input 2
```

```

pruned Mutable_0 = M1A NOT M1A_Anchor
pruned Mutable_1 = M1B NOT M1B_Anchor
DB_anchor_0 = M1A_Anchor OR (pruned Mutable_0 INTERACT M1A_Anchor)
DB_anchor_1 = M1B_Anchor OR (pruned Mutable_1 INTERACT M1B_Anchor)
DB_rotatable_mask0 = pruned Mutable_0 NOT INTERACT M1A_Anchor
DB_rotatable_mask1 = pruned Mutable_1 NOT INTERACT M1B_Anchor

MUTABLE_error = (DB_rotatable_mask0 INTERACT DB_rotatable_mask1) OR
                (DB_rotatable_mask1 INTERACT DB_rotatable_mask0)

// Set up DFM BALANCE
DFM SPEC BALANCE "db_trial" 2
    ANCHOR DB_anchor_0 DB_anchor_1
    MUTABLE DB_rotatable_mask0 DB_rotatable_mask1
    WITH DENSITY >= delta_den
    BACKUP INSIDE OF EXTENT WINDOW window_size STEP window_step

// Run DFM BALANCE
DB_balanced_mask0_delta = DFM BALANCE "db_trial" MASK 1
DB_balanced_mask1_delta = DFM BALANCE "db_trial" MASK 2

Failed_windows_merged { //merged is default
    DFM BALANCE "db_trial"
}
Failed_windows_unmerged {
    DFM BALANCE "db_trial" UNMERGED
}

// DFM BALANCE outputs changes to each mask,
// so a full mask must be reconstructed by post-processing.
DB_balanced_mask0 = OR DB_anchor_0 DB_balanced_mask0_delta
                    (DB_rotatable_mask0 NOT INTERACT DB_balanced_mask1_delta)
DB_balanced_mask1 = OR DB_anchor_1 DB_balanced_mask1_delta
                    (DB_rotatable_mask1 NOT INTERACT DB_balanced_mask0_delta)

```

Example 3 — Specification of Density Expressions

```

DFM SPEC BALANCE m1_balance 2
    ANCHOR mask0_anchors mask1_anchors
    UNCOLORED m1_uncolored
    SEPARATOR m1_separator
    WITH DENSITY MIN_COLOR_OVER_TOTAL_COLOR > 0.42

DFM SPEC BALANCE m1_balance 2
    ANCHOR mask0_anchors mask1_anchors
    UNCOLORED m1_uncolored
    SEPARATOR m1_separator
    WITH DENSITY MAX_DELTA_OVER_TOTAL_COLOR <= 15

```

DFM SPEC MPSTITCH

Multi-Patterning SVRF Commands

Specifies multi-pattern stitching values and constraints. Called by DFM MPSTITCH.

Usage

DFM SPEC MPSTITCH

```
spec_name
mask_number
CONCAVE_SPACING value
CONVEX_SPACING value
STITCH_LENGTH value
WIDTH value
[MIN_AREA value]
[GRID_SPACING value]
[MEASURES constraint [metric]]
```

Arguments

- **spec_name**

A required argument labeling the specifications for subsequent reference by a [DFM MPSTITCH](#) command. Each DFM SPEC MPSTITCH statement must have a unique name. Multiple DFM MPSTITCH commands with different input layers may share a single DFM SPEC MPSTITCH statement.

- **mask_number**

A required number, either 3 or 4, specifying the number of masks to be subsequently generated by DFM MP.

- **CONCAVE_SPACING value**

A required keyword specifying the distance from the inside vertex of a corner where stitch placement is prohibited.

- **CONVEX_SPACING value**

A required keyword specifying the distance from the outside vertex of a corner where stitch placement is prohibited.

- **STITCH_LENGTH value**

A required keyword specifying the length of all generated stitches.

- **WIDTH min_value max_value**

A required keyword specifying the minimum and maximum widths, respectively, of the target polygon for which a stitch can be generated.

- **MIN_AREA value**

An optional keyword specifying the smallest allowable area of any mask shape resulting from stitching.

- **GRID_SPACING** *value*
An optional keyword specifying the grid spacing used for stitch placement.
- **MEASURES** *constraint* [*metric*]
An optional keyword and constraint pair specified to avoid placement of stitches within mutually exclusive stitch locations on the same polygon. Any number of MEASURES keywords may be specified for a single DFM SPEC MPSTITCH statement.
 - constraint* — Establishes the required proximities from stitch to stitch. The constraint values must be positive and be on grid.
 - metric* — An optional keyword specifying the edge measurement method when measuring edge separations. The default metric is EUCLIDEAN.
 - CORNER TO CORNER
 - EUCLIDEAN
 - OPPOSITE
 - OPPOSITE EXTENDED *value*
 - PERPENDICULAR ALSO
 - PERPENDICULAR ONLY

Description

A statement containing values and constraints referenced by subsequent DFM MPSTITCH commands for three- and four-mask layouts. The statement provides stitch sizes, stitch-to-corner proximity and applied target shape widths. After processing the constraints, DFM MPSTITCH generates legal stitch candidates for each potential region. DFM SPEC MPSTITCH must precede DFM MPSTITCH within the SVRF file.

Examples

Example 1 — DFM SPEC MPSTITCH Statement for Three Masks

The DFM SPEC MPSTITCH command shown supports three mask layers. The DFM MPSTITCH command calls this DFM SPEC MPSTITCH statement.

```
DFM SPEC MPSTITCH m1_mpstitch3 3
    STITCH_LENGTH 0.06
    WIDTH 0.02 0.2
    CONCAVE_SPACING 0.05
    CONVEX_SPACING 0.025
    MIN_AREA 0.0035
    GRID_SPACING 0.001
    MEASURES > 0.066 < 0.220 opposite
```

Example 2 — Variable Settings for DFM SPEC MPSTITCH

The DFM SPEC MPSTITCH statement shown here is populated with specified variables.

```
VARIABLE MINIMUM_STITCH_WIDTH 0.022
VARIABLE MAXIMUM_STITCH_WIDTH 0.220
VARIABLE CONCAVE_CORNER_SPACING 0.050
VARIABLE CONVEX_CORNER_SPACING 0.026
VARIABLE STITCH_LENGTH 0.066
VARIABLE MINIMUM_MASK_AREA 0.00339
VARIABLE MASK_GRID_SPACING 0.001

DFM SPEC MPSTITCH m1_mpstitch3 3
    STITCH_LENGTH STITCH_LENGTH
    WIDTH MINIMUM_STITCH_WIDTH MAXIMUM_STITCH_WIDTH
    CONCAVE_SPACING CONCAVE_CORNER_SPACING
    CONVEX_SPACING CONVEX_CORNER_SPACING
    MIN_AREA MINIMUM_MASK_AREA
    GRID_SPACING MASK_GRID_SPACING
    MEASURES > STITCH_LENGTH < MAXIMUM_STITCH_WIDTH opposite
```

Example 3 — Using Variable Settings With Coefficients in DFM SPEC MPSTITCH

The DFM SPEC MPSTITCH statement shown here is populated with specified variables which have values altered by their respective coefficients.

```
DFM SPEC MPSTITCH M4_mpstitch 3
    STITCH_LENGTH MINIMUM_STITCH_LENGTH/2
    WIDTH MINIMUM_STITCH_WIDTH/6 MAXIMUM_STITCH_WIDTH
    CONCAVE_SPACING CONCAVE_CORNER_SPACING/4
    CONVEX_SPACING CONVEX_CORNER_SPACING/4
    MIN_AREA MINIMUM_MASK_AREA/4
    GRID_SPACING .0001
```

INCLUDE TVF tp_model

Multi-Patterning SVRF Commands

Executes the specified encrypted *tp_model.tvf* file and its arguments.

Usage

INCLUDE TVF tp_model.tvf

```
create_tp_model
  -target layer
  -modelSize value
  -stitchCandidates layer
  -targetModel layer
```

INCLUDE TVF tp_model.tvf

```
heal_tp_model
  -masks layers
  -modelMasks layers
  -stitchCandidates layer
```

INCLUDE TVF tp_model.tvf

```
heal_tp_model_conflict
  -target layer
  -modelConflict layer
  -conflict layer
```

INCLUDE TVF tp_model.tvf

```
detect_faces
  -target layer
  -modelSize value
  -internalFaces layer
  -originalFaces layer
  -outsideFaces layer
  -stitchCandidates layer
  -targetModel layer
```

Arguments

- **tp_model.tvf**

A required filename referencing the *tp_model.tvf* file found in the Calibre hierarchy. The path to the encrypted file is built-in and cannot be changed. For Calibre triple-patterning auto-stitching, the filename must be specified as *tp_model.tvf*.

- **create_tp_model**

Generates the stitch model for a given process. One of four function names contained within the *tp_model.tvf* file stored in the Calibre hierarchy. This function name is built-in and cannot be changed. The sub-arguments for `create_tp_model` are:

- **-target layer** — A required user-specified name for the input layer from the source layout design. This argument must be specified first. Shapes on this layer must be polygons.
- **-stitchCandidates layer** — A required keyword and output layer name on which auto-generated stitches are placed. Shapes on this layer must be polygons.
- **-modelSize value** — A required keyword and positive number, divisible by 2 and remaining on grid. This is the size of the stitch.
- **-targetModel layer** — A required keyword and output layer name on which the target model is generated.

- **heal_tp_model**

Corrects the stitch model previously generated by `create_tp_model`. One of four function names contained within the *tp_model.tvf* file stored in the Calibre hierarchy. This function name is built-in and cannot be changed. The sub-arguments for `heal_tp_model` are:

- **-masks layers** — A required user-specified name for the mask output layers. You must specify three layers for triple-patterning.
- **-modelMasks layers** — A required user-specified name for the model mask output layers. You must specify three layers for triple-patterning. Shapes on this layer must be polygons.
- **-stitchCandidates layer** — A required keyword and output layer name on which auto-generated stitches are placed. Shapes on this layer must be polygons.

- **heal_tp_model_conflict**

Generates the stitch model for a given process. One of four function names contained within the *tp_model.tvf* file stored in the Calibre hierarchy. This function name is built-in and cannot be changed. The sub-arguments for `heal_tp_model_conflict` are:

- **-target layer** — A required user-specified name for the input layer from the source layout design. This argument must be specified first. Shapes on this layer must be polygons.
- **-modelConflict layer** — A required keyword and output layer name on which model-related conflicts are placed. Shapes on this layer must be polygons.
- **-conflict layer** — A required keyword and output layer name on which conflicts are placed.

- **detect_faces**

Identifies polygon edges for subsequent manipulation. One of four function names contained within the *tp_model.tvf* file stored in the Calibre hierarchy. This function name is built-in and cannot be changed. The sub-arguments for *detect_faces* are:

- **-target layer** — A required user-specified name for the input layer from the source layout design. This argument must be specified first. Shapes on this layer must be polygons.
- **-internalFaces layer** — A required keyword and output layer name to which the inside target edges are saved.
- **-originalFaces layer** — A required keyword and output layer name to which the target edges are saved.
- **-outsideFaces layer** — A required keyword and output layer name to which the outside facing edges are saved.
- **-modelSize value** — A required keyword and positive number, divisible by 2 and remaining on grid. This is the size of the stitch.
- **-stitchCandidates layer** — A required keyword and output layer name on which auto-generated stitches are placed. Shapes on this layer must be polygons.
- **-targetModel layer** — A required keyword and output layer name on which the target model is generated. Shapes on this layer must be polygons.

Description

The *tp_model.tvf* function as executed by INCLUDE TVF provides four sub-functions:

- *create_tp_model*
- *heal_tp_model*
- *heal_tp_model_conflict*
- *detect_faces*

Each performs successive modifications to the models forming triple-patterning stitches. The *tp_model.tvf* function must be executed after *tp_stitch_gen.tvf*. Arguments listed one per line in the Usage section for each function are for clarity only. Actual command usage prohibits line continuation or formatted whitespace.

Examples

Example 1 — *create_tp_model*

This function generates the stitch model for triple-patterning designs. No line continuation or formatted whitespace is allowed.

```
INCLUDE TVF tp_model.tvf create_tp_model -target CA -stitchCandidates
CA_stitch -modelSize 0.04 -targetModel CA_out
```

Example 2 — heal_tp_model

This function corrects the previously generated stitch model for triple-patterning designs. No line continuation or formatted whitespace is allowed.

```
INCLUDE TVF tp_model.tvf heal_tp_model -stitchCandidates CA_stitch  
-modelMasks mod_mask1 mod_mask2 mod_mask3 -masks mask_1 mask_2 mask_3
```

Example 3 — heal_tp_model_conflict

This function outputs conflicts from the previously generated stitch model for triple-patterning designs. No line continuation or formatted whitespace is allowed.

```
INCLUDE TVF tp_model.tvf heal_tp_model_conflict -target CA -modelConflict  
model_conflict -conflict mask_conflict
```

Example 4 — detect_faces

This function identifies polygon edges for generation of the stitch model for triple-patterning designs. No line continuation or formatted whitespace is allowed.

```
INCLUDE TVF tp_model.tvf detect_faces -target CA  
-stitchCandidates CA_stitch -modelSize 0.04 -targetModel CA_out  
-originalFaces org_face -internalFaces int_face -outsideFaces out_face
```

INCLUDE TVF tp_separator

Multi-Patterning SVRF Commands

Executes the specified encrypted *tp_separator.tvf* file and its arguments.

Usage

INCLUDE TVF tp_separator.tvf

```
-target layer
-defineFace name
  '[{length: constraint [adjacent: constraint] | notFace: name | width: constraint}]'
-measure name
  '['constraint: constraint [option: metric value: value]]'
-modelSeparators layer
-modelSize value
-sqrt2ModelSize value
-stitchCandidates layer
-targetModel layer
-faceCategory value
-passFace layer ...
```

Arguments

- **tp_separators.tvf**

A required filename referencing the *tp_separator.tvf* file found in the Calibre hierarchy. The path to the encrypted file is built-in and cannot be changed. For Calibre triple-patterning auto-stitching, the filename must be specified as *tp_separator.tvf*.

- **-target *layer***

A required input layer name from the source layout design. Shapes on this layer must be polygons.

- **-defineFace *name* '{length: *constraint* [adjacent: *constraint*] | notFace: *name* | width: *constraint*}'**

Classifies edges as directed and applies the specified name. Only one of length, width or notFace arguments may be specified within a -defineFace keyword. Any number of -defineFace keywords may be specified within the INCLUDE TVF command. All arguments provided to -defineFace must be enclosed in a single pair of brackets.

Sub-Arguments

- **length: *constraint* [adjacent: *constraint*]** — An optional keyword and constraint comprised of positive, on-grid values. Classifies edges by the length of the shape containing them. Optionally use adjacent, also a constraint comprised of positive, on-grid values, only used with length, that specifies adjacent distances orthogonal to the length. Similar in function to the CONVEX EDGE command.

- width: *constraint* — An optional keyword and constraint comprised of positive, on-grid values. Classifies edges by the width of shapes containing them.
- notFace: *name* — An optional keyword and name previously defined with other -defineFace or -passFace keywords. Classifies edges of all shapes other than those matching constraints of length or width.
- **-measure *name* ['**constraint: *constraint*** [metric: *name* [value: *value*]]']**
Selects edges by a user-specified constraint and saves the selected edges to their original name. All arguments provided to -measure must be enclosed in a single pair of brackets.
- Sub-arguments**
 - **constraint: *constraint*** — One required constraint establishing numeric boundaries for the edge measurement. The constraint must comprise positive values and be coincident with grid.
 - metric: *name* [value: *value*] — An optional keyword pair defining the applied metrics specified for the edge measurement, and optionally, the orthogonal distance of the metric you specify. The value option must be a positive number and coincident with grid. Not all metrics use values. There are two supported metrics:
 - OPPOSITE
 - OPPOSITE EXTENDED
- **-modelSeparators *layer***
A required keyword specifying the layer to which output model separators are saved.
- **-modelSize *value***
A required keyword specifying the model size. The value must be a positive, on-grid number.
- **-sqrt2ModelSize *value***
A required keyword specifying an approximation of the square root of 2 multiplied by the value of modelSize. The value must be a positive, on-grid number.
- **-stitchCandidates *layer***
A required keyword and input layer name from which stitch candidate shapes are output from the stitch generation operation.
- **-targetModel *value***
A required keyword and output layer name from which the target model shapes are input.
- **-faceCategory *value***
An optional keyword and value to categorize the edges classified as faces. Current values include 3 or 5. The default value is 5.

- -passFace *layer* ...

An optional keyword and layer used to pass previously derived edges to the -define_face and -measure keywords. The specified layer must contain edges only.

Description

The *tp_separator.tvf* function as executed by INCLUDE TVF is provided to generate separators that are aware of triple-patterning stitch candidate placement. This function is run after *tp_model.tvf*. Arguments listed one per line in the Usage section for each function are for clarity only. Actual command usage prohibits line continuation or formatted whitespace.

Examples

This example shows the INCLUDE TVF command executing the *tp_separator.tvf* function with its associated keywords and variables. No line continuation or formatted whitespace is allowed within the INCLUDE TVF command line.

```
// Define model sizing parameter.  
// Must be divisible by 2 and remain on grid.  
// Siemens EDA recommends using a small value.  
VARIABLE MODEL_SIZE 0.002  
  
// Define the square root of 2 * model sizing parameter.  
// Round up to nearest unit.  
VARIABLE MODEL_SIZE_SQUARE_ROOT_2 0.003  
  
INCLUDE TVF tp_separator.tvf -target M1 -targetModel M1_MODEL  
-stitchCandidates M1_SC -modelSize MODEL_SIZE -sqrt2ModelSize  
MODEL_SIZE_SQUARE_ROOT_2 -faceCategory 5 -modelSeparators SEP.M1_MODEL  
-defineFace ALL [length: >= 0.001] -defineFace TIP [length: < 0.042  
adjacent: >= 0.020] -defineFace TIP50 [length: < 0.050] -defineFace W56  
[width: < 0.056] -measure ALL [constraint: < 0.070] -measure ALL  
[constraint: < 0.074 option: OPPOSITE EXTENDED value: 0.026] -measure TIP  
[constraint: < 0.080 option: OPPOSITE EXTENDED value: 0.026] -measure  
TIP50 W56 [constraint: < 0.080 option: OPPOSITE] 0
```

INCLUDE TVF tp_stitch_gen

Multi-Patterning SVRF Commands

Executes the specified encrypted *tp_stitch_gen.tvf* file and its arguments.

Usage

```
INCLUDE TVF tp_stitch_gen.tvf
  -target layer
  -maxStitchLength value
  -maxStitchWidth value
  -minStitchLength value
  -minStitchWidth value
  -stitchCandidates layer
  -anchor1 layer
  -anchor2 layer
  -anchor3 layer
  -concaveCornerSpacing value
  -convexCornerSpacing value
  -gridSpacing mask_space grid_space
  -minArea value
  -separators layer
  -stitchKeepout layer
  -measure '['constraint: constraint [option: metric [value: value]] [rule: name]]']'
```

Arguments

- **tp_stitch_gen.tvf**

A required filename referencing the *tp_stitch_gen.tvf* file found in the Calibre hierarchy. The path to the encrypted file is built-in and cannot be changed. For Calibre triple-patterning auto-stitching, the filename must be specified as *tp_stitch_gen.tvf*.

- **-target *layer***

A required user-specified name for the input layer from the source layout design.

- **-minStitchWidth *value***

A required keyword and value in microns specifying the minimum width a stitch can be generated.

- **-maxStitchWidth *value***

A required keyword and value in microns specifying the maximum width a stitch can be generated.

- **-minStitchLength *value***

A required keyword and value in microns specifying the minimum length a stitch can be generated.

- **-maxStitchLength *value***

A required keyword and value in microns specifying the maximum length a stitch can be generated.

- **-stitchCandidates *layer***

A required keyword and output layer name on which auto-generated stitches are placed.

- **-anchor1 *layer***

An optional keyword and user-specified layer used for anchor shapes to be applied to mask layer 1.

- **-anchor2 *layer***

An optional keyword and user-specified layer used for anchor shapes to be applied to mask layer 2.

- **-anchor3 *layer***

An optional keyword and user-specified layer used for anchor shapes to be applied to mask layer 3.

- **-concaveCornerSpacing *value***

An optional keyword that designates a distance from the vertex of the outside of the corner where no stitches are permitted to be placed.

- **-convexCornerSpacing *value***

An optional keyword that designates a distance from the vertex of the inside of a corner where no stitches are permitted to be placed.

- **-gridSpacing *mask_space grid_space***

An optional keyword used to specify the minimum mask grid spacing and DBU Per User Unit spacing, respectively. You must specify both spacing values.

- **-measure '['constraint: *constraint* [option: *metric* [value: *value*]] [rule: *name*]']'**

An optional keyword used to avoid mutually exclusive stitches placed on the same polygon during stitch generation. You must specify all arguments within a single pair of brackets.

constraint:

A required boundary of measurement between edges of the stitch candidates. The constraint values must be positive and be on grid.

option:

An optional keyword pair specifying the metrics for edge measurement. For OPPOSITE EXTENDED only, value provides the orthogonal distance included in the measurement. The value option must be a positive number and be on grid. Supported metrics include:

OPPOSITE

OPPOSITE EXTENDED [value: *value*]

PERPENDICULAR ALSO

rule:

An optional name reported to the transcript with which to associate the rule.

- `-minArea value`

An optional keyword used for determining the smallest allowable area of any generated stitch.

- `-separators layer`

An optional user-derived layer used for determining potential stitch sites.

- `-stitchKeepout layer`

An optional user-derived layer used to specify areas where stitch placement is not permitted.

Description

The `tp_stitch_gen.tvf` function as executed by INCLUDE TVF is provided to determine legal triple-patterning stitch candidate locations. This function is run prior to `tp_separator.tvf` and `tp_model.tvf` functions. Arguments listed one per line in the Usage section for the function are for clarity only. Actual command usage prohibits line continuation or formatted whitespace.

Examples

Example 1

This example demonstrates usage of the `tp_stitch_gen.tvf` function used for triple-patterning. In this example all variables and input layers are written in uppercase. Variables may be used for keywords supporting numeric values. Whitespace cannot be used for padding in the INCLUDE TVF function.

```
//DRM-specified values for stitch placement
VARIABLE MINIMUM_STITCH_WIDTH 0.022
VARIABLE MAXIMUM_STITCH_WIDTH 0.220
VARIABLE CONCAVE_CORNER_SPACING 0.050
VARIABLE CONVEX_CORNER_SPACING 0.026
VARIABLE MINIMUM_STITCH_LENGTH 0.066
VARIABLE MAXIMUM_STITCH_LENGTH 0.375
VARIABLE MINIMUM_MASK_AREA 0.00339
VARIABLE MASK_GRID_SPACING 0.001
VARIABLE DBU_PER_GRID_SPACING 10

INCLUDE TVF tp_stitch_gen.tvf -target M1 -separators
SEP.M1.ALL -anchor1 M1_E1 -anchor2 M1_E2 -anchor3 M1_E3 -minStitchWidth
MINIMUM_STITCH_WIDTH -maxStitchWidth MAXIMUM_STITCH_WIDTH
-minStitchLength MINIMUM_STITCH_LENGTH -maxStitchLength
MAXIMUM_STITCH_LENGTH -stitchCandidates M1_SC -minArea MINIMUM_MASK_AREA
-concaveCornerSpacing CONCAVE_CORNER_SPACING -convexCornerSpacing
CONVEX_CORNER_SPACING -gridSpacing MASK_GRID_SPACING DBU_PER_GRID_SPACING
-stitchKeepout FORBIDDEN_REGION
```

Example 2

This example demonstrates syntactic usage of the -measure keyword used for measuring potential occurrences of mutually-exclusive stitch candidates. The first example uses the OPPOSITE EXTENDED metric and its associated orthogonal extension value. All arguments must be enclosed within brackets.

```
-measure [constraint: < 0.090 option: OPPOSITE EXTENDED value: 0.026 rule:  
SC.S.3]  
-measure [constraint: < 0.074 option: PERPENDICULAR ALSO rule: SC.S.3a]
```

INCLUDE TVF mp_stitch_gen

Multi-Patterning SVRF Commands

Executes the specified encrypted *mp_stitch_gen.tvf* file and its arguments.

Usage

INCLUDE TVF mp_stitch_gen.tvf

```
mask_count
-target layer
-maxStitchLength value
-maxStitchWidth value
-minStitchLength value
-minStitchWidth value
-stitchCandidates layer
-anchors layer1 layer2 layer3 [layer4]
-concaveCornerSpacing value
-convexCornerSpacing value
-gridSpacing mask_space grid_space
-minArea value
-selfViolatedTarget layer
-separators layer
-stitchKeepout layer
-measure '['constraint: constraint [option: metric [value: value]] [rule: name]]']'
```

Arguments

- **mp_stitch_gen.tvf**

A required filename referencing the *mp_stitch_gen.tvf* file found in the Calibre hierarchy. The path to the encrypted file is built-in and cannot be changed. For Calibre multi-patterning auto-stitching, the filename must be specified as *mp_stitch_gen.tvf*.

- **mask_count**

A required number, either 3 or 4, that specifies the number of masks to be generated by DFM MP.

- **-target layer**

A required user-specified name for the input layer from the source layout design.

- **-minStitchWidth value**

A required keyword and value in microns specifying the minimum width a stitch can be generated.

- **-maxStitchWidth value**

A required keyword and value in microns specifying the maximum width a stitch can be generated.

- **-minStitchLength *value***

A required keyword and value in microns specifying the minimum length a stitch can be generated.

- **-maxStitchLength *value***

A required keyword and value in microns specifying the maximum length a stitch can be generated.

- **-stitchCandidates *layer***

A required keyword and output layer name on which auto-generated stitches are placed.

- **-anchors *layer1 layer2 layer3 [layer4]***

An optional keyword and user-specified layers used for anchor shapes to be applied to all mask layers. The number of layers specified must match the mask_count argument.

- **-concaveCornerSpacing *value***

An optional keyword that designates a distance from the vertex of the outside of the corner where no stitches are permitted to be placed.

- **-convexCornerSpacing *value***

An optional keyword that designates a distance from the vertex of the inside of a corner where no stitches are permitted to be placed.

- **-gridSpacing *mask_space grid_space***

An optional keyword used to specify the minimum mask grid spacing and DBU Per User Unit spacing, respectively. You must specify both spacing values.

- **-measure '['*constraint: constraint [option: metric [value: value]] [rule: name]*']'**

An optional keyword used to avoid mutually exclusive stitches placed on the same polygon during stitch generation. You must specify all arguments within a single pair of brackets.

constraint:

A required boundary of measurement between edges of the stitch candidates. The constraint values must be positive and be on grid.

option:

An optional keyword pair specifying the metrics for edge measurement. For OPPOSITE EXTENDED only, value provides the orthogonal distance included in the measurement. The value option must be a positive number and be on grid. Supported metrics include:

CORNER TO CORNER — Provides a spacing measurement between corner-to-corner shapes (those with no adjacent facing edges).

OPPOSITE — Provides a spacing measurement only between facing edges of shapes.

OPPOSITE EXTENDED [value: *value*] — Provides spacing measurement between the facing edges of shapes, and may be optionally appended with a lateral extension value, broadening the OPPOSITE measurement.

PERPENDICULAR ALSO — Provides a spacing measurement only between shapes with edges perpendicular to each other.

rule:

An optional name reported to the transcript with which to associate the rule.

- -minArea *value*

An optional keyword used for determining the smallest allowable area of any generated stitch.

- -selfViolatedTarget *layer*

An optional input polygon layer, as a subset of the target layer, typically comprising polygons generated by DFM MP using the SELF_CONTAINED_VIOLATION_POLYGONS or SELF_CONFLICT_SEPARATORS error keywords. This increases the number of potential stitch candidates on the target layer.

- -separators *layer*

An optional user-derived layer used for determining potential stitch sites.

- -stitchKeepout *layer*

An optional user-derived layer used to specify areas where stitch placement is not permitted.

Description

The *mp_stitch_gen.tvf* function as executed by INCLUDE TVF is provided to determine legal triple- and quadruple-patterning stitch candidate locations. Arguments listed one per line in the Usage section for the function are for clarity only. Actual command usage prohibits line continuation or formatted whitespace.

Note

 Siemens EDA recommends developing all new recipes using *mp_stitch_gen.tvf* and DFM MP (with its stitching option implemented) instead of *tp_stitch_gen.tvf* and RET TP.

Examples

Example 1

This example demonstrates usage of the *mp_stitch_gen.tvf* function used for triple- and quadruple-patterning. In this example all variables and input layers are written in uppercase. Variables may be used for keywords supporting numeric values. There can be no padding of whitespace used within the INCLUDE TVF function.

```
//DRM-specified values for stitch placement
VARIABLE MINIMUM_STITCH_WIDTH 0.022
VARIABLE MAXIMUM_STITCH_WIDTH 0.220
VARIABLE CONCAVE_CORNER_SPACING 0.050
VARIABLE CONVEX_CORNER_SPACING 0.026
VARIABLE MINIMUM_STITCH_LENGTH 0.066
VARIABLE MAXIMUM_STITCH_LENGTH 0.375
VARIABLE MINIMUM_MASK_AREA 0.00339
VARIABLE MASK_GRID_SPACING 0.001
VARIABLE DBU_PER_GRID_SPACING 10

INCLUDE TVF mp_stitch_gen.tvf 3 -target M1 -separators
SEP.M1.ALL -anchors M1_E1 M1_E2 M1_E3 -minStitchWidth
MINIMUM_STITCH_WIDTH -maxStitchWidth MAXIMUM_STITCH_WIDTH
-minStitchLength MINIMUM_STITCH_LENGTH -maxStitchLength
MAXIMUM_STITCH_LENGTH -stitchCandidates M1_SC -minArea MINIMUM_MASK_AREA
-concaveCornerSpacing CONCAVE_CORNER_SPACING -convexCornerSpacing
CONVEX_CORNER_SPACING -gridSpacing MASK_GRID_SPACING DBU_PER_GRID_SPACING
-stitchKeepout FORBIDDEN_REGION
```

Example 2

This example demonstrates syntactic usage of the -measure keyword used for measuring potential occurrences of mutually-exclusive stitch candidates. The first example uses the OPPOSITE EXTENDED metric and its associated orthogonal extension value. All arguments must be enclosed within square brackets.

```
-measure [constraint: < 0.090 option: OPPOSITE EXTENDED value: 0.026 rule:
SC.S.3]
-measure [constraint: < 0.074 option: PERPENDICULAR ALSO rule: SC.S.3a]
```

RET DPSTITCH

Multi-Patterning SVRF Commands

Calls a LITHO FILE statement containing a setlayer dpstitch command that generates stitch candidate shapes. Used with RET NMDPC and DFM DP.

Usage

```
RET DPSTITCH layer [layer...]  
FILE {filename | name}  
MAP output_type
```

Arguments

- *layer*

A required argument specifying a layer upon which to locate and output stitch candidates. You can specify one or more layers in one statement. The *layer* statements in the LITHO FILE statement determine their function.

- FILE {*filename* | *name*}

A required keyword, followed by one of the following:

filename — A name indicating the path and name of the setup file. The *filename* parameter can contain environment variables. For information regarding the use of environment variables in the *filename* parameter, refer to “[Environment Variables in Pathname Parameters](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

name — A parameter of a LITHO FILE specification statement. Specifications within the LITHO FILE statement are called by the RET NMDPC or DFM DP commands.

- MAP *output_type*

A required keyword, followed by the name of a layer from the setup file. This layer is then mapped to the layer in the rule file for output.

Description

This command is used to find qualified stitch sites and output their respective layers. It can be specified any number of times in the rule file. Instances of the command that are identical, except for the MAP argument, run concurrently.

Examples

Two methods of obtaining stitches are available for the RET DPSTITCH command. Siemens EDA recommends the simplified method invoked with the dpStitchUseV2Syntax keyword.

- Simplified Method — This method requires keywords dpStitchUseV2Syntax and criticalDistance. The order of layers specified by the inputLayerOrder keyword and the RET DPSTITCH commands must match, although the names can be different. Values for tilemicrons, criticalDistance, and other parameters depend on your manufacturing process. All keywords are case-insensitive.

```
VARIABLE stitch_size 0.060
VARIABLE min_s2s 0.054

LITHO FILE sc_setup [
    dpStitchUseV2Syntax;
    tilemicrons = 40;
    tilemicronsMode = adjust;

    dpStitchSize
        ($stitch_size <= length <= 0.120) for (0 < width <= 0.120):
        (0.08 <= length <= 0.250) for (0 < width <= 0.250):
        (0.08 <= length <= 0.5) otherwise;
    dpSpacing (any,any): $min_s2s otherwise;
    dpFaceClass name(tip):
        true for (0.032 <= length <= 0.080):
        false otherwise;
    minIntersectionArmLength 0.040 otherwise;
    intersectionStitch
        true for numArms >= 3 and minWidth >= 0.032
        and maxWidth <= 0.500:
        false otherwise;
    minConcaveCornerOverlap 0.060 for width < 0.060:
        0.020 for width < 0.080:
        0 otherwise;

    criticalDistance = 0.032;
    minArea = 0.0046;
    maxCutLength = 0.5;
    maskGridSpacing = 0.001;
    inputLayerOrder = (target, stitchkeepout, anchor0, anchor1);
]

stitchCandidates = RET DPSTITCH input stitch_keep_out
m1_e1_anchor m1_e2_anchor
FILE sc_setup MAP stitchCandidates
```

- Legacy Method — This method uses dpStitchCandidates, setlayer, and other LITHO FILE arguments including layer, tilemicrons and the options block.

The layers in the options block and those listed in the setlayer dpstitch command are related by order; their names can be different, but their listed order of sequence is critical. In the example, layer M1 appears first in the options block and so must also appear first in the `setlayer dpstitch` command; layer m1_f appears second and therefore must also appear second in the setlayer dpstitch command. The values for tilemicrons and criticalDistance depend on your manufacturing process.

```
VARIABLE stitch_size 0.060
VARIABLE min_s2s 0.054

LITHO FILE sc_setup [
    layer <drawn> visible clear
    layer <other_layers> visible clear //repeat as necessary
    tilemicrons <num>

    svrf_var_import min_s2s
    svrf_var_import stitch_size

    options opt {
        version 1
        layer <drawn> opc clear
        layer <other_layers> visible clear

        dpStitchSize
            ($stitch_size <= length <= 0.120)
                for (0 < width <= 0.120):
                    (0.08 <= length <= 0.5) otherwise;
        dpSpacing (any,any): $min_s2s otherwise;
        dpFaceClass name(tip):
            true for (0.032 <= length <= 0.080):
                false otherwise;
            minIntersectionArmLength 0.040 otherwise;
        intersectionStitch
            true for numArms >= 3 and minWidth >= 0.032
                and maxWidth <= 0.500:
                false otherwise;
            minConcaveCornerOverlap 0.060 for width < 0.060:
                0.020 for width < 0.080:
                0 otherwise;

        dpStitchCandidates criticalDistance <value> \
            -outLayer stitchCandidates
    }

    setlayer stitchCandidates = dpstitch <drawn> <other_layers> \
        OPTIONS opt \
        MAP stitchCandidates
]

stitchCandidates = RET DPSTITCH input stitch_keep_out
m1_e1_anchor m1_e2_anchor
FILE sc_setup MAP stitchCandidates
```

- To control the size of mid-line stitches, add a `dpStitchSize` statement to the options block between the layer and `dpStitchCandidates` lines. This defines the length of the stitch for different line widths.
- To describe geometry-dependent spacing requirements, use `dpFaceClass` to define the geometry and `dpSpacing` to determine where there should be a stitch. The `dpSpacing` command must be after the `dpFaceClass` definition. Both of these go in the options block above the `dpStitchCandidates` command.

- To permit stitches to be placed over intersections, use [minIntersectionArmLength](#) and [intersectionStitch](#). Add these to the options block above the dpStitchCandidates command.
- To define an upper limit on shapes that can receive stitches, add the maxCutLength argument to the dpStitchCandidates command. (The lower limit is set by criticalDistance.)
- To make use of precoloring (anchors) and keepout layers, add the following arguments to the dpStitchCandidates command:

```
anchor0 <mask0_precolor>
anchor1 <mask1_precolor>
stitchKeepout <keepout>
```

Replace the <terms> with your layers. The arguments can go all on one line, or on separate lines if each ends with the command continuation character ()).

The layers used in these arguments must be listed in the layer command and setlayer dpstitch command where the examples shows <other_layers>.

```
LITHO FILE dpSC_setup [
    layer M1
    layer m1_f
    ...

    options OPT {
        layer M1 opc clear
        layer m1_f visible clear

        dpStitchCandidates criticalDistance ... \
            stitchKeepout metal \
            -outlayer stitchCandidates \
            ...
    }

    setlayer stitchCan = dpstitch M1 m1_f MAP stitchCandidates OPTIONS OPT
]

stitchCan = RET DPSTITCH M1 m1_f MAP stitchCandidates FILE dpSC_setup
```

All RET DPSTITCH operations use the same layers and LITHO FILE statement; they differ only in the MAP argument.

Related Topics

[RET NMDPC](#)

RET NMDPC

Multi-Patterning SVRF Commands

Invokes Calibre nmDPC and splits the target layer into two masks.

Usage

```
RET NMDPC action_layer drawn_layer [action_layerN...]  
  FILE {filename | name}  
  MAP type
```

Description

The RET NMDPC command invokes Calibre nmDPC and splits an input target layer into two masks based on the specified action layers. If the target layer cannot be successfully split into two masks, RET NMDPC outputs various error layers.

Action layers determine whether two adjacent shapes must be output to different masks.

When multiple RET NMDPC commands in the SVRF are written identically (including layer ordering but excepting their MAP types), these RET operations are compiled and executed once, simultaneously generating multiple output layers.

Note

 Siemens EDA recommends migrating all RET NMDPC functions to DFM DP. RET NMDPC will be deprecated in a future Calibre release.

Arguments

- *action_layer*

A required argument specifying an action layer. You must specify one *action_layer* that is a separator layer, even if coloring is random. This layer can be a polygon type layer, an edge type layer, or an error type layer. Additional action layers can be specified after the *drawn_layer* argument. For more information on these layer types, refer to the “[Layers](#)” section of the *Calibre Verification User’s Manual*.

- *drawn_layer*

A required argument specifying the layer to be decomposed.

- *action_layerN*

An optional list of supplementary action layers. Each of the layers specified in this list must be a polygon type layer, an edge type layer, or an error type layer. For more information on these layer types, refer to the “[Layers](#)” section in the *Calibre Verification User’s Manual*.

- **FILE {filename | name}**

A required keyword, followed by one of the following:

filename — a name indicating the path and name of the setup file. The filename parameter can contain environment variables. For information regarding the use of environment variables in the filename parameter, refer to “[Environment Variables in Pathname Parameters](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

name — the name of a [LITHO FILE](#) statement. Specifications within the LITHO FILE statement are called by the RET NMDPC command.

Both formats contain setup instructions for Calibre nmDPC and span multiple lines.

- **MAP type**

A required keyword and argument specifying the type of output that is passed to the output layer in the rule file. The output type must be one of the following:

anchor_conflict — Outputs a subset of the conflict layer containing all target shapes in connection with any color conflict between anchors. This error occurs when your design contains two anchors of either color that force an unresolvable coloring choice.

anchor_path — Outputs an error visualization layer containing thin polygon markers associated with conflicts between anchored target shapes. Each marker floats near or touches one side of all target shapes and separators associated with a color alternation conflict between two anchored target shapes.

anchor_self_conflict — Outputs a subset of the conflict layer containing any single target shape with a color conflict between anchors on that shape. This error occurs when a single shape is anchored for both mask0 and mask1.

conflict — Outputs a layer containing all target geometries in violation with input constraints. The conflict layer is a superset of all error output layers.

loop — Outputs a subset of the conflict layer containing the smallest odd cycles formed by separator layers only. This output layer includes the drawn layer and the separator layers.

mask0 — Outputs a layer containing mask0 geometries.

mask1 — Outputs a layer containing mask1 geometries.

ring — Outputs an error visualization layer containing donut polygons. Each donut polygon corresponds to an odd cycle and touches all involved drawn shapes. The locations where rings touch adjacent polygons represent separator locations involved in odd cycles. Increasing the space at these locations within the ring allows the adjacent polygons to be on the same mask, which removes the odd cycle. See “[Debugging DP Layout Errors](#)” on page 327 for more information. For more information regarding ring exclusions see [Figure 3-26](#) on page 145.

ring_tie — Outputs a combination of error layers ring and warning. Either ring or ring_tie can be used in conjunction with the warning error layer. [Figure 3-27](#) on

page 145 shows differences between ring and ring_tie error layers. The ring and ring_tie error layers cannot be used together and they result in a compile error:

ERROR: Output layers ring and ring_tie can not be requested together

self_conflict — Outputs a subset of the conflict layer containing single target geometries in violation with opposite mask constraints. The self_conflict layer can also provide confidence that no flat versus hierarchical error representation inconsistencies exist. This error occurs where one separator touches a single shape in two places.

Figure 3-28 on page 146 depicts a basic rendering of the self_conflict layer and its corresponding separator derivation.

Note

 Although separators connecting to only one polygon (dangling separators) are reported by self_conflict, Siemens EDA cannot guarantee proper treatment of invalid separators.

warning — Outputs an error visualization layer containing donut polygons that touch all target shapes adjacent to the conflict area that form even cycles. The even cycles represent potential errors dependent on how adjacent odd cycles are corrected. The warning markers are not errors, but are hints to help the designer fix the adjacent odd-cycle ring errors more efficiently. The warning error layer can be used with either ring or ring_tie error layers.

Examples

Example 1

The following code shows the LITHO FILE statement followed by RET NMDPC commands to invoke Calibre nmDPC and output either the action conflict layer or the split mask layers. If no action layer conflicts are detected the layout is split and mask0 and mask1 are output. However, if conflicts are found, then the conflict layer captures these conflicts at the lowest level of hierarchy, and this layer is output.

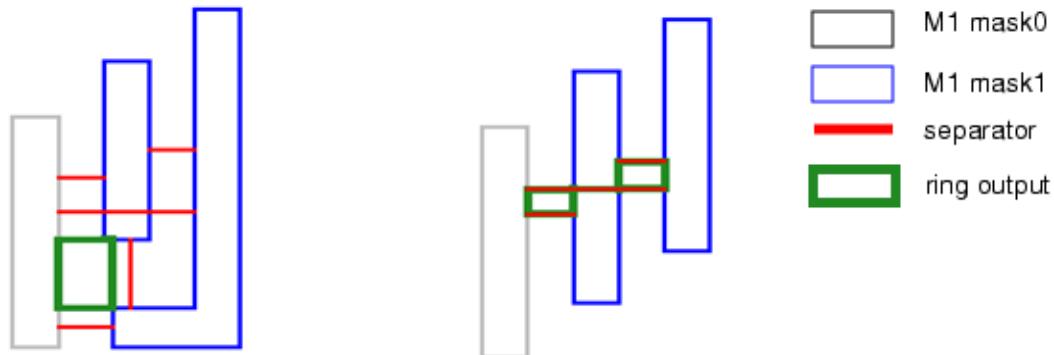
```
LITHO FILE act_file [
    action act_layer 0 1;
    action opt_act_layer 1 0;
    resolve;
]

conflict = RET NMDPC act_layer M1 opt_act_layer FILE act_file MAP conflict
mask0 = RET NMDPC act_layer M1 opt_act_layer FILE act_file MAP mask0
mask1 = RET NMDPC act_layer M1 opt_act_layer FILE act_file MAP mask1
```

Example 2

The ring warning layer is not output where no contained odd-cycle holes are formed or where separator violations cross over intermediate target input shapes:

Figure 3-26. ring Warning Exclusion Scenario

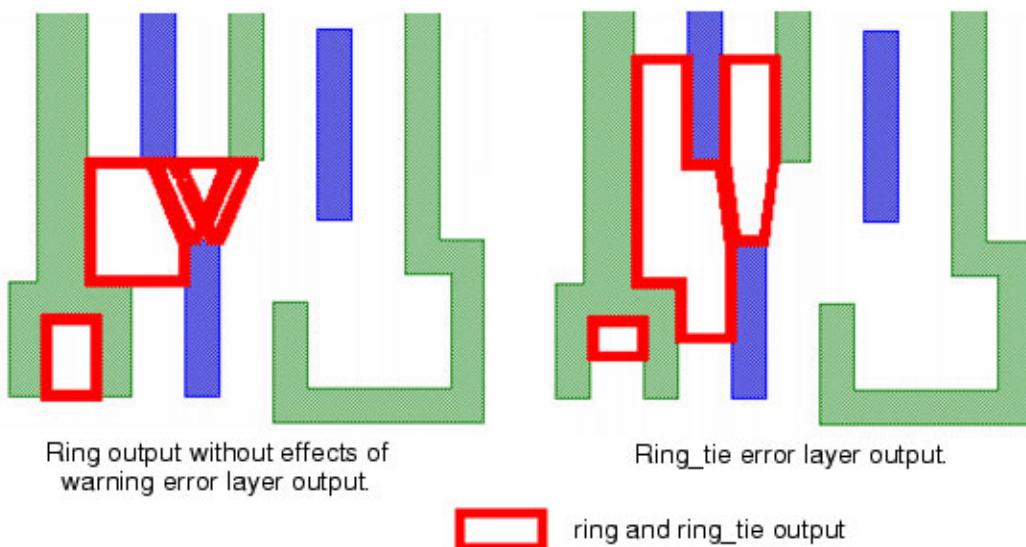


Ring warnings are output wherever odd cycles appear.

Example 3

Differences between ring and ring_tie error layers:

Figure 3-27. ring_tie Appearance



Example 4

Results and derivation of the self_conflict error layer:

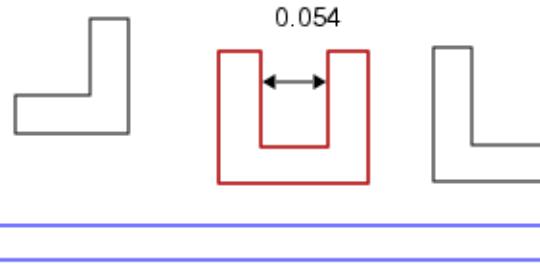
Figure 3-28. self_conflict Error Layer

Self conflict of a single shape from side-to-side separator derivation

 M1 mask0

 M1 mask1

 self_conflict error



```
m1_side2side = EXT m1_side < 0.06 OPPOSITE PARALLEL ONLY PROJ > 0
```

Depending on your rules, self_conflict may also generate errors similar to these:



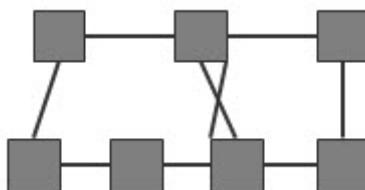
Tip

-  Do not alter any separator spacing spanned by both an odd-cycle ring marker and an even-cycle warning marker. Changing these separators creates a new odd-cycle error. Make corrections near separator spacing along the odd-cycle areas that have no adjacent warning markers. The use of warning markers changes the conformal shape of the adjacent error markers to help you more easily locate the separator spacings that are shared by the warning markers.
-

Crossing Separator Filtering

The filtering algorithm from release 2014.4 visually enhanced odd-cycle rings to facilitate debugging. In specific situations where multiple separators connect the same two polygons and cross each other, NMDPC may output odd-cycle conflict rings appearing to include even cycles as seen in [Figure 3-29](#).

Figure 3-29. Crossing Separator Filtering



Related Topics

[Set Up and Run Calibre Multi-Patterning](#)

RET QP

Multi-Patterning SVRF Commands

Invokes Calibre quadruple-patterning using the RET QP command. Splits the target layer into four masks and generates error layers.

Usage

RET QP

```
target
[additional_layers ...]
FILE {filename | name}
MAP type
```

Arguments

- **target**

A required argument specifying the target mask layer. This layer must contain polygons. The target layer argument must appear first. Additional target layers can be specified in subsequent RET QP statements.

- **additional_layers ...**

Optional layer names specifying separator layers and layers containing user-defined anchor shapes to identify a target shape as a specific mask. Any number of additional layers may be specified.

- **FILE {filename | name}**

A required keyword, followed by one of the following methods:

filename — A name indicating the path and name of the [LITHO FILE](#) statement. The filename parameter can contain environment variables. For information regarding the use of environment variables in the filename parameter, refer to “[Environment Variables in Pathname Parameters](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

name — A unique name for the LITHO FILE statement called by the RET QP command.

- **MAP type**

A required keyword and argument pair specifying the type of output that is passed to the output layer in the rule file. MASK1, MASK2, MASK3, and MASK4 are required. The output type may be one of the following, each specified on a separate RET QP command line:

MASK1 — A required type that outputs a layer containing MASK1 geometries.

MASK2 — A required type that outputs a layer containing MASK2 geometries.

MASK3 — A required type that outputs a layer containing MASK3 geometries.

MASK4 — A required type that outputs a layer containing MASK4 geometries.

For error visualization output, the output type may be one of the following keywords, each specified on a separate RET QP command line. For more information regarding error visualization and the implementation of these keywords, see the section entitled “[MP Visualization and Error Resolution](#)” on page 335.

ANCHOR_SELF_CONFLICT — Outputs a subset of the conflict layer containing any single target shape with a color conflict between anchors on that shape. This error occurs when a single shape is anchored for multiple masks; MASK₁, MASK_n, and so forth. This is a type-1 layer.

CONFLICT — Outputs a layer containing all target geometries in violation with all visualization errors. The layer is formed by coloring violations. It then expands from these initial violations to all polygons with connected separators. It stops expanding at anchors or edges that are removed by reduction. A coloring violation is defined as two polygons on the same mask forcing a separator between them. This is a type-1 layer.

DIRECT_ANCHOR_CONFLICT CLUSTER — Outputs a layer containing separators between two polygons that were both anchored to the same mask. This is a type-3 layer.

REMAINING_VIOLATIONS — Outputs a layer containing merged polygons and separators belonging to components from the CONFLICT layer which have no violations associated with them. This is a type-1 layer.

REMAINING_VIOLATIONS_POLYGONS — Outputs a layer containing polygons belonging to components from the CONFLICT layer which have no violations associated with them. This is a type-1 layer.

REMAINING_VIOLATIONS_SEP CLUSTER — Outputs a layer containing separators belonging to components from the CONFLICT layer which have no violations associated with them. This is a type-3 layer.

SELF_CONFLICT_SEP CLUSTER — Outputs a layer containing the edges of separators marking a violation from one side of a polygon to another side of the same polygon. This is a type-3 layer.

SELF_CONTAINED_POLYGONS — Outputs a layer containing polygons belonging to components that are not colorable. Displays separators and shapes where five shapes each have a separator touching the other four shapes. Identical to a K5 violation. This is a type-1 layer.

SELF_CONTAINED_SEP CLUSTER — Outputs a layer containing separators belonging to components that are not colorable. Displays separators and shapes where five shapes each have a separator touching the other four shapes. Identical to a K5 violation. This is a type-3 layer.

SELF_CONTAINED_VIOLATION — Outputs a layer containing merged polygons and separators belonging to components that are not colorable. Displays separators and shapes where five shapes each have a separator touching the other four shapes. Identical to a K5 violation. This is a type-1 layer.

VIOLATION_SEPARATORS CLUSTER — Outputs all separators between two polygons colored the same. This is a type-3 layer.

Graph Reduction

For graph reduction, the output type may be this keyword, specified on its own RET QP command line:

REDUCED_SEPARATORS_LDR CLUSTER — Reports the edges remaining in the graph after Low Degree Node Removal is performed.

Description

The RET QP command is used to invoke Calibre quadruple-patterning. It splits an input layer into four masks based on optional anchoring and system-determined coloring. If the input layer cannot be split into four masks, RET QP outputs conflict errors on shapes or edges.

When multiple RET QP operations are present and differ only with respect to their MAP type, Calibre performs the RET operation once, generating multiple output layers simultaneously. For important exceptions regarding layer derivation support see “[Guidelines for Deriving Good Separator and Action Layers](#)” on page 346.

Note

 Siemens EDA recommends migrating all RET QP functions to DFM MP. RET QP is deprecated as of Calibre release 2016.1 and will be obsoleted from a future Calibre release.

Examples

The following is an example of the LITHO FILE followed by RET QP commands to invoke Calibre quadruple-patterning and output four decomposed masks and any associated error layers. The masks are output whether conflicts exist or not. If conflicts are found, the error layers capture these conflicts on error shapes (either edges or polygons) at the lowest level of hierarchy possible, and the layer is output.

```
LITHO FILE ca_qp [
    target CA;
    separator SEP.S2S;
    separator SEP.T2T;
    anchor1 CA_anchor1;
    anchor2 CA_anchor2;
    anchor3 CA_anchor3;
    anchor4 CA_anchor4;
]

CA_Mask1_Out = RET QP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 CA_anchor4 FILE ca_qp MAP MASK1
CA_Mask2_Out = RET QP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 CA_anchor4 FILE ca_qp MAP MASK2
CA_Mask3_Out = RET QP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 CA_anchor4 FILE ca_qp MAP MASK3
CA_Mask4_Out = RET QP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 CA_anchor4 FILE ca_qp MAP MASK4

CA_Conflict = RET QP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 CA_anchor4 FILE ca_qp
    MAP CONFLICT

CA_Anchor_Self_Conflict = RET QP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 CA_anchor4 FILE ca_qp
    MAP ANCHOR_SELF_CONFLICT

CA_Remaining_Violations = RET QP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 CA_anchor4 FILE ca_qp
    MAP REMAINING_VIOLATIONS

CA_Self_Contained_Violation = RET QP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 CA_anchor4 FILE ca_qp
    MAP SELF_CONTAINED_VIOLATION
```

RET TP

Multi-Patterning SVRF Commands

Invokes Calibre triple-patterning using the RET TP command. Splits the target layer into three masks and generates error layers.

Usage

RET TP

```
target_layer
[additional_layers ...]
FILE {filename | name}
MAP type
```

Arguments

- *target_layer*

A required argument specifying the target mask layer. This layer must contain polygons. The target layer argument must appear first. Additional target layers can be specified in subsequent RET TP statements.

- *additional_layers ...*

Optional layer names specifying separator layers and layers containing user-defined anchor shapes to identify a target shape as a specific mask. Any number of additional layers may be specified.

- **FILE {filename | name}**

A required keyword, followed by one of the following:

filename — A name indicating the path and name of the [LITHO FILE](#) statement. The filename parameter can contain environment variables. For information regarding the use of environment variables in the filename parameter, refer to “[Environment Variables in Pathname Parameters](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

name — A unique name for the LITHO FILE statement called by the RET TP command.

- **MAP type**

A required keyword and argument pair specifying the type of output that is passed to the output layer in the rule file. MASK1, MASK2, and MASK3 are required. For mask output, the output type may be one of the following keywords, each specified on a separate RET TP command line:

MASK1 — A required type that outputs a layer containing MASK1 geometries.

MASK2 — A required type that outputs a layer containing MASK2 geometries.

MASK3 — A required type that outputs a layer containing MASK3 geometries.

For error visualization output, the output type may be one of the following keywords, each specified on a separate RET TP command line. For more information regarding error visualization and the implementation of these keywords, see the section entitled “[MP Visualization and Error Resolution](#)” on page 335.

ANCHOR_SELF_CONFLICT — Outputs a subset of the conflict layer containing any single target shape with a color conflict between anchors on that shape. This error occurs when a single shape is anchored for multiple masks; MASK1, MASK_n, and so forth. This is a type 1 layer.

CONFLICT — Outputs a layer containing all target geometries in violation with all visualization errors. The layer is formed by coloring violations. It then expands from these initial violations to all polygons with connected separators. It stops expanding at anchors or edges that are removed by reduction. A coloring violation is defined as two polygons on the same mask forcing a separator between them. This is a type-1 layer.

DIRECT_ANCHOR_CONFLICT CLUSTER — Outputs a layer containing separators between two polygons that were both anchored to the same mask. This is a type 3 layer.

EQUIV_LOOP — Outputs a layer containing merged polygons and edge separators of an equivalence loop where two polygons have a forced equivalence path and a separator between them. This is a type 1 layer.

EQUIV_LOOP_POLYGONS — Outputs a layer containing all the polygons of an equivalence loop. An equivalence loop violation is defined as two polygons having a forced equivalence path and a separator between them. This is a type 1 layer.

EQUIV_LOOP_EQUIV — Outputs a layer containing only those polygons of an equivalence loop that have a forced equivalence. An equivalence loop violation is defined as two polygons having a forced equivalence path and a separator between them. This is a type 1 layer.

EQUIV_LOOP_SEP CLUSTER — Outputs a layer containing all separators of an equivalence loop. An equivalence loop violation is defined as two polygons having a forced equivalence path and a separator between them. This is a type 3 layer.

EQUIV_LOOP_VIOLATION_SEP CLUSTER — Outputs a layer containing the violation separators of an equivalence loop. An equivalence loop violation is defined as two polygons having a forced equivalence path and a separator between them. This is a type 3 layer.

OPPOSITE_ANCHOR_EQUIV — Outputs only polygons reduced to equivalence within an opposite-anchor path. This is a type 1 layer.

OPPOSITE_ANCHOR_PATH — Outputs merged polygons and separators in an opposite-anchor path. This is a type 1 layer.

OPPOSITE_ANCHOR_POLYGONS — Outputs polygons within an opposite-anchor path. This is a type 1 layer.

OPPOSITE_ANCHOR_SEP CLUSTER — Outputs separators within an opposite-anchor path. This is a type 3 layer.

REMAINING_VIOLATIONS — Outputs a layer containing merged polygons and separators belonging to components from the CONFLICT layer which have no violations associated with them. This is a type 1 layer.

REMAINING_VIOLATIONS_POLYGONS — Outputs a layer containing polygons belonging to components from the CONFLICT layer which have no violations associated with them. This is a type 1 layer.

REMAINING_VIOLATIONS_SEP CLUSTER — Outputs a layer containing separators belonging to components from the CONFLICT layer which have no violations associated with them. This is a type 3 layer.

SAME_ANCHOR_EQUIV — Outputs only polygons reduced to equivalence within a same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This is a type 1 layer.

SAME_ANCHOR_PATH — Outputs merged polygons, anchors and violation separators of a same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This is a type 1 layer.

SAME_ANCHOR_POLYGONS — Outputs all polygons of a same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This is a type 1 layer.

SAME_ANCHOR_SEP CLUSTER — Outputs all separators of the same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This is a type 3 layer.

SAME_ANCHOR_VIOLATION_SEP CLUSTER — Outputs violating separators of the same-anchor path. A same-anchor violation is defined as two polygons anchored to the same mask and separated by equivalences and one additional separator. This is a type 3 layer.

SELF_CONFLICT_SEP CLUSTER — Outputs a layer containing the edges of separators marking a violation from one side of a polygon to another side of the same polygon. This is a type 3 layer.

SELF_CONTAINED_POLYGONS — Outputs a layer containing polygons belonging to components that are not colorable. Displays separators and shapes where four shapes each have a separator touching the other three shapes. Identical to a W4 or K4 violation. This is a type 1 layer.

SELF_CONTAINED_SEP CLUSTER — Outputs a layer containing separators belonging to components that are not colorable. Displays separators and shapes where four shapes each have a separator touching the other three shapes. Identical to a W4 or K4 violation. This is a type 3 layer.

SELF_CONTAINED_VIOLATION — Outputs a layer containing merged polygons and separators belonging to components that are not colorable. Displays separators and shapes where four shapes each have a separator touching the other three shapes. Identical to a W4 or K4 violation. This is a type 1 layer.

VIOLATION_SEPARATORS CLUSTER — Outputs all separators between two polygons colored the same. This is a type 3 layer.

Graph Reduction

For graph reduction, the output type may be this keyword, specified on its own RET TP command line:

REDUCED_SEPARATORS_LDR CLUSTER — Reports the edges remaining in the graph after Low Degree Node Removal is performed.

Description

The RET TP command is used to invoke Calibre triple-patterning. It splits an input layer into three masks based on optional anchoring and system-determined coloring. If the input layer cannot be split into three masks, RET TP outputs conflict errors on shapes or edges.

When multiple RET TP operations are present and differ only with respect to their MAP type, Calibre performs the RET operation once, generating multiple output layers simultaneously. For important exceptions regarding layer derivation support see “[Guidelines for Deriving Good Separator and Action Layers](#)” on page 346.

Note

 Siemens EDA recommends migrating all RET TP functions to DFM MP. RET TP is deprecated as of Calibre release 2016.1 and will be obsoleted from a future Calibre release.

Examples

The following example shows a LITHO FILE followed by RET TP commands to invoke Calibre triple-patterning and output three decomposed masks and any associated error layers. The masks are output whether conflicts exist or not. If conflicts are found, the error layers capture these conflicts on error shapes (either edges or polygons) at the lowest level of hierarchy possible, and the layer is output.

```
LITHO FILE ca_tp_file [
    target CA;
    separator SEP.S2S;
    separator SEP.T2T;
    anchor1 CA_anchor1;
    anchor2 CA_anchor2;
    anchor3 CA_anchor3;
]

CA_Mask1_Out = RET TP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 FILE ca_tp_file MAP MASK1
CA_Mask2_Out = RET TP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 FILE ca_tp_file MAP MASK2
CA_Mask3_Out = RET TP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 FILE ca_tp_file MAP MASK3

CA_Conflict = RET TP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 FILE ca_tp_file
    MAP CONFLICT

CA_Equiv_Loop = RET TP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 FILE ca_tp_file
    MAP EQUIV_LOOP

CA_Self_Contained_Violation = RET TP CA SEP.S2S SEP.T2T
    CA_anchor1 CA_anchor2 CA_anchor3 FILE ca_tp_file
    MAP SELF_CONTAINED_VIOLATION
```

LITHO FILE Statement and Commands

The FILE argument of RET-based commands call their respective LITHO FILE statement. The LITHO FILE statement contains application-specific commands and settings called by RET NMDPC, RET TP, and RET QP.

LITHO FILE commands may be specified in a separate file or specified within the same SVRF file within a LITHO FILE statement. Commands must be specified one per line.

Table 3-6. LITHO FILE Statement and RET-Based Commands

Command	Description
LITHO FILE	Specifies pattern decomposition commands. Used only with RET NMDPC, RET TP, and RET QP.
action	Declares layers for separators and stitches, and invokes coloring. Used for RET NMDPC only.
anchor	Forces decomposed shapes to be saved to a specific mask. Called by RET NMDPC, RET TP, and RET QP.
collector	Disables mask balancing. Used for RET NMDPC only.
layer	Allows layers to be available for the setlayer commands. Used for RET NMDPC only.
options	Passes stitching keywords with free-format to setlayer. Used for RET NMDPC and RET DPSTITCH only.
resolve	Causes the generated mask layers to be output. Used for RET NMDPC only.
seed	Disables alternating coloring for non-separated target shapes. Used for RET NMDPC only.
setlayer dpstitch	Generates layers for dpStitchCandidates. Used for RET DPSTITCH only.
target	Declares the input target layer. Used for RET TP and RET QP only.
tilemicrons	Specifies the size of a processing tile. Used for RET NMDPC and RET DPSTITCH only.

LITHO FILE

LITHO FILE Statement and Commands

Specifies pattern decomposition commands. Used only with RET NMDPC, RET TP, and RET QP.

Usage

```
LITHO FILE name '['  
    statement_keywords  
  ']'
```

Arguments

- *name*

A required name allowing commands to be referenced by [RET NMDPC](#), [RET DPSTITCH](#), [RET TP](#), and [RET QP](#) commands. No LITHO FILE statements may share the same name.

- '[' *statement_keywords* ']'

A required group of keywords placed between brackets ([]). Any term on the same line as a bracket is ignored.

Each keyword in the LITHO FILE must be on a line of its own. You span keywords over multiple lines by ending the line with a backslash (\). The backslash character must not be followed by any other characters, including whitespace.

Description

A required statement called by the FILE argument of RET NMDPC, RET DPSTITCH, RET TP, and RET QP commands. The LITHO FILE can be an independent file or specified in a separate LITHO FILE statement. The LITHO FILE statement is processed separately from SVRF commands and can be called by multiple RET commands. Different Calibre RET commands use different keywords in the LITHO FILE statements and therefore are not always shared between commands.

For a list of LITHO FILE commands, see “[LITHO FILE Statement and Commands](#)” on page 156.

You can use variables for any names or numbers in the LITHO FILE statement as described in “[Using SVRF Variables within a LITHO FILE Statement](#)” on page 323.

Examples

Example 1 — RET NMDPC

The following example demonstrates the use of the RET NMDPC command. Other commands that might appear in the RET NMDPC LITHO FILE include the control options: [collector](#), [resolve](#), [seed](#), and [setlayer dpstitch options](#) Commands.

Figure 3-30. RET NMDPC LITHO FILE and Command

```
// LITHO FILE for action and anchor layers
LITHO FILE nmdpc [
    action layer1 0 1;
    action layer2 has nicety of 5 1;
    action stitch_candidate :stitch 5 1;

    # anchor layers are optional
    anchor layer3 has nicety of 3 is for mask0;

    # control options
    ...
]
mask0 = RET NMDPC layer1 M1 opt_act_layer FILE nmdpc MAP mask0
mask1 = RET NMDPC layer2 M1 opt_act_layer FILE nmdpc MAP mask1
```

Example 2 — RET DPSTITCH

The following example demonstrates the use of RET DPSTITCH for stitch candidate generation.

Figure 3-31. RET DPSTITCH LITHO FILE and Command

```
// LITHO FILE for dpStitchCandidates
LITHO FILE dpSC_setup [

    layer M1 visible clear
    layer ...
    tilemicrons 40 adjust

    options opts {
        version 1
        layer M1 split_action opc clear
        layer ...

        dpStitchCandidates ... -outLayer cutCandidateRegion ...
    }

    setlayer cutCanReg = dpstitch M1 MAP cutCandidateRegion OPTIONS opts
]

stitchCan = RET DPSTITCH M1 MAP stitchCandidates FILE dpSC_setup
```

Example 3 — RET TP

This example shows an abbreviated LITHO FILE statement used for a RET TP command. Each layer specified in the LITHO FILE statement must appear in the accompanying RET TP or RET QP command.

```
LITHO FILE ca_tp_file [
    target CA;
    anchor1 CA_Mask1;
    separator SEP.S2S;
]
ca_mask1_out = RET TP CA SEP.S2S CA_Mask1 FILE ca_tp_file MAP MASK1
    ...
```

action

LITHO FILE Statement and Commands

Declares layers for separators and stitches, and invokes coloring. Used for RET NMDPC only.

Usage

```
action layer_name
  [:stitch]
  [has nicety of | nicety] {0 | priority}
  [and requires] {[opposite | same | 1 | 0]} [mask[s]] ;
```

Arguments

- *layer_name*

A required argument specifying a layer containing separators by name. Action layers contain either polygons or edge pairs.

- :stitch

An optional keyword that sets priority control over stitching within NMDPC. When :stitch mode is specified, shapes from the target layer are added to the output masks to inhibit or complete stitching as directed. Layers input to the action command in stitch mode must be polygons.

- [has nicety of | nicety] 0 | *priority*

An optional set of keywords and an argument specifying the priority of target shape separation. The value must be an integer. The default value is 0. The strings “has nicety of” or “nicety” are provided for readability only and are optional. Nicety is a legacy term for priority.

Use 0 or less to identify a critical shape-to-shape separation requirement. All shapes with separators using a priority of 0 or less are considered simultaneously for conflicts. Shapes with separators using a priority greater than 0 are considered noncritical.

Note



Only target shapes actually requiring separation should be given a priority of 0. If a large shape touches a separator with a priority of 0, it may cause many conflicts due to its proximity to more shapes.

- [and requires] {[opposite | same | 1 | 0]} [mask[s]]

An optional keyword set that inclines the coloring algorithm to assign target shapes to the same mask or different masks:

0 | same — Inclines the coloring algorithm to same mask assignment.

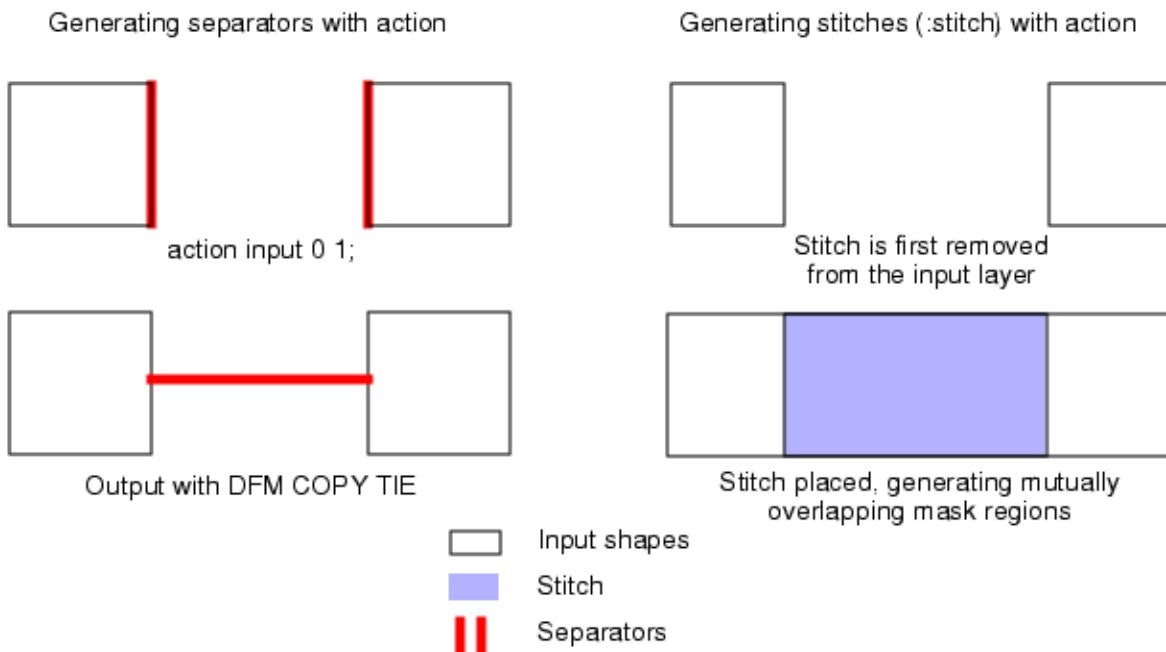
1 | opposite — Inclines the coloring algorithm to different mask assignment. This is the default value.

The extra terms are not required.

Description

The action command invokes coloring through separation checking and stitch implementation. Without the action command, no coloring (decomposition) takes place. For separators, the declared input layer may contain polygons or edge pairs. For stitches, the declared input layer must be polygonal. Resulting separator edge pairs can be coincident with, or overlap, the edges on the target layer. Resulting stitches are first removed from the original input layer and then placed where the stitch acts as an overlapping extension to both masks.

Figure 3-32. Generated Stitches and Separator Edge Pairs



Separators input to the action command are derived using operations in the SVRF rule file before calling [RET NMDPC](#). Stitches input to the action keyword can be drawn, derived with SVRF, or generated by dpStitchCandidates.

For important exceptions regarding layer derivation support see “[Guidelines for Deriving Good Separator and Action Layers](#)” on page 346 in the [Calibre Multi-Patterning Best Practices](#) chapter.

Examples

Example 1 — Prioritized Action Layers

The first rule in this example establishes some critical separators. Contacts touching the separators on this layer are assigned to the other mask. The second rule defines optional separators. Shapes touching the separators on this layer are assigned to the same mask. Because the first rule has a priority of 0, the first rule is applied first, followed by the second rule.

```
action must_separator 0 opposite;
action optional_separator 1 same;
```

Example 2 — Stitch Action Layer

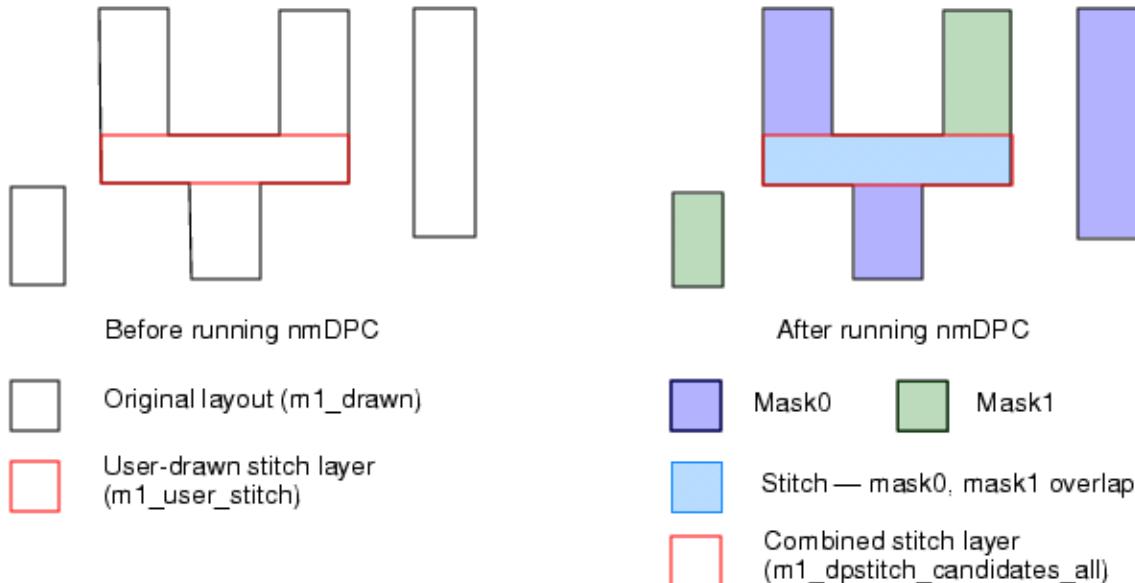
The following action layer specifies the stitch option. Priority and opposite mask settings can be used to insure user stitches are placed. In this example stitch candidates are first generated with RET DPSTITCH. The action keyword used with the stitch option then declares the input layers for stitches with specified priority and mask settings:

```
dpstitch_candidates = RET DPSTITCH target stitch_keepout
    anchor_0 anchor_1 user_sep FILE dpstitch_setup MAP stitchcandidates
...
action dpstitch_candidates :stitch 1 same;
action user_stitches :stitch 1 opposite;
```

Example 3 — User Stitch Use

This example declares the user stitch layer and imports the layer as an argument to the action keyword.

Figure 3-33. User-Defined Stitch Candidate Example



```
LAYER m1_user_stitch 1002 //Metall User Stitches

m1_dpstitch_candidates = RET DPSTITCH m1_target ...
    FILE dpstitch_setup_M1 MAP stitchcandidates

m1_dpstitch_candidates_all = m1_dpstitch_candidates OR m1_user_stitch

LITHO FILE m1dpcfile [
    action m1_dpstitch_candidates_all :stitch 0 1;
    ...
]
```

Example 4 — Equivalent Forms

The following action commands (using 0 and 1 for priority and mask assignment, respectively) are equivalent:

```
action metall has nicety of 0 and requires opposite masks;
action metall      nicety 0                      opposite masks;
action metall      nicety 0                      opposite mask;
action metall          0                      opposite;
action metall          0                      1;
action metall;           #(this default reduction only for 0, 1)

action user_stitch :stitch has nicety of 1 and requires same masks;
action user_stitch :stitch      nicety 1          same;
action user_stitch :stitch                  1          0;
```

anchor

LITHO FILE Statement and Commands

Forces decomposed shapes to be saved to a specific mask. Called by RET NMDPC, RET TP, and RET QP.

Usage

For RET NMDPC:

```
anchor layer_name [{has nicety of} | nicety] priority [[is for] mask] {0 | 1};
```

For RET TP:

```
anchor1 layer_name;  
anchor2 layer_name;  
anchor3 layer_name;
```

For RET QP:

```
anchor1 layer_name;  
anchor2 layer_name;  
anchor3 layer_name;  
anchor4 layer_name;
```

Arguments

- *layer_name*

A required argument specifying the layer the anchor is drawn on. The layer is specified by either name or the order it appears in the RET NMDPC operation. [RET NMDPC](#) anchor layers are typically passed in after the drawn layer as one of the optional action layers. [RET TP](#) and [RET QP](#) anchor layers are specified after the target layer as one of several optional layers. Anchor layers may be either polygon or edge layers, but not edge pairs.

- [{has nicety of} | nicety] *priority*

An optional set of keywords and a required argument used by RET NMDPC only, specifying the priority of an anchor layer. The priority value must be an integer. The strings “has nicety of” or “nicety” are provided for readability but are not necessary. Nicety is a legacy term for priority.

Use 0 or less to identify a critical anchor layer. (All critical layers are considered simultaneously for conflicts.) Anchor layers with a priority greater than 0 are noncritical.

Note

 Only a critical layer should be given a priority of 0. If a non-critical layer with many anchors is specified with a priority of 0, it is more likely to host increased conflicts and slow the debugging process.

- [[is for] mask]{0 | 1}

A required argument with optional keywords used by RET NMDPC only, that assigns anchor shapes to mask1 or mask0. Anchor layers must specify one mask or the other; you cannot mix mask assignments.

Description

An optional command used by RET NMDPC, RET TP, and RET QP to force assignment of shapes to specific masks which remain permanent. If anchor assignments cause coloring conflicts during mask decomposition, analysis output is generated. You may output these layers to your database to view the conflicts in Calibre WORKbench™. For MAP names representing these anchor-based conflict layers, see each specific RET command.

The shapes in an anchor layer may cover all or a portion of the target shapes to assign them to a particular mask. Some foundries refer to precoloring anchors as identical to the target shapes they anchor. Other foundries use seed anchors that are fully enclosed by or overlapping the target shape.

Only one anchor layer of each type is allowed within a single LITHO FILE statement. You can derive or save anchor layers to analyze them in a layout viewer.

Examples

Example 1 — Action and Anchor Layer Usage for RET NMDPC

In the following code, the LITHO FILE statement assigns a greater priority to the anchor layer than to the action layer. This code outputs the conflict errors for masks and anchors, and both masks (using resolve).

The LITHO FILE statement is called by all three RET NMDPC commands. Because all arguments other than the MAP types are identical, the operations are run concurrently. The MAP types indicate the internally generated layers which are assigned to the layer names on the left. This makes the internal mask and conflict error data available to the SVRF operations outside Calibre nmDPC.

```
LITHO FILE nmdpc.in[
    resolve;
    action actionlayer 1 opposite;
    anchor anchorlayer 0 mask0;
]

maskA = RET NMDPC actionlayer drawn anchorlayer FILE nmdpc.in MAP mask0
maskB = RET NMDPC actionlayer drawn anchorlayer FILE nmdpc.in MAP mask1
conflict = RET NMDPC actionlayer drawn anchorlayer FILE nmdpc.in
           MAP anchor_conflict
```

Example 2 — Equivalent Forms of Usage for RET NMDPC

The following anchor layer specifications are equivalent:

```
anchor anchor_layer1 has nicety of 1 is for mask0;
anchor anchor_layer1 has nicety of 1           mask0;
anchor anchor_layer1      nicety    1           0;
anchor anchor_layer1                  1           0;
```

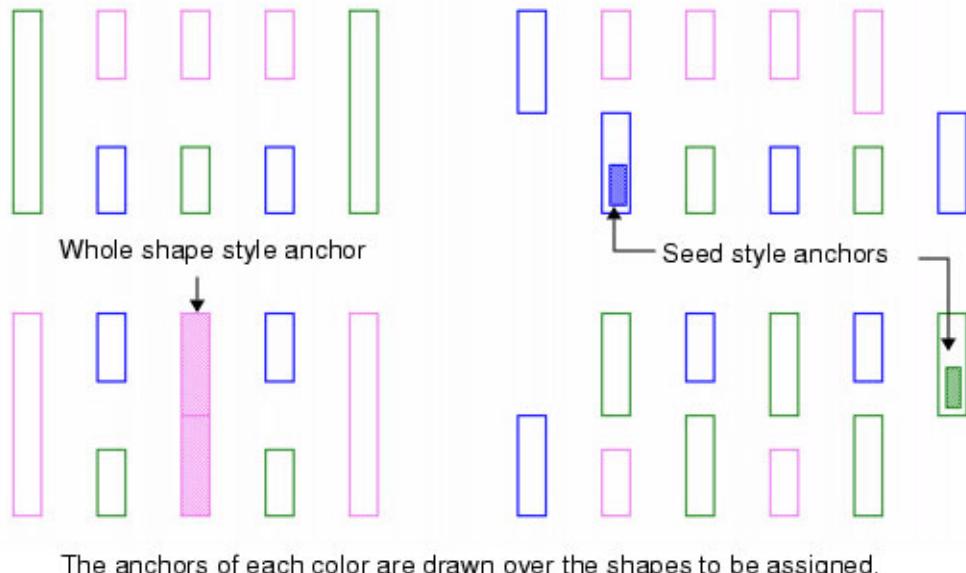
Example 3 — Anchor Layer Usage for RET TP

The LITHO FILE statement is called by all three **RET TP** commands specified in the example. Because all arguments other than the MAP types are identical, the operations are run concurrently. The MAP types output mask and error shapes to the specified output layers. These output layers are then available to the SVRF operations outside Calibre Multi-Patterning.

```
LITHO FILE ca_tp [
    target CA;
    anchor1 CA_Anchor1;
    anchor2 CA_Anchor2;
    anchor3 CA_Anchor3;
    separator SEP.S2S;
    separator SEP.T2T;
]
CA_Mask1_Out = RET TP CA SEP.T2T SEP.S2S CA_Anchor1 CA_Anchor2 CA_Anchor3
    FILE ca_tp MAP MASK1
CA_Mask2_Out = RET TP CA SEP.T2T SEP.S2S CA_Anchor1 CA_Anchor2 CA_Anchor3
    FILE ca_tp MAP MASK2
CA_Mask3_Out = RET TP CA SEP.T2T SEP.S2S CA_Anchor1 CA_Anchor2 CA_Anchor3
    FILE ca_tp MAP MASK3
```

Figure 3-34 depicts two application types of anchors determining three mask assignments.

Figure 3-34. Anchor Applications for Triple-Patterning



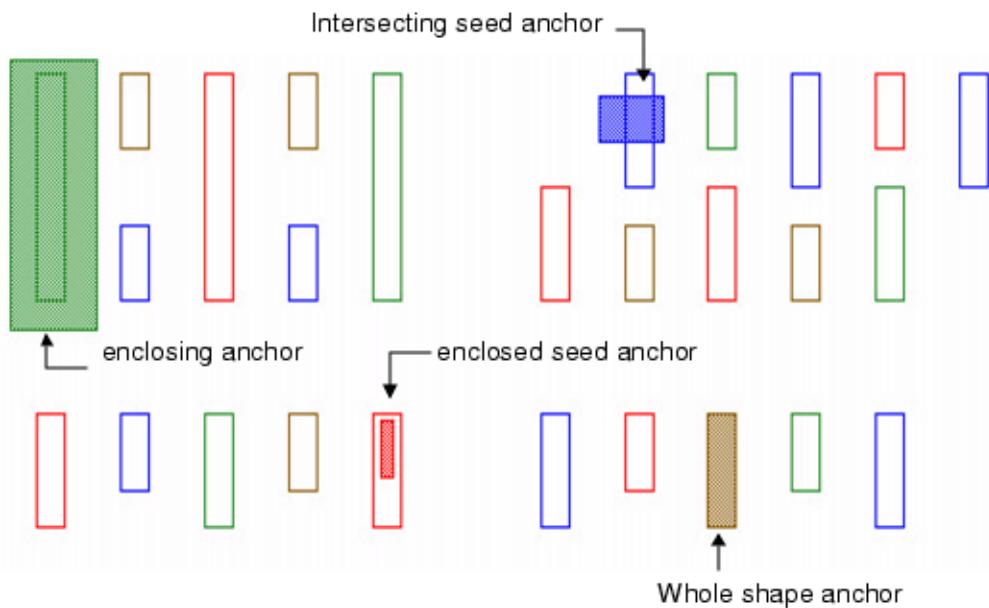
Example 4 — Anchor Layer Usage for RET QP

The LITHO FILE statement is called by all four RET QP commands specified in the example. Because all arguments other than the MAP types are identical, the operations are run concurrently. The MAP types output mask and error shapes to the specified output layers. These output layers are then available to the SVRF operations outside Calibre Multi-Patterning.

```
LITHO FILE ca_qp [
    target CA;
    anchor1 CA_Anchor1;
    anchor2 CA_Anchor2;
    anchor3 CA_Anchor3;
    anchor4 CA_Anchor4;
    separator SEP.S2S;
    separator SEP.T2T;
]
CA_Mask1_Out = RET QP CA SEP.T2T SEP.S2S
    CA_Anchor1 CA_Anchor2 CA_Anchor3 CA_Anchor4
    FILE ca_qp MAP MASK1
CA_Mask2_Out = RET QP CA SEP.T2T SEP.S2S
    CA_Anchor1 CA_Anchor2 CA_Anchor3 CA_Anchor4
    FILE ca_qp MAP MASK2
CA_Mask3_Out = RET QP CA SEP.T2T SEP.S2S
    CA_Anchor1 CA_Anchor2 CA_Anchor3 CA_Anchor4
    FILE ca_qp MAP MASK3
CA_Mask4_Out = RET QP CA SEP.T2T SEP.S2S
    CA_Anchor1 CA_Anchor2 CA_Anchor3 CA_Anchor4
    FILE ca_qp MAP MASK4
```

Figure 3-35 depicts four different types of anchors determining four mask assignments.

Figure 3-35. Anchor Applications for Quadruple-Patterning



The anchors of each color are drawn over the shapes to be assigned.

collector

LITHO FILE Statement and Commands

Disables mask balancing. Used for RET NMDPC only.

Usage

```
collector {mask0 | 0 | mask1 | 1} ;
```

Arguments

- mask0 | 0 | mask1 | 1

One of four arguments must be specified indicating which mask to use as a collector mask.

Description

An optional command that disables mask balancing and places the majority of decomposed mask shapes on the specified mask.

layer

LITHO FILE Statement and Commands

Allows layers to be available for the setlayer commands. Used for RET NMDPC only.

Usage

layer *layer_name* *layer_type* *transmission*

Arguments

- ***layer_name***

A required argument specifying the name of the layer. The tool uses this name in the rest of the setup file to refer to this layer.

Layer names must be alphanumeric strings less than 32 characters in length.

- ***layer_type***

A required parameter that specifies how the layer is used by the tool. Valid types are listed here:

Table 3-7. layer Command Layer Types

Layer Type	Function
hidden (optional)	By default, any layer not specified in the setup file is marked as a hidden layer.
visible (required)	Visible layers are used in the setup file.

At least one layer of type visible must be specified.

- ***transmission***

A required argument that defines the mask layer's optical transmission type. This can be any value for hidden layers. Valid choices for visible layers are **clear** or **dark**.

Description

A required command that names and describes optical properties of an input layer. Every layer to be used in a [setlayer dpstitch](#) command must be declared in the LITHO FILE using this command. Transmission only affects layers with a layer type of visible.

Examples

This example declare a clear layer named M1.

```
layer M1 visible clear
```

Related Topics

[RET DPSTITCH](#)

options

LITHO FILE Statement and Commands

Passes stitching keywords with free-format to setlayer. Used for RET NMDPC and RET DPSTITCH only.

Usage

```
options name '{  
    options_list  
}'
```

Arguments

- *name*

A required argument naming the options block so it may be referenced by [setlayer dpstitch](#).

- *options_list*

A required list of keywords surrounded by required braces ({ }). The keywords must appear one per line. A pound sign (#) comments the remainder of the line.

The layers specified in the options block and those listed in the setlayer dpstitch command are related explicitly by order; their names can be different, but their listed order of sequence is critical.

Refer to the “[setlayer dpstitch options Commands](#)” section for a complete list of supported commands.

Description

A required list of keywords called by the setlayer dpstitch and [RET DPSTITCH](#) commands.

Examples

With regard to layer order in this example, layer target appears first in the options block and so must also appear first in the setlayer dpstitch command; layer keep_out appears second and therefore must also appear second in the setlayer dpstitch command.

```
LITHO FILE dpstitch.in [
    layer target visible clear
    layer keep_out visible clear
    options opt_block {
        version 1
        layer target opc clear
        layer keep_out visible clear
        dpStitchSize \
            (0.060 <= length <= 0.120) for width <= 0.060: \
            (0.120 <= length <= 0.240) for (0.060 < width <= 0.120): \
            (0.240 <= length <= 0.500) otherwise :
        dpFaceClass name(tip): \
            true for (0.030 <= length <= 0.060) and minAdjacent >= 0.030: \
            false otherwise :
        dpSpacing (tip, side) metric(opposite, extended 0.010): \
            0.060 for minWidth < 0.030 and maxLength >= 0.060: \
            0.080 otherwise:
        minIntersectionArmLength 0.040 otherwise :
        intersectionStitch true for numArms >= 2 and minWidth >= 0.032 \
            and maxWidth <= 0.500: \
            false otherwise :
        minConcaveCornerOverlap 0.030 for width < 0.060: \
            0.020 for width < 0.080: \
            0 otherwise :
        dpStitchCandidates criticalDistance 0.60 minArea 0.04 \
            stitchkeepout keep_out \
            -outLayer stitchcandidates -outLayer separators
    } # end of options block
    setlayer stitchcandidates = dpstitch target keep_out \
        MAP stitchcandidates OPTIONS opt_block
    setlayer separators = dpstitch target keep_out \
        MAP separators OPTIONS opt_block
] //End of LITHO FILE statement

candidate = RET DPSTITCH target keep_out FILE dpstitch.in
MAP stitchcandidates
separator = RET DPSTITCH target keep_out FILE dpstitch.in
MAP separators
```

resolve

LITHO FILE Statement and Commands

Causes the generated mask layers to be output. Used for RET NMDPC only.

Usage

```
resolve [critical conflict] ;
```

Arguments

- critical conflict

An optional string that can be specified for increased readability. It does not change the behavior of the command.

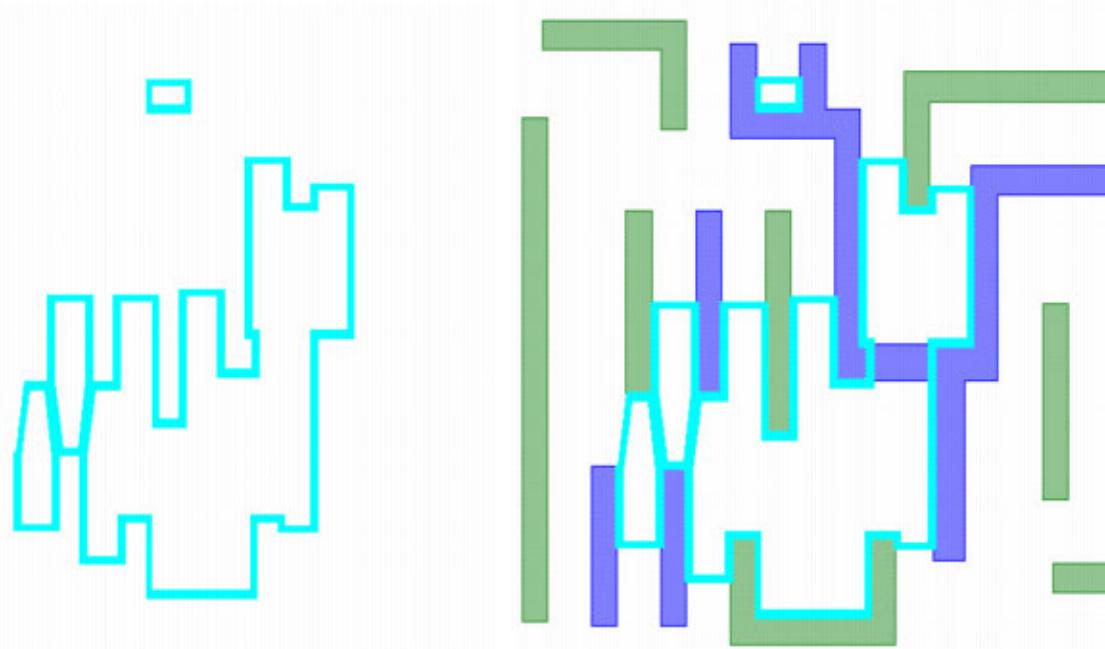
Description

An optional command used to enable automatic resolution which, in the presence of conflicts, outputs the two mask layers and the conflict layer. By default, only the conflict layer is output if conflicts are found.

Examples

Figure 3-36 demonstrates the resolve keyword behavior.

Figure 3-36. Resolve Behavior



resolve not specified, outputting ring shapes only

resolve specified, outputting ring shapes and generated masks

seed

LITHO FILE Statement and Commands

Disables alternating coloring for non-separated target shapes. Used for RET NMDPC only.

Usage

`seed value ;`

Arguments

- *value*

Any integer may be specified.

Description

An optional command that disables the standard alternating coloring algorithm and enables random mask coloring only for those target shapes not touching any separator. This command randomly colors non-separated shapes as mask0 or mask1. There is no default for this command.

setlayer dpstitch

LITHO FILE Statement and Commands

Generates layers for dpStitchCandidates. Used for RET DPSTITCH only.

Usage

setlayer *output_layer* = dpstitch *target* [*layer ...*] MAP *type* OPTIONS *block_name*

Arguments

- ***output_layer***

A required argument naming the derived layer (*output_layer*) that receives the new dpStitchCandidates data.

- **dpstitch**

A required keyword that invokes this command instead of other setlayer commands.

- ***target* [*layer ...*]**

At least one layer name is required that specifies the target (drawn) layer with shapes to be decomposed. Any number of subsequent layers may be specified for other mask shapes such as anchors, separators and keepout areas.

The layers in the options block and those listed in the setlayer dpstitch command are related explicitly by order; their names can be different, but their listed order of sequence is critical.

- **MAP *type***

One required argument that specifies the type of stitch-related output. The output type can be one of:

separators — Returns a separator shape that touches two target shapes indicating they must be colored differently (saved to different masks). The separator shapes are a useful debugging tool to ensure the dpSpacing commands correctly evaluate target shape distances of various face classes for decomposition.

stitchCandidates — Returns shapes showing potential stitches. These stitches may or may not be used, determined by decomposition needs.

- **OPTIONS *block_name***

A required argument indicating the name of the [options](#) block to use.

Description

A required command that must appear at least once per [RET DPSTITCH](#) setup file. The setlayer dpstitch command is used to output dpStitchCandidates layers. To output [dpStitchCandidates](#) layers, you must specify at least one target layer after the dpstitch keyword.

The setlayer dpstitch construct restricts the file contents to only what is required for dpStitchCandidates. Using this method, you cannot load optical models and you do not have to set image properties. You can use a [LITHO DENSEOPC](#) operation if you also set the

appropriate values. For more information on LITHO DENSEOPC, refer to the [*Calibre nmOPC User's and Reference Manual*](#).

Calibre® dpStitch identifies areas in the input where shapes can be split. The split areas must be large enough to receive a stitch (overlapped region) without risk of pinching and far enough from neighboring shapes to not introduce additional coloring conflicts.

Calibre dpStitch examines the layout using rules you define to identify stitch candidates. The rules can be simple, for example, defining only minimum overlap area and distance from features, or more complex, for example, defining different minimum spacings from each type of intersection.

Examples

The following example uses the setlayer dpstitch command to output the specified dpStitchCandidates layers. In this example, a required mask layer (m1, the target layer) and optional keepout layer (m1_ko) are specified after the dpstitch command.

With regard to layer order in this example, layer m1 appears first in the options block and so must also appear first in the setlayer dpstitch command; layer m1_ko appears second and therefore must also appear second in the setlayer dpstitch command.

```
options opt_block {  
    version 1  
    layer m1 opc clear  
    layer m1_ko visible clear  
    ...  
  
    setlayer stitchCand = dpstitch m1 m1_ko MAP stitchCandidates OPTIONS OPTS  
    setlayer separators = dpstitch m1 m1_ko MAP separators      OPTIONS OPTS
```

target

LITHO FILE Statement and Commands

Declares the input target layer. Used for RET TP and RET QP only.

Usage

```
target layer_name ;
```

Arguments

- *layer_name*

The name of the layer on which the target shapes are drawn.

Description

A required keyword used to name the input target layer. The target layer must consist of polygons. The target layer must be specified as the first argument to the [RET TP](#) and [RET QP](#) commands.

The shapes in target layer constitute the entirety of the design to be decomposed by Calibre triple- and quadruple-patterning. The decomposed target layer is output to three or four masks; the MAP types using the MASK_n keywords output mask shapes to the specified RET TP or RET QP output layers. These output layers can be further manipulated by SVRF operations.

Examples

The LITHO FILE statement is called by three or four RET TP or RET QP commands, respectively, outputting the necessary masks containing the decomposed portions of the target layer. Because the layer arguments in this example are identical, the operations run concurrently. This example demonstrates the target keyword with a triple-patterning case.

```
LITHO FILE ca_triple [
    target CA;
    anchor1 CA_Anchor1;
    anchor2 CA_Anchor2;
    anchor3 CA_Anchor3;
    separator SEP.S2S;
    separator SEP.T2T;
]
CA_Mask1_Out = RET TP CA SEP.T2T SEP.S2S CA_Anchor1 CA_Anchor2 CA_Anchor3
    FILE ca_triple MAP MASK1
CA_Mask2_Out = RET TP CA SEP.T2T SEP.S2S CA_Anchor1 CA_Anchor2 CA_Anchor3
    FILE ca_triple MAP MASK2
CA_Mask3_Out = RET TP CA SEP.T2T SEP.S2S CA_Anchor1 CA_Anchor2 CA_Anchor3
    FILE ca_triple MAP MASK3
```

tilemicrons

LITHO FILE Statement and Commands

Specifies the size of a processing tile. Used for RET NMDPC and RET DPSTITCH only.

Usage

`tilemicrons tile_size [adjust | exact]`

Arguments

- `tile_size`

A required argument that specifies a preferred tile size in microns. When this command is not in the setup file, tiles are 100 microns per side.

When processing a layer, Calibre Litho and RET operations work on only a portion at a time in order to have all geometries in memory simultaneously. The portion is a square, referred to as a tile. For more on tile management, see the [Calibre OPCpro User's and Reference Manual](#).

- `adjust | exact`

An optional argument controlling whether the `tile_size` is used as specified. Only one of `adjust` or `exact` may be specified. The default behavior if neither is specified is `adjust`.

- When `adjust` is specified, Calibre may adjust the tile size slightly to optimize performance.
- When `exact` is specified, Calibre uses only the specified tile size. This is not generally recommended.

Description

An optional command, but strongly recommended for process nodes that benefit from multi-patterning. The default behavior when the command is not used is tiles of approximately 100 microns, roughly twice the size recommended for the 45-nm node.

The `tilemicrons` command must appear in a LITHO FILE statement for [RET DPSTITCH](#).

Recommended Tile Sizes per Technology Node

The selection of tile size is critical for optimum processing. If the tile size is too large or too small, performance is slowed by memory swapping. The optimum tile size settings based on benchmarks correlate to the technology node and balance the size of data per tile. Use [Table 3-8](#) to set `tilemicrons` to a more optimal value. For multi-patterning, use the smaller values listed in the table.

If you use CPUs that support simultaneous multi-threading (SMT), double the memory per core to support hyperscaling. For example, use 8 GB per core at 22 nm. For an 8-core remote, this would be 64 GB. For a 12-core remote, this is 96 GB.

Table 3-8. Hardware Requirements and tilemicrons Settings

Technology Node	Hardware Requirement	tilemicrons Setting
>=45 nm	2 GB/core	45 to 55
32 nm	3 GB/core	35 to 40 (30 for M1)
22 nm	4 GB/core	25 to 30 (20 for M1)

Examples

The following code shows the initial portions of a LITHO FILE statement for RET DPSTITCH.

```
LITHO FILE dpstitch.in [
    layer fabriclayer visible clear
    layer otherlayer visible clear
    tilemicrons 20
    options dp_opt {
        ...
    }
    ...
]
```

setlayer dpstitch options Commands

The setlayer dpstitch options block contains keywords that classify edges and specify dimensional constraints. The options block is referenced by the OPTION argument of a setlayer dpstitch command.

Commands in this section are only used in an options block called by setlayer dpstitch within a LITHO FILE statement in turn called by RET DPSTITCH. Any of the commands in the table may appear in the options block, one per line. Many of the command syntaxes use the backslash (\) and colon (:) characters. The backslash allows the command to wrap across multiple lines. The colon separates different conditions. The colon is always required where shown. The backslash is only required if you format the command to wrap. When used, it must always be the last character on the line, including spaces.

LITHO FILE statements executed by Calibre releases prior to 2012.2 may contain an options block with the original name of denseopc_options instead of the later name, options. From release 2012.2 forward, either name may be used and they function identically.

Table 3-9. setlayer dpstitch options Commands

Command	Description
dpFaceClass	Classifies edges into groups.
dpFaceSet	Accumulates various user-defined face classes into a single set.
dpMinStitchWidth	Specifies the minimum width of stitch candidates.
dpSpacing	Specifies spacing requirements between two face classes.
dpStitchCandidates	Generates shapes representing regions where overlapping decomposed layers may be joined.
dpStitchSize	Specifies stitch length requirements for various line widths.
minConvexCornerSpacing	Defines the minimum distance from every outside corner to a stitch candidate rectangle.
intersectionStitch	Specifies the required conditions that make an intersection suitable for stitches.
minConcaveCornerOverlap	Defines the minimum overlap required for concave corners to receive stitches.
minIntersectionArmLength	Specifies the minimum length of an intersection extension that is considered for stitching.
dpStitchToViaSpacing	Specifies the spacing from vias to stitches.

Table 3-9. setlayer dpstitch options Commands (cont.)

Command	Description
minConcaveCornerSpacing	Defines the minimum distance from every inside corner to a stitch candidate rectangle.
version	Declares the version of the LITHO FILE statement.

dpFaceClass

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Classifies edges into groups.

Usage

```
dpFaceClass name('faceclass_name'):
    [true for constraint [and constraint...] ...:]
    {false | true} otherwise:
```

Arguments

- **name('name'):**

A required argument specifying the name of the user-defined face class. The name must be enclosed in parentheses and, after all options, be terminated with a colon (:). All face classes must have a unique name.

Names may only contain alphanumeric characters, hyphens (-), and underscores (_). No dpFaceClass name may contain the string dpstitch_, or be named ‘any’, ‘jog’, or ‘side’; these are reserved names.

- {true for constraint [and constraint...] } ...:

An optional argument specifying one or more constraints which must be true for an edge to be considered a member of the named face class. Constraints are comprised with the following constraint keywords, an operator and a value, and terminated by a colon (:). For more information regarding the constraint keywords see [Figure 3-37](#) in the Description section.

length — The length of the edge between, or adjacent to, corners or jogs. Values specified must be positive.

width — The width of the mask shape classifying a perpendicular edge to a face class. Values specified must be positive.

maxAdjacent — The larger of the two adjacent side measurements. Values specified may be positive or negative.

maxAdjacentWidth — The maximum width of a polygon segment containing what has been classified as an adjacent edge. Values specified may be positive or negative.

minAdjacent — The smaller of the two adjacent side measurements. Values specified may be positive or negative.

minAdjacentWidth — The minimum width of a polygon segment containing what has been classified as an adjacent edge. Values specified may be positive or negative.

- {false | true} otherwise:

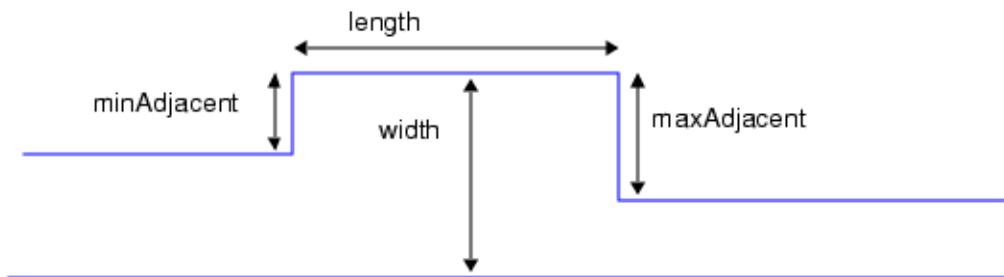
A required argument ensuring that edges not meeting any condition specified within the dpFaceClass command are classified. The otherwise argument must be terminated by a colon (:).

Description

Classifies edges of shapes into groups referred to as face classes. Up to 32 dpFaceClass definitions may be classified in one NMDPC run. The face class method classifies edges of polygons for subsequent spacing measurements by the dpSpacing command. Face classes can overlap and a single edge can belong to multiple face classes.

Face classes are defined in a RET DPSTITCH option block using the dpFaceClass command. Different classes can be created based on the characteristics seen in the following figure.

Figure 3-37. Face Class Constraint Keywords



There are two built-in face classes that can be referenced by the dpFaceSet and dpSpacing commands:

- **jog** — Any edge that has one convex corner, one concave corner, and is shorter than the spanJogLength setting in dpStitchCandidates, or is otherwise shorter than $0.293 * criticalDistance$ (defined as a parameter of dpStitchCandidates).
- **side** — Any other edge not classified by dpFaceClass.

Note

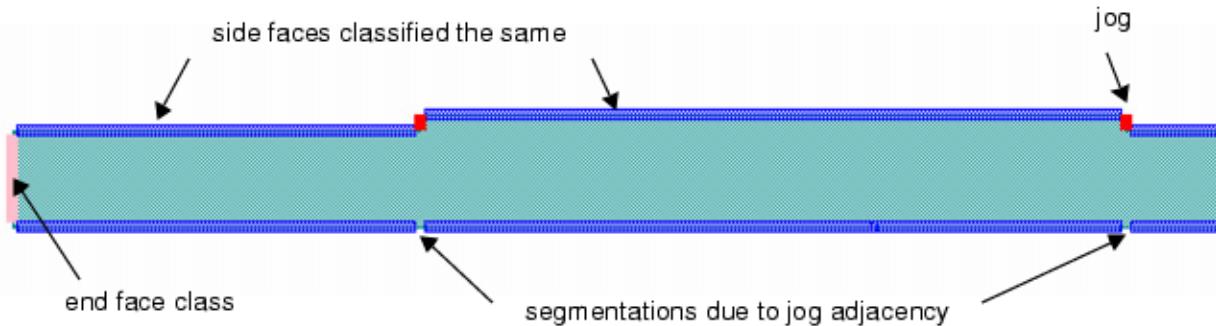
 Although the reserved face class, side, is provided as a convenience for rule development and testing, Siemens EDA does not recommend its application in production rule files.

Polygon faces can include jogs when the jog length is less than minAdjacent. If two or more jogs are connected in sequence, the widest side of the shape is used for the measurement. You can control what is considered a jog by setting the spanJogLength parameter in the dpStitchCandidates command. This also affects the built-in jog face class. To see the edges that are part of a particular face class, use:

```
dpStitchCandidates ... -outLayer name
```

The following figure demonstrates an example of the edges selected for measurement where jogs exist:

Figure 3-38. Face Classes and Jogs



Examples

Example 1

This example shows two dpFaceClass specifications. The first creates a user-defined face class called wideMetal and the second generates a user-defined face class named lineend. Both of these dpFaceClass commands join two conditions with the ‘and’ keyword.

```
# Example using constant values in its length and width conditions
dpFaceClass name(wideMetal)
    true for length >= 0.75 and width >= 0.1:
    false otherwise:

# Example using an independent measurement in its width condition
dpFaceClass name(lineend)
    true for (0.03 <= length <= 0.7) and minAdjacent < 0.04:
    false otherwise:
```

Example 2

This example code and illustration in [Figure 3-39](#) demonstrates the use of the maxAdjacentWidth argument. The dpFaceClass command identifies edge segments that act as jogs in polygons considered to be wide. In this example, the edge in consideration:

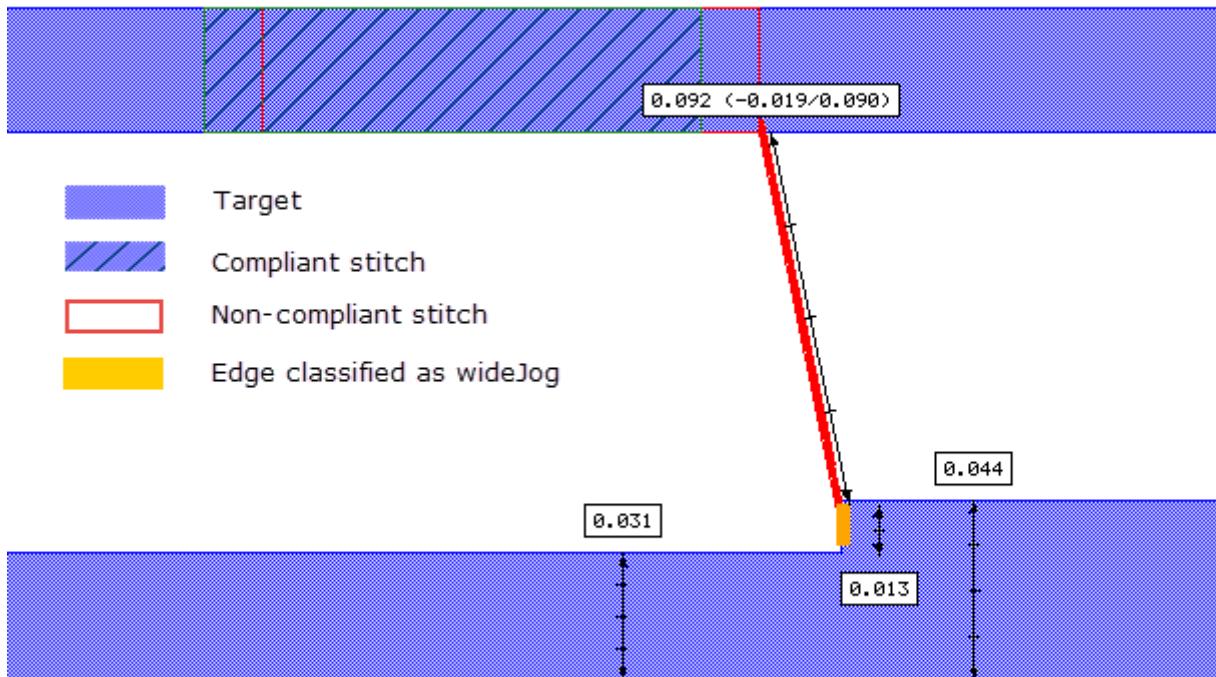
- Meets the length criteria:
 $0.010 \leq \text{length} < 0.030$
- Has two connected adjacent edges, one convex and one concave, that satisfy the minAdjacent and maxAdjacent arguments:
 $\text{minAdjacent} \leq -0.010 \text{ and } \text{maxAdjacent} \geq 0.010$
- And validates the wide jog is adjacent to at least one edge of a polygon within the constraint as specified:
 $\text{maxAdjacentWidth} \geq 0.035$

The dpSpacing command then measures the distance from these wide jogs to any line-end considered to be a tip.

```
dpFaceClass name(wideJog)
  true for (0.010 <= length < 0.030) and minAdjacent <= -0.010 and
  maxAdjacent >= 0.010 and maxAdjacentWidth >= 0.035:
    false otherwise:

dpSpacing (tip, wideJog) : 0.095 otherwise:
```

Figure 3-39. Use of maxAdjacentWidth With dpFaceClass



Related Topics

[Key Concepts](#)

[dpFaceSet](#)

dpFaceSet

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Accumulates various user-defined face classes into a single set.

Usage

```
dpFaceSet name('faceset_name'):  
    faceclass [,faceclassN ...]:
```

Arguments

- **name('faceset_name'):**
A required argument specifying the name of the accumulated user-defined set of face classes. The face class set name must be enclosed in parentheses. The dpFaceSet name must be unique. The name argument must be terminated by a colon (:).
- **faceclass [, faceclassN...]:**
A required argument specifying a previously declared face class name. One or more previously-declared face class names must be specified, and any number of face class names may be specified. The faceclass name(s) must be terminated by a colon (:).

Description

An optional command that accumulates any number of previously-defined face classes into a single face class set. One or more previously declared face class names must be specified. The dpFaceSet command must follow dpFaceClass commands declaring face classes. All face class sets must have a unique name.

All dpFaceClass commands must precede dpFaceSet commands specifying their respective face class names.

Names may only contain alphanumeric characters, hyphens (-), and underscores (_).

Note

 The reserved class named ‘side’ is a repository for non-classified polygon faces. After use by a dpFaceSet list, the side class is emptied of all faces and has no effect for subsequent use by dpSpacing commands.

Examples

This example shows two dpFaceClass commands generating two face classes named tip and wideMetal. Both of these face classes are subsequently combined into a single face class set with the dpFaceSet command.

```
dpFaceClass name(tip):
    true for length <= 0.03 and minAdjacent >= 0.03 and width <= 0.032:
    false otherwise:

dpFaceClass name(wideMetal):
    true for width >= 0.150:
    false otherwise:

dpFaceSet name(all): tip, wideMetal:
```

Related Topics

[Key Concepts](#)

[dpFaceClass](#)

dpMinStitchWidth

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Specifies the minimum width of stitch candidates.

Usage

```
dpMinStitchWidth {[stitch_width for target_width:] ... [stitch_width otherwise:]}
```

Arguments

- *stitch_width for target_width:*

A constraint specifying the minimum stitch width for a given target width. The *stitch_width* value may be a constant, variable, expression, and the *targetWidth* internal variable. The same specification requirements apply to the *target_width* value, except the *targetWidth* internal variable must be specified. Expressions with both upper and lower bounds must be enclosed in parentheses. One or both ‘*stitch_width for target_width*’ or ‘*stitch_width otherwise*’ constraints must be specified. The *target_width* constraint portions are highlighted in red:

```
0.081 for 0.080 < targetWidth:  
0.05 for (0.080 < targetWidth <= 0.120):  
0.5 * targetWidth for (0.080 < targetWidth <= 0.120):
```

- *stitch_width otherwise:*

A constraint specifying the default stitch width. The *stitch_width* value may include a constant, variable, expression, and the *targetWidth* reserved variable. One or both ‘*stitch_width for target_width*’ or ‘*stitch_width otherwise*’ constraints must be specified and only one ‘*stitch_width otherwise*’ constraint may be specified, and must appear last.

Examples include:

```
targetWidth otherwise:  
0.09 otherwise:
```

Description

An optional command that specifies the minimum stitch candidate width for its respective target shape width. The *dpMinStitchWidth* command provides precise control of stitch width relative to the width of target shapes to be stitched, especially in cases of partial stitch overlap of target shapes. The associated target shape width values are automatically stored in the *targetWidth* internal variable for use in expressions. Multiple expressions may be specified in a single *dpMinStitchWidth* keyword. The specified expressions are order dependent; the first qualifying expression that matches is used to establish the stitch width. No stitch of a width less than *criticalDistance* is ever output. One or either of the ‘*stitch_width for target_width*’ or ‘*stitch_width otherwise*’ constraints must be specified.

Examples

```
dpMinStitchWidth 0.5*targetWidth for (0.080 < targetWidth <= 0.120): \
    targetWidth otherwise:
dpMinStitchWidth 0.081 for 0.080 < targetWidth : \
    0.09 otherwise:
dpMinStitchWidth 0.05 for (0.080 < targetWidth <= 0.120): \
    targetWidth otherwise:
dpMinStitchWidth targetWidth otherwise:
```

dpSpacing

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Specifies spacing requirements between two face classes.

Usage

dpSpacing ‘(*class1, class2*)’ [*class_options*]:

[*space for condition [and condition] ...:*]

space otherwise:

Arguments

- ‘(*class1, class2*)’ [*class_options*]:

A required pair of arguments enclosed in parentheses specifying the combination of face classes to which the settings apply. The specified classes are order-independent. Both classes can be the same. The face classes can be one of three names:

- A user-defined face class classified by [dpFaceClass](#).
- The reserved class ‘side’, referencing any polygon face of a cut not defined by [dpFaceClass](#).
- The reserved wildcard ‘any’, referencing any defined face class or side, not including jogs.

Note



Although the reserved face class, side, is provided as a convenience for rule development and testing, Siemens EDA does not recommend its application in production rule files.

- [*class_options*]

The class options provide important adjunct functions to dpSpacing. If used, these options must be specified in the order as listed here and in the Usage section. The last class option must terminate the class arguments with a colon (:).

name‘(*spacing_name*)’

An optional argument specifying a name to use for the space between face classes established by the names of class1 and class2. The name argument must be specified after the class names and precede all remaining optional arguments.

spacing_name — The name associated to the spacing pair. The *spacing_name* must be enclosed in parentheses. Only alphanumeric characters, hyphens, and underscores are allowed in the name. The name parameters ‘any’ and ‘side’ are predefined.

Note

 Although optional, the name argument allows declared spaces to be segregated by their respective names for independent analysis and subsequent layer output. By default, all spaces are combined into one output layer. For this reason, Siemens EDA recommends always using the name argument in dpSpacing commands.

`metric('euclidean | {opposite [, extended value]}')`

An optional keyword specifying the type of metric to use in measuring the spacing between face classes. Only one metric may be specified. The metric type must be enclosed in parentheses. Valid extended values must be a single number or expression. The metric keyword must be specified after the mask and before the condition keywords. For more information, see “[Metrics](#)” in the *Calibre Verification User’s Manual*.

euclidean — Forms regions with quarter-circle boundaries that extend past the corners of selected edges. This is the default metric.

opposite — Forms regions with right-angle boundaries that do not extend past corners of the selected edges.

opposite, extended value — Forms regions with right-angle boundaries that laterally extend past corners of selected edges, dependent on the specified extension value.

- *space* for *condition* [and *condition*] ...:

An optional set of arguments specifying the required space in microns between two face classes where the specified condition is met. The condition is specified with at least one of five arguments:

minLength — The shorter of the two face lengths.

maxLength — The longer of the two face lengths.

minWidth — The width of the target shapes associated with the face classes.

maxWidth — The larger of the two widths.

runLength — The shared continuous edge projection of both face classes. Perpendicular edges are considered as 0 runLength.

The condition must take the form of:

argument operator value

For example:

```
minWidth < 0.030:
```

Multiple conditions can be joined with the ‘and’ modifier:

```
minWidth <= 0.032 and runLength > 0.060:
```

You can specify any space as a variable within the dpSpacing command. For example:

```
VARIABLE minTrackWidth 0.024
LITHO FILE [
    dpSpacing(class1, class2) name(c2c) :
        minWidth < $min_track_width:
        $default_space otherwise:
    ...
]
```

- ***space otherwise:***

A required argument ensuring that edges not meeting any condition specified within the dpFaceClass command are classified. The space-otherwise argument must be terminated by a colon (:). For example:

```
0.030 otherwise:
```

You can specify both space-for-condition and space-otherwise together. Both argument sets must be individually terminated with a semicolon (;). For example:

```
minWidth < 0.045:
0.030 otherwise:
```

Description

A required command that defines spacing dimensions between face classes established by [dpFaceClass](#). Use dpSpacing to measure distances between shapes on the target layer to determine if the layer must be decomposed to opposite masks, or if stitches must be inserted. The dpSpacing command insures decomposed layers and stitches do not result in MRC violations. Only one measurement is allowed for each classified edge pair; if dpSpacing(A, B) is specified, dpSpacing(B, A) would result in a compile error.

When you apply dpSpacing commands to user-defined face classes, you must declare dpSpacing commands after dpFaceClass.

Examples

Example 1 — Single Value with Default Metric

This example uses dpSpacing to ensure spacing of 54 nm between two face classes declared as side.

```
dpSpacing (side,side) name(s2s) : 0.054 otherwise:
```

Both command sections are terminated with a colon.

Example 2 — Single Value with User-Defined Class

The following example first uses the dpFaceClass command to create a face class called “widelineend”. Then, the dpSpacing command is used to establish a spacing of 0.065 between two “widelineend” face classes.

```
# The spacing between widelineend must be 0.065
dpFaceClass name(widelineend):
    true for (0.080 < length <= 0.100) and minAdjacent >= 0.020:
    false otherwise:

dpSpacing (widelineend, widelineend) name(wle2wle): 0.065 otherwise:
```

Example 3 — Conditional Values

This example uses the dpSpacing command to find all narrow and wide metal1 with a spacing of at least 30 nm but requiring a contiguous edge projection of at least 130 nm.

```
dpSpacing (narrowM1,wideM1) name(n2w):
    0.030 for runLength >= 0.130:
    0.020 otherwise:
```

Example 4 — Extended Metric

This example uses the opposite extended metric with an extension of 10 nm to define a spacing of 60 nm unless the faces are on wide metal (the minimum width \geq 30 nm). Using the opposite extended metric for line-end measurements often provides better corner accuracy than the default metric, euclidean.

```
dpSpacing (tip, side) name(t2s) metric(opposite, extended 0.010):
    0.060 for minWidth < 0.030 and maxLength >= 0.060:
    0.080 otherwise:
```

Example 5 — Exhaustive Permutations

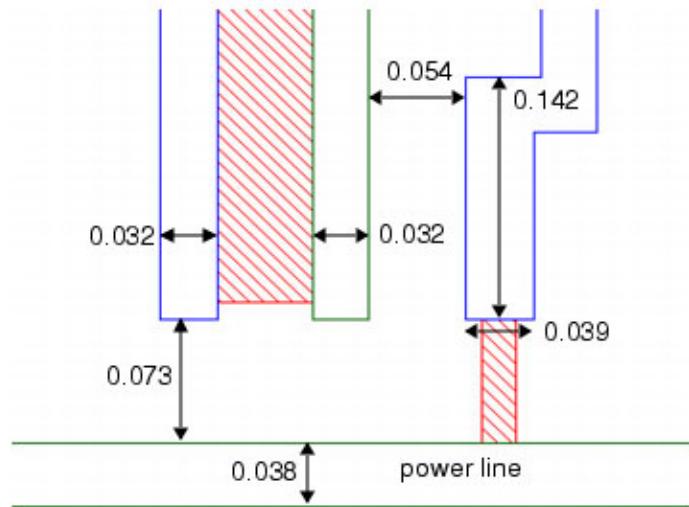
These code examples exercise all syntax permutations of the dpSpacing command.

```
dpSpacing (tip,side) name(t2s): 0.06 otherwise:  
dpSpacing (tip,side) name(t2s): 0.032 for minWidth <= 0.032:  
dpSpacing (tip,side) name(t2s):  
    0.032 for minWidth <= 0.032: 0.06 otherwise:  
dpSpacing (tip,side) name(t2s) metric(opposite): 0.06 otherwise:  
dpSpacing (tip,side) name(t2s) metric(opposite):  
    0.032 for minWidth <= 0.032:  
dpSpacing (tip,side) name(t2s) metric(opposite, extended 0.010):  
    0.032 for minWidth <= 0.032:  
    0.06 otherwise:  
dpSpacing (tip,side) name(t2s) metric(opposite, extended 0.010):  
    0.032 for minWidth <= 0.032 and runLength > 0.060:  
    0.06 otherwise:
```

Example 6 — runLength and Conjunction

The runLength argument outputs separators by mutual side length (exceeding the 142 nm segment length shown), and separators output by minLength (tip face length) in conjunction with minWidth (the power line width of 38 nm):

Figure 3-40. runLength and Conjunction

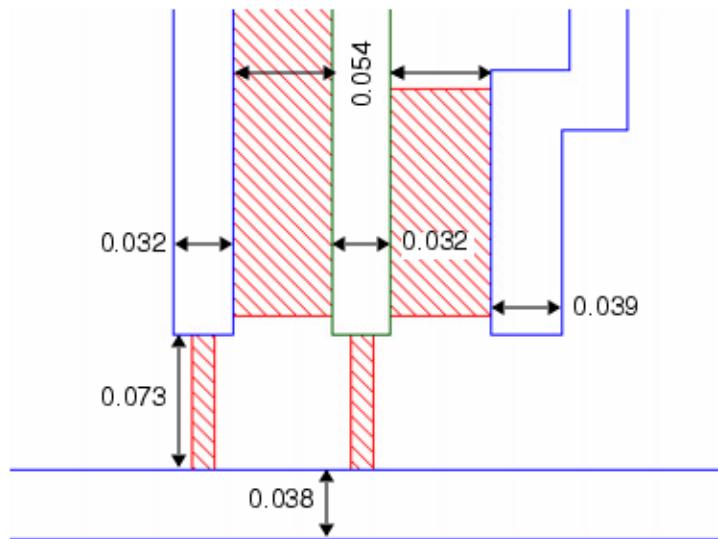


```
dpSpacing (tip,side): 0.074 for minWidth = 0.038 and minLength > 0.032:  
dpSpacing (side,side): 0.06 for runLength > 0.150:
```

Example 7 — otherwise and minLength Usage

The otherwise argument outputs separators by spacing alone, and minLength selectively outputs separators by tip face length:

Figure 3-41. otherwise and minLength Usage



```
dpSpacing (tip,side) : 0.074 for minLength <= 0.032:  
dpSpacing (side,side) : 0.060 otherwise:
```

dpStitchCandidates

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Generates shapes representing regions where overlapping decomposed layers may be joined.

Usage

dpStitchCandidates *criticalDistance value*

```
[anchor0 layer]  
[anchor1 layer]  
[anchork layer]  
[dpStitchMode combinedCycleAwareTargetNodePairs | cycleAware | targetNodePairs]  
[maskGridSpacing value]  
[maxStitchWidth value]  
[minArea value]  
[spanJogLength value]  
[stitchKeepout layer]  
[useMetalWidthForStitchRestrictions true | false]  
[userSeparator layer]  
-outLayer stitchCandidates  
[-outLayer separators]  
[-outLayer userFaceClass]
```

Arguments

- **criticalDistance** *value*

A required keyword specifying the size of the minimum feature to be resolved. The size is measured in microns.

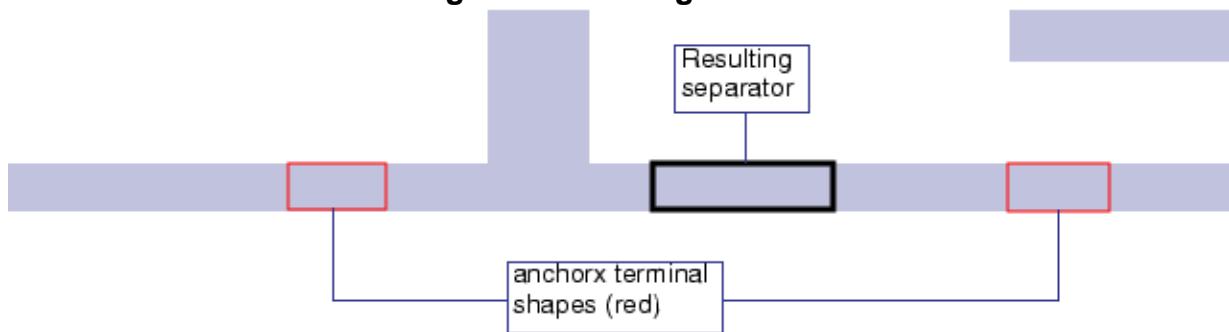
- *anchor0 layer anchor1 layer*

Optional arguments specifying a layer name that assign target shapes to either mask0 or mask1. The target shapes are properly assigned when they touch, overlap or enclose the anchor shapes. The area between anchor0 and anchor1 shapes on a single target shape may be potentially recognized as a stitch candidate area by dpStitchCandidates. Overlapping anchors force the generation of a stitch candidate in the overlapped region.

- *anchork layer*

An optional keyword for creating user-defined separators. The layer specified must contain polygons. You place two drawn shapes as terminals marking the extent of an area forcing the generation of a separator somewhere between them. This capability allows separators to be formed where the dpStitchCandidates rules cannot. The anchork shapes and formed separator are shown in [Figure 3-42](#).

Figure 3-42. Using anchorx



- dpStitchMode combinedCycleAwareTargetNodePairs | cycleAware | targetNodePairs

An optional keyword specifying the mode to select stitch candidate locations. Each mode uses different criterion to establish stitch candidate sites. This may affect dpStitchCandidates run-time and vary output of resulting stitch candidates.

combinedCycleAwareTargetNodePairs — Invokes both cycleAware and targetNodePairs modes. Initially, cycleAware candidates are constructed and then combined with targetNodePairs. This combination is then applied to areas of the input layer that can be legally stitched.

cycleAware — Uses even and odd-cycle detection to prioritize candidate insertion. This mode is more computationally expensive than targetNodePairs, and more likely to find candidate sites, effectively resolving odd cycles. See [Figure 3-43](#) for more information regarding mode behavior.

targetNodePairs — The default mode. This mode typically reduces dpStitchCandidates processing time over cycleAware mode for selection of stitch candidates. It uses contiguous edge pairs instead of path traces. See [Figure 3-43](#) for more information regarding mode behavior.

Figure 3-43. dpStitchMode cycleAware Behavior

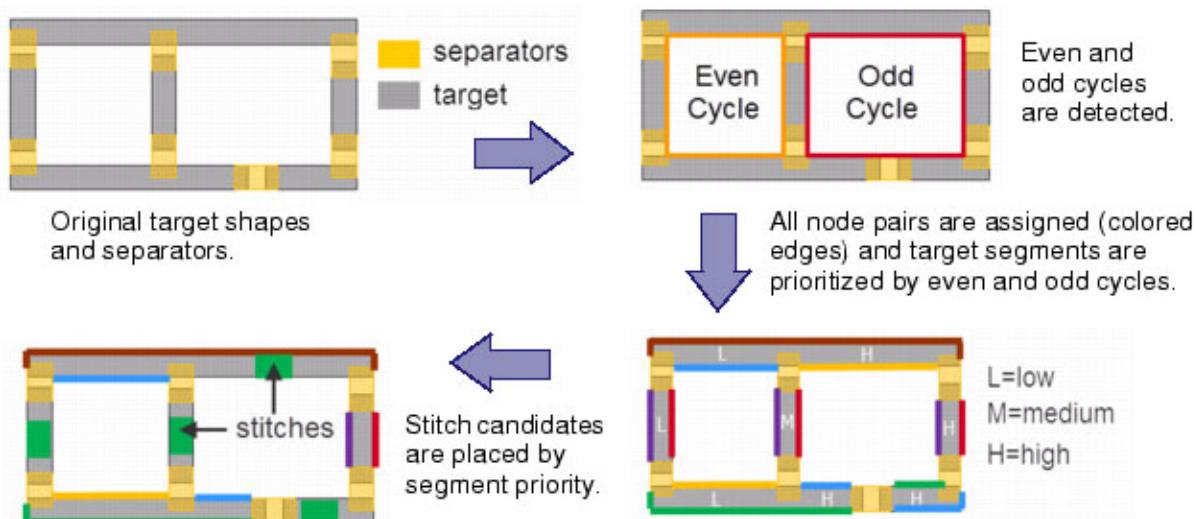
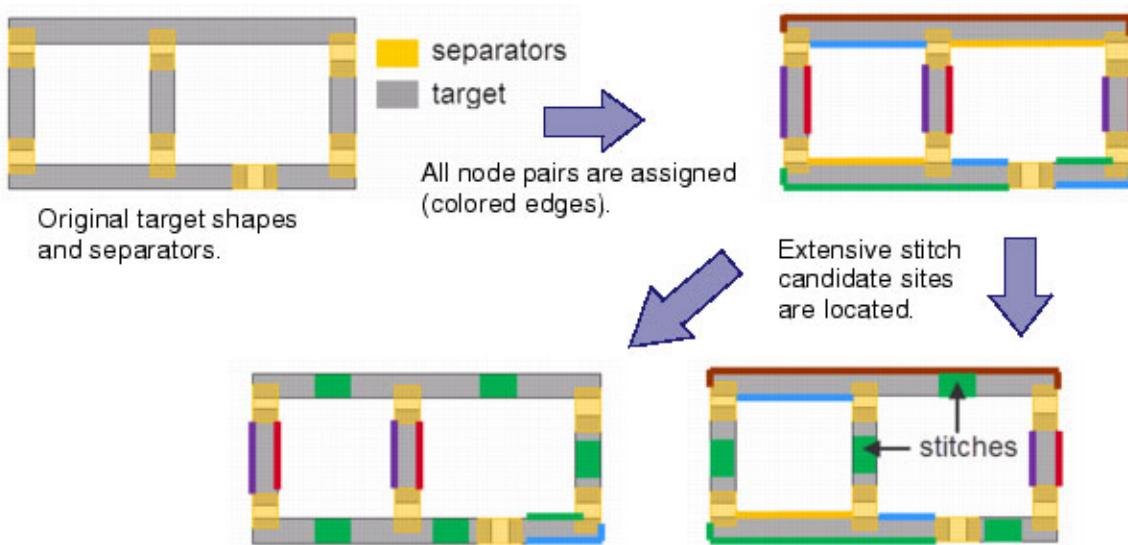


Figure 3-44. dpStitchMode targetNodePairs Behavior



- **maskGridSpacing value**

An optional keyword specifying the grid units for placing stitch candidates in microns. The maskGridSpacing keyword avoids grid unit ambiguity between DRC checks and the generation of stitch candidates. The value specified must be a multiple of the database unit setting or stitch candidates are not placed.

The length of all generated stitches are multiples of the maskGridSpacing value. If target shapes potentially receiving a stitch do not fall on the maskGridSpacing grid, results are unpredictable and may preclude the stitch candidate from proper placement. User-defined stitch candidates are always aligned to target shapes, even if the shapes are not on the maskGridSpacing grid.

By default, maskGridSpacing is 1 dbu (database unit), matching any precision and magnification settings of the run.

- **maxStitchWidth value**

An optional keyword that limits the width of a shape that is considered for a stitch candidate region. The maximum width is specified in microns. Target locations with widths less than or equal to maxStitchWidth and with lengths equal to or greater than the criticalDistance argument are evaluated for possible stitch candidates. By default, maxStitchWidth is set to 4*criticalDistance.

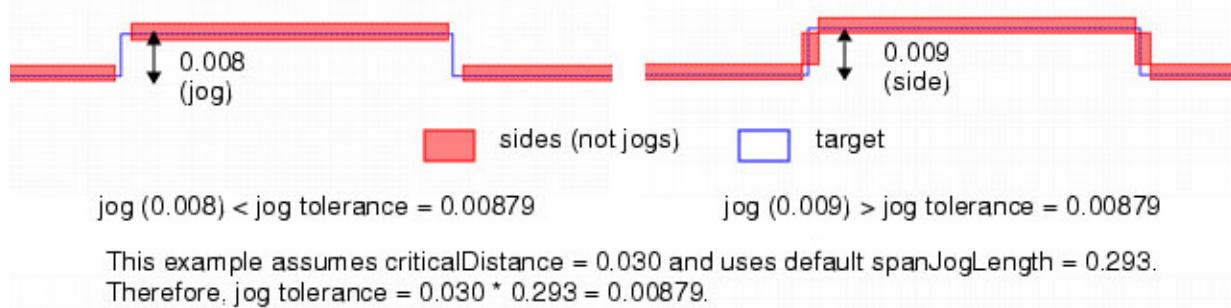
- **minArea value**

An optional keyword that specifies the minimum area of a stitched target segment (not including intersections) in square microns. If any target shape potentially overlapped by a stitch candidate could result in a polygon on the final mask that is smaller than the specified value, then that stitch candidate is not inserted. If minArea is not specified, stitch candidates are placed equidistant from each separator touching a given line.

- `spanJogLength value`

An optional keyword specifying the maximum length for an edge to be considered a jog. This aids in determining tips and sides. By default, `spanJogLength` equals $0.293 * criticalDistance$. Siemens EDA recommends using this computed default setting.

Figure 3-45. Determination of Jogs



- `stitchKeepout layer`

An optional keyword and layer pair specifying a single layer representing an area in which no stitches are permitted to be generated.

- `useMetalWidthForStitchRestrictions true | false`

An optional keyword that specifies the direction stitches are allowed to be generated relative to the aspect ratio of the metal shape by which they are contained. By default, the width of the metal containing the stitch is used to compute the allowed stitch length (intuitively, width is smaller than length, and the stitch bisects the shape's width). When set to false, the stitch is allowed to be placed parallel to the length of the metal shape, where the stitch would resolve coloring errors. True is the default setting. Figures 3-46 and 3-47 show the effect of the keyword on three cases:

Figure 3-46. True (Default) Setting

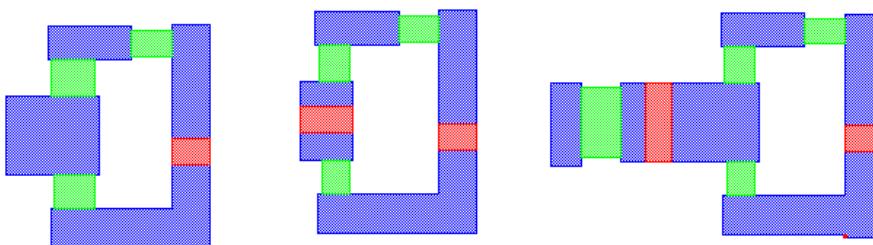
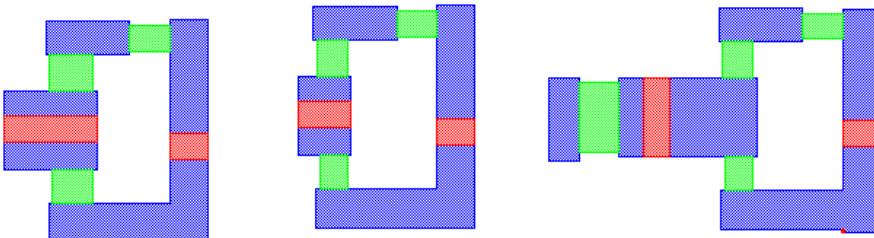


Figure 3-47. False Setting



`useMetalWidthForStitchRestrictions false`

- **userSeparator *layer***

An optional keyword and layer pair specifying a single layer representing all user-defined separators. This layer directs `dpStitchCandidates` to require separation of the target polygons that cannot be found through use of the `dpSpacing` commands. The `userSeparator` keyword should be used only if no programmatic solution is found with `dpSpacing` commands.

Note

 You must use caution with this capability for generic target shape interactions.
RET DPSTITCH modifies the input target shapes by defining stitch locations that may be used in a later coloring stage, and then guarantees these stitch candidates may be used without violating spacing requirements.

- **-outLayer *stitchCandidates***

A required keyword that outputs the stitch candidates to a layer to be ultimately mapped to the output database by the `setlayer dpstitch` and RET DPSTITCH commands. The `-outLayer stitchCandidates` keyword pair must precede all other `-outLayer` keywords.

- **-outLayer *separators***

An optional keyword that directs RET NMDPC to output separator shapes generated by `dpSpacing`. The `separators` keyword is helpful for debugging spacing dimensions established by `dpSpacing`. The output layer name saved by `setlayer dpstitch` must also be `separators`. In turn, RET DPSTITCH outputs separators using this same keyword.

To obtain loops and rings for debugging, however, you need edge pairs to act as separators and not polygonal shapes. For more information see “[Polygon to Edge Conversion](#)”.

- **-outLayer *userFaceClass***

An optional keyword that outputs a layer showing the members of the specified `userFaceClass`. This argument can be set to any `dpFaceClass` name, including the predefined `side`, `jog`, or `any`.

- -outLayer unusableUserSeparator

An optional keyword that outputs userSeparator shapes that fail this criteria:

- All userSeparator shapes intersecting target shapes must be overlapped by a polygon on the designated anchorx input layer.
- Every userSeparator shape must interact with two or more anchorx polygons.

The unusableUserSeparator keyword is helpful for debugging incorrectly drawn userSeparator shapes. This keyword is compatible only with dpStitchMode arguments cycleAware and combinedCycleAwareTargetNodePairs. RET DPSTITCH outputs separators using this same keyword.

Description

Generates stitch candidate shapes that overlap the mutually overlapping regions of decomposed target layers resulting in different masks. Stitch candidates are only used to form stitches as determined by the RET NMDPC command. Other debugging layers may be output as specified.

Examples

Example 1 — Outputting Layers From dpStitchCandidates to RET NMDPC

The following example uses dpStitchCandidates in the options block and outputs each layer using the setlayer dpstitch and RET DPSTITCH commands.

The layers in the options block and those listed in the setlayer dpstitch command are related explicitly by order; their names can be different, but their listed order of sequence is critical. In the example shown, layer m1 appears first in the options block and so must also appear first in the setlayer dpstitch command; layer KO appears second and therefore must also appear second in the setlayer dpstitch command.

```
LITHO FILE dpstitch.in [
...
    options OPTS {
        layer m1 opc      clear
        layer KO visible clear

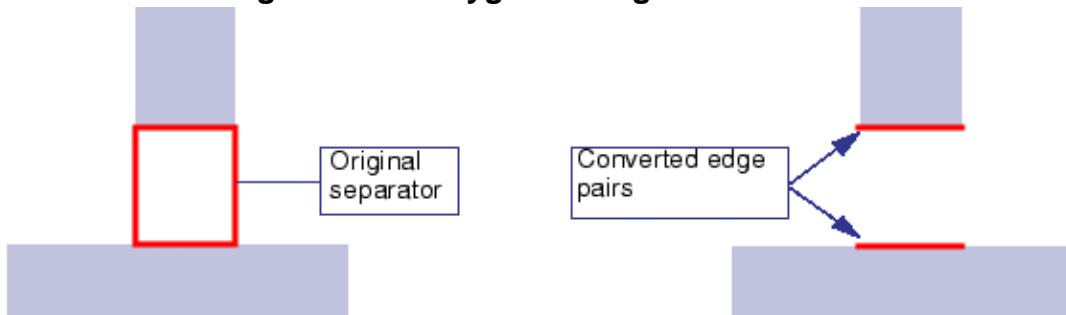
        dpStitchCandidates criticalDistance 0.32 \
            stitchKeepout KO \
            -outLayer stitchCandidates \
            -outLayer separators
    }

    setlayer stitchCan = dpstitch m1 KO MAP stitchCandidates OPTIONS OPTS
    setlayer separators = dpstitch m1 KO MAP separators           OPTIONS OPTS
]
stitchCan = RET DPSTITCH m1 KO FILE dpstitch.in MAP stitchCan
...
LITHO FILE nmdpc.in [
...
]
mask0 = RET NMDPC separator m1 stitchCan FILE nmdpc.in MAP mask0
mask1 = RET NMDPC separator m1 stitchCan FILE nmdpc.in MAP mask1
```

Example 2 — Converting Separator Layer for Use With Rings and Loops

The separator layer uses polygons to pair shapes that need to go on different masks. To create conflict loops and warning rings, however, requires edge pairs. The following example uses SVRF layer processing to convert the polygon layer to an edge pair layer.

Figure 3-48. Polygon to Edge Conversion



The second line uses a process-dependent value, 0.082, which you must set for your own topology.

```
temp_sep = polygon_separator COINCIDENT OUTSIDE EDGE m1_target
edge_pair_separator = INTERNAL temp_sep <= 0.082
```

dpStitchSize

[setlayer](#) [dpstitch](#) options Commands

Specifies stitch length requirements for various line widths.

Usage

```
dpStitchSize {[('stitch_length for width_condition'):] ... [stitch_length otherwise:]}
```

Arguments

- ‘*stitch_length for width_condition*’:

An optional argument specifying the valid combinations of length and width for a stitch. The *stitch_length* and *width_condition* parameters can be specified as constants, variables, or expressions in microns.

One or more ‘*stitch_length for width_condition*’ constraints may be specified. Each constraint must be terminated by a colon. A backslash is required to terminate the end-of-line when the command wraps to the next line. One or both ‘*stitch_length for width_condition*’ or ‘*stitch_length otherwise*’ constraints must be specified. Refer to the examples for various *stitch_length* constraint formats.

- *stitch_length otherwise*:

An optional argument specifying the stitch length to use when no preceding conditions are met. The value can be a constant or a variable. One or both ‘*stitch_length for width_condition*’ or ‘*stitch_length otherwise*’ constraints must be specified and only one ‘*stitch_length otherwise*’ constraint may be specified, and must appear last.

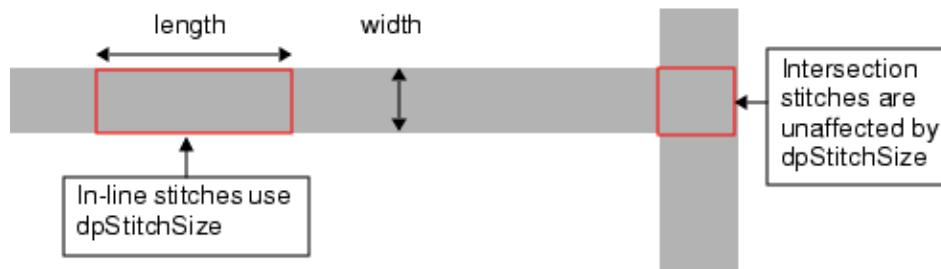
Description

An optional command that specifies stitch length for given target line widths. One or either of the ‘*stitch_length for width_condition*’ or ‘*stitch_length otherwise*’ constraints must be specified. Stitches at intersections are handled by commands [intersectionStitch](#) and [minIntersectionArmLength](#).

Note

 dpStitchSize does not apply to stitches within intersections.

Figure 3-49. dpStitchSize



Examples

Example 1 — Three Combinations

This example uses three width bins: up to 0.060 um, 0.060 through 0.120 um, and widths greater than 0.120 um. If width bins overlap, ambiguity arises with the application of associated stitch lengths. Each constraint is terminated with a colon.

```
dpStitchSize \
(0.060 <= length <= 0.120) for width <= 0.060: \
(0.120 <= length <= 0.240) for (0.060 < width <= 0.120): \
(0.240 <= length <= 0.500) otherwise:
```

Example 2 — Stitch Length Specified by Width

This example uses the width variable in an expression in the stitch_length constraint for the first argument.

```
dpStitchSize \
1.1 * width + 0.030 for width <= 0.050: \
0.055 for (0.050 < width < 0.080): \
0.040 otherwise:
```

minConvexCornerSpacing

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Defines the minimum distance from every outside corner to a stitch candidate rectangle.

Usage

```
minConvexCornerSpacing [strict:] \
[min_dist for width_constraint:] ... \
[alt_dist otherwise:]
```

Arguments

- strict:

An optional argument that prohibits singularities ($spacing = 0$) between the stitch candidate and the convex corner of the target shape. It is equivalent to enforcing the placement at no less than min_dist.

By default, the corners of stitch candidates and convex corners are allowed to overlay. If this optional argument is used, it must appear as the first argument.

- *min_dist* for *width_constraint*:

An optional argument that specifies the conditions how near a stitch can be placed to a convex corner on a shape.

min_dist — The minimum distance between the stitch and the corner. The distance is measured radially (euclidean metric).

width_constraint — How wide the shape needs to be in order for a stitch to be considered.

The min_dist and width_constraint values can be a constant or an expression, for example

```
0.06 for (0.045 <= width < 0.06) :
```

This argument can appear multiple times. Each instance must be terminated by a colon. Use the line continuation character (\) at the end of each line if placing them on separate lines.

- *alt_dist* otherwise:

An optional argument specifying a minimum distance when no width_constraints are satisfied. The alt_dist can be a value or expression. If used, this argument must appear last.

Description

An optional command that defines the minimum distance from every outside corner to a stitch candidate shape. By default, stitch candidates are positioned equidistant from the nearest vertices, either concave or convex.

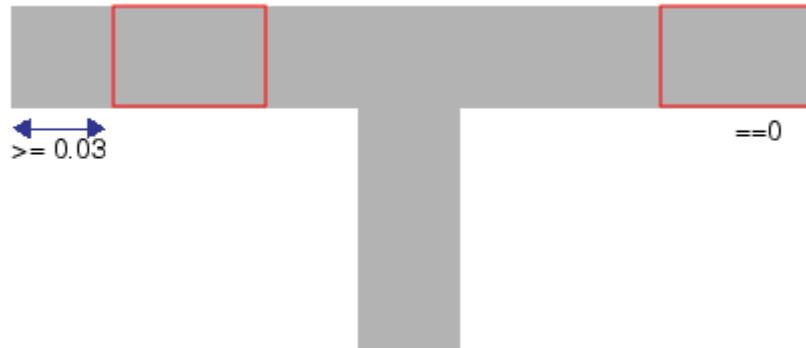
Examples

Example 1 — Typical Usage

The following example shows a typical spacing condition and potential output.

Figure 3-50. minConvexCornerSpacing Usage

```
minConvexCornerSpacing \
  0.030 for width <= 0.030: \
  0.045 otherwise:
```

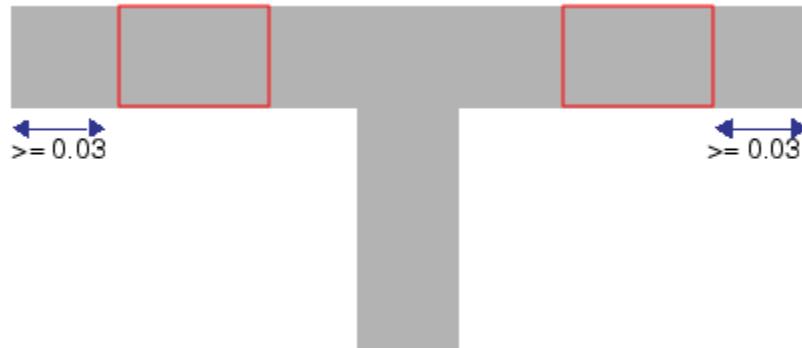


Example 2 — Effect of Strict Option

In contrast to Example 1, this code includes the strict argument. Every stitch candidate must be placed at least min_dist away from the convex corners.

Figure 3-51. minConvexCornerSpacing with strict Option

```
minConvexCornerSpacing \
  strict: \
  0.030 for width <= 0.030: \
  0.045 otherwise:
```



Example 3 — Minimum Distance In All Cases

```
minConvexCornerSpacing 0.035 otherwise:
```

Related Topics

[minConcaveCornerOverlap](#)
[intersectionStitch](#)

intersectionStitch

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Specifies the required conditions that make an intersection suitable for stitches.

Usage

```
intersectionStitch \
  {true for condition [and condition]...:} ... \
  false otherwise:
```

Arguments

- *condition*

An argument specifying at least one condition. Use the ‘and’ keyword to create a list of conditions. These must be satisfied for the intersection to be considered for stitching. The following keywords are valid arguments to the conditions:

`numArms` — The number of branches off the intersection that satisfy the [minIntersectionArmLength](#) requirement. Two arms make an L, three a T, and 4 a +.

`minWidth` — The width of the narrowest arm.

`maxWidth` — The width of the widest arm.

`maxJog` — The maximum permissible difference in arm width for opposing arms.

Setting `maxJog` to 0 prevents stitches if the arms are not the same width.

- `false` otherwise:

An argument terminated by a colon ensuring the intersection is not stitched.

Description

An optional command that defines any number of conditions that must be satisfied in order for an intersection to be considered for stitching. If a stitch is placed, it is equal to the size of the intersection. By default, stitches placed in intersections adhere to `dpStitchSize` specifications.

Examples

Example 1 — Single Condition

```
intersectionStitch \
    true for minWidth >= 0.45: \
    false otherwise:
```

Example 2 — List of Conditions

```
intersectionStitch \
    true for numArms >= 2 and minWidth >= 0.35 and maxWidth <= 0.55: \
    false otherwise:
```

Example 3 — Multiple Cases

```
intersectionStitch \
    true for numArms >= 3 and maxWidth <= 0.45: \
    true for numArms >= 2 and minWidth >= 0.30 and maxWidth <= 0.50: \
    false otherwise:
```

Example 4 — Use of maxJog

The first instance disables jogs when maxJog is set to 0. In the second instance, maxJog is parameterized as a function of minWidth.

```
intersectionStitch \
    true for numArms >= 3 and maxJog = 0: \
    true for numArms = 2 and maxJog < 0.1*minWidth: \
    false otherwise:
```

Related Topics

[dpStitchSize](#)

[minIntersectionArmLength](#)

minConcaveCornerOverlap

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Defines the minimum overlap required for concave corners to receive stitches.

Usage

```
minConcaveCornerOverlap [strict:] \
[min_dist for width_constraint:] ... \
[alt_dist otherwise:]
```

Arguments

- strict :

An optional argument that prohibits singularities ($overlap = 0$, also referred to as kissing corners) between the stitch candidate and the concave corner of the target shape. It is equivalent to enforcing the placement at no less than min_dist. See “[Example 2 — Effect of Strict Option](#)” for a comparison.

By default, the corners of stitch candidates and concave corners are allowed to touch. If this optional argument is used, it must appear as the first argument.

- *min_dist for width_constraint:*

An optional argument that specifies the conditions under which a stitch can be placed on a shape with concave corners. The min_dist and width_constraint values can be a constant, a range, or an expression.

This argument can appear multiple times. Each instance must be terminated by a colon. Use the line continuation character () at the end of each line if placing them on separate lines.

- *alt_dist otherwise:*

An optional argument specifying a minimum amount of overlap when no width_constraints are satisfied. The alt_dist can be a value or expression. If used, this argument must appear last.

Description

An optional command that is used to prevent jogs from being created on the decomposed mask shape. This requires that the overlap of a stitch beyond an inside (concave) corner on the original layer must be a certain amount. By default, stitch candidates are positioned equidistant from the nearest vertices, either concave or convex.

Examples

Example 1 — Setting a Single Value For All Widths

The following causes all stitch candidates to be 0.035 from a concave corner:

```
minConcaveCornerOverlap 0.035 otherwise:
```

Example 2 — Effect of Strict Option

The following figures provide a demonstration of minConcaveCornerOverlap behavior. The stitch candidate is shown by the red rectangle. As shown in [Figure 3-52](#), the default behavior (that is, without the strict argument) permits corners to touch, as in the upper right of the stitch candidate. With strict, as shown in [Figure 3-53](#), the candidate must overlap all concave corners.

Figure 3-52. minConcaveCornerOverlap Usage

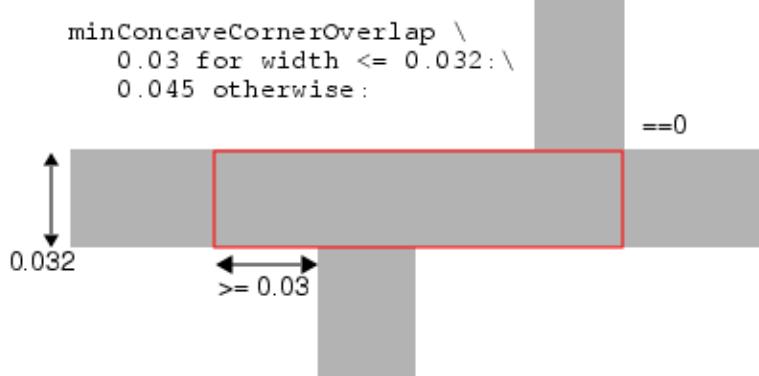
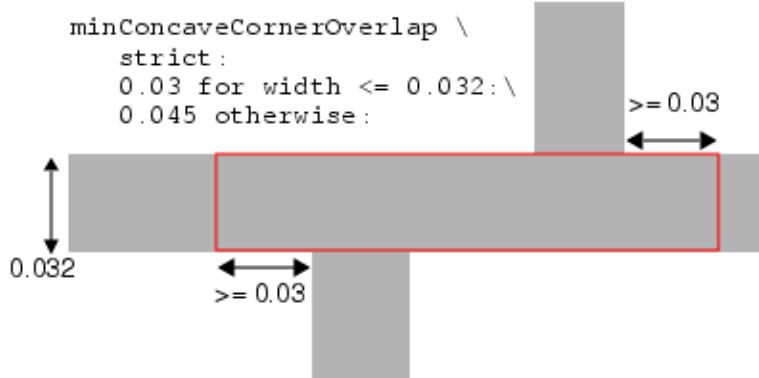


Figure 3-53. minConcaveCornerOverlap with strict Argument



Related Topics

[intersectionStitch](#)

[minConcaveCornerSpacing](#)

minIntersectionArmLength

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Specifies the minimum length of an intersection extension that is considered for stitching.

Usage

```
minIntersectionArmLength \
[minLength for width_constraint:] ... \
[altLength otherwise:]
```

Arguments

- *minLength* for *width_constraint*:

An optional argument that specifies for different widths of the arm the minimum distance the arm must extend before the intersection is considered for stitching.

The minLength and width_constraint values can be a constant or an expression:

```
0.06 for (0.045 <= width < 0.06) :
```

This argument can appear multiple times. Each instance must be terminated by a colon. Use the line continuation character () at the end of each line if placing them on separate lines.

- *altLength* otherwise:

An optional argument specifying a minimum arm length when no width_constraints are satisfied. The altLength can be a value or expression. If used, this argument must appear last.

Description

An optional command that defines the minimum length of an arm extending from an intersection that is considered for stitching. This command is usually used in conjunction with [intersectionStitch](#).

Examples

Example 1

```
minIntersectionArmLength \
0.040 for width <= 0.075: \
0.5*width for width <= 0.025: \
0.100 otherwise:
```

Example 2

```
minIntersectionArmLength 0.04 otherwise:
```

Related Topics

[intersectionStitch](#)

dpStitchToViaSpacing

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Specifies the spacing from vias to stitches.

Usage

```
dpStitchToViaSpacing viaLayer "('via_layer')": \
    [value for minViaEnclosure condition:]... \
    value otherwise:
```

Arguments

- viaLayer "('via_layer')"

An argument and layer name pair specifying the name of the via layer from which to measure stitch distance.

- *value* for minViaEnclosure *condition*:

An optional group argument specifying the stitch to via spacing based on the amount shapes on the via_layer are enclosed by their conductor. The distance is measured radially (by euclidean metric). You specify the spacing with the value argument. You specify the via enclosure with the condition argument. Vias extending outside the conductor layer assume the default value (specified with otherwise). You can specify multiple minViaEnclosure conditions within a single dpStitchToViaSpacing keyword. When specifying multiple minViaEnclosure conditions, you must specify the condition with the largest spacing value first:

```
...
0.038 for minViaEnclosure = 0.0: \
0.035 for minViaEnclosure < 0.010: \
...
```

You can also specify minViaEnclosure conditions that bound ranges of values for a single spacing value (if specifying an equivalence, you must include greater or less than). Bounded conditions must be enclosed within parentheses:

```
...
0.030 for (0.005 > minViaEnclosure >= 0.0): \
0.025 for (0.010 > minViaEnclosure >= 0.005): \
...
```

- *value* otherwise:

You specify this argument pair last in the argument order to provide a default spacing value.

Description

An optional keyword establishing the spacing distance from all stitch candidates to vias on the named layer. The dpStitchToViaSpacing keyword may only be specified once in a LITHO FILE statement.

Examples

Example 1 — Basic Usage

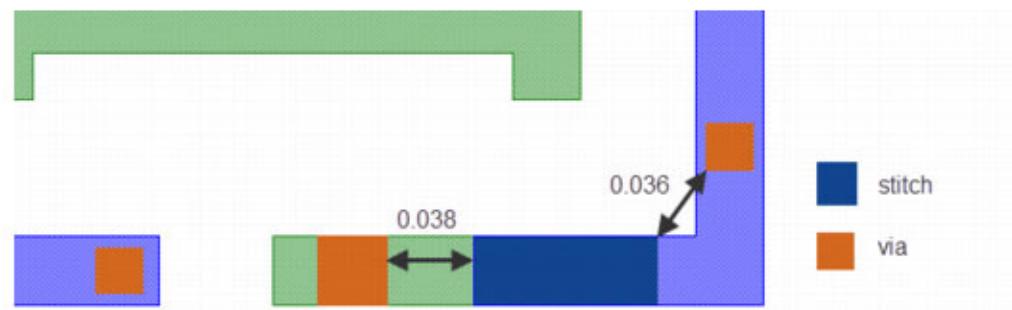
Figure 3-54. Basic dpStitchToViaSpacing Usage



```
dpStitchToViaSpacing viaLayer(V0) : \
  0.035 for minViaEnclosure < 0.010: \
  0.030 otherwise:
```

Example 2 — Multiple minViaEnclosure Conditions

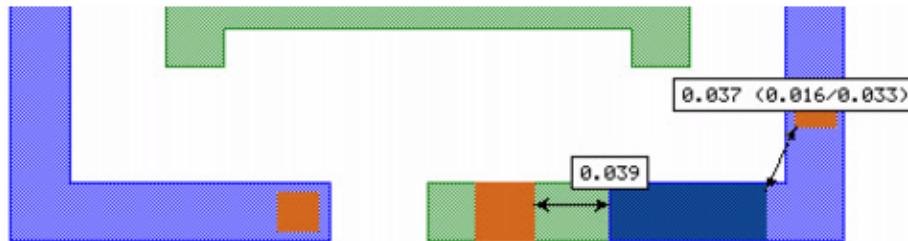
Figure 3-55. dpStitchToViaSpacing with Two Via Enclosures



```
dpStitchToViaSpacing viaLayer(V0) : \
  0.038 for minViaEnclosure = 0.0: \
  0.035 for minViaEnclosure < 0.010: \
  0.032 otherwise:
```

Example 3 — Bounded minViaEnclosure Condition

Figure 3-56. dpStitchToViaSpacing with a Bounded Via Enclosure



```
dpStitchToViaSpacing viaLayer(V0_in) : \
  0.038 for (0.005 > minViaEnclosure >= 0.0): \
  0.033 for minViaEnclosure >= 0.005: \
  0.030 otherwise:
```

minConcaveCornerSpacing

[setlayer](#) [dpstitch](#) [options](#) [Commands](#)

Defines the minimum distance from every inside corner to a stitch candidate rectangle.

Usage

```
minConcaveCornerSpacing [strict:] \
[min_dist for width_constraint:] ... \
[alt_dist otherwise:]
```

Arguments

- strict:

An optional argument that prohibits singularities ($overlap = 0$, also referred to as kissing corners) between the stitch candidate and the concave corner of the target shape. It is equivalent to enforcing the placement at no less than min_dist.

By default, the corners of stitch candidates and concave corners are allowed to touch. If this optional argument is used, it must appear as the first argument.

- min_dist for width_constraint :

An optional argument that specifies the conditions how near a stitch can be placed to a concave corner on a shape.

min_dist — The minimum distance between the stitch and the corner. Unless strict is specified, singularities (kissing corners) are allowed.

width_constraint — How wide the shape adjoining the concave corner needs to be in order for a stitch to be considered.

The min_dist and width_constraint values can be a constant or an expression, for example:

```
0.06 for (0.045 <= width < 0.06) :
```

This argument can appear multiple times. Each instance must be terminated by a colon. Use the line continuation character (\) at the end of each line if placing them on separate lines.

- alt_dist otherwise:

An optional argument specifying a minimum distance when no width_constraints are satisfied. The alt_dist can be a value or expression. If used, this argument must appear last.

Description

An optional command that defines the minimum distance from every inside (concave) corner to a stitch candidate. Different minimums can be specified for different shape widths. By default, stitch candidates are positioned equidistant from the nearest vertices, either concave or convex.

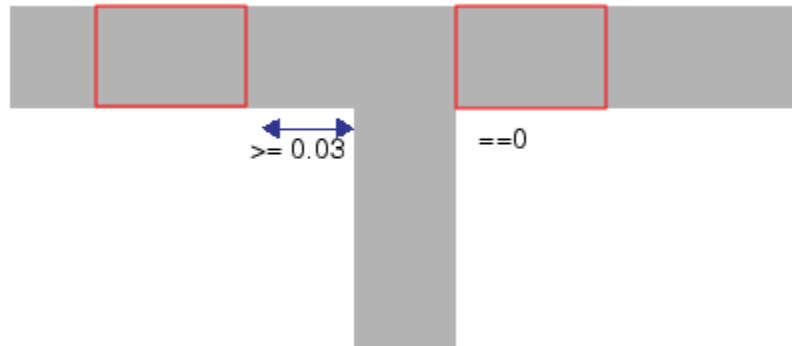
Examples

Example 1 — Typical Usage

The following example shows a typical spacing condition and potential output.

Figure 3-57. minConcaveCornerSpacing Usage

```
minConcaveCornerSpacing \
  0.030 for width <= 0.030: \
  0.045 otherwise:
```

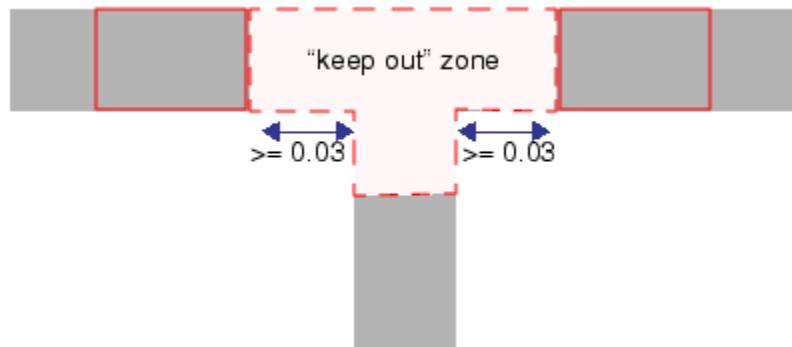


Example 2 — Effect of Strict Option

In contrast to Example 1, this code includes the strict argument. Every stitch candidate must be placed at least min_dist away from the concave corners.

Figure 3-58. minConcaveCornerSpacing with strict Option

```
minConcaveCornerSpacing \
  strict: \
  0.030 for width <= 0.030: \
  0.045 otherwise:
```



Related Topics

[minConcaveCornerOverlap](#)
[minConvexCornerSpacing](#)

version

setlayer dpstitch options Commands

Declares the version of the LITHO FILE statement.

Usage

version {1 | year.quarter}

Arguments

- **1 | year.quarter**

Specify either 1 or the release in the format year.quarter.

Description

A required keyword that declares the version of the LITHO FILE statement. This keyword must appear first in the options block.

Examples

```
options OPT {
    version 1
    layer ca visible
    ...
}
options OPT {
    version 2016.4
    layer ca visible
    ...
}
```

Chapter 4

Self-Aligned Double-Patterning Concepts and Command Reference

RET SIDCUTFILL is the Calibre command developed to support SADP and SAQP methodologies using the sidewall-is-dielectric (SID) process.

This methodology converts target shapes into tracks and selects some tracks as mandrels. Mandrel tracks are imaged on the wafer, normally wider than the intended target shape. The mandrel material is etched to the actual target width. Sidewalls are grown around the mandrels and then etched to their desired width (the gap between target shapes). The mandrel is then removed. Before metalization, a cut mask must be added to block the deposition of metal where required, to define the intended target ends. RET SIDCUTFILL identifies the mandrel shapes and defines the cut mask needed prior to metalization.

Self-Aligned Double-Patterning Concepts	218
Introduction to SID Cut-Fill Methodology	218
Cuts	235
Trim Mask.....	241
Derived Target	244
Error Handling	245
Operational Modes of Control	251
Self-Aligned Double-Patterning Command Reference	254
RET SIDCUTFILL.....	255
LITHO FILE.....	268
SADP-Specific Spacing Parameters	288

Self-Aligned Double-Patterning Concepts

The Calibre RET SIDCUTFILL command and its associated keywords generate topologies to adhere to Spacer Is Dielectric (SID) processes for Self-Aligned Double-Patterning (SADP). These commands address track generation and coloring, cut generation, track termination, track layout, and output error layers to help resolve SADP-related issues during SADP topology generation.

Introduction to SID Cut-Fill Methodology	218
Cuts	235
Trim Mask	241
Derived Target	244
Error Handling	245
Operational Modes of Control	251

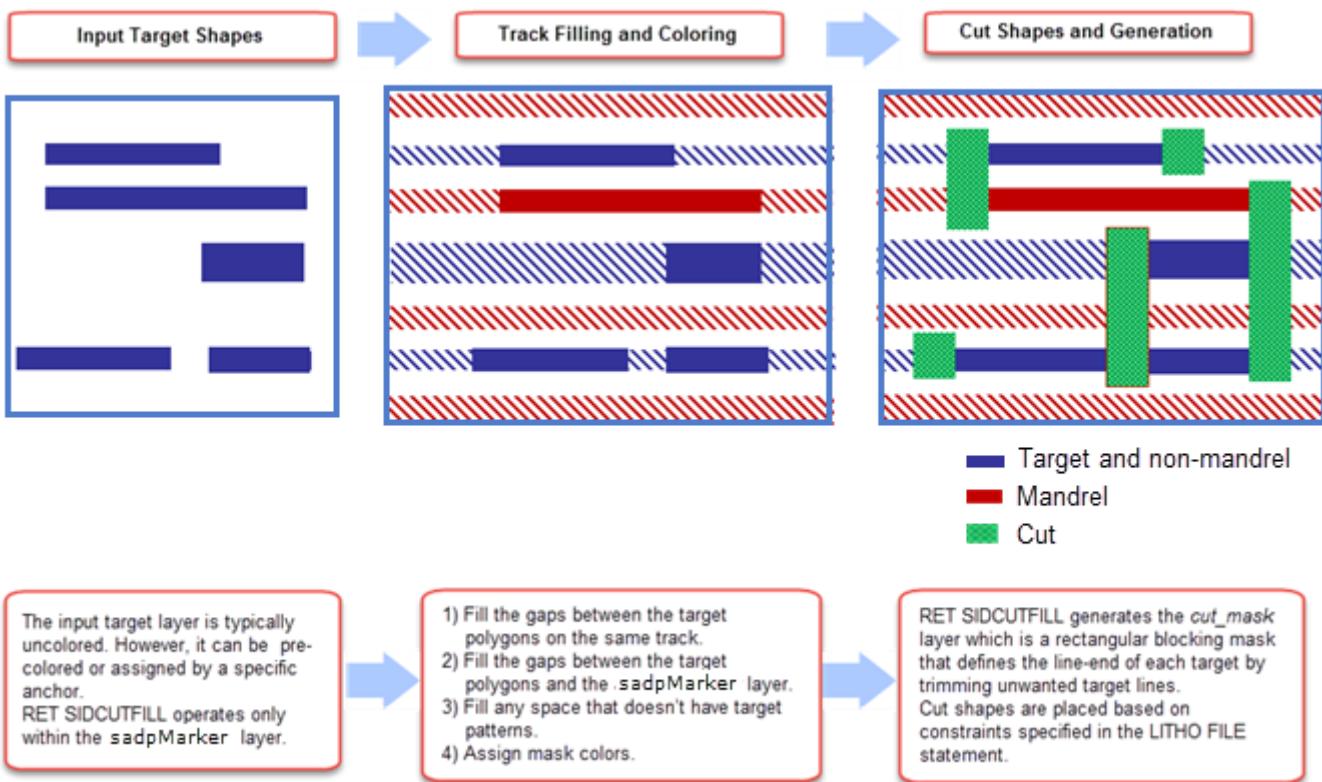
Introduction to SID Cut-Fill Methodology

The RET SIDCUTFILL command generates mandrel and non-mandrel tracks, and cuts to build design topologies satisfying the SID Cut-Fill process.

The Calibre SID Cut-Fill Solution Flow Overview

RET SIDCUTFILL uses the input target shapes of the design as a template for track filling and mask assignment. After tracks are established, cuts are generated that terminate the target shapes.

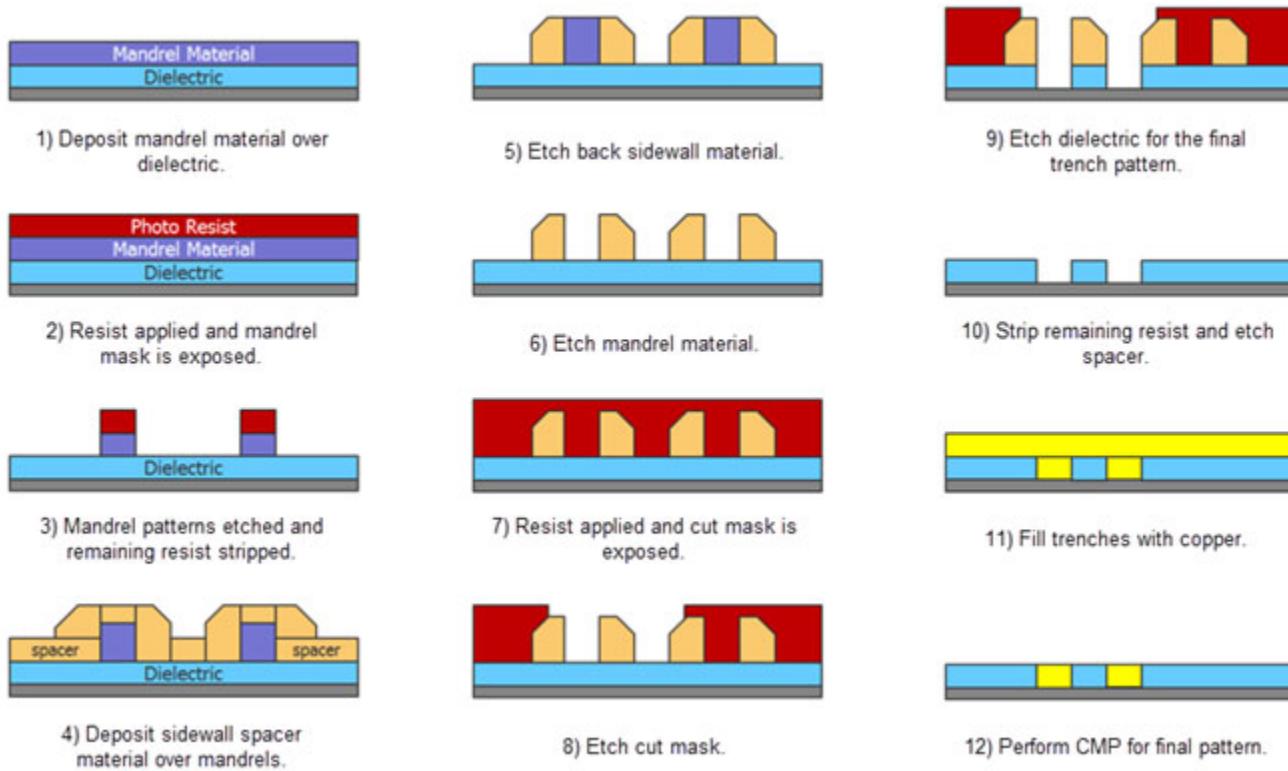
Figure 4-1. RET SIDCUTFILL Flow



Pattern Transfer

To build the SID-based design, the track pattern is transferred to the wafer in a specific order, as depicted here:

Figure 4-2. Pattern Transfer Overview



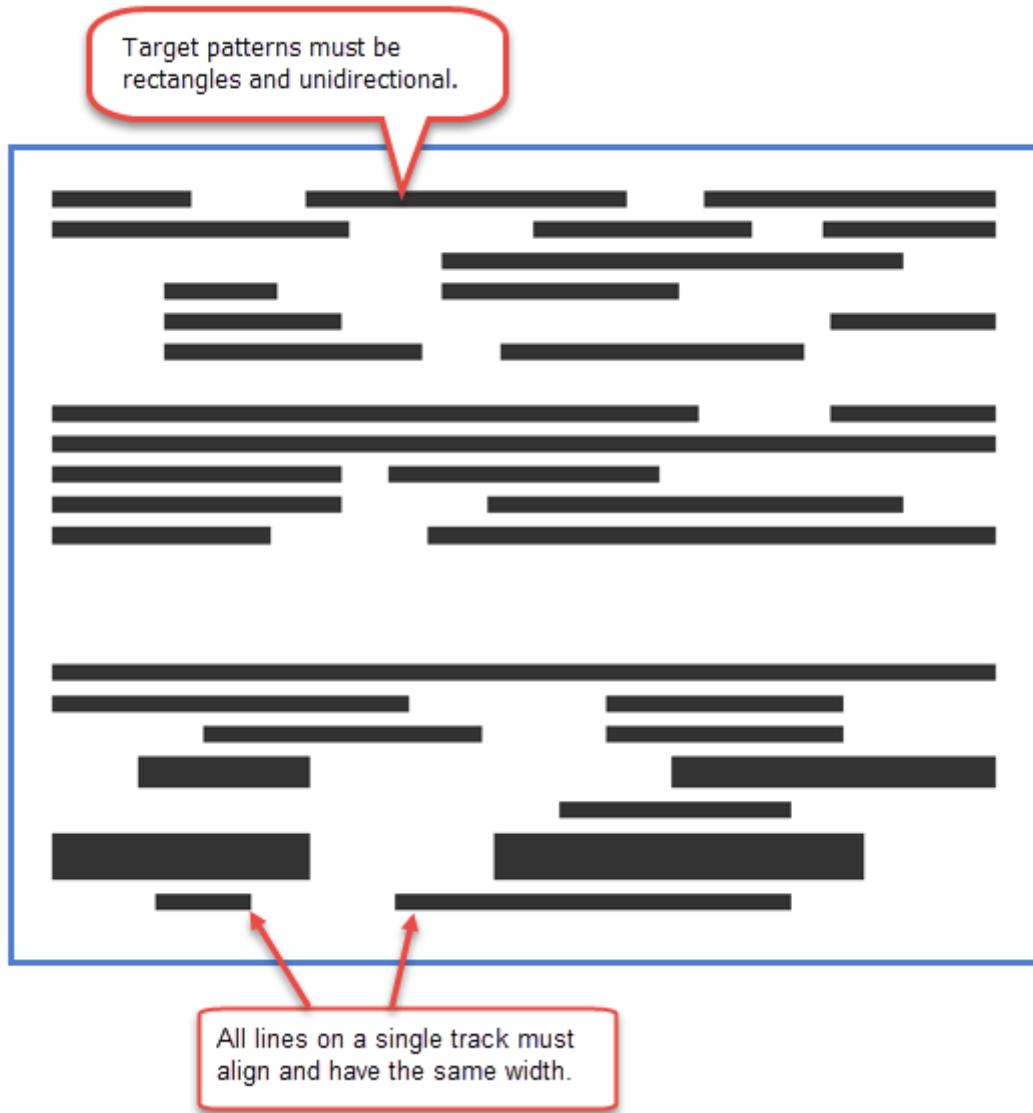
- **Steps 1-3: Mandrel Processing** — Every other track in the design is designated as a mandrel. The mandrel material is laid over dielectric (acting as an insulator and hard mask). Photoresist is laid over the mandrel material, the mandrel mask is exposed and then etched back to reveal the mandrel patterns. All remaining resist is then stripped away.
- **Steps 4-6: Sidewall Spacer Processing** — Grow the sidewall material around the mandrel material, then etch the mandrel material away, simultaneously etching the sidewalls back to the required width.
- **Steps 7-9: Cut Mask Processing** — Apply photoresist over the remaining spacer material, then expose the cut mask. Wash resist from the exposed cut areas and etch through the dielectric (hard mask) for the resulting trench pattern.
- **Steps 10-12: Metalization Processing** — Strip remaining photoresist and etch spacer material away. Sputter copper over wafer surface, filling trenches. Perform Chemical Mechanical Polishing (CMP) to planarize wafer surface and reveal final design pattern.

The SADP Target Design

An example target design is illustrated in [Figure 4-3](#) showing two design constituents:

- **Target Layer** — The black horizontal rectangles are the target shapes. They represent the patterns to be printed on the wafer. They must all be oriented to the same axis, either X or Y (with the possible exception of jogs). All target shapes must be manhattan geometries. Excepting jogs and pads, all target shapes sharing the same track must be the same width.
- **SADP Boundary** — The blue border shows the track containment region, a polygon on the sadpMarker layer. The RET SIDCUTFILL command generates mandrel and non-mandrel tracks, and cuts only within the sadpMarker polygon.

Figure 4-3. Target Shapes Within a Boundary



sadpMarker Input Layer

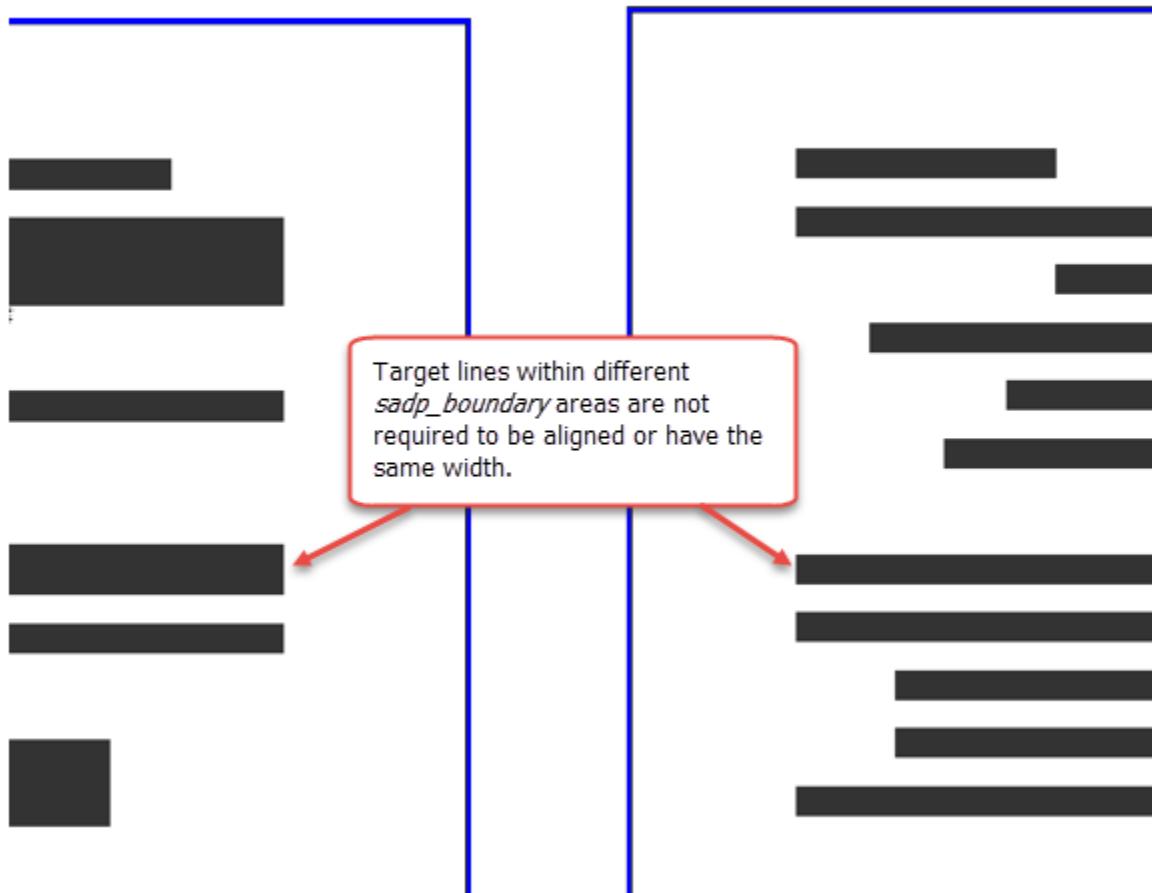
As previously stated, RET SIDCUTFILL only operates within the confines of the sadpMarker layer. Specific attributes of the sadpMarker layer include:

- **Multiple Boundary Shapes** — A maximum of 128 sadpMarker polygon shapes may be specified. If this limit is exceeded, check the order of keywords in the inputLayerOrder list. It is rare to require more than ten sadpMarker shapes. If necessary, this environmental variable can be set to the number of shapes necessary:

```
setenv CALIBRE_RET_cutFill_maxSadpMarkers 200
```

- **Individual Decomposition** — RET SIDCUTFILL performs track ordering and cut shape placement within each sadpMarker layer individually. This provides for separate sadpMarker shapes to contain target designs of independent track alignment and width.
- **Rectangular Boundaries** — RET SIDCUTFILL supports input of arbitrary polygon sadpMarker shapes. These arbitrary shapes are converted to manhattan polygons enclosing the original arbitrary shapes that are input. The converted manhattan sadpMarker shapes are output to the effectiveSadpMarker layer. Boundary shapes may be drawn in any alignment to each other.

Figure 4-4. Separate sadpMarker Layers



mandrelAnchor and nonMandrelAnchor Input Layers

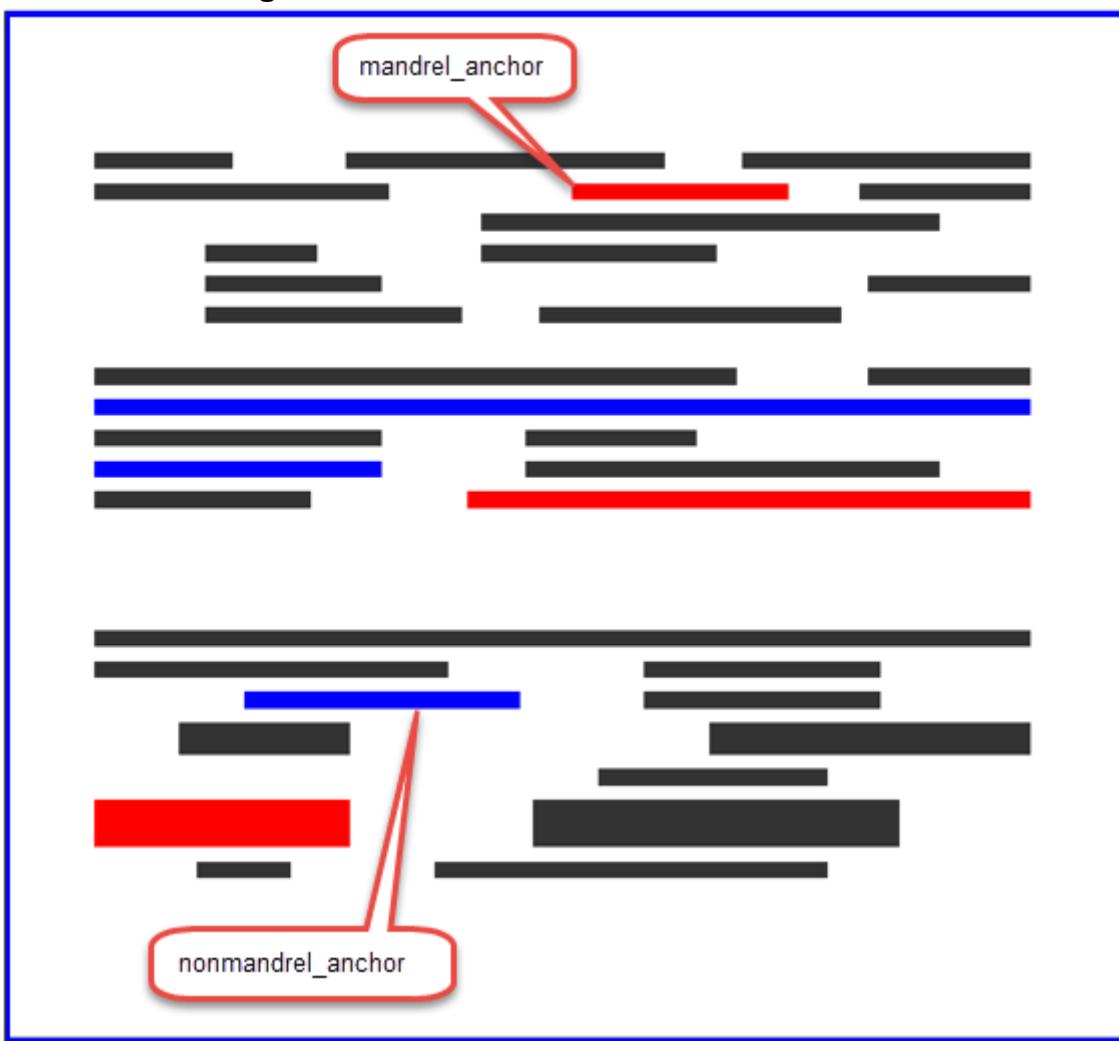
The mandrel and non-mandrel comprise alternating tracks required by the SID-based SADP process (see “[Pattern Transfer](#)” on page 220 for more information). The assignment of target shapes to mandrel and non-mandrel tracks requires anchors specifically used for this purpose. The anchors shapes are logically OR’ed with the target shapes. The effective target shape is equivalent to:

target OR mandrelAnchor OR nonMandrelAnchor

- **mandrelAnchor Layer** — Any target polygon overlapping a shape on the mandrelAnchor layer is assigned to the mandrel track.
- **nonMandrelAnchor Layer** — Any target polygon overlapping a shape on the nonMandrelAnchor layer is assigned to the non-mandrel track.

In [Figure 4-5](#), red and blue rectangles are drawn on the mandrelAnchor and nonMandrelAnchor layers, respectively. These anchors assign the target shapes they overlap to the mandrel and non-mandrel tracks.

Figure 4-5. Mandrel and Non-mandrel Tracks



Track Layers

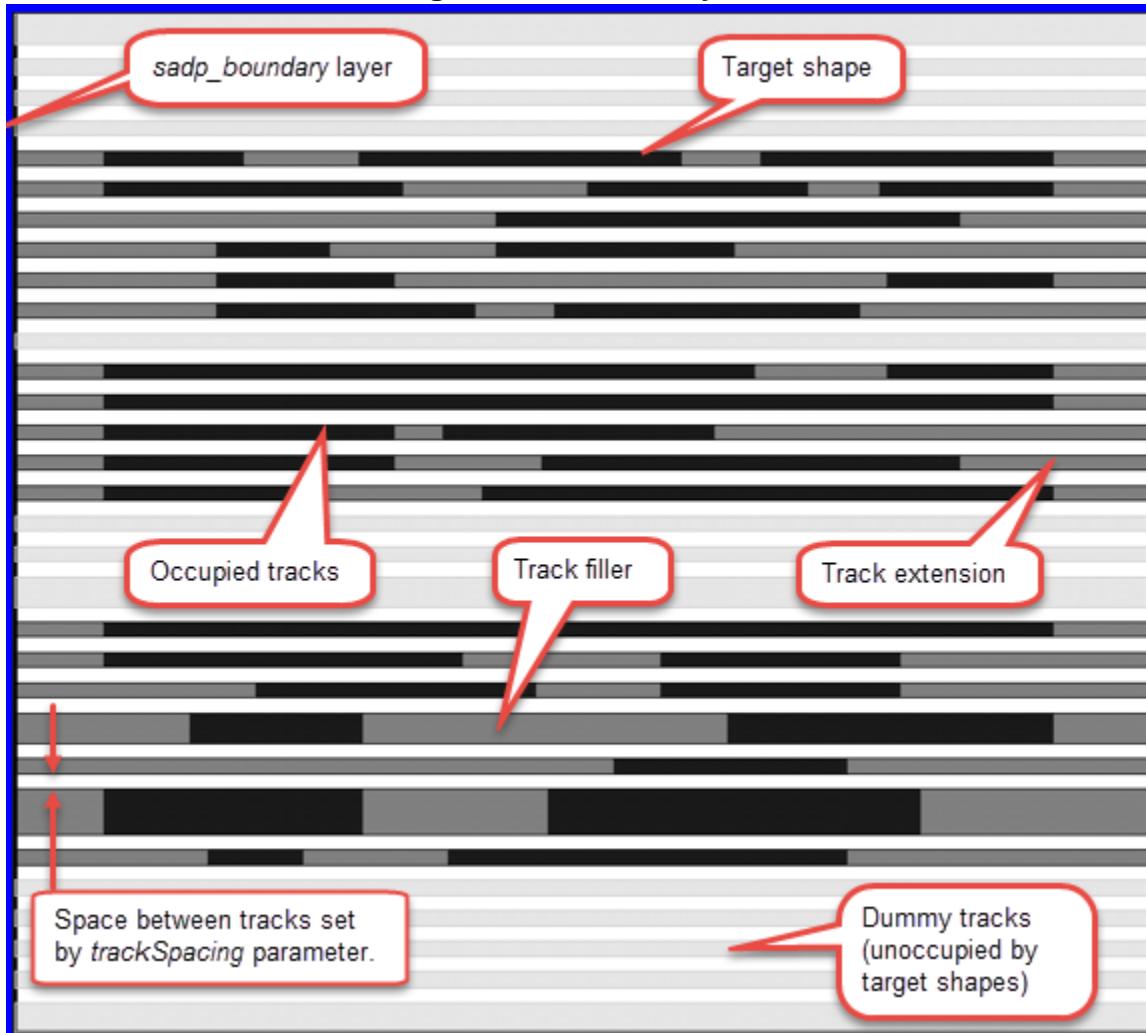
RET SIDCUTFILL generates tracks for each target shape to occupy. Every track spans an `sadpMarker` polygon. RET SIDCUTFILL generates tracks of the same width as the target polygons. If some target polygons do not conform to the established track patterns or specified RET SIDCUTFILL parameters, those target shapes are rejected and output to the `illegalTarget` error layer keyword. If there is unoccupied space between occupied tracks, that space is filled with additional dummy tracks. The space between tracks is set by the `trackSpacing` parameter (for more information, see `trackSpacing` in “[LITHO FILE](#)” on page 268). This is the width of the sidewalls chemically grown around the mandrels after etching back to the desired width. The smallest track width is set by the parameter `minTrackWidth` or `trackWidth`¹. Wider tracks

1. The `minTrackWidth` parameter supports a single value; the `trackWidth` parameter supports a list of legal track widths.

are used if necessary to fill the space. the RET SIDCUTFILL tool generates tracks inside the sadpMarker polygons as shown in [Figure 4-6](#).

- **Track Generation Limits** — RET SIDCUTFILL only generates tracks within the sadpMarker layer.
- **Occupied Track** — A track that is coincident with target shapes.
- **Dummy Track** — A track that is unoccupied, or has no target shapes coincident with it. Dummy tracks fill the space outboard of target shapes to the sadpMarker layer.
- **Track Extension** — A track segment added between the outermost target shape end to the sadpMarker edge.
- **Track Filler** — A track segment added between two target shapes.

Figure 4-6. Track Layers



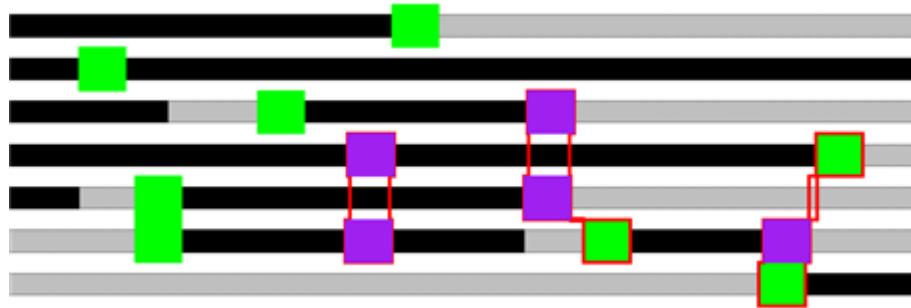
The Track Containment Region

Tracks are generated within each polygon on the `sadpMarker` layer. The `sadpMarker` layer is required input to RET SIDCUTFILL. It determines the regions of the design it needs to process. For more information, see the section entitled “[sadpMarker Input Layer](#)” on page 222.

The Cut Mask

To ensure correct connectivity of the design, RET SIDCUTFILL generates shapes on the `cutMask` layer that terminate target shapes according to the parameters provided. Without the `cutMask` layer, the final wafer pattern would only consist of parallel lines (generated tracks); all the target patterns on each track would be shorted together end-to-end. A small portion of a design with a `cutMask` layer added is shown in Figure 4-7.

Figure 4-7. Generated Output and Error Layers

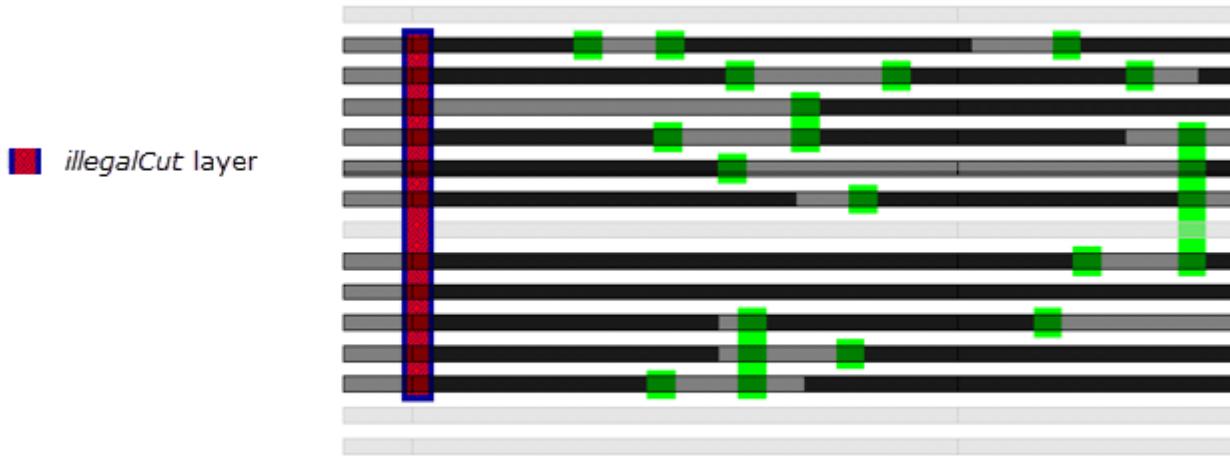


These are the key layers and their respective colors as shown in Figure 4-7:

- **Tracks** — Generated tracks span the width of each `sadpMarker` polygon. In Figure 4-7, the target shapes are shown in black coinciding with portions of the track layer. The generated track layer is always a superset of the target shapes (gray).
- **Target Shapes** — The shapes of the original design (black).
- **cutMask** — Shapes on this layer are placed to terminate tracks so the generated tracks correctly render target shapes (green) without short circuits or undue parasitic effects. Cuts may also be placed over dummy tracks to reduce parasitic effects as well. Cuts may be shared between multiple tracks (up to the value specified by the `maxCutLength` parameter) to avoid spacing errors between cut shape ends.
- **cutPlacementError** — Polygons covering required cuts that failed legal placement. This layer joins other cuts (legal or otherwise) to indicate spacing errors that determine these cuts to be illegal. At least one `illegalCut` comprises every `cutPlacementError` but there is no guarantee which of the cuts enclosed are illegal. Also, every illegal cut resides within a cut placement error (red).

- **illegalCuts** — Error shapes indicating cutMask shapes that violate placement rules (magenta). The illegal cuts that are shown in Figure 4-7 violate one of the specified conditions:
 - Every cut mask shape must be a rectangle of a specified width.
 - Spacing between cut mask shapes must meet the restrictions specified in the setup file (see “[The cutMask Shape Spacing Requirement](#)” on page 229).
 - The maximum cut length, if specified.

Figure 4-8. maxCutLength and the illegalCut Layer



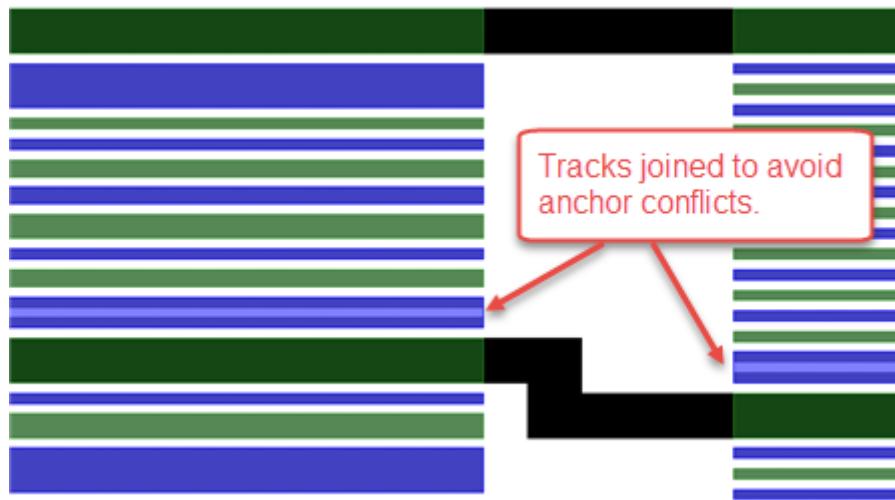
Where the adjusted cut exceeds the *maxCutLength* parameter.

Track Coloring (Mandrel and Non-Mandrel Layer Assignments)

When required, RET SIDCUTFILL can also assign mask colors as seen in [Figure 4-9](#). Key behavioral issues include:

- Target shapes that touch or overlap the mandrel anchors (shapes on the mandrelAnchor layer) are assigned to the mandrel mask (green in the figure.)
- Tracks within different sadpMarker regions do not have to align.
- The two mandrel anchors applied in this example would result in an anchor conflict because the number of tracks between the two anchored mandrels is even. RET SIDCUTFILL avoids this conflict by automatically joining two adjacent dummy tracks into a single track. Joining tracks is legal where no target shapes are overlapped by the tracks and where the combined track width equals one of the legal track widths. When using minTrackWidth, this always true; when using a trackWidth list, the combined track width must equal one width in the list. These tracks are joined in [Figure 4-9](#) just above the lower anchored shape in both sadpMarker regions.

Figure 4-9. Track Coloring

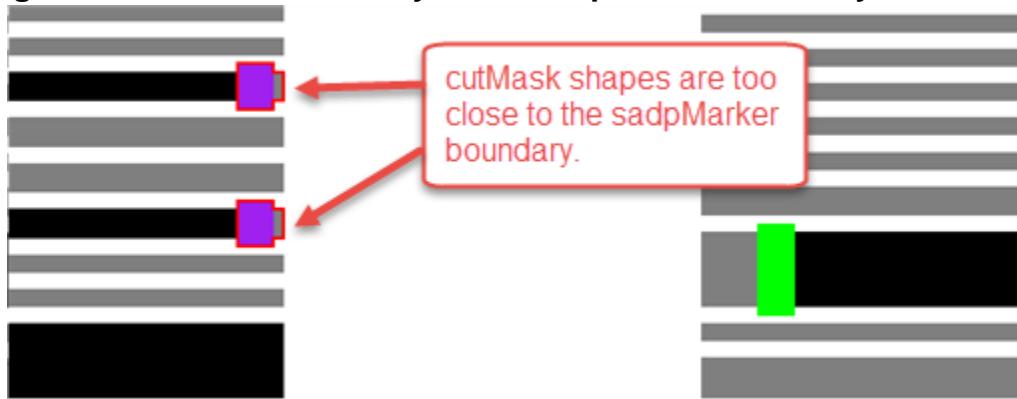


Interaction Between `sadpMarker` and `cutMask` Layers

Figure 4-10 illustrates the interaction between `cutMask` shapes and edges of `sadpMarker` polygons. The tracks are generated to touch the boundaries of their containing polygons. The `cutMask` shapes do not extend outside the containing polygons (except as specified by the `cutOverlap` parameter if a target edge is aligned horizontally with the containing polygon). In Figure 4-10, there is one target line that reaches the edge of the `sadpMarker` region. No `cutMask` shape is added and, in this case, no error marker is generated.

If a target shape is close enough to the boundary of the `sadpMarker` boundary so that a `cutMask` shape cannot be placed (either because of failing the `minArea` parameter requirement on the leftover track, or because the cut would extend beyond the boundary of the marker), then an error marker is added to the `cutPlacementError` output layer. The next figure shows two cuts that cannot be placed because of `minArea` violations at the edge of an `sadpMarker` polygon.

Figure 4-10. The `cutMask` Layer and `sadpMarker` Boundary Interaction



As seen in Figure 4-10, tracks are always generated to the edge of the `sadpMarker` polygon boundary. Like tracks, the `cutMask` layer is generated only within the `sadpMarker` polygon. The error layers within the left-side `sadpMarker` region represent `cutMask` shapes that were

generated too close to the `sadpMarker` edge. The `sadpMarker` region must be increased or the target lines must be redrawn.

The `cutMask` Shape Spacing Requirement

The spacing specifications between cuts may require multiple cut shapes to be joined and merged into a single rectangle. Figure 4-11 demonstrates two cuts at line-ends (*a* and *b*) that is too close to each other if placed discretely; RET SIDCUTFILL assigns a cut to the intermediate track between track *a* and *b* and then merges all three cuts to form a legal cut shape. For these design rules, if the two cuts (*a* and *b*) were placed without merging them, a spacing error would result between them.

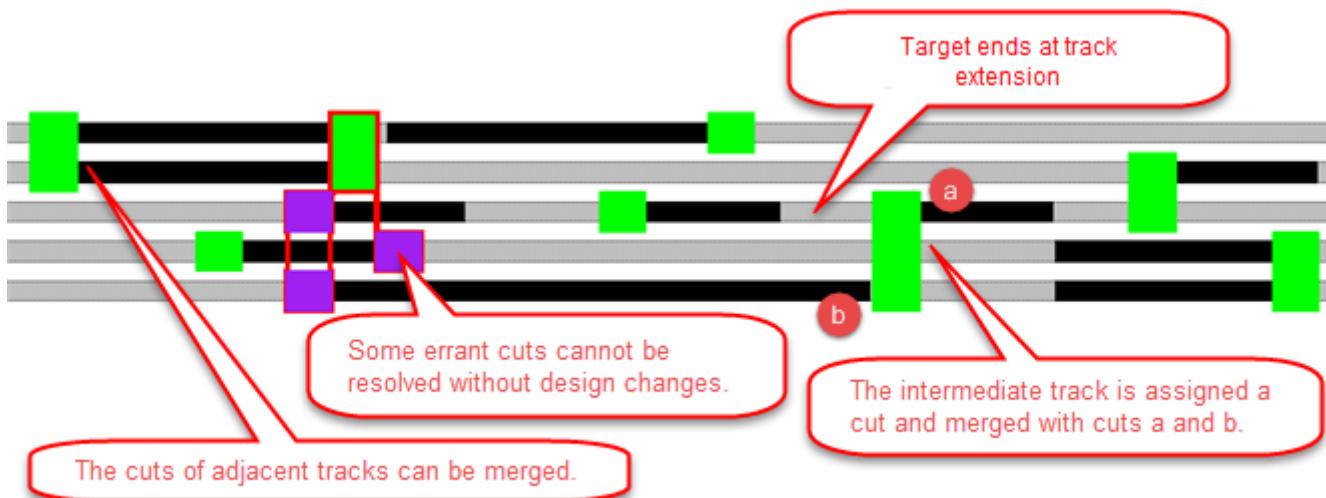
On the left side of Figure 4-11, four cuts highlighted by error layers need to be merged or separated to satisfy SADP rule constraints. However, merging or separating them is not possible given the SADP rule restrictions. The only solution is to redraw the target shapes.

Maximum Line End Extension

Ideally, a cut could be placed at every line-end. However, spacing requirements between cuts often make this impossible. The `maxLineEndExtension` parameter determines the distance a cut can be placed from a line-end and still be legal. RET SIDCUTFILL attempts to minimize the length of these extensions, but uses the flexibility supplied by `maxLineEndExtension` and places cuts that prevent short circuits, even where they are not on a line-end. There may be only one cut between two target line-ends, or there may be more. In any case, these cuts can be moved away from the line-ends in order to satisfy spacing requirements between cuts.

Several target lines in Figure 4-11 have no cuts placed at their line-ends. There are no cuts required at their line-ends as long as the design intent is not compromised and the remaining unoccupied track is terminated at a `cutMask` shape or `sadpMarker` boundary.

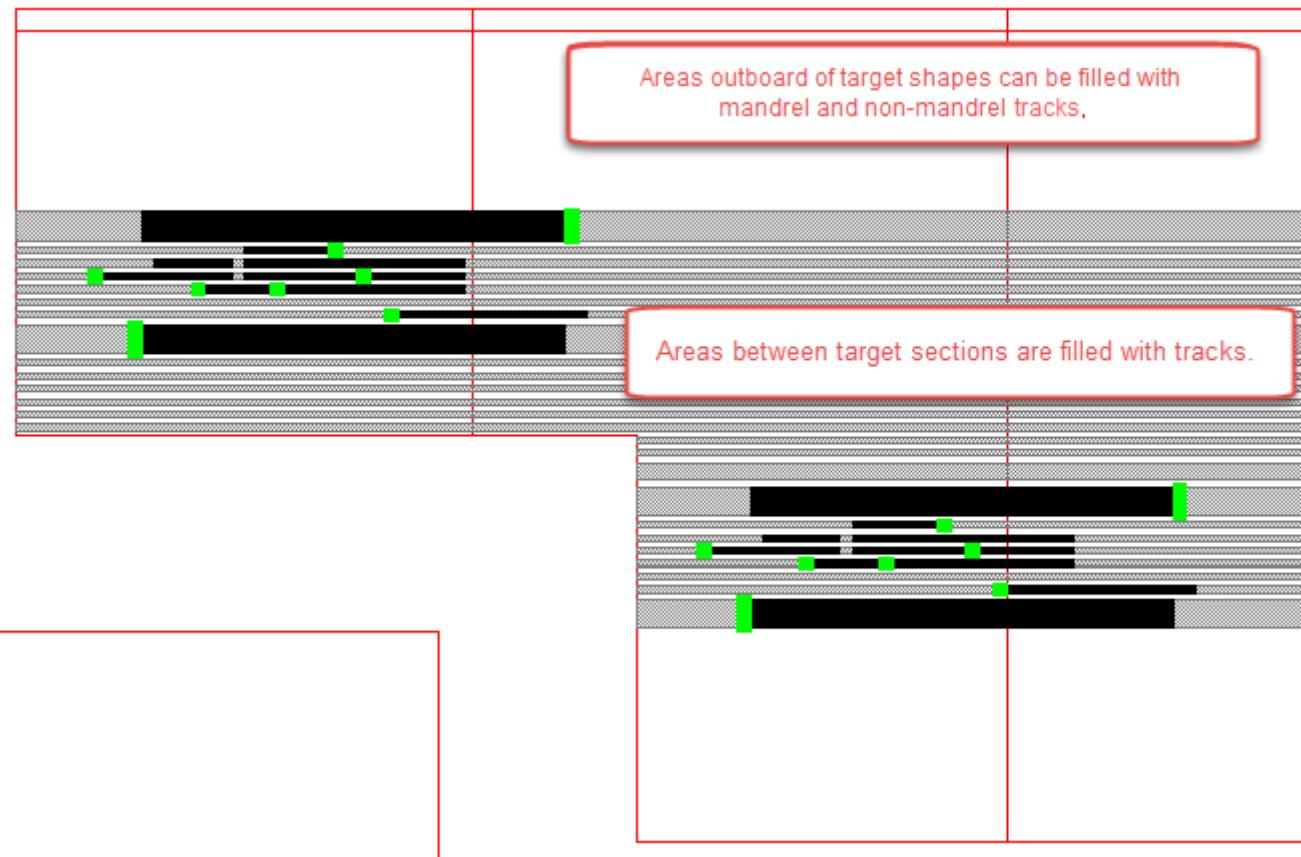
Figure 4-11. `cutMask` Shape Spacing Requirements



sadpMarker Layer Containment

The sadpMarker region can be drawn as any orthogonal polygon. Tracks will be automatically placed from the ends of the target lines to the sadpMarker boundary. As seen in [Figure 4-12](#), RET SIDCUTFILL fills the interim void between multiple sections of drawn target lines with appropriately spaced tracks, of specified widths, inside the sadpMarker region shape.

Figure 4-12. sadpMarker Containment



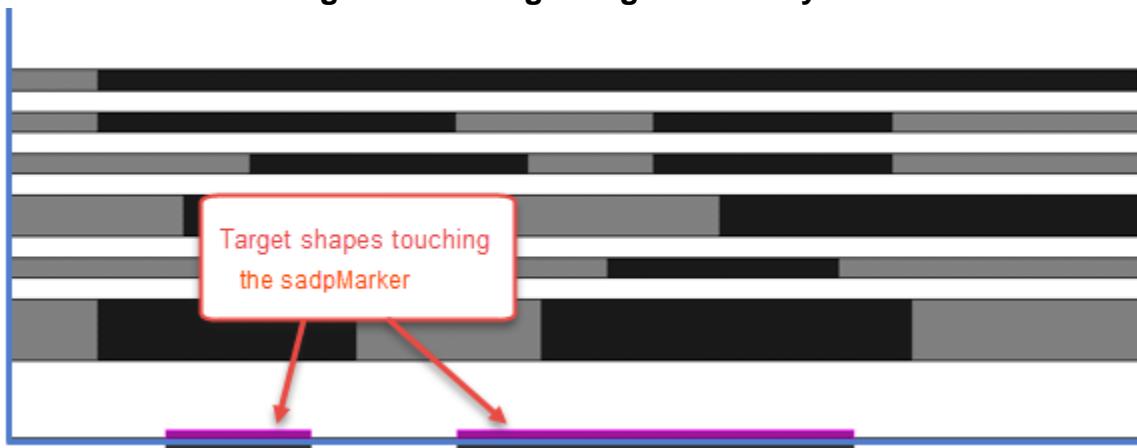
In all cases, the two outermost tracks inside the sadpMarker layer are generated equal to the terminalDummyTrack parameter, and are referred to as terminal tracks. The space between these terminal tracks and target shapes are populated with tracks equaling the minTrackWidth² parameter. RET SIDCUTFILL generates an even or an odd number of tracks to force the outer two tracks to be mandrels.

Target Shapes Touching the *sadpMarker* Layer

If any target polygons touch the sadpMarker layer, they are output to the illegalTarget error layer as shown in [Figure 4-13](#).

2. Or the smallest (first) value specified in the trackWidth list.

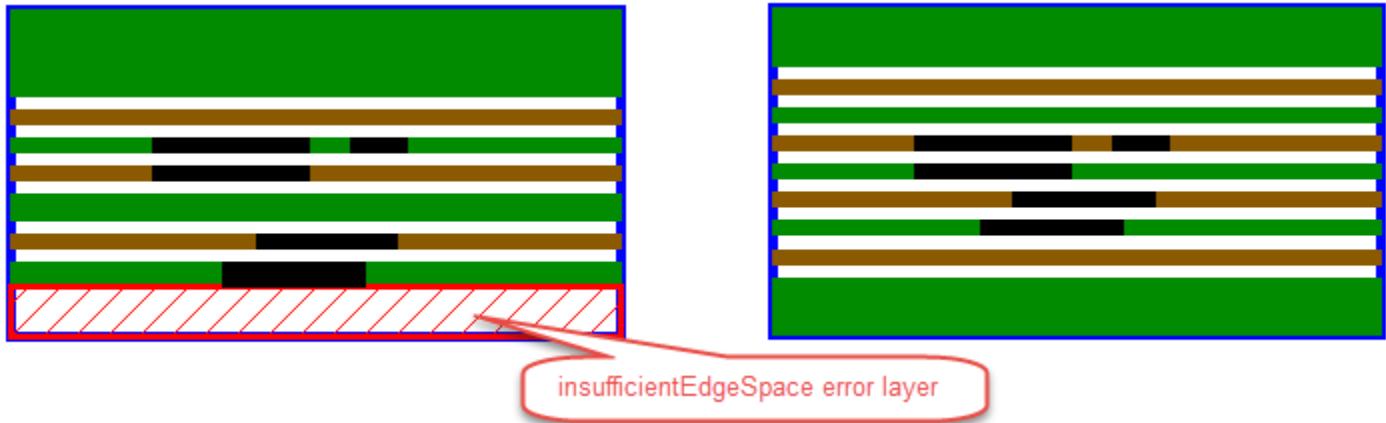
Figure 4-13. illegalTarget Error Layer



Insufficient Terminal Mandrel Space

If an insufficient amount of space remains within the sadpMarker region outboard of the drawn target lines is too small to accommodate terminal dummy tracks, that space can be highlighted with the insufficientEdgeSpace error layer as shown in [Figure 4-14](#). For example, the design on the right side has enough space to support the required terminal dummy mandrel tracks on both top and bottom. The design on the left side does not have enough space for the bottom terminal dummy mandrel track, so the insufficientEdgeSpace error layer can highlight the area in violation, as required.

Figure 4-14. Insufficient Space for Terminal Mandrels



Filler Tracks

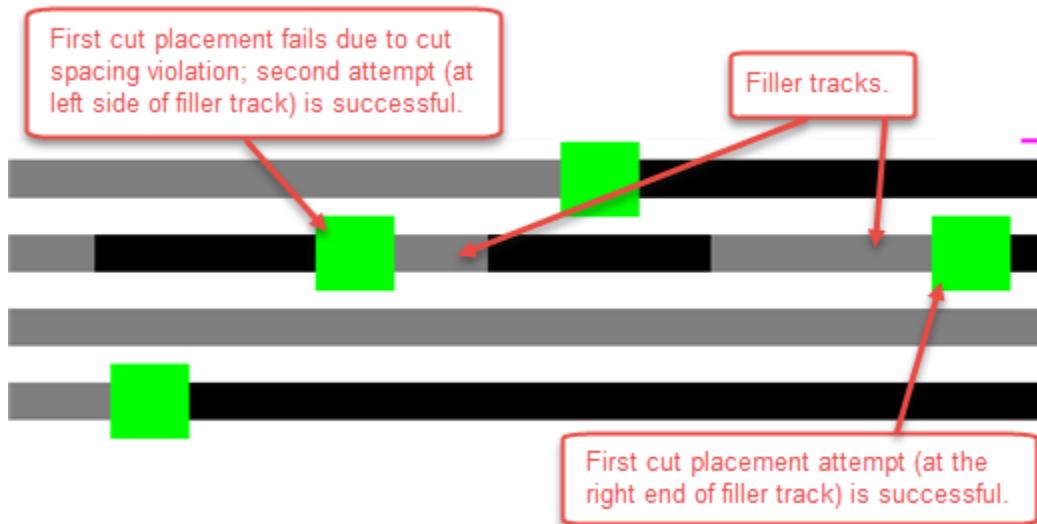
A filler track is bounded on both ends by a target line of the same width. Various examples are illustrated in [Figure 4-15](#). Normally, a cut is required at both ends of a filler track. However, there may not be enough space at each end of the filler track for a cut for these reasons:

- Placing both cuts violates the minArea parameter for the filler track remaining between the two cuts.

- The spacing requirement between the two cuts only leaves room for one.
- Spacing from one cut to another in a neighboring region causes a spacing violation.

A single cut on a filler track is legal where one cut is required to comply with the minArea parameter or if the target shape line-end extension is no longer than the maxLineEndExtension parameter.

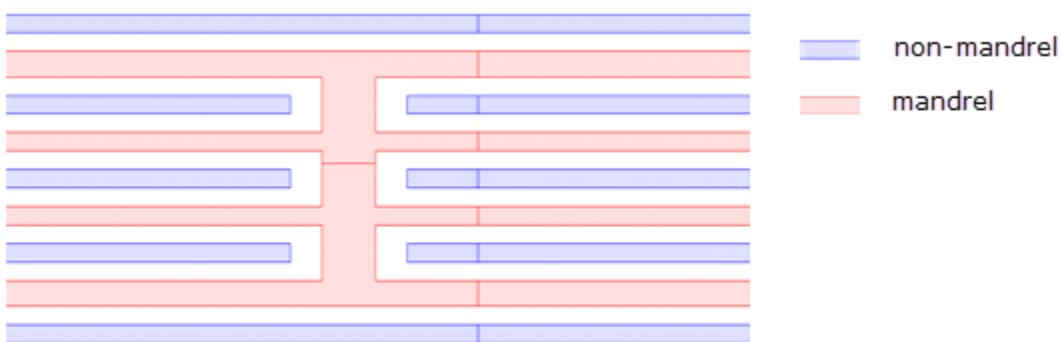
Figure 4-15. Filler Tracks



Jogs and Landing Pads

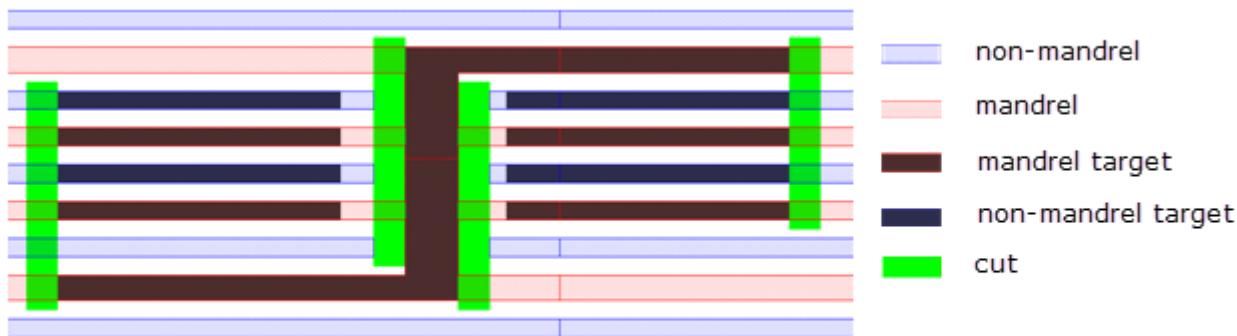
RET SIDCUTFILL handles jogs and landing pads in SADP designs. Each of these jogs or landing pads may span an even or odd number of tracks. To allow jogs or pads to span an even number of tracks, the allowEvenJogs parameter must be set to true. This is illustrated in [Figure 4-16](#) on page 232.

Figure 4-16. Odd-Track Spans



The mandrel and non-mandrel tracks are generated. When jogs traverse an odd number of tracks, the entire 2D shape is assigned to either mandrel or non-mandrel track.

Figure 4-17. 2D Target Shape Placement



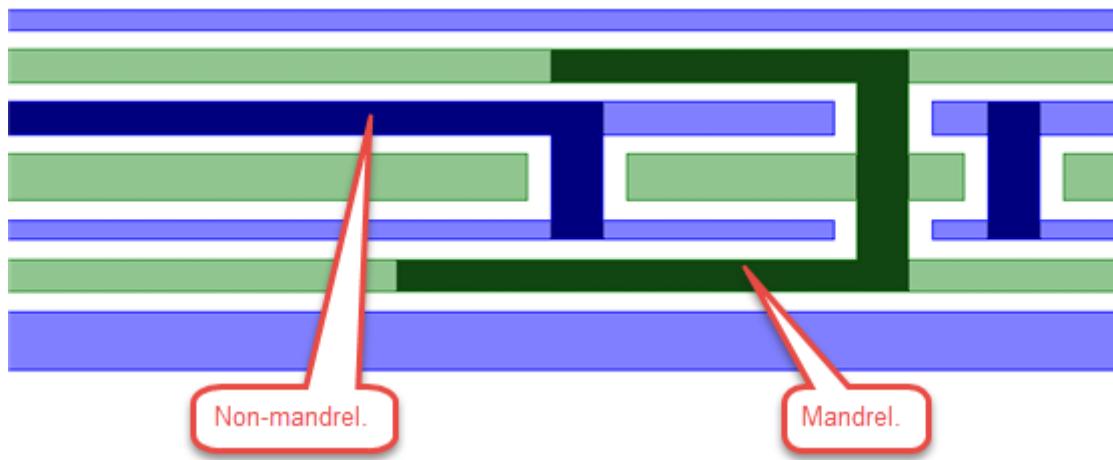
After establishing the mandrel and non-mandrel tracks, the target shapes are placed with terminations at cut shapes.

Figure 4-18. Jogs and Pads



After errors are resolved, the mandrel and non-mandrel tracks are trimmed away from the jogs and pads. The jogs and pads are assigned to masks based on their top and bottom positions. In [Figure 4-19](#), the mandrel track is shown as green and the non-mandrel track is blue. The dummy mandrel tracks are light green and the dummy non-mandrel tracks are light blue.

Figure 4-19. Final Pattern with Jogs

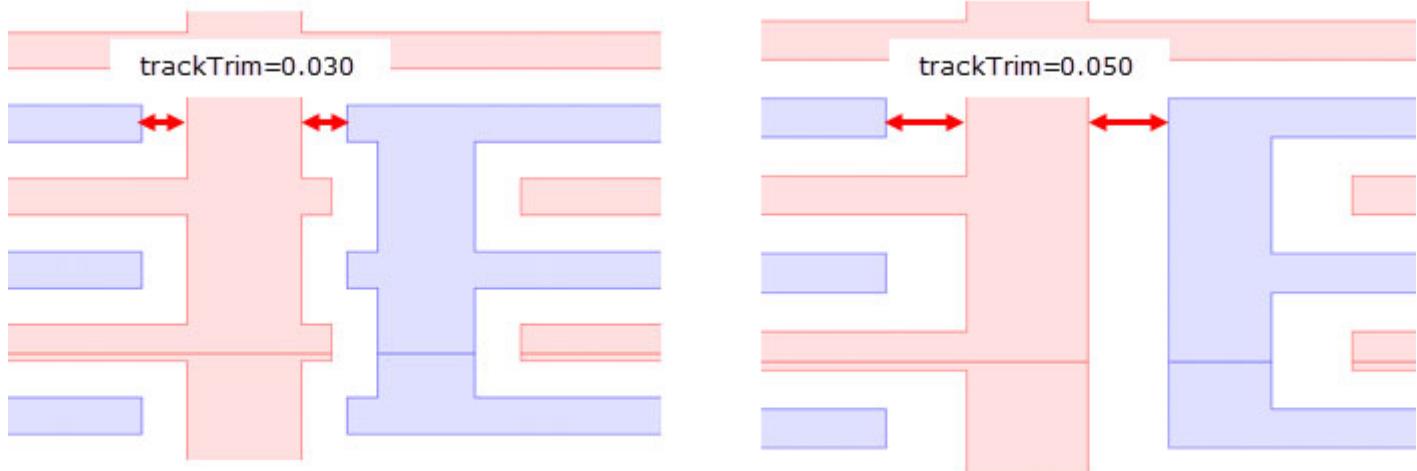


Mandrel and Non-mandrel Tracks

The generated mandrel track is the combination of the dark and light green shapes that are shown in Figure 4-19, while the non-mandrel track is the combination of the dark and light blue shapes. It is the mandrel that is imaged on the wafer as a form to grow the sidewall material.

Mandrel tracks are trimmed away from non-mandrel target shapes the distance specified by the `trackTrim` parameter, as depicted in Figure 4-20. This is required so sidewalls do not intrude into non-mandrel target areas. The same considerations regarding the mandrel tracks apply to the non-mandrel tracks.

Figure 4-20. `trackTrim` Parameter



The `trackTrim` parameter determines the distance from tracks to jogs. This illustration shows its effects on a design with `trackDirection` set to `X`.

Cuts

Cuts provide termination for correct track connectivity.

Sliding Cuts

The `slidingCuts` parameter controls potential resolution of cut placement problems. The effect of this parameter is illustrated in [Figure 4-21](#). If sliding cuts are not allowed, an illegal cut is output to the `illegalCut` layer (purple rectangle). This error could be resolved by sliding the top cut over to the right to, aligned with the lower cut and joined as a single cut, as shown in the left figure. The `slidingCuts` parameter is enabled by default.

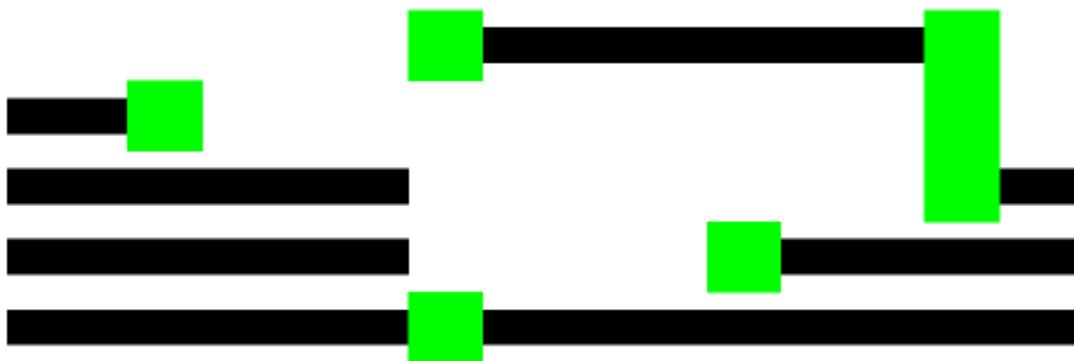
Figure 4-21. Sliding Cut Implementation



Sliding Cuts Behavior

Setting `slidingCuts` to `false` may still allow some cuts to be eliminated as shown in [Figure 4-22](#). The algorithm first places cuts on the right-hand side of gaps between targets (or the top, if tracks are oriented vertically). It then adds cuts to the left-hand side only if required to avoid violating the `maxLineEndExtension` parameter. In [Figure 4-22](#), there are two line-ends with no cuts. This is not because the cuts slid to the right; the cuts are not needed, so they are not placed.

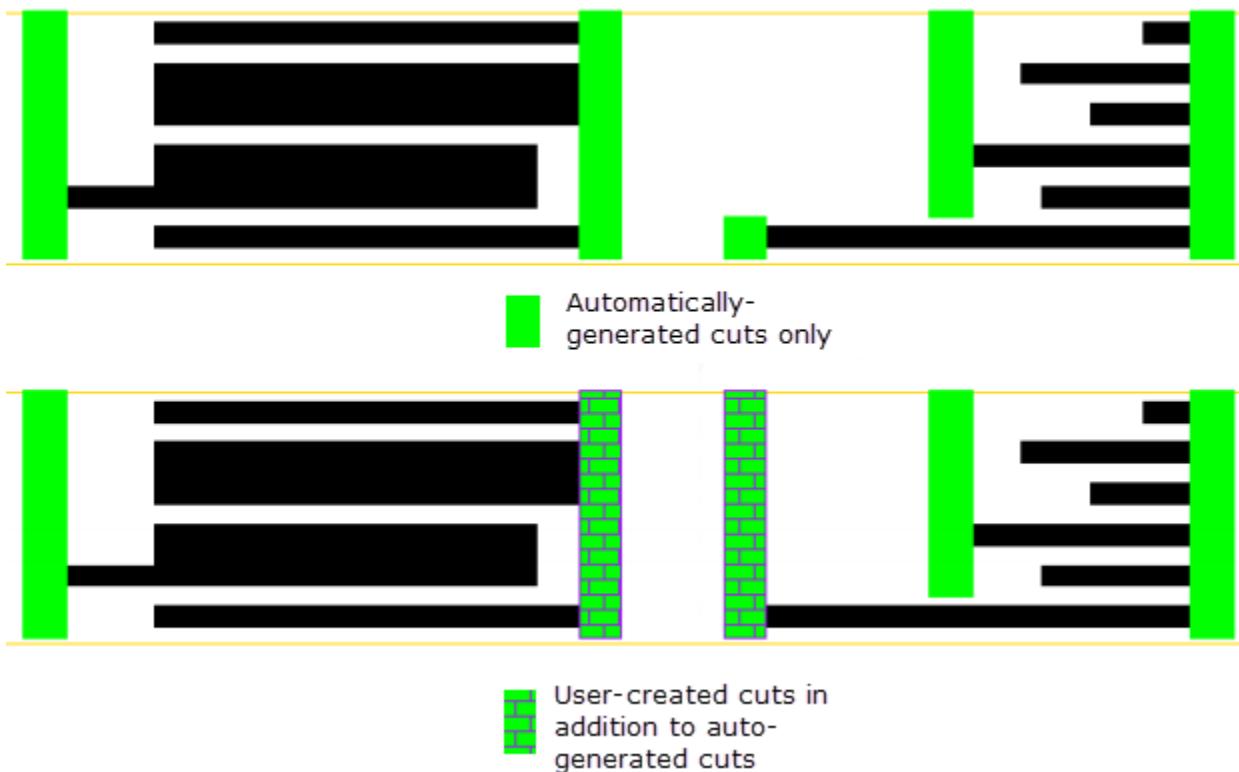
Figure 4-22. Behavior of Sliding Cuts False



User Cuts

The user cut layer permits users to specify their own cuts that are output to the `cutMask` layer and saved as valid cuts. You specify the user cut layer in the LITHO FILE statement. The effect of this parameter is illustrated in [Figure 4-23](#). User cuts may be non-rectangular, but are handled internally by their respective rectangular bounding box. For more information regarding usage of user cuts see “[LITHO FILE](#)” on page 268.

Figure 4-23. A User-Cut Implementation



Cut Keepout Layer

An optional input layer, `cutKeepout`, is used to restrict areas where cuts may be placed. In these designated areas, no cuts can be placed that intersect polygons on the `cutKeepout` layer, with specific exceptions. The `cutKeepout` implementation steps are shown here:

- The `cutKeepout` intersects the generated tracks. The intersection constrains cut placements.
- Cuts are forbidden to be placed in the keepout regions formed between logical overlap of the `cutKeepout` layer and tracks.
- Cuts are allowed to touch keepout regions.
- The `cutKeepout` layer may also overlap target polygons. This has no effect since cuts never intersect target shapes.
- If two `cutKeepout` shapes intersect a single track where cuts are legal, a generated keepout is placed over the span of the track between the two original `cutKeepout` shapes.

Logically, the effective `cutKeepout` for a given mask x is:

$cutKeepout \text{ OR } cutKeepout_x$

Dual Cut Masks

You can generate one or two cut masks. RET SIDCUTFILL is triggered to output the required number of masks by the specification of the output layers. With a few exceptions, keywords with a numeric suffix (_0, _1), are normally used with that suffix processing dual cut masks:

- **Single cut mask** — Maps out layer cutMask.
- **Dual cut masks** — Maps out layers cutMask_0 and cutMask_1.

For one mask, cutMask is the output layer containing cuts generated for both the mandrel and non-mandrel targets. For dual masks, cutMask_0 is the output layer containing cuts generated for the mandrel target; cutMask_1 is the layer containing cuts generated for the non-mandrel target. The mandrel and non-mandrel target shapes are computed and generated internally by RET SIDCUTFILL.

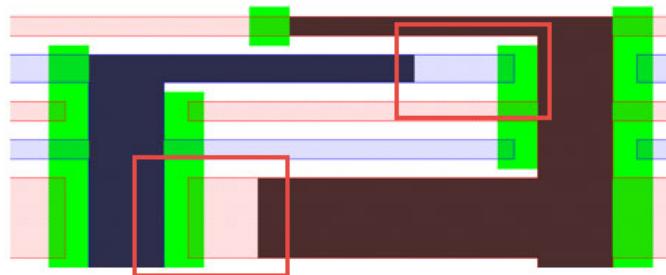
You can use the input layers mandrelAnchor and nonMandrelAnchor to control which shapes end up on each mask. If there is room inside the sadpMarker polygon, RET SIDCUTFILL assigns the outermost dummy tracks to be on the mandrel mask, and with a width specified by the terminalDummyTrackWidth parameter. Specifying dual cut masks determines which input and error layers are used. In most cases, single mask keywords and dual mask keywords are mutually exclusive. For example, specifying cutMask_0, cutMask_1, and cutMask together result in a parse error. An exception is the pitchAlign keyword, which is permitted to be used with both single and dual masks. When dual masks are specified, many keywords specified in the LITHO FILE statement must be suffixed with _0 and _1, otherwise a parse error results. Exceptions are cutKeepout, trimMaskKeepout and derivedTarget:

- **cutCandidate** — Specify input keywords cutCandidates_0 and cutCandidates_1.
- **cutPlacementError** — Specify error keywords cutPlacementError_0 and cutPlacementError_1.
- **dummyFillCutMarkers** — Specify output keywords dummyFillCutMarkers_0 and dummyFillCutMarkers_1.
- **illegalCutCandidates** — Specify error keywords illegalCutCandidates_0 and illegalCutCandidates_1.
- **illegalCuts** — Specify error keywords illegalCuts_0 and illegalCuts_1.
- **longCuts** — Specify error keywords longCuts_0 and longCuts_1.
- **trackPattern** — Specify input keywords trackPattern_0 and trackPattern_1.
- **trappedCuts** — Specify output keywords trappedCuts_0 and trappedCuts_1.
- **trimMask** — Specify output keywords trimMask_0 and trimMask_1.

Cuts for 2D Structures

As shown in Figure 4-24, cuts play a part in the trimming of tracks back from jogs. The cuts themselves also act as terminations for the target shapes (jogs) running in the non-preferred orientation.

Figure 4-24. Cuts in 2D Structures



Cuts for Automatic Short Resolution

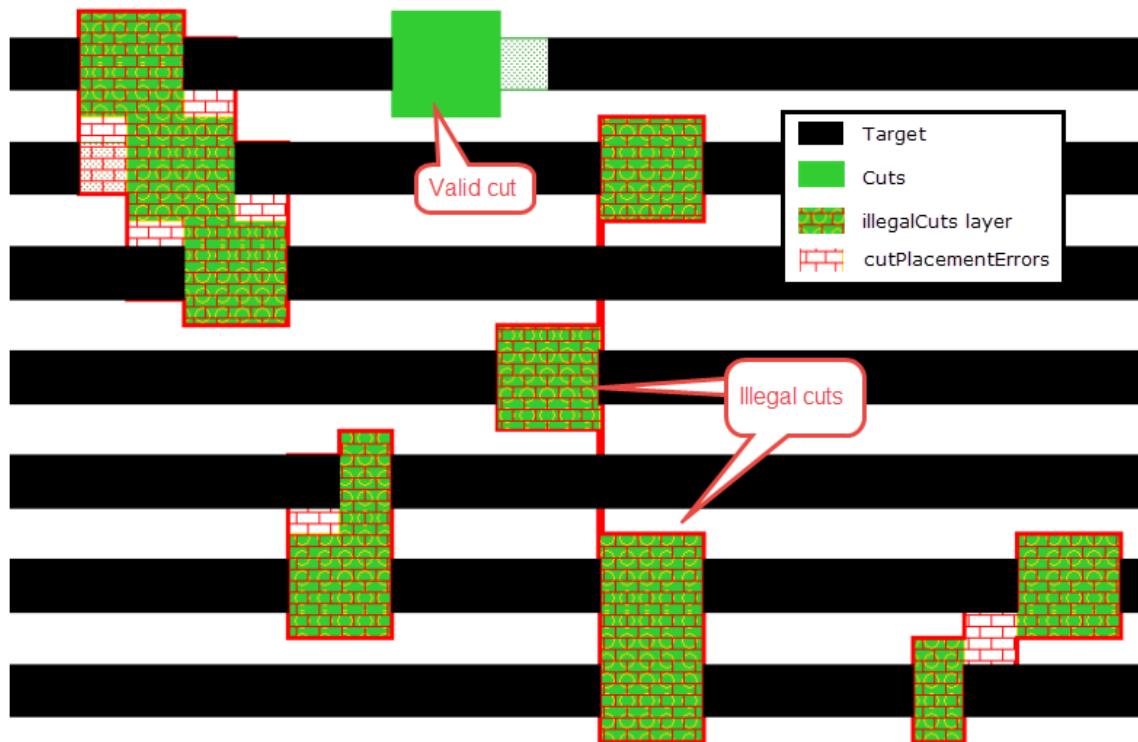
RET SIDCUTFILL generates cuts that solve short circuits, even if those cuts cause violations. If the cuts cause violations, they are placed on the illegalCuts output layer. No cuts are generated between a line-end and the edge of the sadpMarker layer because it is assumed a block mask exists outside the sadpMarker layer.

A short circuit indicates a case where two target shapes designed with independent connectivity are electrically tied together. Without the inserted cut, the two target shapes are connected; consequently, the cut is inserted regardless of legality, to avoid compromising the design network.

Automatic cut placement is not performed for the isolation of a line-end from the edge of the sadpMarker layer. Assuming the existence of large block masks at the edge of the sadpMarker layer, absent cuts would not allow short circuits.

As seen in Figure 4-25, all but one cut is illegal. The illegal cuts are output to the cutMask layer even though they were needed to resolve short circuits.

Figure 4-25. Cuts Automatically Placed to Resolve Shorts



Cuts may be considered illegal for various reasons:

- **Gap violations** — Where the gap between target shapes is smaller than the cutWidth parameter. In this case, a cut is generated anyway, but does not overlap the target, and therefore violates the required cutWidth parameter.
- **Spacing violations** — Where any spacing requirements are violated. These are indicated by shapes on the cutPlacementErrors marker.
- **Non-rectangular cuts** — Errors forced by the proximity of nearby required cuts.
- **User-cut priority** — User cuts are always placed on the cutMask even if their placement results in a violation.
- **Prohibited cut location** — Cuts intersecting a cut-keepout layer.
- **Violations of minArea** — Cuts too close to the sadpMarker layer or cuts on either end of a target shape violating minArea.

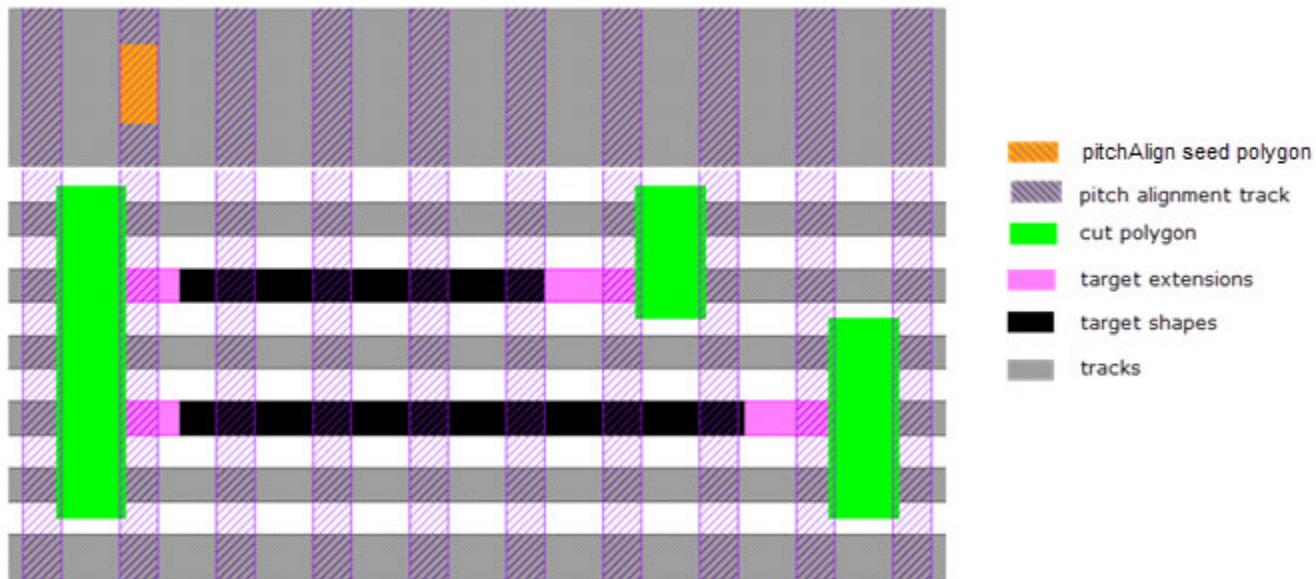
Cut Pitch Alignment

An optional input layer, pitchAlign layer, contains a polygon acting as a seed location for the generation of pitch alignment tracks. The pitch alignment capability provides for regular placement of cuts placed on predefined tracks running perpendicular to the target input shapes, instead of line-ends. The pitch alignment track pitch is established by values you provide to the

`pitchAlignTrackWidth` and `pitchAlignTrackSpacing` parameters. The generated pitch alignment tracks must run perpendicular to the target shapes.

By default, RET SIDCUTFILL places cuts as close to target line-ends as possible. In some cases, however, it may be necessary to align cuts with other input shapes, requiring the cuts to be moved from target line-ends. The `pitchAlign` input layer must consist of rectangles that define tracks oriented perpendicular to the target shapes. At least one rectangle on the `pitchAlign` layer within the `sadpMarker` layer is required as a seed to specify the location of the initial pitch align track, but typically several should be specified. Remaining tracks are generated from the initial track using the `pitchAlignTrackWidth` and `pitchAlignTrackSpacing` parameters you specified. Placement of cuts are centered over the spaces between the pitch align tracks. The example in [Figure 4-26](#) illustrates cut pitch alignment constituents.

Figure 4-26. Pitch Alignment for Cuts



You define pitch alignment for cut placement using four inputs:

- **`pitchAlign`** — A required layer name containing shapes that act as seeds for track alignment generation.
- **`pitchAlignTrackSpacing`** — A required parameter specifying the spacing between two adjacent pitch alignment tracks.
- **`pitchAlignTrackWidth`** — A required parameter specifying the width of pitch alignment tracks.
- **`pitchAlignCutOffset`** — An optional list of cut location offsets from the pitch alignment centerline established by `pitchAlignTrackSpacing`.

The pitch alignment period is defined as:

$$pitch = pitchAlignTrackSpacing + pitchAlignTrackWidth$$

In [Figure 4-26](#) there is only one polygon on the pitchAlign layer. Typically, there may be several, but only one is required because the required parameters in conjunction with this single polygon are enough to determine placement of all pitch alignment tracks.

As shown in [Figure 4-26](#), the generated cuts are always centered over the spaces between shapes on the pitchAlignTracks output layer. To guarantee centering, the difference between specified values of cutWidth and pitchAlignTrackSpacing must be an even number of database units (they may be zero or negative), which determines the overlap of the cut shapes with the pitchAlignTracks output layer. If the differences between them does not equal an even multiple of the database units, a parsing error is issued.

Polygons on the pitchAlignTrackErrors layer may cover a full polygon from the sadpMarker layer, in which case no alignment tracks are generated for that region. The pitchAlignTrackErrors layer may also include polygons smaller than one of the sadpMarker shapes, marking either an illegal pitch align track width, or an illegal pitch align track spacing.

Note

 Regarding pitch alignment and related usage:

- If the pitchAlign layer is specified, slidingCuts must be set to true, otherwise RET SIDCUTFILL issues a parse error.
 - Normally, the illegalTargetGap error layer consists of polygons filling the gaps between target shapes that are smaller than the specified cutWidth. If the pitchAlign layer is declared, then some illegalTargetGap error shapes may be longer than cutWidth. This indicates (after accounting for required pitch alignment) there is no room for the cut between two target shapes.
 - If jog and pad target shape edges are not aligned with the locations established by the pitch align parameters (typically they require cuts to be placed at their edges), these shapes are extended to the next legal edge location like normal target line-ends. Cuts are placed on these relocated jog or pad edges when possible.
-

Trim Mask

Removes mandrel and non-mandrel dummy fill.

Introduction

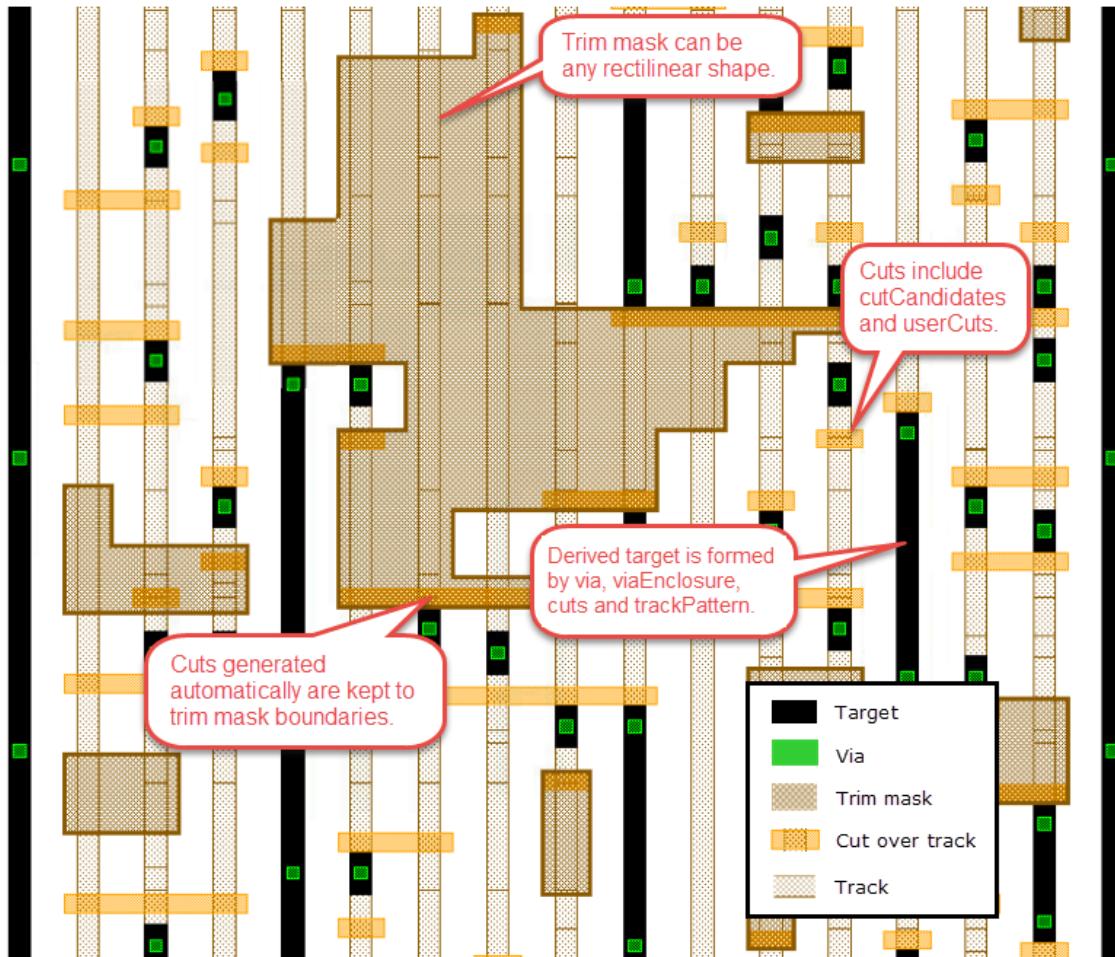
The trim mask is rectilinear, and can be any shape, adhering to its specified parameters (see [“LITHO FILE” on page 268](#)). The trim mask and its associated parameters control removal of mandrel and non-mandrel dummy fill.

Removal of passive track areas where no target shapes are required is performed with the trim mask. The trim mask is used to remove as much unneeded dummy fill as possible. Generated trim masks may enclose cut masks, or be placed outside of cut masks, adhering to enclosure or

spacing rules, respectively. RET SIDCUTFILL automatically generates cuts when one of the output layers and keywords (trimMask, trimMask_0, trimMask_1) are specified along with parameters and input layers to define legal results.

Figure 4-27 shows an example of the trim mask with various layers output after processing.

Figure 4-27. Trim Mask Implementation



Trim Mask Spacing Rules

The trim mask requires spacing rules. Spacing requirements between ordinary cuts and the trim mask is also required. This is managed by adding a new qualifier to the dpFaceClass specification called the layerType. Its use is illustrated in the following spacing rules:

```
dpFaceClass name(cut_side) orientation(vertical): true otherwise;  
dpFaceClass name(cut_end) orientation(horizontal): true otherwise;  
dpFaceClass name(trimmask_side) layerType(trimMask)  
    orientation(vertical): true otherwise;  
dpFaceClass name(trimmask_end) layerType(trimMask)  
    orientation(horizontal): true otherwise;
```

```

dpSpacing (trimmask_side, trimmask_side) metric(opposite):
    $trimmaskToTrimmaskSide otherwise;
dpSpacing (trimmask_end, trimmask_end):
    $trimmaskToTrimmaskEnd otherwise;
dpSpacing(trimmask_side, cut_side) metric(opposite):
    $trimmaskToCutSide otherwise;
dpSpacing(trimmask_end, cut_end):
    $trimmaskToCutEnd otherwise;

dpSpacing (cut_side, cut_side) metric(euclidean):
    0.046 otherwise;
dpSpacing (cut_side, cut_side) metric(opposite):
    0.082 otherwise;
dpSpacing (cut_end, cut_end) metric(opposite):
    0.110 otherwise;

```

In this example, the first two dpFaceClass definitions do not include a layerType; they apply to standard rectangular cuts, not to the trim mask. The second two dpFaceClass definitions specify layerType(trimMask); they apply only to the trim mask.

Each dpSpacing definition specifies a pair of dpFaceClass names (one or both of which may actually be a named dpFaceSet.) For example if the pair is (cut_end, cut_end) then the dpSpacing only applies between ordinary cuts, while if the pair is (trimmask_side, trimmask_side) it applies only between trim mask shapes. In contrast, if the pair is, for example, (trimmask_side, cut_side) then the spacing rule only applies between ordinary cuts and trim mask shapes.

Note

 Regarding dpFaceClass, dpSpacing, and dpFaceSet interactions:

- A dpFaceSet is a named collection of dpFaceClass definitions. Although it is legal to collect cut_side and cut_end into a dpFaceSet, it is not legal to put cut_side and trimmask_side into the same dpFaceSet because they have different layerType specifications. This results in an parse time error message.
 - Two generalized keywords may be used instead of a specific dpFaceClass name in a dpSpacing statement. The more general one is *any*. It references any face class or edge. The other is *side* which references any polygon face that was not previously classified. Neither of these generalized names should be used where multiple layerTypes are present, to avoid unpredictable results.
-

Parameter Restrictions Related to Trim Mask and Cut Mode

Some parameter values are restricted when the trim mask is specified depending on the cut mode specified:

- In single cut-mask mode when trimMask is specified, an error message is output if:

`2*cutOverlap > trackSpacing`

- In two cut-mask mode when trimMask is specified, an error message is output if:

```
2*cutOverlap >= 2*trackSpacing + minTrackWidth
```

These parameters and their usage is detailed in “[LITHO FILE](#)” on page 268.

Trim Mask Parameters

The parameters directly affecting generation of the trim mask is illustrated here:

```
cutOverlap = 0.020;
trimMaskMinJog = 0.030;
trimMaskMinWidth = 0.040;
trimMaskMinRectWidth = 0.031;
viaEnclosure = 0.025;
```

The parameters and their respective usage is detailed in “[LITHO FILE](#)” on page 268.

Derived Target

Provides patterns for target derivation.

Introduction

Typically, the target layer is provided to RET SIDCUTFILL. In processes where the target layer is not available, RET SIDCUTFILL, in conjunction with other input layers, can derive the target layer. This is the only case where the target layer does not have to be specified. In no case, however, can the target layer be specified and the target layer be derived in a single recipe.

Input Layers to Facilitate Derived Target Generation

If the target input layer is not specified, RET SIDCUTFILL can derive the target layer. Because there is no target input layer to determine the location of tracks, for single masks, the track pattern can be specified with trackPattern, or trackPattern_0 and trackPattern_1. Some cut shapes must be placed to break electrical connectivity where required, and determining where the target must be absent in each track. Cut locations can be specified using various cut and cut candidate layers. Via locations can be specified with the via layer. For an example of an outputted derivedTarget layer, see [Figure 4-27](#) on page 242. These input layers are used to derive the target layer:

- **via** — Input layer for the via locations.
- **trackPattern** — For single masks, the track pattern must be specified with trackPattern.
- **trackPattern_0, trackPattern_1** — For dual masks, these layers must determine the mandrel and non-mandrel tracks, respectively.
- **userCuts, cutCandidates** — For single cut masks, either or both of these layers must be used. An internal OR of userCuts and cutCandidates layers is performed for the application of the cut layer.

- **userCuts_0, userCuts_1, cutCandidates_0, cutCandidates_1** — For dual cut masks, any combination of these layers must be used. The userCuts_0 and cutCandidates_0 layers are applied to the mandrel track, and userCuts_1 and cutCandidates_1 layers are applied to the non-mandrel tracks. An internal OR of all of these layers is performed for the application of the cut layer.
- **viaEnclosure** — An optional, positive parameter defining the distance of enclosure the target must overlap each via in the direction of track orientation. The viaEnclosure parameter extends target line-ends to obtain the required enclosure. In addition, if the parameter trimTargetToMinViaEnclosure is equal to true, the target line-ends that extend past the required enclosure distance are trimmed back to match the specified enclosure. If the target line-end extensions result in gaps too small for cuts to be inserted, those areas are reported in the illegalTargetGap output layer (see “[Error Layers](#)” on page 245). However, the target line-ends are still extended.

If the cutMask output keyword is specified, userCuts appear on their respective output layers unmodified. The cutCandidates input layers are not modified for the purposes of generating the derivedTarget. However, after generating the derivedTarget, they may be modified if any cutMask output is also specified.

Output Layers for the Derived Target

The output MAP keywords are optionally specified to output the derived target layer:

- derivedTarget for single mask processes.
- derivedTarget_0 for the mandrel mask.
- derivedTarget_1 for the non-mandrel mask.

Error Handling

Various error layers help resolve SADP patterning problems quickly.

This section covers error outputs and debugging aids.

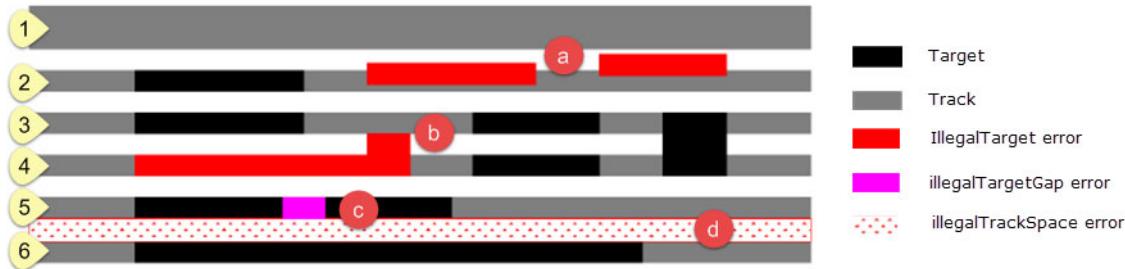
Error Layers

The example in [Figure 4-28](#) illustrates a number of common error layer types generated by Calibre RET SIDCUTFILL:

- **illegalTarget** — Polygons on the original target layer that do not conform to the tracks computed by Calibre RET SIDCUTFILL. There are two conditions that cause a target polygon to be declared illegal:
 - A horizontal target edge not coincident with any horizontal track edge (a).
 - A horizontal target edge abutting a horizontal track edge, and not intersecting the track at that location (b).

- **illegalTargetGap** — A gap between two target shapes that is less than cutWidth (c). The illegalTargetGap error output layer includes illegal gaps enforced by pitch alignment rules, where gaps are less than or equal to the cut width, or narrowed by target line-end extensions. If a pitchAlign layer or maskGridSpacing is specified, resulting line-end extensions may result in additional illegalTargetGaps. Cuts will still be placed, but are marked as illegal.
- **illegalTrackSpace** — Where the space between tracks is less than trackSpacing or less than $(2 * \text{trackSpacing}) + \text{minTrackWidth}$ (d).
- **illegalTrackWidth** — The width of a track as seeded by a target shape that does not match any width value specified in the trackWidth list, or a track of a width that is less than the value specified for the minTrackWidth parameter.

Figure 4-28. Target- and Track-Related Spacing Errors

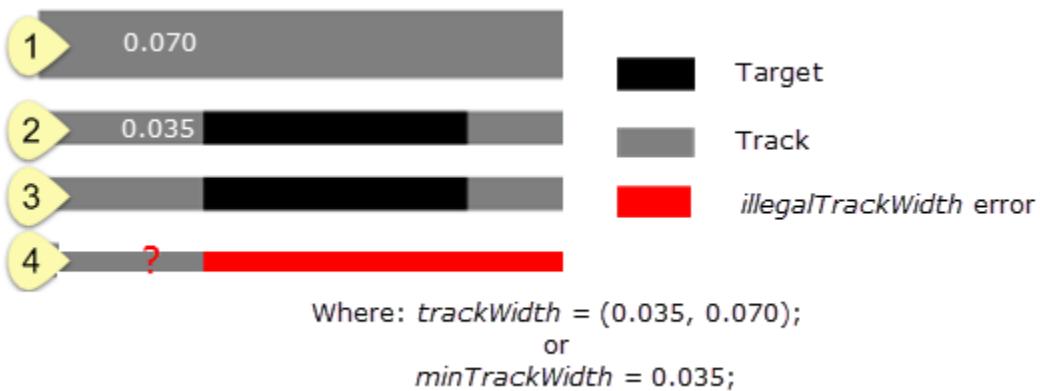


In Figure 4-28, track 1 is wider than minTrackWidth because that track was required to fill the area above the regular tracks. It contains no target shapes. Track 2 intersects three target shapes, two of which are off-track (a). All three target shapes intersecting that track have the same height. RET SIDCUTFILL finds all vertical target edges with two convex corners. If two of these vertical edges project onto each other, the smaller one is kept and the larger one is ignored. In Figure 4-28, there are three vertical edges all the same height that project on each other. RET SIDCUTFILL ignores all three of these polygons for the purposes of computing tracks. The left-most of these three polygons becomes part of the legal target because it aligns with a track created by RET SIDCUTFILL which, at this stage of the processing, was considered as a dummy track.

The illegalTarget polygon in track 4 is not legal because part of the target projects upward into the space between the tracks, but does not fully intersect track 3.

In Figure 4-29, track 4 is generated from a target shape that does not conform to any value specified in the trackWidth list, or is less than minTrackWidth. This entire track and target is saved to the illegalTrackWidth error layer.

Figure 4-29. illegalTrackWidth Error



Illegal target shapes are saved to the illegalTarget and illegalTrackWidth error layers only after RET SIDCUTFILL computes all tracks.

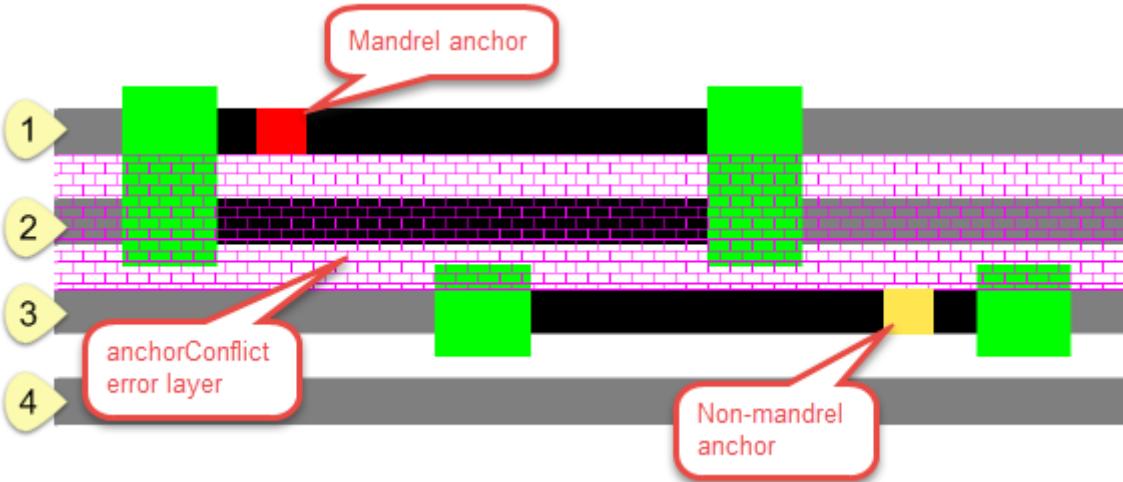
Jogs and Pads

Jogs and pads are identified by RET SIDCUTFILL after track computation is complete. Jogs and pads are not used to determine track locations or to determine which tracks are occupied.

Anchor Conflicts

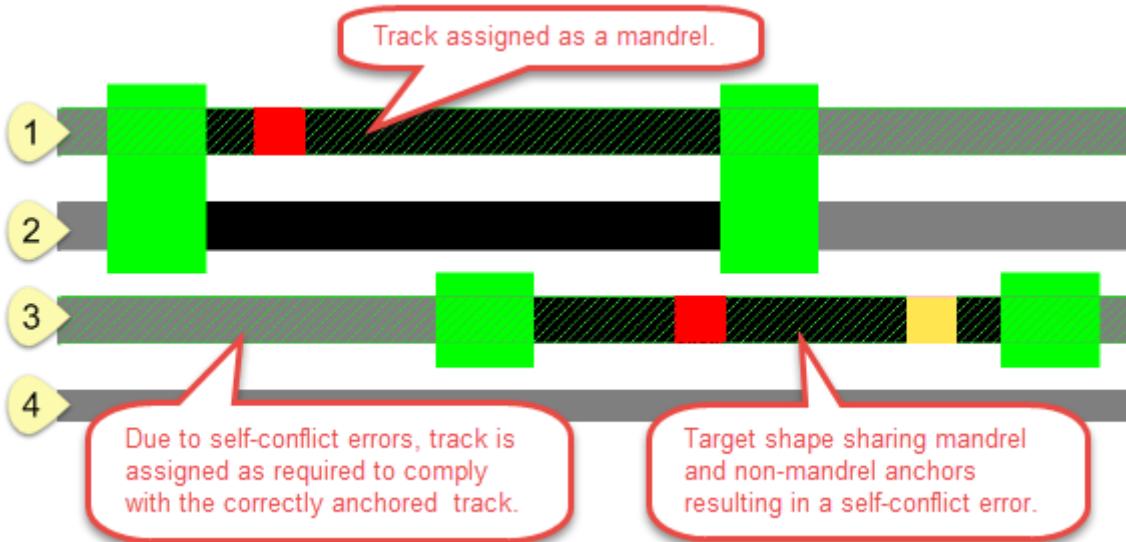
Both mandrel_anchor and nonmandrel_anchor input layers may be used by RET SIDCUTFILL to assign certain target shapes to the mandrel and non_mandrel layers, respectively. The example in [Figure 4-30](#) illustrates a coloring conflict caused by anchors that are poorly placed. The mandrel and non-mandrel anchors force a coloring conflict between tracks 1 and 3. The resulting anchor conflict is output as a shape on the anchorConflict error layer (shown as a purple brick stipple). It spans the entire area between the tracks in conflict. The cuts and all other processing are still performed, despite the event of the anchor conflict.

Figure 4-30. Poorly Placed Anchors Result in Anchor Conflict Error



RET SIDCUTFILL also checks for target shapes containing both mandrel and non-mandrel anchors, referred to as self-conflict errors. Figure 4-31 illustrates a mandrel anchor sharing the same target shape with a non-mandrel anchor on track 3. This target shape is output to the anchorSelfConflict output layer (not shown) and both anchors are subsequently ignored. The correctly placed mandrel anchor on the target shape of track 1 forces the assignment of that track to be a mandrel (shown as diagonal green stipple), which also forces track 3 to be assigned to the mandrel mask to comply with the design coloring requirements. It may appear the mandrel anchor on track 3 is being enforced, but actually both the anchors on track 3 are ignored due to the self-conflict error.

Figure 4-31. Self-Conflict Anchor Error and Resulting Track Assignment



In addition, if both mandrel and non-mandrel anchors are found anywhere on the same track (even when not sharing the same target shape), that track is output to the anchorTrackConflict error layer and all anchors on that track are subsequently ignored.

Named Spacing Errors

The dpSpacing and dpFaceClass commands allow users to specify spacing requirements between generated cuts. Optionally, a name may be assigned to each dpSpacing command. This allows greater clarity when using debug output layers to establish why two cuts generate spacing errors.

In addition to determining standard, single-direction spacing rules, dpSpacing can optionally detect violations with multiple neighboring cuts for additional rule stringency.

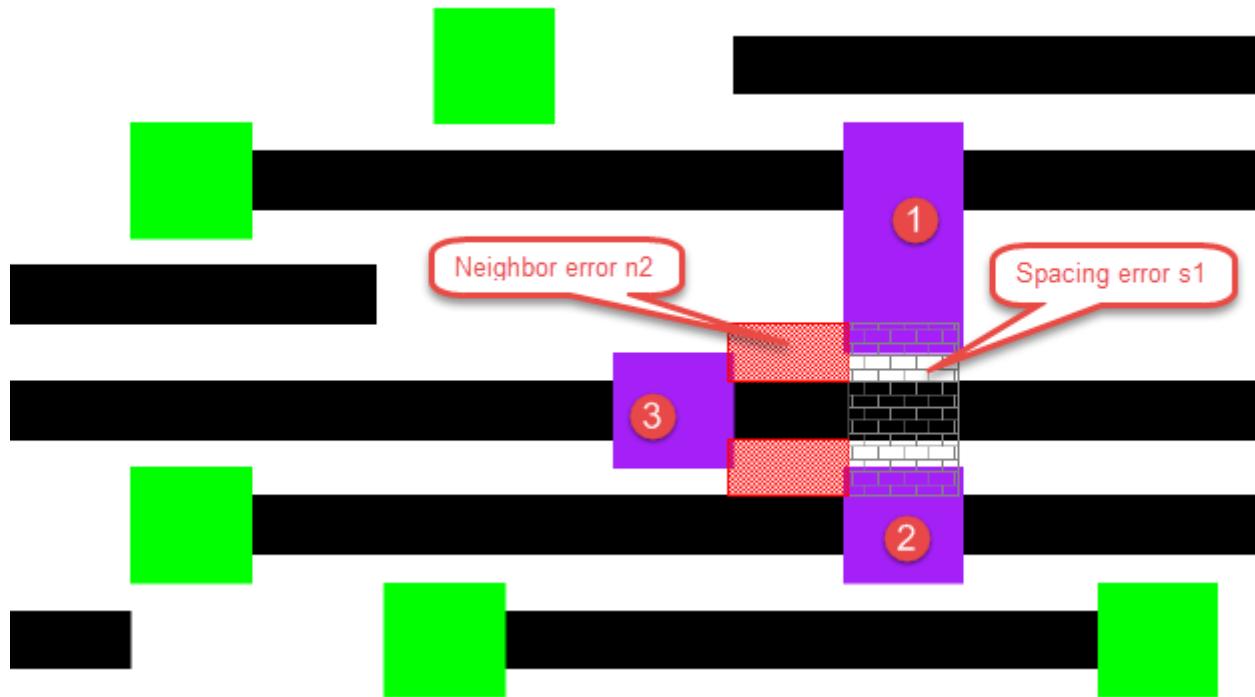
The code example represents the results shown in Figure 4-32. A spacing error is output to error layer s1 between cuts 1 and 2. Cut 3 is in violation to both neighboring cuts 1 and 2 and then output to error layer n2.

```

dpSpacing (any, basic_cut) name(s1); $mxCut_s1 otherwise;
dpSpacing (any, verticalCutFace) name(n2);
metric(opposite, extended 0.001) condition(numNeighbors 2);
$mxCut_n2 otherwise;

```

Figure 4-32. Spacing Errors

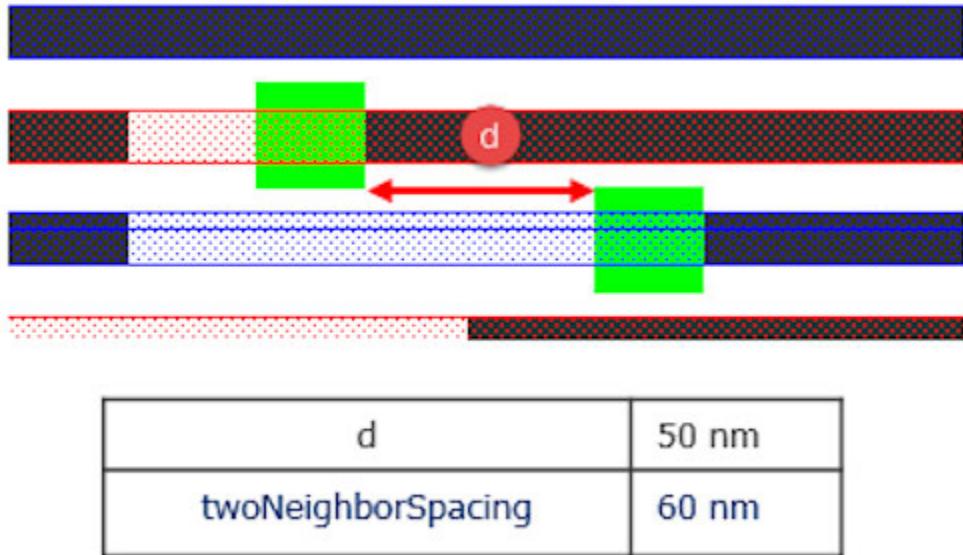


Neighbor Spacing

Neighbor spacing is a convenient way of determining potential resolution for invalid cut proximities. Neighbor spacing errors are applied only where two or more cuts are within spacing constraints. Neighbor spacing values can be specified from 2 to 4 through the use of `dpSpacing numNeighbors` parameter (see “[dpSpacing](#)” on page 294 for more information).

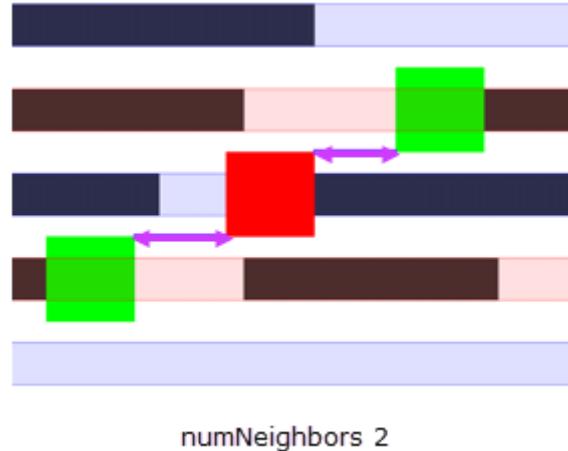
In [Figure 4-33](#), the required spacing constraint is set to 60 nm, but the actual spacing between the two cuts is 50 nm, resulting in a violation.

Figure 4-33. Two-Neighbor Spacing Criteria



In [Figure 4-34](#), the numNeighbors rule finds cuts that have more than one neighboring cut and performs a spacing check of 60 nm. The cut highlighted in red is closer than the constraint and is in violation.

Figure 4-34. Two-Neighbor Spacing Errors



[Figure 4-35](#) and [Figure 4-36](#) demonstrate three neighboring cuts, where numNeighbors is 3. In Figure 4-35, The required inter-track spacing (d) between two cuts is 50 nm and adjacent-track spacing (L) is 60 nm. In Figure 4-36 there is a cluster of cuts exceeding three cuts. The constraint of 70 nm applies these cuts, resulting in four cuts output to the illegalCuts error layer.

Figure 4-35. Three-Neighbor Spacing Criteria

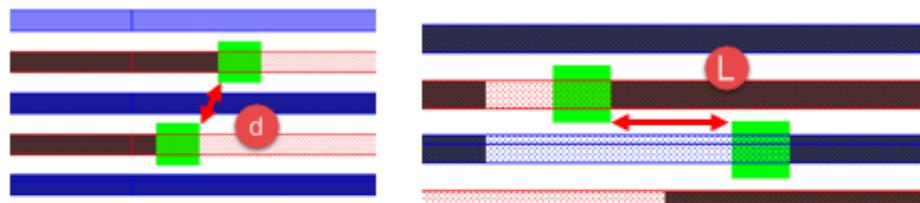
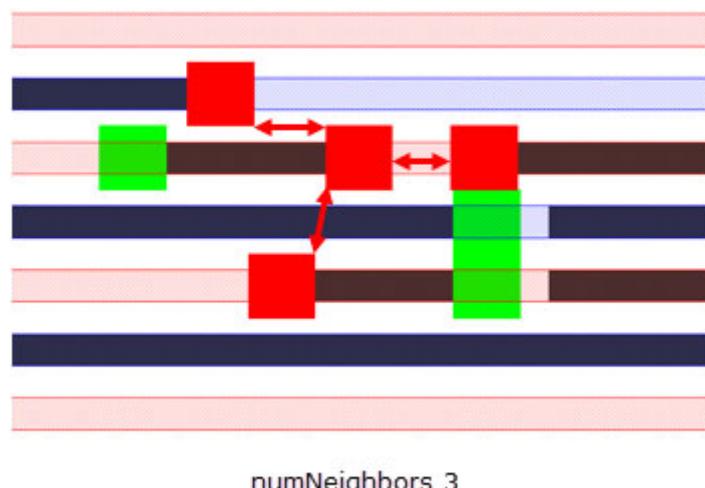


Figure 4-36. Three-Neighbor Spacing Errors



Operational Modes of Control

Operational modes enable control over cut placement, cut length, and track generation.

Cut Length Control

The `maxCutLength` parameter controls the lengths of cuts. Use the application methods in [Table 4-1](#) as a quick reference.

Table 4-1. Cut Length Control Reference

Desired Cut Length Behavior	<code>maxCutLength</code> Setting
Allow merging of cuts with no limit to their length.	unspecified (default) or <code>maxCutLength = unlimited</code>
Generate cuts less than a specified value.	<code>maxCutLength = <positive value></code>

Table 4-1. Cut Length Control Reference (cont.)

Desired Cut Length Behavior	maxCutLength Setting
Generate single cuts only.	maxCutLength = 0 or maxCutLength = doNotMerge or <positive value < cutLength*2>

For a comprehensive description of all these parameters, see “[LITHO FILE](#)” on page 268. For more behavioral information see “[Cuts](#)” on page 235.

Cut Placement Control

The slidingCuts and maxLineEndExtension parameters control cut sliding and cut placement tolerance, respectively. Use the application methods in [Table 4-2](#) as a quick reference.

Table 4-2. Cut Placement Control Reference

Desired Cut Placement Behavior	slidingCuts Setting	maxLineEndExtension Setting
Disable cut sliding and cut dropping.	slidingCuts = false	maxLineEndExtension = 0
Disable cut sliding and enable cut dropping.	slidingCuts = false	maxLineEndExtension >= (cutWidth + minimum-same-track spacing*)
Enable cut sliding and cut dropping.	unspecified (default) or slidingCuts = true	maxLineEndExtension >= (cutWidth + minimum-same-track spacing*)

*Where minimum-same-track spacing is the required spacing between cuts on the same track or minArea constraint.

For a comprehensive description of all these parameters, including cut dropping, see “[LITHO FILE](#)” on page 268. For more behavioral information see “[Cuts](#)” on page 235.

Track Generation Control

Several parameters control the generation of tracks. Use the application methods in [Table 4-3](#) as a quick reference.

Table 4-3. Track Generation Control Reference

Desired Track Generation Behavior	Controlling Parameter	Optional Control Parameters
Generate tracks of any width greater than or equal to the minimum track width.	minTrackWidth = <positive value>	<ul style="list-style-type: none"> If terminalDummyTrackWidth is specified, terminalDummyTracks are generated with a width greater than or equal to terminalDummyTrackWidth. If terminalDummyTrackWidth is not specified, terminalDummyTracks are generated with a width equal to minTrackWidth.
Generate tracks of widths equal to a series of track widths.	trackWidth = <vector of increasing track width values>	The terminalDummyTrack width is equal to the largest value of the trackWidth vector.
Generate dummy tracks between target shapes with widths in specific order.	trackWidthSequence = <vector of track widths ordered between target shapes>	<ul style="list-style-type: none"> Used by both minTrackWidth and trackWidth. Several trackWidthSequence parameters may be specified. trackWidthSequence values need not match minTrackWidth or trackWidth.
Generate tracks according to track pattern polygon locations and widths.	trackPattern <input layer>	None.

For a comprehensive description of all these parameters, see “[LITHO FILE](#)” on page 268. For more behavioral information see “[Error Handling](#)” on page 245.

Self-Aligned Double-Patterning Command Reference

The Calibre RET SIDCUTFILL command and associated keywords are an implementation of the spacer-is-dielectric (SID) process for Self-Aligned Double-Patterning (SADP). These commands address topological generation requirements for the SID process. Calibre SADP-specific commands adhere to SVRF standards. A typical SVRF rule file containing SADP-specific commands may also contain other SVRF commands that specify input layers, perform preliminary layer manipulation, and save output layers.

RET SIDCUTFILL	255
LITHO FILE	268
SADP-Specific Spacing Parameters.....	288

RET SIDCUTFILL

Self-Aligned Double-Patterning Command Reference

Calibre RET SIDCUTFILL builds a correct-by-design topology of tracks and cuts, adhering to the spacer-is-dielectric process, and complies to all specified SADP-based design rules. Calibre SADP-specific commands adhere to SVRF standards. A typical SVRF rule file containing SADP-specific commands may also contain other SVRF commands that specify input layers, perform preliminary layer manipulation, and save output layers.

Usage

Note

 The input layers specified to RET SIDCUTFILL are user-defined and can be any name, matching those previously specified within the SVRF file. Each input layer name shown here has a corresponding keyword that must be specified in the inputLayerOrder list in the LITHO FILE statement.

```
output_layer = RET SIDCUTFILL
    sadpMarker
    [target]
    [criticalTimingMarkers]
    [cutCandidates] [cutCandidates_0] [cutCandidates_1]
    [cutKeepout] [cutKeepout_0] [cutKeepout_1]
    [mandrelAnchor]
    [nonMandrelAnchor]
    [pitchAlign]
    [sinuousCutRegion] [sinuousCutRegion_0] [sinuousCutRegion_1]
    [terminationRegion]
    [trackPattern] [trackPattern_0] [trackPattern_1]
    [trimMaskKeepout] [trimMaskKeepout_0] [trimMaskKeepout_1]
    [unmovableCutCandidates] [unmovableCutCandidates_0] [unmovableCutCandidates_1]
    [userCuts] [userCuts_0] [userCuts_1]
    [via] [via_a] [via_b] [via_c]
FILE name
MAP keyword
```

Description

The RET SIDCUTFILL command generates a correct-by-design topology of tracks implied by target shapes, cuts for all target shapes, and outputs errors to assist with debugging where the intended generation of these layers fails. RET SIDCUTFILL also generates alternating mandrel and non-mandrel layers as required. All tracks and cuts adhere to the spacer-is-dielectric (SID) process, and complies with all manufacturer-specified SADP design rules.

Arguments

Output Layer

- *output_layer*

A required polygonal output layer containing generated shapes relevant to all mask-related cut, trim, and mandrel shapes to be printed on the wafer.

RET SIDCUTFILL can output any track, rectangular cut, or error layer. The output is always contained within the boundaries of each polygon on the sadpMarker layer.

Input Layers

- *sadpMarker*

A required input polygonal layer encompassing a region of target shapes that RET SIDCUTFILL processes. No shapes outside this layer are processed. You can define multiple sadpMarker shapes in a single design. The corresponding inputLayerOrder keyword is sadpMarker.

- *target*

A required input polygonal layer comprising patterns to be printed on the wafer. The target layer typically has a logical description:

```
preliminaryTarget = OR target mandrelAnchor nonMandrelAnchor
```

RET SIDCUTFILL requires these shapes to be rectangles (plus possible jogs between tracks and pads that span an odd number of tracks), and all rectangles on the same track (aligned with X- or Y-axis, depending on the specified trackDirection orientation) must be the same width. This is enforced within, and not across, the boundaries of each polygon on the sadpMarker layer. The corresponding inputLayerOrder keyword is target.

Note

 The target layer is not required if RET SIDCUTFILL is used to derive the target layer. In this case, trackPattern, or trackPattern_0 and trackPattern_1 layers must be specified.

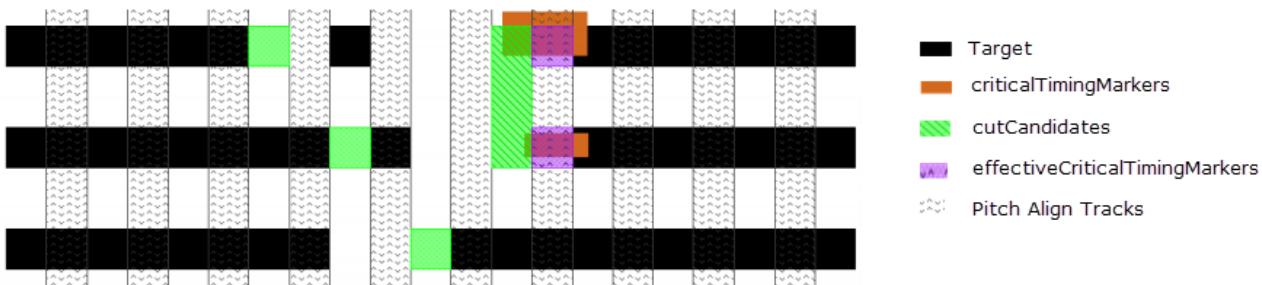
- *criticalTimingMarkers*

An optional input polygon layer used to identify nets considered critical for cut placement. Shapes on the criticalTimingMarkers layer must intersect with the associated critical net. The post-processed output layer, effectiveCriticalTimingMarkers, is subsequently generated internally for final application of the user-input markers. Cuts are placed at the ends of each effectiveCriticalTimingMarkers polygon where possible. No initial cuts are required. Where cuts are placed precisely at line ends sharing a criticalTimingMarkers shape, only a sliver at the line-end remains for the shape on the effectiveCriticalTimingMarkers layer.

RET SIDCUTFILL minimizes the line-end extension for all target shapes. However, those nets identified with shapes on the criticalTimingMarkers layer are given priority. This prioritization provides line-end extensions on these target segments to be minimized, potentially at the expense of longer line-end extensions of target segments that are not

identified as critical. Shapes on the criticalTimingMarkers layer must be a subset of the target layer. The corresponding inputLayerOrder keyword is criticalTimingMarkers.

Figure 4-37. criticalTimingMarkers



- *cutCandidates*

An optional input polygon layer used as candidates for cut locations. Enables repair flows where most cut locations are known, with some needing to be fixed or moved because of local target shape changes. The input cutCandidates are used where possible, but may be rejected because of spacing errors to other cutCandidates, or to additional cuts that are required to satisfy local constraints. The corresponding inputLayerOrder keyword is cutCandidates.

Shapes on the cutCandidates layer must comply with the same parameter requirements within the LITHO FILE as generated cuts. Any cutCandidates shapes that fail compliance are output to the illegalCutCandidates error layer. These criteria include:

- Cuts must be rectangular.
- Cuts must not intersect any cutKeepout layer.
- Cut width must comply to the specification in the LITHO FILE.
- Cuts must not intersect target shapes (mask-specific if multiple cut masks are used).
- Cut length must comply to the specification in the LITHO FILE.
- All cuts must comply with the specified cutOverlap parameter.

- *cutCandidates_0, cutCandidates_1*

Optional input polygon layers used for the same function as cutCandidates, but differentiating dual mask colors. The cutCandidates_0 layer is used for the mandrel cut mask and the cutCandidates_1 layer for the non-mandrel cut mask. The corresponding inputLayerOrder keywords are cutCandidates_0 and cutCandidates_1.

- *cutKeepout*

An optional input polygon layer used to prohibit the placement of cuts for both masks. All cutKeepout shapes that touch or overlap generated tracks define regions where no cut shapes can be placed. The corresponding inputLayerOrder keyword is cutKeepout.

- *cutKeepout_0, cutKeepout_1*

Optional input polygon layers performing the same function as cutKeepout, but differentiating dual mask colors. Used to prohibit the placement of cuts for dual mask layers. The cutKeepout_0 layer is used for mandrel cut keepouts and the cutKeepout_1 layer for non-mandrel cut keepouts. The corresponding inputLayerOrder keywords are cutKeepout_0 and cutKeepout_1.

- *mandrelAnchor*

An optional input polygon layer comprising shapes used to assign target shapes to a mandrel track. Any target shape touching or overlapping any shape on the mandrelAnchor layer is assigned to the mandrel mask. The corresponding inputLayerOrder keyword is mandrelAnchor.

- *nonMandrelAnchor*

An optional input polygon layer comprising shapes used to assign target shapes to a non-mandrel track. Any target shape touching or overlapping any shape on the nonMandrelAnchor layer is assigned to the non-mandrel mask. The corresponding inputLayerOrder keyword is nonMandrelAnchor.

- *sinuousCutRegion*

An optional input polygon layer used to define the region where sinuous cut exceptions are allowed. This input layer may be used in single cut mask or two cut mask mode. The corresponding inputLayerOrder keyword is sinuousCutRegion.

- *sinuousCutRegion_0, sinuousCutRegion_1*

Optional input polygon layers used to define the region where sinuous cut exceptions are allowed, but differentiating two mask colors. In two cut mask mode, the sinuous cut regions may be specified for two masks independently. If sinuousCutRegion is also specified, then the effective region of allowed sinuous cuts is the logical OR of the sinuousCutRegion layer and the sinuousCutRegion_0 or sinuousCutRegion_1 layers. The sinuousCutRegion_0 layer identifies regions allowing sinuous cuts over non-mandrel tracks and the sinuousCutRegion_1 layer identifies regions allowing sinuous cuts over mandrel tracks. The corresponding inputLayerOrder keywords are sinuousCutRegion_0 and sinuousCutRegion_1.

- *terminationRegion*

An optional input polygon layer containing shapes that surround areas of previously determined tracks that may require subsequent modification outside the sadpMarker layer by either terminalDummyTrackWidth or terminationWidthSequence parameters. The terminationRegion layer can only include polygons touching one track-direction edge of the sadpMarker layer. If it includes polygons touching two or more different track-direction edges, an error message is output, and the terminationRegion is not extended to cover those edges.

- *trackPattern*

An optional input polygon layer used to define all tracks. By default, tracks are computed as implied by shapes on the target. Since RET SIDCUTFILL may fill various gaps between shapes on the target and the sadpMarker layer, if the computed tracks do not match user intent, or the dummy tracks do not satisfactorily fill the spaces between tracks, the trackPattern layer may be used to define the tracks instead. The trackPattern layer consists of polygons inside the sadpMarker delineating all track locations. These polygons do not have to span the entire sadpMarker region. All shapes on the trackPattern layer must be rectangles. The corresponding inputLayerOrder keyword is trackPattern.

Note

 If the target layer is not specified, the trackPattern, or trackPattern_0 and trackPattern_1 layers must be specified for target layer derivation by RET SIDCUTFILL.

Although allowable, Siemens EDA recommends the trackPattern layer not contain jogs. If the trackPattern layer is present, no further attempt is made by RET SIDCUTFILL to compute tracks.

- *trackPattern_0, trackPattern_1*

Optional input polygon layers performing the same function as trackPattern, but differentiating dual mask colors. These input layers can be used in conjunction where the derivedTarget output layer is required. The trackPattern_0 layer is used for mandrel track patterns and the trackPattern_1 layer for non-mandrel track patterns. The corresponding inputLayerOrder keywords are trackPattern_0 and trackPattern_1.

- *trimMaskKeepout*

Optional input polygon layer for single masks used to prohibit the placement of trim masks. Where trimMaskKeepout shapes touch or overlap generated tracks define regions in which no trim mask can be placed. The corresponding inputLayerOrder keyword is trimMaskKeepout.

- *trimMaskKeepout_0, trimMaskKeepout_1*

Optional input polygon layers differentiating dual mask colors used to prohibit the placement of trim masks. Where trimMaskKeepout shapes touch or overlap generated tracks define regions in which no trim mask can be placed. The trimMaskKeepout_0 layer is used for mandrel trim mask keepouts and the trimMaskKeepout_1 layer for non-mandrel trim mask keepouts. The corresponding inputLayerOrder keywords are trimMaskKeepout_0 and trimMaskKeepout_1.

- *unmovableCutCandidates*

An optional input polygon layer used as candidates for cut locations that cannot be moved. Cuts on this layer disable repair flows where cuts may need to be fixed or moved because of local target shape changes or spacing errors. The corresponding inputLayerOrder keyword is unmovableCutCandidates.

Shapes on the `unmovableCutCandidates` layer must comply with the same parameter requirements within the LITHO FILE as generated cuts, and must pass compliance in the same way as `cutCandidates`. All `unmovableCutCandidates` shapes that fail compliance are output to the `illegalCutCandidates` error layer. For compliance information, see `cutCandidates`.

- *unmovableCutCandidates_0, unmovableCutCandidates_1*

Optional input polygon layers differentiating dual mask colors used as candidates for cut locations that cannot be moved. Cuts on these layers disable repair flows where cuts may need to be fixed or moved because of local target shape changes or spacing errors. The `unmovableCutCandidates_0` layer is used for the mandrel cut mask and the `unmovableCutCandidates_1` layer for the non-mandrel cut mask. The corresponding `inputLayerOrder` keywords are `unmovableCutCandidates_0` and `unmovableCutCandidates_1`.

- *userCuts*

An optional input polygon layer used to define a layer comprising user-defined cut shapes. Although non-rectangular user-defined cuts are processed, it is advisable to define cuts only as rectangles to avoid unexpected processing results. The corresponding `inputLayerOrder` keyword is `userCuts`.

- *userCuts_0, userCuts_1*

Optional input polygon layers, identical in function to `userCuts`, but used to define cut layers for multiple cut masks. The `userCuts_0` layer is used for the mandrel cut mask and the `userCuts_1` layer for the non-mandrel cut mask. The corresponding `inputLayerOrder` keywords are `userCuts_0` and `userCuts_1`.

- *via, via_a, via_b, via_c*

Optional input polygon layers used to define layers containing via shape locations. The `via` layers are typically used for generation of the `derivedTarget` layer where no target layer is specified, and also for defining minimum spacing constraints between via and cut layers using the `viaEnclosure` parameter. The corresponding `inputLayerOrder` keywords are `via`, `via_a`, `via_b` and `via_c`, respectively.

LITHO FILE Statement Name

- **FILE name**

A required keyword followed by the name of a LITHO FILE statement. The LITHO FILE statement is called by the RET SIDCUTFILL command.

MAP Output Keywords

- **MAP keyword**

A required keyword and its argument that specifies the application of the RET SIDCUTFILL output layer. The output keywords are used to generate mask layers

Mask-Related Output Keywords

`cutMask` — An optional keyword that generates legal cuts to the cut mask layer.

`cutMask_0, cutMask_1` — Optional keywords that generate legal cuts to dual cut mask layers. The `cutMask_0` keyword is used for the mandrel cut mask and the `cutMask_1` keyword for the non-mandrel cut mask.

`mandrel` — An optional keyword that generates mandrel tracks. See [Figure 4-9](#) on page 228 for mandrel and non-mandrel process-related information.

Derivation-Related Output Keywords

`derivedTarget` — An optional keyword that generates the derived target layer. For more information, see “[Derived Target](#)” on page 244.

`derivedTarget_0, derivedTarget_1` — Optional keywords that generate the derived target layer to dual mask layers. The `derivedTarget_0` keyword is used for the mandrel cut mask and the `derivedTarget_1` keyword for the non-mandrel cut mask.

`effectiveCriticalTimingMarkers` — An optional keyword that generates an `effectiveCriticalTimingMarkers` layer with orthogonal edges from a `criticalTimingMarkers` input shape that may contain non-orthogonal edges. The resulting `effectiveCriticalTimingMarkers` layer always contains the original `criticalTimingMarkers` shape.

`effectiveSadpMarker` — An optional keyword that generates an `effectiveSadpMarker` layer with orthogonal edges from an `sadpMarker` shape that may contain non-orthogonal edges. The resulting `effectiveSadpMarker` layer always contains the original `sadpMarker` shape.

`effectiveTerminationRegion` — An optional keyword that reflects the actual termination regions as computed and used by RET SIDCUTFILL. The input termination regions are assigned by the `terminationRegion` layer.

`pitchAlignTracks` — An optional keyword that generates a layer containing the pitch align tracks computed from the `pitch_align` input layer and the parameters `pitchAlignTrackWidth` and `pitchAlignTrackSpacing`. If RET SIDCUTFILL cannot compute any pitch align tracks within an `sadpMarker` layer, that `sadpMarker` shape contains no tracks, no aligned cuts, and may completely fail all computation.

`trimMask` — An optional keyword that generates the trim mask. For more information, see “[Trim Mask](#)” on page 241.

`trimMask_0, trimMask_1` — Optional keywords that generate the trim masks for dual mask layers. The `trimMask_0` keyword is used for the mandrel cut mask and the `trimMask_1` keyword for the non-mandrel cut mask.

Other Output Keywords

`dummyFillCutMarkers` — An optional keyword that generates markers for those cuts placed by the `maxDummyFillLength` parameter.

`nonMandrel` — An optional keyword that generates non-mandrel tracks.

`sinuousCutMarkers` — An optional keyword that generates an error layer for single cut masks, marking cuts that satisfy all spacing requirements except those that are waived by the `sinuousCutRegion` layer. RET SIDCUTFILL moves sinuous cuts as close to each other as possible, making a smooth transition. The solution for sinuous cuts is to

make a smooth transition, on average equally distributing spaces according to the surrounding geometries.

sinuousCutMarkers_0, sinuousCutMarkers_1 — An optional keyword that generates an error layer for two cut masks. The **sinuousCutMarkers_0** keyword is used for the mandrel cut mask and the **sinuousCutMarkers_1** keyword for the non-mandrel cut mask.

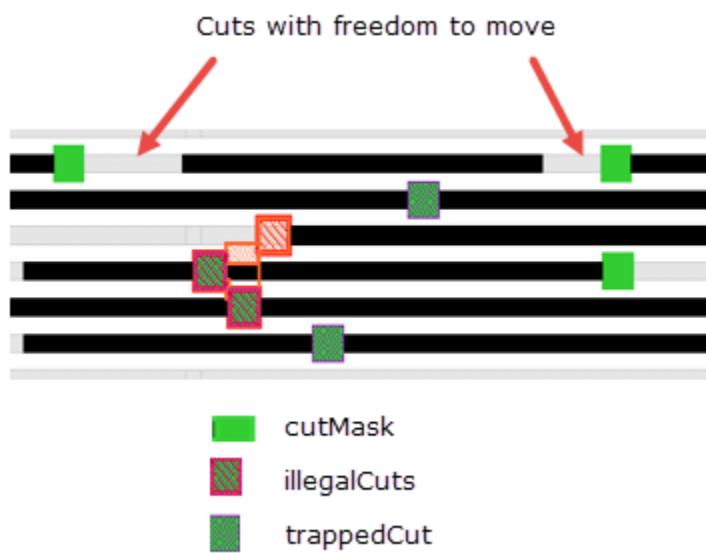
sinuousCutConnectors — An optional keyword that generates an error layer marking violated spacing requirements which are waived by the **sinuousCutRegion** layer.

sinuousCutConnectors_0, sinuousCutConnectors_1 — Optional keywords that generate an error layer marking violated spacing requirements for their respective cuts which are waived by the **sinuousCutRegion** or **sinuousCutRegion_0** and **sinuousCutRegion_1** layers. The **sinuousCutConnectors_0** keyword is used for the mandrel cut mask and the **sinuousCutConnectors_1** keyword for the non-mandrel cut mask.

track — An optional keyword that generates tracks to the track layer. Tracks fill the extent of each polygon on the **sadpMarker** layer spaced by the **trackSpacing** parameter. RET SIDCUTFILL generates tracks based on the arguments **trackDirection**, **minTrackWidth**, **terminalDummyTrackWidth**, **trackSpacing**, and the target polygons.

trappedCuts — An optional keyword that outputs cuts between two target shapes equal to the **cutWidth** parameter. Trapped cuts may extend across empty tracks between other trapped cuts where spacing parameters require two cuts to be joined into a single cut; in this case, the joined cut is output. By default, trapped cuts are placed on the **cutMask** output layer (or **cutMask_0** and **cutMask_1** for dual mask layers), but may also be output to the **illegalCuts** layer if they cause spacing errors. Trapped cuts may be extended to nearby tracks as required, but the **trappedCuts** error layer will not include these extensions.

Figure 4-38. trappedCuts Layer



`trappedCuts_0`, `trappedCuts_1` — Optional keywords that output cuts for dual mask layers between two target shapes of a spacing equal to the `cutWidth` parameter. Equivalent in function to `trappedCuts`. The `trappedCuts_0` keyword is used for the mandrel cut mask and the `trappedCuts_1` keyword for the non-mandrel cut mask.

`unalignedCuts` — An optional warning layer to detect those cuts not aligned with `pitchAlignTrack` layer. This keyword may be used for single-cut or two-cut masks.

`unalignedCuts_0`, `unalignedCuts_1` — Optional warning layers to detect those cuts not aligned with `pitchAlignTrack_0`, and `pitchAlignTrack_1` layers, respectively. In two cut mask mode, the unaligned cuts may be specified for two masks independently.

Error Output Keywords

`anchorConflict` — An optional keyword that generates a layer containing rectangles defined by regions between two anchored tracks that are in conflict. Anchors trigger coloring conflicts over an even number of tracks.

`anchorSelfConflict` — An optional keyword that generates a layer containing all target shapes that have been assigned both mandrel and a non-mandrel anchors. The anchors are not included in the output layer.

`anchorTrackConflict` — An optional keyword that generates a layer containing all tracks that have been assigned both mandrel and a non-mandrel anchors. The anchors are not included in the output layer.

`cutErrorCluster` — An optional keyword that generates polygons that connect cuts on the `illegalCuts` layer with polygons on the `cutMask` layer where cut spacing errors are found. These errors may indicate potential interactions between illegal cuts and legal cuts.

`cutErrorCluster_0`, `cutErrorCluster_1` — Optional keywords that generate polygons that connect cuts on the `illegalCuts_0` and `illegalCuts_1` layers with polygons on the `cutMask_0` and `cutMask_1` layers, respectively, where cut spacing errors are found. The `cutErrorCluster_0` keyword is used for the mandrel cut mask and the `cutErrorCluster_1` keyword for the non-mandrel cut mask.

`cutPlacementError` — An optional keyword that generates polygons spanning cuts (shapes on the output layer generated by the `cutMask` keyword) indicating the presence of spacing errors that cannot be resolved. The `cutPlacementError` keyword also generates shapes to the `illegalCuts` layer.

`cutPlacementError_0`, `cutPlacementError_1` — Optional keywords that generate polygons spanning cuts on dual mask layers (shapes on the output layers generated by the `cutMask_0` and `cutMask_1` keywords) indicating the presence of spacing errors that cannot be resolved. The `cutPlacementError_0` and `cutPlacementError_1` keywords also generate shapes to the `illegalCuts_0` and `illegalCuts_1` layers, respectively. The `cutPlacementError_0` keyword is used for the mandrel cut mask and the `cutPlacementError_1` keyword for the non-mandrel cut mask.

`illegalCutCandidates` — An optional keyword that generates an error layer containing polygons from the `cutCandidates` input layer which do not comply with the same requirements for generated cuts within the LITHO FILE.

`illegalCutCandidates_0`, `illegalCutCandidates_1` — Optional keywords that generate error layers containing polygons from the `cutCandidates_0` and `cutCandidates_1` input layers which do not comply with the same requirements for generated cuts within the LITHO FILE. The `illegalCutCandidates_0` keyword is used for the mandrel cut mask and the `illegalCutCandidates_1` keyword for the non-mandrel cut mask.

`illegalCuts` — An optional keyword that generates a layer consisting of illegally generated cut shapes, typically those that violate cut spacing requirements, minimum area requirements, or cut dimension requirements (`cutWidth` or `maxCutLength` parameters).

`illegalCuts_0`, `illegalCuts_1` — Optional keywords that generate layers consisting of illegally generated cut shapes on dual mask layers, typically those that violate cut spacing requirements or the `maxCutLength` setting. The `illegalCuts_0` keyword is used for the mandrel cut mask and the `illegalCuts_1` keyword for the non-mandrel cut mask.

`illegalTarget` — An optional keyword that generates a layer consisting of shapes from the target layer that do not conform to the tracks generated by RET SIDCUTFILL. See [Figure 4-28](#) on page 246 for more information.

`illegalTargetGap` — An optional keyword that generates a layer containing shapes formed by gaps between two legal target shapes with widths less than the specified `cutWidth` parameter. If pitch align or `maskGridSpacing` is used, then the line-end extensions forced by that usage may result in additional `illegalTargetGaps`. In these cases cuts will still be placed, but will also be marked as illegal. These gap shapes are added to their respective target shapes for further processing. See [Figure 4-28](#) on page 246 for more information.

`illegalTrackSpace` — An optional keyword that generates a layer containing shapes formed by spaces between tracks that do not exactly adhere to the spacing specified by the `trackSpacing` parameter. This spacing check is performed after RET SIDCUTFILL adds dummy tracks. There are two ways track spacing can fail:

- Two target shapes define a pair of tracks with spacing less than the value specified by `trackSpacing`.
- Two target shapes define a pair of tracks (with no other occupied tracks between) with a gap greater than the `trackSpacing` value, but less than $2(trackSpacing) + minTrackWidth$.

See [Figure 4-28](#) on page 246 for more information.

`illegalTrackWidth` — An optional keyword that generates track shapes implied by target input shapes of illegal width, or due to existing space between target shapes, requiring tracks of illegal width to be placed. When `minTrackWidth` is used to determine track widths, tracks of widths smaller than its specified value are illegal. When `trackWidth` is used to specify the list of all legal track widths, tracks not matching one of the specified values in the list are illegal.

`insufficientEdgeSpace` — An optional keyword that generates shapes formed by the spaces between the inside edge of an `sadpMarker` layer and the nearest occupied track

(parallel to the considered edge) that is not sufficient for the required mandrel track of width terminalDummyTrackWidth to be placed. Because the outside tracks must be always be mandrel tracks, a larger space than terminalDummyTrackWidth + trackSpacing may be required due to an additional non-mandrel track needed to make the mask assignment correct. The final mandrel track is allowed to be larger than the specified value, but is generated at the smallest width required.

longCuts — An optional error keyword that generates a layer consisting of cut shapes exceeding the maxCutLength parameter, that would otherwise be legal.

longCuts_0, **longCuts_1** — Optional error keywords that generate layers consisting of cut shapes exceeding the maxCutLength parameter, that would otherwise be legal. The longCuts_0 keyword is used for the mandrel cut mask and the longCuts_1 keyword for the non-mandrel cut mask.

PitchAlignTargetExtensions — An optional keyword that generates a layer containing polygons of extended target line-ends required to meet maskGridSpacing and PitchAlign constraints.

pitchAlignTrackErrors — An optional keyword that generates an error layer containing pitchAlignTracks computation errors. These errors may output tracks of a width not equal to the pitchAlignTrackWidth parameter or of a spacing between tracks not equal to the pitchAlignTrackSpacing parameter. The pitchAlignTrackErrors layer may also contain the sadpMarker layer indicating the pitch align tracks could not be computed for that sadpMarker shape.

Debugging Output Keywords

effectiveTarget — An optional keyword that generates an output layer representing the target layer used by RET SIDCUTFILL to place cuts. The layer generated by effectiveTarget may vary from the input target layer:

- The layer contains input target, mandrelAnchor, and nonMandrelAnchor shapes.
- Any of these aforementioned shapes not aligning with internally-derived or user-specified track patterns are deleted from the effectiveTarget layer. The illegalTarget output layer contains these deleted shapes.
- Extensions of the target line-ends forced by pitch alignment rules, if present, are included in the effectiveTarget layer.
- The maskGridSpacing parameter is a special case of pitch alignment, and may force extension of line-ends, adding to the effectiveTarget layer.
- If a pitchAlign layer or maskGridSpacing parameter is specified, resulting line-end extensions may result in additional shapes on the illegalTargetGap layer. Cuts will still be placed, but are marked as illegal. The effectiveTarget layer includes all line-ends corresponding to valid positions for the aligned cuts. For more information regarding illegalTargetGap, see “[Error Handling](#)” on page 245.

effectiveTarget_0, **effectiveTarget_1** — Optional keywords that generate output layers representing the target layers used by RET SIDCUTFILL to place cuts. The

effectiveTarget_0 keyword is used for the mandrel cut mask and the effectiveTarget_1 keyword for the non-mandrel cut mask.

modifiedTrackTerminationRegions — An optional keyword that generates a debugging layer containing regions of the sadpMarker shape that have had track terminations modified.

Examples

In this example, layer names specified in the RET SIDCUTFILL commands are identical to the names declared in the inputLayerOrder list. This is not required, since the layers named within RET SIDCUTFILL are sequentially mapped to layers declared in the inputLayerOrder list by position, and not by name. The declaration of layers in the inputLayerOrder list can be arranged in any order. In this example code, the MAP keywords are highlighted in red.

```
LITHO FILE scf_setup [  
  
    inputLayerOrder = (target, sadpMarker, mandrelAnchor,  
                      sinuousCutRegion);  
  
    dpFaceClass name(basic_cut) orientation(perpTrackDirection):  
        true for length <= 0.048;  
        false otherwise;  
  
    dpFaceClass name(long_cut) orientation(perpTrackDirection):  
        true for length > 0.048;  
        false otherwise;  
  
    dpFaceClass name(basic_cut_end) orientation(trackDirection):  
        true for length <= 0.048;  
        false otherwise;  
  
    dpFaceClass name(long_cut_end) orientation(trackDirection):  
        true for length > 0.048;  
        false otherwise;  
  
    dpSpacing (any, any): 0.046 otherwise;  
  
    dpSpacing (any, basic_cut_end) metric(euclidean): 0.050 otherwise;  
  
    dpSpacing (any, basic_cut_end) metric(opposite): 0.065 otherwise;  
  
    dpSpacing (any, long_cut_end) metric(euclidean): 0.054 otherwise;  
  
    dpSpacing (any, long_cut_end) metric(opposite): 0.096 otherwise;  
  
    dpSpacing (any, basic_cut) metric(opposite):  
        0.064 for runLength >= 0.024;  
        0 otherwise;  
  
    dpSpacing (any, long_cut) metric(opposite):  
        0.070 for runLength >= 0.024;  
        0 otherwise;
```

```
trackSpacing          = 0.024;
trackDirection        = X;
cutOverlap            = 0.012;
minTrackWidth         = 0.024;
terminalDummyTrackWidth = 0.048;
minArea               = 0.0024;
maxLineEndExtension  = 0.150;
cutWidth              = 0.050;
]

cut_mask = RET SIDCUTFILL target sadpMarker mandrelAnchor
           sinuousCutRegion
           FILE scf_setup MAP cutMask

cutErrors = RET SIDCUTFILL target sadpMarker mandrelAnchor
           sinuousCutRegion
           FILE scf_setup MAP cutPlacementError

tracks = RET SIDCUTFILL target sadpMarker mandrelAnchor
           sinuousCutRegion
           FILE scf_setup MAP track

mandrel = RET SIDCUTFILL target sadpMarker mandrelAnchor
           sinuousCutRegion
           FILE scf_setup MAP mandrel

nonMandrel = RET SIDCUTFILL target sadpMarker mandrelAnchor
           sinuousCutRegion
           FILE scf_setup MAP nonMandrel

badSpace = RET SIDCUTFILL target sadpMarker mandrelAnchor
           sinuousCutRegion
           FILE scf_setup MAP illegalTrackSpace

anchorErr = RET SIDCUTFILL target sadpMarker mandrelAnchor
           sinuousCutRegion
           FILE scf_setup MAP anchorConflict
```

LITHO FILE

Self-Aligned Double-Patterning Command Reference

Parameters specified within the LITHO FILE statement include designated layer order, and numeric values or variables used by RET SIDCUTFILL. The LITHO FILE statement may be written anywhere within the SVRF file. Parameters are specified one per line in the LITHO FILE statement.

Usage

```
LITHO FILE name '['  
    inputLayerOrder = '(`sadpMarker, target [, keywordN, ...]`);  
    cutWidth = width;  
    dpFaceClass name`(`face_class_name`)...  
    dpSpacing `(`class1, class2`)...  
    minTrackWidth = width; | trackWidth = `(`width1, width2 [, widthN, ...]`);  
    trackDirection = {x | y};  
    sidewallWidth = `(`width1 [, widthN, ...]`);  
    allowEvenJogs = {true | false};  
    boundaryCuts = boundary_argument;  
    cutOverlap = extension;  
    dpFaceSet name`(`face_set_name`)...  
    extendCutOverlapToCenterOfNextTrack = {any | false | symmetric};  
    ignoreOptionalSpacing = `(`false | true`);  
    invertedSpacingCheck = `(`error | false | warning`);  
    maskGridSpacing = value;  
    maxCutLength = {length | doNotMerge | unlimited};  
    maxDummyFillLength = length;  
    maxLineEndExtension = {extension | unlimited};  
    maxSadpMarkers = limit;  
    minArea = area;  
    minLithoGap = {gap | mandrel_gap, nonmandrel_gap};  
    optimizeLineEnds = `(`critical | false | true`);  
    patternMultiplier = value;  
    pitchAlignCutOffset = `(`offset0 [, offsetN, ...]`);  
    pitchAlignTrackSpacing = space;  
    pitchAlignTrackWidth = width;  
    slidingCuts = {true | false};  
    slidingJogCuts = {true | false};  
    terminalDummyTrackJogWidth = width;  
    terminalDummyTrackWidth = width;  
    terminationWidthSequence = `(`trackwidth1, trackwidth2 [, trackwidthN, ...]`);  
    trackSpacing = space;  
    trackTrim = distance;  
    trackWidthSequence = `(`targetwidth1, trackwidth1 [, trackwidthN, ...], targetwidth2`);  
    trimMaskMinJog = distance;
```

```

trimMaskMinWidth = distance;
trimTargetToMinViaEnclosure = { true | false };
viaEnclosure = enclosure;
viaEnclosure_a = enclosure;
viaEnclosure_b = enclosure;
viaEnclosure_c = enclosure;
]

'
```

Description

Parameters are specified within the LITHO FILE statement to provide the layers and values needed by RET SIDCUTFILL to generate correct SADP-related topography and perform SADP-related dimensional checking.

You can use variables for any names or numbers as further described in “[Using SVRF Variables within a LITHO FILE Statement](#)” on page 323. For example:

```

VARIABLE minTrackWidth 0.024
LITHO FILE [
    minTrackWidth = $minTrackWidth,
    ...
]
```

The LITHO FILE statement called by RET SIDCUTFILL can be specified anywhere within the same SVRF file. The LITHO FILE statement is typically called by multiple RET SIDCUTFILL commands. Each keyword in the LITHO FILE must be on a line of its own.

Different Calibre RET tools use different keywords in the LITHO FILE statements and therefore cannot generally be shared between tools. For more information regarding commands specified within the LITHO FILE statement, see “[SADP-Specific Spacing Parameters](#)” on page 288.

Table 4-4. SADP LITHO FILE Parameter Summary

Name	Req/Opt	Type	Default	Description
allowEvenJogs	optional	boolean	false	Inputs target shapes with jogs spanning an even number of tracks.
boundaryCuts	optional	keyword	notRequired	Determines how RET SIDCUTFILL handles boundary-related cuts.
cutOverlap	optional	numeric	sidewallWidth/2	Sets the perpendicular cut extension distance beyond tracks.
cutWidth	required	numeric length	none	Sets fixed cut width.
dpFaceClass	required	name and numeric length	none	Associates specified cut edges with a face class name.

Table 4-4. SADP LITHO FILE Parameter Summary (cont.)

Name	Req/Opt	Type	Default	Description
dpSpacing	required	name vector and spacing	none	Sets spacing between specified face classes.
dpFaceSet	optional	name vector	none	Associates specified face classes into a single face set.
extendCutOverlapToCenterOfNextTrack	optional	keyword	false	Controls cut extensions.
ignoreOptionalSpacing	optional	boolean	false	Disables consideration of optional spacing conditions.
inputLayerOrder	required	name vector	none	Declares the order of input layer names.
invertedSpacingCheck	optional	keyword	error	Controls inverted spacing checks.
maskGridSpacing	optional	numeric	0	Enabled when slidingCuts is true. Restricts cut placement to the desired grid.
maxCutLength	optional	numeric or keyword	unlimited	Controls cut lengths.
maxDummyFillLength	optional	numeric	none	Sets dummy track fill length with added cuts.
maxLineEndExtension	optional	numeric or keyword	cutWidth + maxSpacing	Sets maximum allowable extension to the line-end in case of cut dropping or cut sliding. The default value is also the minimum allowable extension in case of cut sliding. If cut sliding is not allowed, the minimum value is zero.
maxSadpMarkers	optional	numeric	none	Sets the maximum limit of the number of sadpMarker shapes that may be processed from the input layout.
minArea	optional	numeric area	0	Sets minimum allowed area of a track segment.
minLithoGap	optional	numeric	none	Sets the minimum gap between target line-ends.
minTrackWidth	required	numeric vector	none	Sets minimum allowable track width value.

Table 4-4. SADP LITHO FILE Parameter Summary (cont.)

Name	Req/Opt	Type	Default	Description
optimizeLineEnds	optional	boolean	critical	Controls the line-end optimization algorithm.
patternMultiplier	required	numeric	2	Sets multiplier to SADP or SAQP.
pitchAlignCutOffset	optional	numeric vector	0	Adds cuts at offset from the center lines of pitchAlignTrackSpacing.
pitchAlignTrackSpacing	optional	numeric	none	Sets space between pitchAlign tracks.
pitchAlignTrackWidth	optional	numeric vector	none	Sets width of pitchAlign tracks.
sidewallWidth	required	numeric	none	Sets a fixed width of two or more sidewalls (effectively track spacing) for SADP or SAQP and higher. As of the 2020.1 release, this parameter replaces trackSpacing.
slidingCuts	optional	boolean	true	Allows cuts to move from line ends in order to comply with spacing requirements and cutKeepout regions.
slidingJogCuts	optional	boolean	allowEvenJogs	Enables sliding cuts associated to jog shapes.
terminalDummyTrackJogWidth	optional	numeric	0	Sets trackWidth jog transition step between regular tracks and the termination track.
terminalDummyTrackWidth	optional	numeric	minTrackWidth or smallest trackWidth vector.	Sets the width of terminal dummy tracks (tracks at the extremities of the sadpMarker shape), and assigns them to the mandrel mask by default.
terminationWidthSequence	optional	numeric vector	none	Sets the termination track widths and their sequence.
trackDirection	optional	x y	none	Sets x or y track orientation.
trackSpacing	optional	numeric	none	Sets a fixed single value of spacing between tracks. Used for SADP only. This parameter is deprecated as of the 2020.1 release. Use sidewallWidth instead.

Table 4-4. SADP LITHO FILE Parameter Summary (cont.)

Name	Req/Opt	Type	Default	Description
trackTrim	optional	numeric	cutWidth	Sets the distance a mandrel track is trimmed away from a nonMandrel pad or jog.
trackWidth	required	numeric vector	none	Sets multiple track widths.
trackWidthSequence	optional	numeric vector	none	Defines a sequence of track widths that is used to fill gaps between target shapes.
trimMaskMinJog	optional	numeric	none	Sets the minimum concave edge length of a trim mask polygon.
trimMaskMinWidth	optional	numeric	none	Sets the minimum width in the track direction of a trim mask polygon.
trimTargetToMinViaEnclosure	optional	boolean	false	Specifies whether line-ends that extend farther than the required distance past a via may be trimmed back to match the required enclosure.
viaEnclosure, viaEnclosure_a, viaEnclosure_b, viaEnclosure_c	optional	numeric	none	Sets the minimum distance between vias and end of target for four via enclosures.

Arguments

- **name**

A required name used to pass the LITHO FILE statement to the [RET SIDCUTFILL](#) command. All LITHO FILE statements must have a unique name.

- **inputLayerOrder = ('sadpMarker, target [, keyword]...');**

A required list of one or more keywords that determines the order of the input layers specified in the RET SIDCUTFILL command. There are two required keywords, sadpMarker and target; all other keywords are optional. The keywords are case-insensitive. The layer names specified within the SVRF and those input to RET SIDCUTFILL must match, and can be any name or case.

The following example correctly orders the layers declared by the LAYER command to the RET SIDCUTFILL command (colors shown map the layer and keywords). There is no case-sensitivity or layer order requirements:

```
LAYER M1 1
LAYER sadp_Boundary 2
LAYER mandrelAnchor 3
```

```
LITHO FILE [
    inputLayerOrder = (target, mandrelanchor, sadpMARKER) ;
    ...
]

cut_mask = RET SIDCUTFILL M1 Mandrelanchor SADP_boundary ...
```

All keywords that order the layer names specified in the SVRF file and in the RET SIDCUTFILL command are listed here (for more information regarding the layer usage, see “[RET SIDCUTFILL](#)” on page 255).

- **sadpMarker** — A required keyword for ordering the layer containing the SADP area to process.
- **target** — A required keyword for ordering the target layer name. If the target layer is derived by RET SIDCUTFILL, this keyword is not specified. For more information regarding target layer derivation, see “[Derived Target](#)” on page 244.
- **criticalTimingMarkers** — An optional keyword for ordering the criticalTimingMarkers layer name.
- **cutCandidates** — An optional keyword for ordering the cut candidate layer name. May be used for deriving target shapes, if the target layer is not specified.
- **cutCandidates_0**, **cutCandidates_1** — Optional keywords for ordering the cut candidate layer names for dual masks. May be used for deriving target shapes, if the target layer is not specified.
- **cutKeepout** — An optional keyword for ordering the cut keepout layer name.
- **cutKeepout_0**, **cutKeepout_1** — Optional keywords for ordering the cut keepout layer names for dual masks.
- **faceClassRegion_a** through **faceClassRegion_h** — Optional keywords for naming the face class regions for the dpFaceClass command.
- **mandrelAnchor** — An optional keyword for ordering the mandrel anchor layer name.
- **nonMandrelAnchor** — An optional keyword for ordering the non-mandrel anchor layer name.
- **pitchAlign** — An optional keyword for ordering the pitch align seed(s) layer name.
- **sinuousCutRegion** — An optional keyword for ordering the region where sinuous cut exceptions are allowed. This keyword may be used in single cut mask or two cut mask mode.
- **sinuousCutRegion_0**, **sinuousCutRegion_1** — Optional keywords for ordering the regions where sinuous cut exceptions are allowed. In two cut mask mode, the sinuous cut regions may be specified for two masks independently. If **sinuousCutRegion** is also specified, then the effective region of allowed sinuous cuts

is the logical OR of the sinuousCutRegion layer and the sinuousCutRegion_0 or sinuousCutRegion_1 layers.

- trackPattern — An optional keyword for ordering the track pattern layer name. For single mask recipes, the trackPattern keyword is required where the target layer is derived.
 - trackPattern_0, trackPattern_1 — Optional keywords for ordering the track pattern layer names for dual masks. These layers can be used to define track color in single-cut mask mode. For dual mask recipes, the trackPattern_0 and trackPattern_1 keywords are required where the target layer is derived.
 - trimMaskKeepout — An optional keyword for ordering the trim mask keepout layer name.
 - trimMaskKeepout_0, trimMaskKeepout_1 — Optional keywords for ordering the trim mask keepout layer names for dual masks.
 - userCuts — An optional keyword for ordering the user-defined cut layer name. May be used for deriving target shapes, if the target layer is not specified.
 - userCuts_0, userCuts_1 — Optional keywords for ordering the user cut layer names for dual masks. May be used for deriving target shapes, if the target layer is not specified.
 - via, via_a, via_b, via_c — Optional keywords for ordering the via layer names. The via layers must be used for generation of the derivedTarget layer where no target layer is specified, and can be used for defining minimum spacing constraints between via and cut layers using the viaEnclosure parameter.
- **cutWidth = *width*;**
A required parameter that specifies the width of all generated cuts.
 - **dpFaceClass *name*‘(*face_class_name*‘) ...**
A required command that classifies edges into a named group. For more information see “[dpFaceClass](#)” on page 289.
 - **dpSpacing ‘(*class1,class2*‘) ...**
A required command that performs design rule checks on edges previously classified by dpFaceClass. For more information see “[dpSpacing](#)” on page 294.
 - **minTrackWidth = *width*; or trackWidth = ‘(*width1, width2* [, *widthN*, ...])’;**
One of two required parameters that set the minimum width of generated tracks.
The minTrackWidth parameter specifies only the minimum track width, allowing larger tracks of unspecified width to be generated (typically with dummy tracks).
The trackWidth parameter specifies a list of a minimum of two track widths for track generation. The track width values must be specified in ascending order, with the smallest value reflecting what would be the minTrackWidth parameter. The minTrackWidth and

trackWidth parameters are mutually exclusive. The trackWidth parameter (with its multiple values) may be specified as a single variable:

```
VARIABLE trw "(0.024,0.028,0.032,0.038,0.050)"
```

- **trackDirection = x | y;**

A required parameter that specifies the orientation of the tracks. You specify x or y for horizontal and vertical track orientations, respectively.

- **sidewallWidth = '(*width1* [, *widthN*, ...])';**

A required parameter that sets the sidewall widths (equivalent to track spacing) for SADP and SAQP processes. The first parameter is the side wall width for SADP processes, or the first grown spacer for SAQP processes and beyond. The second parameter is the second grown spacer for SAQP processes. Any number of parameters may be specified.

Note



The sidewallWidth parameter has replaced trackSpacing, which has been deprecated as of the 2020.1 release.

- **allowEvenJogs = true | false;**

An optional parameter that allows jogs or pads to span an even number of tracks. If allowEvenJogs is false, jogs or pads must span an odd number of tracks or they are invalid and output to the illegalTarget layer. Specifying allowEvenJogs disables the debugJogShapes layer. The debugMultiTrackTargetMarkers layer highlights portions of valid target shapes that span an even number of tracks. The default is false.

- **boundaryCuts = *boundary_argument*;**

An optional parameter that determines the behavior in which RET SIDCUTFILL handles boundary-related cuts. One of the following boundary_arguments must be specified.

allowZeroArea

Enables cuts to be moved coincident with, but remaining inside, the sadpMarker boundary where the placed cut violates the minArea parameter. The allowZeroArea argument requires an optimally designed sadpMarker boundary.

ignoreMinArea

Disables the minArea parameter from applying to cut placement. Ordinarily, the minArea parameter specifies the minimum area of the track between the cut and the sadpMarker boundary.

maskGridSpacing

Forces cut placement on the specified grid relative to line-end locations.

notRequired

Disables cut placement in all boundary gaps, except where violating the maxLineEndExtension parameter. Cuts placed by the notRequired option are saved only to the illegalCuts error layer. This is the default setting.

`pitchAlign`

Forces cut placement on locations established through the `pitchAlign` layer and various pitch alignment parameters. For more information, see “[Cut Pitch Alignment](#)” on page 239.

`required`

Forces cut placement in all boundary gaps. If placed cuts violate spacing constraints, the cuts are also output to the `illegalCuts` error layer.

- `cutOverlap = extension;`

An optional parameter that extends both ends of a cut perpendicularly beyond the track sides it terminates by the extension value. The default extension amount is the last parameter specified by `sidewallWidth/2`. It also extends the trim mask perpendicularly to the track by the extension value, guaranteeing a valid width for the trim mask polygon. Used for standard cuts and trim mask generation.

- `dpFaceSet name('face_set_name')` ...

Accumulates various user-defined face classes into a single set. For more information see “[dpFaceSet](#)” on page 292.

- `extendCutOverlapToCenterOfNextTrack = {any | false | symmetric};`

An optional parameter that extends two ends of each cut perpendicular to their track to the center of the next track only where no spacing errors are introduced.

`any`

Extends cuts where only one of two extensions of the same cut are valid. This argument reflects the most lenient cut extension behavior.

`false`

Cuts are extended only by the dimension specified by `cutOverlap`. This is the default.

`symmetric`

Extends cuts only where both perpendicular extensions of the same cut are valid. This argument reflects the most restrictive cut extension behavior.

- `ignoreOptionalSpacing = {false | true};`

An optional parameter that disables optional spacing constraints during cut placement. For recipes in which optional spacing constraints are present, and optimizing line-ends is of greater priority, this parameter allows those optional spacing constraints to be ignored with minimal or no changes to the recipe. This parameter also makes it easier to compare results, and decide whether to choose optional spacing constraints or not. The default is `false`; optional spacing constraints are included during cut placement.

- `invertedSpacingCheck = ('error | false | warning');`

An optional parameter that controls how inverted spacing checks are rendered. An inverted spacing check is defined as a cut spacing error that can be resolved by moving two cuts closer together instead of farther apart.

RET SIDCUTFILL makes an initial assumption that there are three ways to solve a spacing violation between two cuts which include:

- a. Moving the cut shapes farther apart.
- b. Merging the two cut shapes into a single cut.
- c. Marking the cut shape pair as sinuous cuts where allowed.

RET SIDCUTFILL checks for common occurrences of inverted spacing rules and if it finds any, issues parsing error messages and halts the run. Specifically, RET SIDCUTFILL checks all dpSpacing commands using the runLength keyword. Two dpSpacing examples are shown that lead to potential problems. The first example shows a runLength that is less than 0, and the second example shows a spacing of 0 for any positive runLength:

```
dpSpacing (any,any) name(CorToCor) metric(euclidean) :
  0.050 for runLength < 0:
  0 otherwise:

dpSpacing (cut_side,cut_side) name(oddSpacing) metric(opposite) :
  0 for runLength > 0.005:
  0.050 otherwise:
```

Three options are provided to control inverted spacing messaging:

error

Generates runtime error messages and halts the run. This is the default setting.

false

Disables inverted spacing checks.

warning

Generates warning messages and continues the run to completion.

Note

 Although Siemens EDA recommends not using runLength within dpSpacing, or isolating smaller spacing rules for long edge run lengths, if these rules are necessary, specify invertedSpacingCheck to provide a safety net to prevent problematic and unrealistic constraints for SADP recipes.

- maskGridSpacing = *value*;

An optional parameter that, when slidingCuts is set to true, allows cuts to be placed anywhere along their tracks. If the precision specified for the Calibre run is smaller than the desired output grid, you use the maskGridSpacing parameter to force placement of cuts on the desired grid. For example, setting maskGridSpacing to 0.001 results in cuts being output to a 1 nm grid. If the database precision is 0.25 nm, cuts are placed on a 4 nm grid. Because cut placement is relative to target shapes, the input target shapes must be on the same grid specified by maskGridSpacing. The default is 0.

The maskGridSpacing parameter interacts with pitch alignment and target line-end extension algorithms:

- Target line-ends are extended as required to put line-ends on grid.
- Line-ends extended to match grid are included in the output layer, PitchAlignTargetExtensions, which is also used for pitch align operations.
- The debugReducedTarget layer includes target line-end extensions.
- The illegalTargetGap layer shows locations where the grid cannot be matched between two line-ends.
- maxCutLength = {*length* | doNotMerge | unlimited};

An optional parameter that either sets the maximum length of a cut (in the direction perpendicular to track orientation), disables, or allows unlimited cut lengths. There are three settings:

length

A positive number allowing cuts to be generated to a maximum of the specified value. The applied cut length is a multiple of a single, minimum cut length. All single tracks are cut, irrespective of their width. When the length of a cut exceeds maxCutLength the cut is reported on both the longCut and illegalCuts error layers. For cases where maxCutLength is less than the length of a cut required for spanning two adjacent minimum width tracks, maxCutLength is set to doNotMerge and applied.

doNotMerge

Disables merging of two cut ends, prohibiting cuts from crossing two or more tracks. However, all single tracks are cut, irrespective of their width, and cuts on jogs or pads are always allowed.

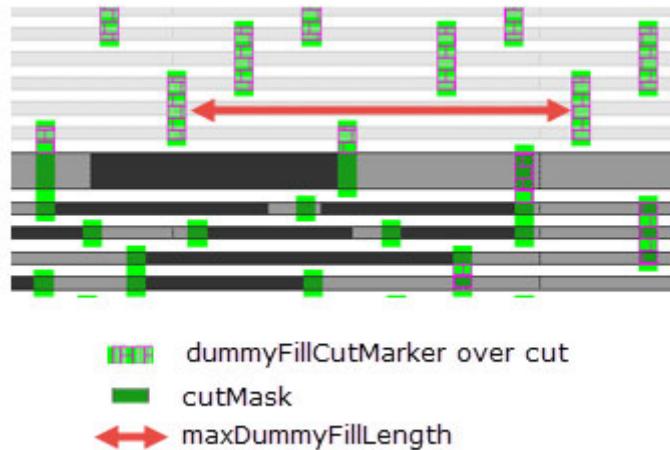
unlimited

Allows generation of cuts of any length. This is the default setting.

- maxDummyFillLength = *length*;

An optional parameter that limits the length of dummy fill tracks by adding cuts to terminate the fill. Added cuts may be placed anywhere along the tracks by maxDummyFillLength, limiting the tracks to a length less than or equal to the value specified. Cuts added by maxDummyFillLength are marked with the dummyFillCutMarkers layer. There is no default length.

Figure 4-39. maxDummyFillLength



- `maxLineEndExtension = {extension | unlimited};`

The `maxLineEndExtension` parameter determines the distance a cut may be placed from a line-end and still be legal. RET SIDCUTFILL attempts to minimize the length of these extensions, but uses the flexibility supplied by `maxLineEndExtension` and places cuts that prevent short circuits, even where they are not on a line-end. Each filler track polygon touches two target shapes. Normally, a cut is desired at both ends of every filler track.

If `slidingCuts` is set to true, then RET SIDCUTFILL always minimizes the line-end extensions it generates, even though `maxLineEndExtension` may be relatively large. The default value is `cutWidth + sidewallWidth`, where the last specified parameter of `sidewallWidth` is equal to the largest same-track spacing between cuts specified by either a `dpSpacing` command, or implied by `minArea` or `minTrackWidth`. There are two settings:

extension

Specifies the value for `maxLineEndExtension`. The value must be greater than the default value (`cutWidth + sidewallWidth`). However, if `maxLineEndExtension` is 0 and `slidingCuts` is false, cuts are always placed at line-ends. If these cuts happen to be illegal placements, errors are output, but all illegal cuts remain. For more information on cut dropping, see “[Operational Modes of Control](#)” on page 251.

unlimited

Specifies an incremental process applying extension values and their respective effects. Given that the minimum, default `maxLineEndExtension` is

$$minLE = cutWidth + sidewallWidth$$

and the `unlimited` argument is specified, the final `maxLineEndExtension` value is established by this algorithm:

- a. The cut solver is first run with the smallest reasonable value.

- b. The cut solver is iteratively run with incrementally increasing multiples of minLE on unsolved regions.
 - c. Remaining short circuits are detected for longer line-end extensions by a final run of the cut solver on these longer extensions using all previously located cuts as `unmovableCutCandidates`.
- `maxSadpMarkers = limit;`
An optional parameter that specifies the maximum limit of the number of `sadpMarker` shapes that may be processed from the input layout. The maximum value permitted is 1024. The default number is 128.
 - `minArea = area;`
An optional parameter that specifies the minimum area of a track remaining between two cuts. If two cuts leave an area of track between them less than `minArea`, only one of the two cuts is placed. One or both of the cuts that violate `minArea` may be put in the `illegalCuts` output layer. This validation also applies to cuts that are too close to an outer edge of the `sadpMarker` polygon that contains them. The `minArea` parameter does not apply to jogs or pads placed between two cuts. The default value is 0.
 - `minLithoGap = {gap | mandrel_gap, nonmandrel_gap};`
One of two optional numeric parameters specifying the minimum gap between target line-ends, below which require cut placement to lithographically resolve. The `minLithoGap` gap parameter applies to both mandrel and nonmandrel gaps. The `minLithoGap` mandrel gap and nonmandrel gap parameters are uniquely applied to mandrel and nonmandrel gaps, respectively. The `minLithoGap` parameter places cuts in gaps smaller than the specified `minLithoGap` value between line-ends, and between line-ends and an edge of an `sadpMarker`. If pitch alignment is enforced, the `minLithoGap` parameter is applied after the target line-ends are extended to legal cut locations in preparation for cut placement. For gaps between line-ends and the `sadpMarker` boundary, cuts are subject to existing `boundaryCuts` specifications, except that cuts are not placed in the event the gap is greater than or equal to the `minLithoGap` value.
If `minLithoGap` and `maxLineEndExtension` are both specified, a warning message is issued and `maxLineEndExtension` is ignored. If `minLithoGap` is less than `cutWidth`, a parsing error is issued, and the run terminates. Cuts not able to be legally placed with respect to the `minLithoGap` parameter are copied to the `illegalCuts` error layer. There is no default gap value.
 - `optimizeLineEnds = {critical | false | true};`
An optional parameter that controls line-end optimization. In most cases, using line-end optimization obviates most optional spacing constraints. When `optimizeLineEnds` is set to critical or true, and optional spacing constraints are included (`ignoreOptionalSpacing` is false), a warning message is issued. Set `ignoreOptionalSpacing` to true explicitly if required.

Note

 Siemens EDA recommends using `optimizeLineEnds` for optimal cut placement at line-ends especially where `criticalTimingMarkers` are present.

critical

Enables the optimization algorithm only for critical line-ends where `criticalTimingMarkers` are present. If no `criticalTimingMarkers` are present, the optimization algorithm does nothing, equivalent to setting `optimizeLineEnds` to false. This is the default.

false

Disables the optimization algorithm, including cuts where `criticalTimingMarkers` are present.

true

Enables the optimization algorithm to place cuts as close to all line-ends as possible. Placement priority is always given for cuts at critical line-ends over other line-ends. This setting normally results in slightly longer run times.

Note

 Line-end extension optimization and optional spacing constraints are mutually conflicting. Implementation of optional spacing constraints typically increase cut spacing to enhance manufacturability, and consequently place cuts farther from their ideal locations. Line-end optimization ignores optional constraints, resulting in cut placement relative to the line-end. In most cases, this eliminates the benefits of specifying optional spacing constraints. Consider one of three methods:

- Enable optional spacing constraints — This may enhance manufacturability, but at the cost of additional run time, and may result in cuts generated at a higher level in the hierarchy. To reduce run times, set `ignoreOptionalSpacing` to true.
 - Enable line-end optimization — Line-end optimization of critical line-ends is enabled by default. If optional spacing constraints are required, consider disabling line-end optimization by setting `optimizeLineEnds` to false.
 - Disable optional spacing constraints — Where a recipe prioritizes line-end locations, enabling optional spacing constraints increases run times, potentially with little process improvement. Consider disabling optional spacing constraints by setting `ignoreOptionalSpacing` to true.
-

- `patternMultiplier = value;`

An optional parameter that specifies the assignment frequency of the mandrel track. The multiplier may be set to any integer greater than or equal to 2. If the multiplier is set to 4, every fourth track would be assigned to the mandrel mask. If `patternMultiplier` is greater than 2, a warning message is issued when the `nonMandrelAnchor` input layer is present, even when the layer is empty. In this case, all non-mandrel anchors are ignored. The default pattern multiplier is 2, resulting in the default generation of alternating mandrel tracks. For

self-aligned quadruple-patterning (SAQP) applications, the patternMultiplier parameter must be set to 4.

- `pitchAlignCutOffset = ('offset0 [, offsetN, ...]');`

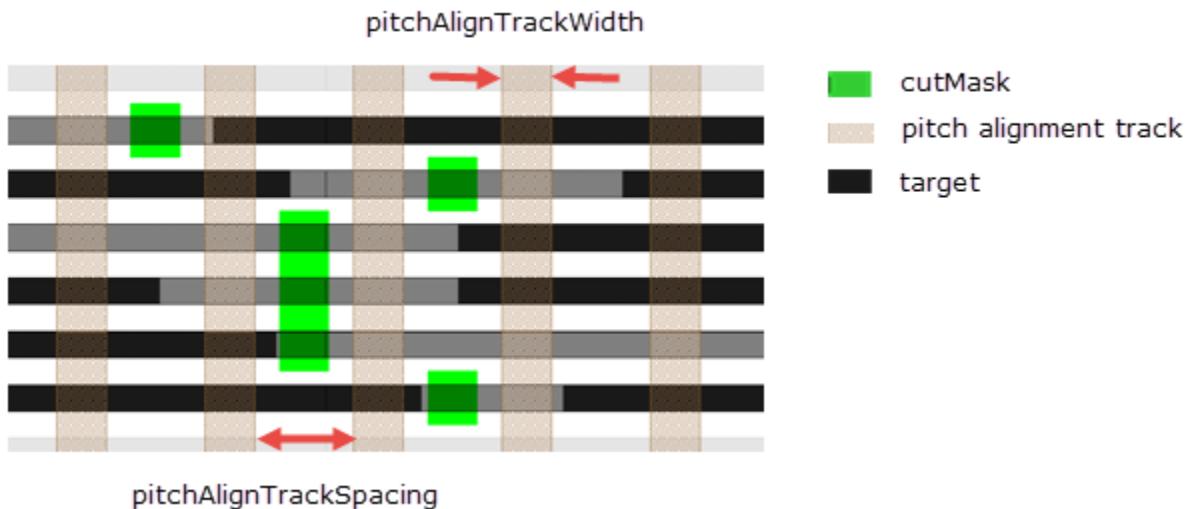
An optional list-based parameter that specifies an ascending sequence of positive, floating-point values for cut location offsets used to place cuts that terminate target shape ends. all specified offset values must be less than the pitch align period. The list must be specified with a minimum of two offset values, and any number of offset values may be specified.

The `pitchAlignCutOffset` list must be accompanied by specifications of both `pitchAlignTrackWidth` and `pitchAlignTrackSpacing`. The first offset is typically 0, indicating no offset, where the alignment of the cut is on the centerline established by `pitchAlignTrackSpacing`. Remaining offset values are qualified and selected as required.

- `pitchAlignTrackSpacing = spacing;`

An optional parameter that specifies the spacing between pitch alignment tracks. This parameter is applicable only when the `pitchAlign` layer is specified.

Figure 4-40. Pitch Alignment Track Parameters



- `pitchAlignTrackWidth = width;`

An optional parameter that specifies the width of pitch alignment tracks. This parameter is applicable only when the `pitchAlign` layer is specified.

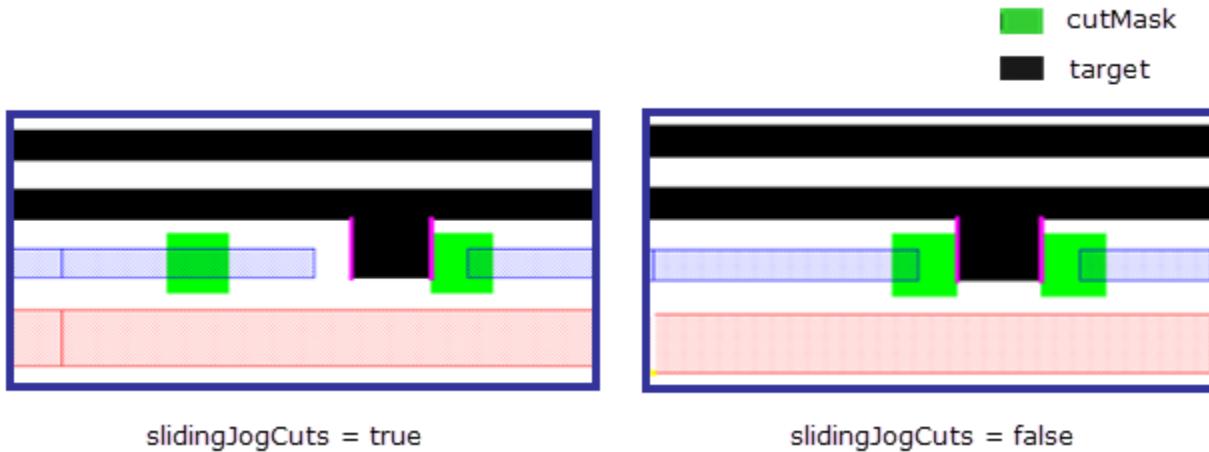
- `slidingCuts = {true | false};`

An optional parameter that specifies whether cuts are allowed to move away from line-ends to comply with design rule check constraints. The amount of movement allowed is determined by the `maxLineEndExtension` parameter. The default setting is true.

- `slidingJogCuts = {true | false};`

An optional parameter that specifies whether cuts are allowed to slide away from jogs. The default setting is the `allowEvenJogs` parameter setting.

Figure 4-41. slidingJogCuts



- `terminalDummyTrackJogWidth = width;`

An optional parameter that sets the minimum width of jogs within the last dummy track added to finish filling the sadpMarker layer. The dummy tracks are placed at the top and bottom for a mask of X-oriented tracks, or left and right sides for a mask of Y-oriented tracks. For jogs extending outside concave sadpMarker corners, you must use also specify the `terminationRegion` layer (see “[RET SIDCUTFILL](#)” on page 255). The default value is 0.

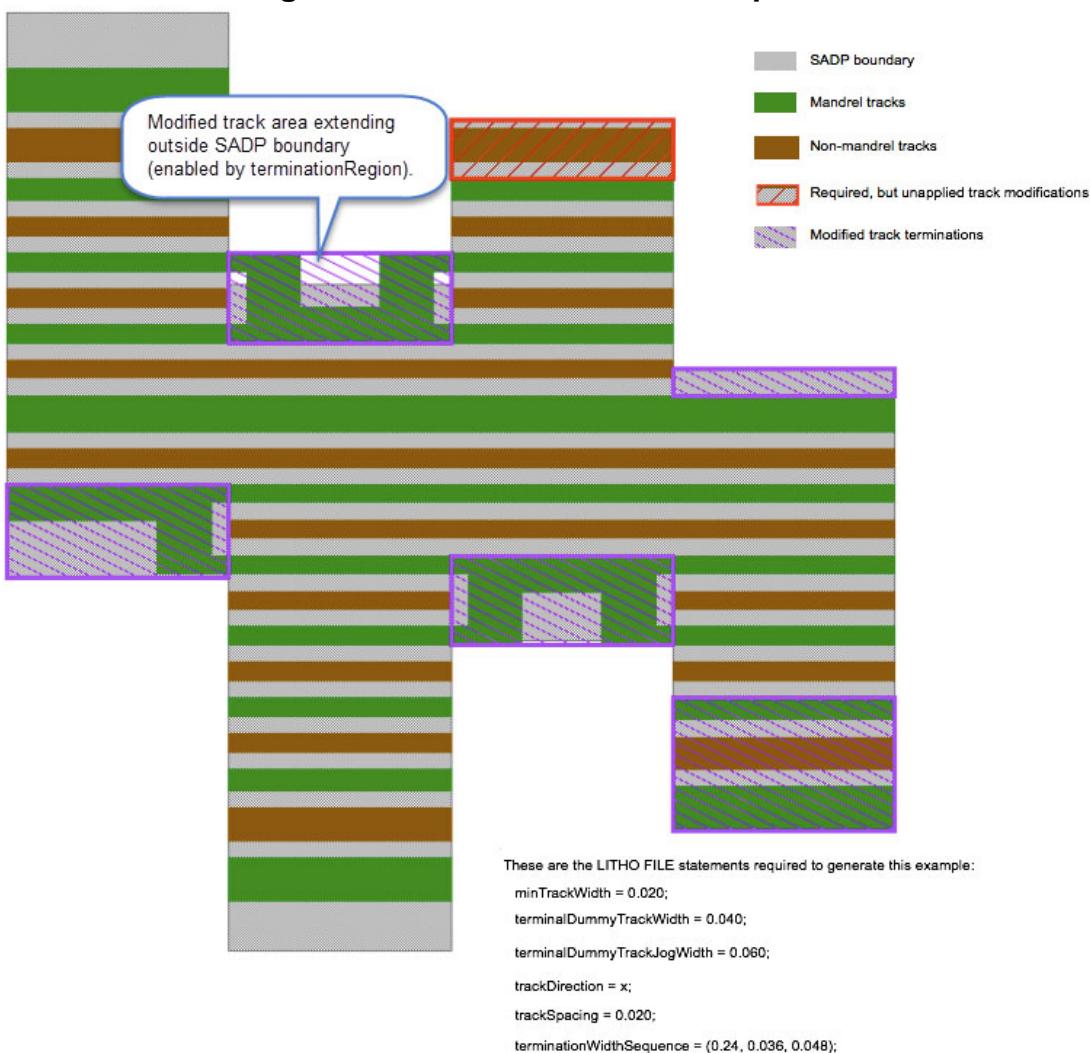
- `terminalDummyTrackWidth = width;`

An optional parameter that sets the minimum width of the last dummy track added to finish filling the sadpMarker layer. The dummy tracks are placed at the top and bottom for a mask of X-oriented tracks, or left and right sides for a mask of Y-oriented tracks. For tracks extending outside concave sadpMarker corners, you must use also specify the `terminationRegion` layer. The default is the value of `minTrackWidth`.

- `terminationWidthSequence = ('trackwidth1, trackwidth2 [, trackwidthN, ...]');`

An optional list-based parameter that specifies a sequence of track termination widths used to fill gaps at the end of sadpMarker regions. Any number of `terminationWidthSequence` lists may be specified, being tried in order by RET SIDCUTFILL from the largest total span to the smallest total span. The list is specified with at least two width values. The list must end with the desired track width closest to the sadpMarker edge. Any listed parameters less than the `minTrackWidth` parameter, or that are not specified within `trackWidth`, are considered legal.

Figure 4-42. terminationWidthSequence



- `trackSpacing = space;`

An optional parameter that sets the spacing between tracks for SADP.

Note

The `trackSpacing` parameter has been deprecated as of the 2020.1 release and is to be obsoleted as of the 2020.4 release. Use the `sidewallWidth` parameter instead.

- `trackTrim = distance;`

An optional parameter that specifies the distance a mandrel track is trimmed away from a non-mandrel pad or jog. This also applies to non-mandrel tracks that are trimmed away from mandrel pads and jogs. The default is equal to the `cutWidth` value as specified.

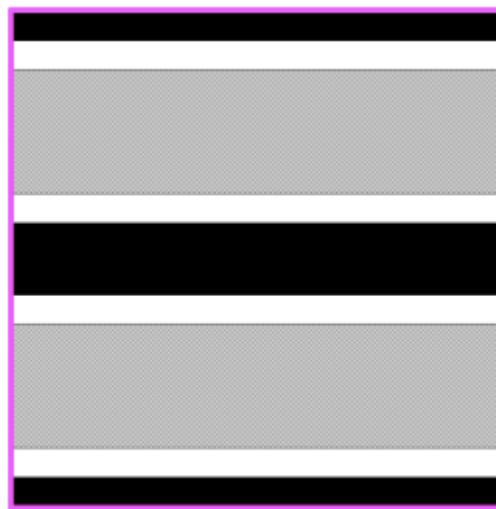
- `trackWidthSequence = ('targetwidth1, trackwidth1 [, trackwidthN, ...], targetwidth2');`

An optional list parameter that specifies a sequence of track widths used to fill gaps between target shapes. Any number of trackWidthSequence lists may be specified. The list is specified with at least three width values. The list must start and end with target shape widths, and there must be at least one intermediate track width value specified. If a specified track width is less than minTrackWidth, the specified value is still used and overrides minTrackWidth.

Figure 4-43. trackWidthSequence



`trackWidthSequence = (0.024, 0.032, 0.036, 0.024, 0.024, 0.036, 0.032, 0.024);`



`trackWidthSequence = (0.024, 0.100, 0.056, 0.100, 0.024);`

- `trimMaskMinJog = distance;`

An optional parameter that specifies the shortest length of an edge of a trim mask polygon with at least one concave corner. If both corners of the edge are convex, then

trimMaskMinWidth applies instead. Used for trim mask generation. An example is shown in [Figure 4-44](#) on page 286.

- trimMaskMinWidth = *distance*;

An optional parameter that specifies the smallest width allowable in the track direction for a trim mask polygon. Used for trim mask generation. An example is shown in [Figure 4-44](#) on page 286.

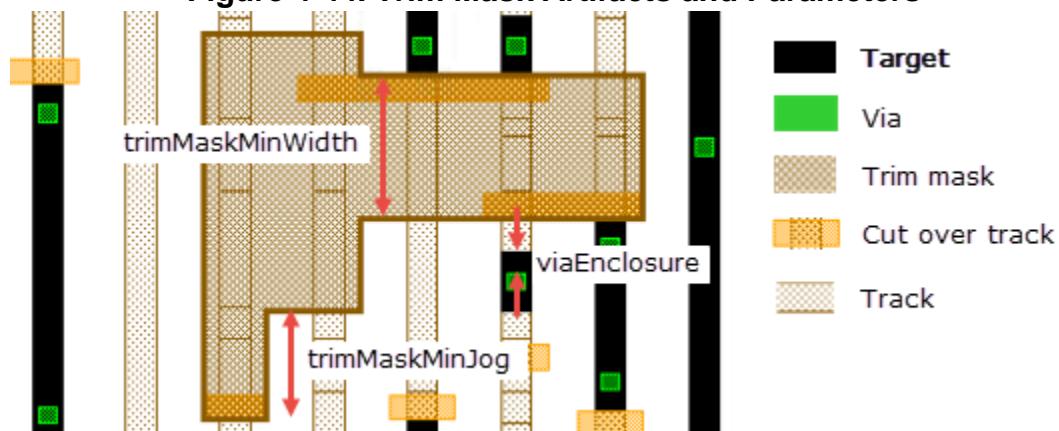
- trimTargetToMinViaEnclosure = {true | false};

An optional parameter used with viaEnclosure that specifies whether line-ends that extend farther than the required distance past a via may be trimmed back to match the required enclosure. Used for trim mask generation. An example is shown in [Figure 4-44](#) on page 286.

- viaEnclosure = *enclosure*; viaEnclosure_a = *enclosure*; viaEnclosure_b = *enclosure*; viaEnclosure_c = *enclosure*;

Optional parameters that specify positive, minimum distances of enclosure (length-wise over the track) of their associated vias and the end of the enclosing target apron. Used by the trim mask parameter trimTargetToMinViaEnclosure. An example is shown in [Figure 4-44](#) on page 286. The viaEnclosure parameter is not used to generate the trim mask, but is required during generation of the derivedTarget layer.

Figure 4-44. Trim Mask Artifacts and Parameters



Examples

Example 1

The following example demonstrates the use of the RET SIDCUTFILL command. This example is simple; many other keywords can be specified in the LITHO FILE statement for RET SIDCUTFILL.

Figure 4-45. RET SIDCUTFILL LITHO FILE and Commands

```
LAYER M1 1
LAYER sadp_boundary 2
LAYER mandrel_anchor 3
```

```
LITHO FILE sadp [
    inputLayerOrder = (target, sadpMarker, mandrelAnchor);

    dpFaceClass name(basic_cut) orientation(vertical):
        true for length <= 0.048;
        false otherwise;
    dpSpacing (any, any): 0.046 otherwise;
    dpSpacing (any, basic_cut) metric(opposite):
        0.064 for runLength >= 0.024;
        0 otherwise;

    sidewallWidth = 0.024;
    trackDirection = x;
    trackWidth = (0.024, 0.036, 0.048);
    trackWidthSequence = (0.048, 0.036, 0.024, 0.036, 0.048);
    cutWidth = 0.050;
]

mandrel = RET SIDCUTFILL M1 sadp_boundary mandrel_anchor
FILE sadp MAP mandrel
nonMandrel = RET SIDCUTFILL M1 sadp_boundary mandrel_anchor
FILE sadp MAP nonMandrel
```

SADP-Specific Spacing Parameters

SADP-specific cut spacing parameters are written within the LITHO FILE statement and referenced by the RET SIDCUTFILL command.

These parameters are only for application of SADP-specific cut spacing. With these parameters, edges of cuts may be grouped into classes, sets of classes, and varied spacing values then assigned between them.

Table 4-5. SADP-Specific LITHO FILE Statement and Parameters

Parameter	Description
dpFaceClass	Classifies cut edges into a named group.
dpFaceSet	Accumulates various user-defined cut face classes into a single set.
dpSpacing	Specifies spacing requirements between two face classes for SADP.

dpFaceClass

SADP-Specific Spacing Parameters

Classifies cut edges into a named group.

Usage

```
dpFaceClass name‘(‘faceclass_name‘)’ [options]:  
[true for constraint [and constraint...]:]  
{false | true} otherwise:
```

Arguments

- **name‘(‘faceclass_name‘)’ [options]:**

A required argument specifying the name of the user-defined face class. It must be enclosed in parentheses. All face classes must have a unique name.

Names may only contain alphanumeric characters, hyphens (-), and underscores (_). If name is specified only with otherwise, it must be followed by a colon (:).

One reserved face class, side, is defined as any polygon face that is not classified by dpFaceClass. The side face class can be referenced by the dpFaceSet and dpSpacing commands.

Note

 Although the reserved face class, side, is provided as a convenience for rule development and testing, Siemens EDA does not recommend its application in production rule files.

- **[class_options]**

The name argument has several options, the last of which must terminate all name arguments with a colon (:):

inRegion‘(‘faceclass_region_layer‘)

An optional argument specifying up to eight named regions with different spacing requirements inside a single sadpMarker layer. Any face class region declared must be mapped to an existing layer, even if that layer is empty. After one or more region arguments are declared, the layers mapped to them are used by dpSpacing for region-based measurements. See [Example 2 - Multiple Region Classifications and Measurements](#) for more information. The group of arguments comprising inRegion, orientation, and layerType must be terminated by a colon (:).

faceclass_region_layer — The layer specified by inRegion. The layers must adhere to the form faceClassRegion_?, where ? is one of a, b, c, d, e, f, g, or h.

Note



Face class regions impact cuts:

- Face class regions prevent merging of cuts between different regions.
 - Face class regions only apply if cut mask generation is specified.
 - User cuts and trapped cuts are never modified by face class regions and may span multiple face class boundaries. This may result in cuts being outside all defined face class regions.
 - Cuts may have representative markers output for each face class by mapping the dpFaceClass name as specified in the LITHO FILE.
-

`layerType('trimMask | via')`

An optional argument specifying a keyword to restrict the face class for trim mask operations.

trimMask — Restricts the face class to polygons on the generated trim mask.

via — Restricts the face class to polygons on the via input layer.

`orientation('relative_direction')`

An optional argument specifying one of two keywords of orthogonal directions by which to classify edges.

trackDirection, horizontal — Classifies only those edges that run parallel to the specified track orientation.

perpTrackDirection, vertical — Classifies only those edges that run perpendicularly to the specified track orientation.

- true for *constraint* [and *constraint...*]:

An optional argument specifying one or more constraints which must be true for an edge to be considered a member of the name face class. See the examples in this section for more information. The constraint argument must be terminated by a colon (:).

- {**false** | **true**} **otherwise**:

A required argument ensuring that edges not meeting any condition specified within the dpFaceClass command are classified. The otherwise argument must be terminated by a colon (:).

Description

A required command that classifies edges of shapes into groups referred to as face classes. Up to 32 dpFaceClass definitions may be classified in one LITHO FILE called by RET SIDCUTFILL.

Examples

Example 1 - Cut Edge Classification

This example shows two dpFaceClass specifications. The first creates a user-defined face class called long_cut, where those edge lengths perpendicular to the specified trackOrientation exceed standard track widths, indicating a terminal or multi-track cut. The second is a face class named std_cut, where edge lengths perpendicular to the specified trackOrientation indicate a standard track cut.

```
# Example using length and orientation conditions

dpFaceClass name(long_cut) orientation(perpTrackDirection):
    true for length > 0.048:
    false otherwise:

dpFaceClass name(std_cut) orientation(perpTrackDirection):
    true for length <= 0.048:
```

Example 2 - Multiple Region Classifications and Measurements

This example shows inRegion declaring two face_class_region_layers, faceClassRegion_a and faceClassRegion_b. The inRegion argument requires specification of a face_class_region_layer, and listing it with the inputLayerOrder keyword. In the example, the face class layers, cut_side_a and cut_side_b, are mapped to the inRegion arguments faceClassRegion_a and faceClassRegion_b, respectively. The face classes are then used in the subsequent dpSpacing command, where spacing values are specified selectively within the different face class regions.

```
# The regions are declared by dpFaceClass
dpFaceClass name(cut_side)
    orientation(perpTrackDirection): true otherwise:
dpFaceClass name(cut_side_a) orientation(perpTrackDirection)
    inRegion(faceClassRegion_a): true otherwise:
dpFaceClass name(cut_side_b) orientation(perpTrackDirection)
    inRegion(faceClassRegion_b): true otherwise:
# Wildcard use for inRegion (may lead to unintended results)
dpFaceClass name(cut_side_b) orientation(perpTrackDirection)
    inRegion(any): true otherwise:

# Various measurements are applied within regions by dpSpacing
dpSpacing (cut_side, cut_side) name(c2c): 0.050 otherwise:
dpSpacing (cut_side, cut_side_a) name(c2ca): 0.040 otherwise:
dpSpacing (cut_side_a, cut_side_a) name(ca2ca): 0.030 otherwise:
dpSpacing (cut_side_a, cut_side_b) name(ca2cb): 0.045 otherwise:
```

dpFaceSet

SADP-Specific Spacing Parameters

Accumulates various user-defined cut face classes into a single set.

Usage

```
dpFaceSet name('faceset_name'):
    faceclass [,faceclassN...]:
```

Arguments

- name('faceset_name'):

A required argument specifying the output name of the accumulated user-defined face class set. The faceset_name must be enclosed in parentheses. The dpFaceSet name must be unique. The name argument must be terminated by a colon (:).

- faceclass [, faceclassN...]:

A required argument specifying a previously declared cut face class name. Any optional number of additional face class names may be specified within the comma-separated list. The name of the new set must be unique. The faceclass name(s) must be terminated by a colon (:).

Description

An optional command that accumulates any number of previously defined face classes into a single face class set. One or more previously declared face class names must be specified. The dpFaceSet command must follow dpFaceClass commands declaring face classes. The resulting faceset_name can be used by subsequent dpSpacing and dpFaceClass commands. All face classes specified within a single dpFaceSet command must represent shapes of the same orientation and layer type. All face class sets must have a unique name.

Names may only contain alphanumeric characters, hyphens, and underscores.

Note

 The reserved class named 'side' is a repository for non-classified polygon faces. After use by a dpFaceSet list, the side class is emptied of all faces and has no effect for subsequent use by dpSpacing commands.

Examples

This example shows two dpFaceClass commands generating two face classes named basic_cut and long_cut. Both face classes are subsequently combined into a single face class set named verticalCutFace by dpFaceSet. The resulting face set is then used by dpSpacing.

```
dpFaceClass name(basic_cut) orientation(vertical):
    true for length <= 0.024:
    false otherwise; dpFaceClass name(long_cut) orientation(vertical):
    true for length > 0.024:
    false otherwise:

dpFaceSet name(verticalCutFace): basic_cut, long_cut:

dpSpacing (any, verticalCutFace) metric(opposite, extended 0.001)
    condition (numNeighbors > 1):
    0.030 otherwise:
```

dpSpacing

SADP-Specific Spacing Parameters

Specifies spacing requirements between two face classes for SADP.

Usage

dpSpacing ‘(*class1, class2*)’ [*class_options*]:

[*space for condition [and condition] ...:*]

space otherwise:

Arguments

- ‘(*class1, class2*)’ [*class_options*]:

A required pair of arguments enclosed in parentheses specifying the combination of face classes to which the settings apply. The specified classes are order-independent. Both classes can be the same. The face classes can be any of three class names:

- A user-defined face class classified by [dpFaceClass](#).
- The reserved class ‘side’, referencing any polygon face of a cut not defined by [dpFaceClass](#).
- The reserved wildcard ‘any’, referencing any defined face class or side.

Note



Although the reserved face class, side, is provided as a convenience for rule development and testing, Siemens EDA does not recommend its application in production rule files.

- [*class_options*]

The class options provide important adjunct functions to dpSpacing. If used, these options must be specified in the order as listed here and in the Usage section. The last class option must terminate the class arguments with a colon (:).

name ‘(*spacing_name*)’

An optional argument specifying a name to use for the space between face classes established by the names of class1 and class2. The name argument must be specified after the class names and precede all remaining optional arguments. The name argument is mutually exclusive with the condition argument numNeighbors.

spacing_name — The name associated to the spacing pair. The *spacing_name* must be enclosed in parentheses. Only alphanumeric characters, hyphens, and underscores are allowed in the name. The name parameters ‘any’ and ‘side’ are predefined.

Note

 Except when specifying the condition argument numNeighbors, the name argument allows declared spaces to be segregated by their respective names for independent analysis and subsequent layer output. By default, all spaces are combined into one output layer. When not specifying the condition argument numNeighbors, Siemens EDA recommends always using the name argument in dpSpacing commands.

`mask('0 | 1')`

An optional argument that applies spacing constraints to the specified mask. 0 and 1 applies the constraints to the mandrel, and non-mandrel cut masks, respectively. The 0 mask constraint must be specified first. If the mask keyword is not specified, the constraint is applied to both cut masks. This argument only applies to multi-cut masks. The mask keyword must be specified after the name and before the metric keyword.

0 — Applies the spacing only to mask 0.

1 — Applies the spacing only to mask 1.

`metric('euclidean | {opposite [, extended value]}')`

An optional keyword specifying the type of metric to use in measuring the spacing between face classes. Only one metric may be specified. The metric type must be enclosed in parentheses. Valid extended values must be a single number or expression. The metric keyword must be specified after the mask and before the condition keywords. For more information, see “[Metrics](#)” in the Calibre Verification User’s Manual.

euclidean — Forms regions with quarter-circle boundaries that extend past the corners of selected edges. This is the default metric.

opposite — Forms regions with right-angle boundaries that do not extend past corners of the selected edges.

opposite, extended value — Forms regions with right-angle boundaries that laterally extend past corners of selected edges, dependent on the specified extension value.

`condition('{numNeighbors value [, optional]} | optional')`

An optional keyword indicating either a non-required constraint, or the number of neighboring cuts to consider with the constraint. Either argument must be enclosed in parentheses. The condition keyword must be specified after the metric keyword.

`numNeighbors value [, optional]`

A value indicating the minimum number of neighboring cuts that are considered for error checking. The specified value must be 2, 3, or 4. The ‘optional’ argument may be added to indicate non-required application of spacing checks. The numNeighbors argument is mutually exclusive with the ‘name’ keyword.

`optional`

Applied only where the spacing is correct but not required. The ‘optional’ argument only applies when the slidingCuts parameter is true.

- *space for condition [and condition] ...:*

An optional set of arguments specifying the required space in microns between two face classes where the specified condition is met. The condition is specified with at least one of five arguments:

minLength — The shorter of the two face lengths.
maxLength — The longer of the two face lengths.
minWidth — The width of the target shapes associated with the face classes.
maxWidth — The larger of the two widths.
runLength — The shared continuous edge projection of both face classes. Perpendicular edges are considered as 0 runLength.

The condition must take the form of:

argument operator value

For example:

```
minWidth < 0.030:
```

Multiple conditions can be joined with the ‘and’ modifier:

```
minWidth <= 0.032 and runLength > 0.060:
```

You can specify any space as a variable within the dpSpacing command. For example:

```
VARIABLE minTrackWidth 0.024
LITHO FILE [
    dpSpacing(class1, class2) name(c2c) :
    minWidth < $min_track_width:
    $default_space otherwise:
    ...
]
```

- ***space otherwise:***

A required argument ensuring that edges not meeting any condition specified within the dpFaceClass command are classified. The space-otherwise argument must be terminated by a colon (:). For example:

```
0.030 otherwise:
```

You can specify both space-for-condition and space-otherwise together. Both argument sets must be individually terminated with a semicolon (;). For example:

```
minWidth < 0.045:
0.030 otherwise:
```

Description

A required command that measures distances between cut shapes. The dpSpacing command defines spacing dimensions between face classes established by [dpFaceClass](#). Only one measurement is allowed for each classified edge pair; if dpSpacing(A, B) is specified,

dpSpacing(B, A) would result in a compile error. The optional group of arguments comprising name, mask, metric, and condition must be terminated by a colon (:). To apply user-defined face classes, all dpSpacing commands must follow dpFaceClass commands in the SVRF file.

Examples

Example 1 — Single Value with Default Metric

This example uses dpSpacing to check spacing of 54 nm between edges of the reserved face class named ‘side’ (no name is declared):

```
dpSpacing (side, side) 0.054 otherwise:
```

Example 2 — Single Value with User-Defined Class

This example uses the dpFaceClass command to create a face class called basic_cut. The dpSpacing command then establishes a spacing of 0.030 um between the edges of the basic_cut face class and any other edge:

```
dpFaceClass name(basic_cut) orientation(vertical) :  
    true for length <= 0.024:  
    false otherwise:  
    dpSpacing (any, basic_cut): 0.030 otherwise:
```

Example 3 — Conditional Values

This example uses the dpSpacing command to check spacing between edges of classes jog and mandrelTrack for 30 nm only where a contiguous edge projection of at least 130 nm exists:

```
dpSpacing (jog, mandrelTrack) name(j2m) :  
    0.030 for runLength >= 0.130:  
    0.020 otherwise:
```

Example 4 — Extended Metric

Using the opposite extended metric for line-end measurements often provides better corner accuracy than euclidean, the default metric. This example uses the opposite extended metric with an extension of 10 nm to define a spacing of 45 nm unless the faces are on compliant shapes (the minimum where width is greater than or equal to 30 nm):

```
dpSpacing (any, cut) name(cut_space) metric(opposite, extended 0.010) :  
    0.045 for minWidth < 0.030 and maxLength >= 0.060:  
    0.030 otherwise:
```

Example 5 — More Permutations

These code examples illustrate further syntax permutations of the dpSpacing command.

```
dpSpacing (side,side) name(s2s): 0.06 otherwise:  
dpSpacing (any,cut) name(cut_space): 0.032 for minWidth <= 0.032:  
    0.06 otherwise:  
dpSpacing (cut,cut) name(c2c) metric(opposite): 0.06 otherwise:  
dpSpacing (any, basic_cut_end) name(ce_space)  
    metric(opposite, extended 0.024):  
    $mxCut_s_2 otherwise:  
dpSpacing (any, verticalCutFace)  
    metric(opposite, extended 0.001)  
    condition(numNeighbors 2):  
    $twoNeighborSpacing otherwise:  
dpSpacing (any, verticalCutFace) name(s_3) metric(opposite):  
    $mxCut_s_3 for runLength >= $minTrackWidth:  
    0 otherwise:
```

Example 6 — numNeighbors Condition

The numNeighbors argument is helpful to locate cuts with multiple spacing violations. This code example demonstrates two- and three-neighbor spacing checks.

```

dpFaceClass name(basic_cut) orientation(vertical):
    true for length <= $smallCutLength:
    false otherwise:

dpFaceClass name(long_cut) orientation(vertical):
    true for length > $smallCutLength:
    false otherwise:

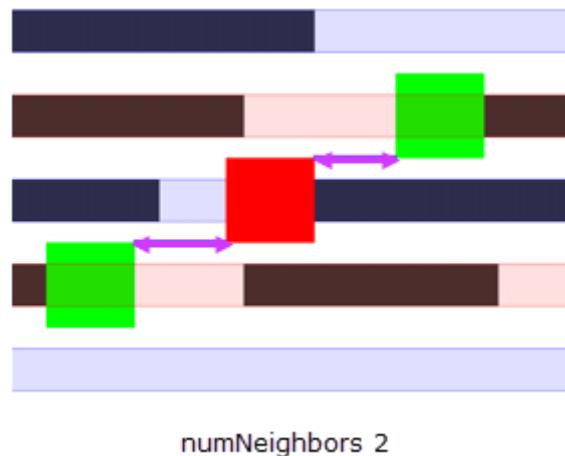
dpFaceSet name(veritcalCutFace): basic_cut, long_cut:

dpSpacing (any, veritcalCutFace) metric(opposite, extended 0.001)
    condition(numNeighbors 2):
    $twoNeighborSpacing otherwise:

dpSpacing (any, veritcalCutFace) metric(opposite, extended 0.001)
    condition(numNeighbors 3):
    $threeNeighborSpacing otherwise:

```

Figure 4-46. numNeighbors Condition



Chapter 5

Set Up and Run Calibre Multi-Patterning

You use Calibre Multi-Patterning to perform mask decomposition. Because Calibre Multi-Patterning tools run in batch mode, you configure all settings in a LITHO FILE statement, which is called from an SVRF file.

calibre	302
Decomposing Whole Shapes	304
Decomposing Stitched Shapes.....	306
Anchor Applications for Double-Patterning.....	309
Stitch Applications for Double-Patterning	311
Creating User-Drawn Stitches	312
Creating Pre-colored Stitches	312
Generating SVRF Custom Stitches.....	313
Auto-generating Stitches with DPSTITCH	314
Validation of Stitches	315
Control of Stitch Behavior with Priority and Mask Settings	320
Stitch and Priority Interaction.....	322
Using SVRF Variables within a LITHO FILE Statement.....	323
DP Debugging Methodology	324
Debugging DP Layout Errors.....	327
Fixing DP Layout Errors	333
MP Visualization and Error Resolution	335

calibre

Linux® command

Invokes the SVRF-based Multi-Patterning command set.

Usage

calibre -drc

```
[-hier]
[-hyper [remote]]
[-remotedata [-recoveroff | -recoverremote]]
[-remote hostname [,hostname ...]]
[-remotefile filename]
[-remotecommand filename #CPUs]
[-turbo [n]]
svrf_filename
```

Arguments

- **-drc**
A required argument that specifies a Calibre nmDRC run. This calls the Calibre database constructor.
- **-hier**
An option that specifies hierarchical DRC (Calibre nmDRC-H). Although optional, Siemens EDA recommends its use.
- **-hyper [remote]**
An optional switch that enables hyperscaling mode. It can only be specified in conjunction with -turbo. The -hyper argument functionality requires Calibre® MTflex™ capability to be available. The remote option to hyper triggers remote pseudo-hierarchical database technology. The number of parallel operations increases when -hyper remote is used in conjunction with the RDS technology enabled by -remotedata, and this argument must be used in conjunction with the -remotedata switch. If you specify -turbo 1 -hyper or if you only have one CPU available, Calibre issues a warning that only one CPU is used and continues processing.
- **-remotedata [-recoveroff | -recoverremote]**
An optional argument that enables Remote Data Server (RDS) technology. This argument is required for -hyper remote to function.
- **-remote *hostname* [,*hostname* ...]**
An optional argument that is used in a homogeneous network environment to specify one or more remote host names to use for the Calibre MTflex run.

- **-remotecmd *filename #CPUs***

An optional argument that enables auto-generation of a Calibre MTflex configuration file. The purpose of the -remotecmd option is to simplify usage in LSF and GRID-based environments by eliminating the need for a configuration file. You must specify remote command file and number of CPUs to use. The -turbo option is required for -remotecmd.

- **-remotefile *filename***

An optional argument that enables multi-threaded operation of a distributed network (Calibre MTflex mode) by specifying a filename containing a list of primary or remote hosts of any platform. The -turbo option is a prerequisite for -remotefile.

- **-turbo [n]**

An optional argument that enables multi-threaded processing (Calibre® MT mode) for DRC and DFM operations with the number of CPUs you specify. By default, all available CPUs are used. For best performance, Siemens EDA recommends you specify no value with -turbo. The -hier option is a prerequisite to -turbo. The -turbo option alone handles most SVRF processes except LITHO. If you set a value for -turbo that is higher than the available number of CPUs, Calibre runs on the maximum number of CPUs available.

- ***svrf_filename***

A required argument that specifies the SVRF filename.

Description

Executes any number of Calibre-related commands and their options from the command line.

Table 5-1. Command Compatibility of Processing Modes

Command	MT	MTflex	-hyper [remote]
RET dpStitch	Yes	Yes	Compatible
RET nmDPC	Yes	Yes	Yes, requires -hyper remote and HDB memory to exceed 1/4 of host
RET SIDCUTFILL	Yes	Yes	Yes, excepting cut location computations
DFM BALANCE	Yes	Yes	Yes
DFM DP	Yes	Yes	Yes
DFM MP	Yes	Yes	Yes
RET TP	Yes	Yes	Compatible
RET QP	Yes	Yes	Compatible

Note

 Before invoking Calibre, you must set your environment variables. Calibre tools require that the MGC_HOME environment variable be set. For more information regarding Calibre MT, MTflex and Hyperscaling processing modes, see the [Calibre Administrator's Guide](#).

Decomposing Whole Shapes

Whole-shape layer decomposition is performed using Calibre nmDPC. You control whether specific sets of shapes go onto same or different masks, or allow Calibre to automatically place them.

Prerequisites

- Layout in GDS or OASIS format.
- Minimum spacing rules between shapes on the same mask.
- SVRF file defining input, deriving any necessary layers, and naming at least one results database (RDB).

Procedure

1. If you want to embed the setup file in the SVRF file, add a LITHO FILE statement to the SVRF file.

Where commands are identified for inclusion in the setup file, add them between the opening and closing brackets, each on their own line. Each LITHO FILE statement in an SVRF file must have a unique name.

2. Identify any pairs of shapes that need to go on separate layers by deriving a layer in SVRF and identifying it in the setup file.
 - a. Derive the layer(s) in SVRF. You can have multiple layers, and assign them different priorities.

Most often the separation requirement is due to spacing violations. The pairs are marked with separators, edge pairs you can derive programmatically using SVRF operations, for example:

```
m1Separator = EXTERNAL m1 < 0.06 OPPOSITE PARALLEL ONLY PROJ > 0
```

This generates an edge pair along facing edges of two m1 shapes that are less than 60 nm apart. Assuming that the minimum spacing between two shapes on the same mask is 60 nm, these two shapes must go on separate mask layers.

- b. Identify the layer(s) in the setup file with action statements.

The action statement for the example in (a) could take the following form:

```
action m1Separator has nicety of 0 and requires opposite masks;
```

The priority is set by the number. Priorities of 0 and negative values are considered critical, which means that if the shapes cannot be separated onto separate masks a violation is reported. Priorities of 1 or higher do not generate a violation.

3. Identify any shapes that must be placed on the same layer as another shape or must be placed on a particular mask.
 - If you can identify groups of shapes, use action layers with same instead of opposite in the setup file. You may need to create an original layer in the layout with polygons to identify these groups if they cannot be determined by rule checks.
 - To associate a shape with a particular mask, use anchor layers. Some foundries require that the shapes on the anchor layer are coincident with the shapes on the drawn layer, but Calibre anchors any drawn shape that overlaps a shape on the anchor layer. Assuming you have derived a layer m1_vdd to identify power nets and that you want them on mask 0, the corresponding anchor statement in the setup file is as follows:

```
anchor m1_vdd has nicety of 0 is for mask0;
```

To minimize conflicts, use anchors sparingly.

4. Add the calls to Calibre nmDPC to the SVRF file with RET NMDPC. This SVRF operation calls the setup file and outputs various types of layers. The output can be captured with either rule checks or derived layer statements, which you then save to the RDB.

When a drawn layer cannot be separated into two masks satisfying the conditions given by critical action layers, a conflict layer is created. The conflict layer contains only the shapes that cannot be resolved. For easier resolution, also output the ring and warning layers, or the loop layer. The different layers are indicated by the MAP keyword in the RET NMDPC statement.

```
conflict = RET NMDPC action drawn FILE setup.in MAP conflict
mask0   = RET NMDPC action drawn FILE setup.in MAP mask0
mask1   = RET NMDPC action drawn FILE setup.in MAP mask1
errors   = RET NMDPC action drawn FILE setup.in MAP ring
at_risk  = RET NMDPC action drawn FILE setup.in MAP warning
```

Conflicts caused by critical anchor layers are output separately, using the anchor_conflict and anchor_path MAP keywords.

If the RET NMDPC statements are identical aside from the MAP keyword, the operation is run only a single time to produce all the different layer outputs.

Note

 Calibre nmDPC may not always begin its processing from the same shape. This can lead to different, but correct mask assignment results between runs. In particular, flat and hierarchical runs are unlikely to start at the same shape. Both results are still valid.

When using derived layers, write the layers to the RDB with DRC CHECK MAP:

```
conflict {COPY conflict} DRC CHECK MAP conflict 5
mask0   {COPY mask0}      DRC CHECK MAP mask0 10
mask1   {COPY mask1}      DRC CHECK MAP mask1 11
errors  {COPY errors}    DRC CHECK MAP errors 20
at_risk {COPY at_risk}   DRC CHECK MAP at_risk 21
```

5. Run DRC checks to validate output.

Results

After the run completes, the results database contains the layers you have written, although either the mask layers or the conflict layers may be empty.

Related Topics

[LITHO FILE](#)

[RET NMDPC](#)

[action](#)

Decomposing Stitched Shapes

Stitching shapes is performed by Calibre nmDPC. Valid stitch areas can be determined manually or automatically using RET DPSTITCH. RET DPSTITCH calls the LITHO FILE statement and outputs a stitch candidates layer. RET NMDPC then inputs the stitch candidates layer to produce final stitches.

Prerequisites

- Layout in GDS or OASIS format
- Minimum spacing rules between shapes
- SVRF file defining input, deriving any necessary layers, and naming at least one results database (RDB)

Procedure

1. If you want to embed the setup files in the SVRF file, add two LITHO FILE statements to the SVRF file. RET NMDPC and RET DPSTITCH require separate setup files.

Where commands are identified for inclusion in the setup file, add them between the opening and closing brackets, each on their own line. Each LITHO FILE statement in an SVRF file must have a unique name.

2. If the layout includes precoloring, manual stitches, or keepout markers, add the necessary SVRF commands. For example:

```

LAYER M1 200 //Metall Layer
LAYER M1_FILL 300 //Fill Layer
LAYER M1_0 1000 //Metall Pre-Color0
LAYER M1_1 1001 //Metall Pre-Color1
LAYER M1USERSTITCH 1002 //Metall User Stitches
LAYER M1USERSTITCHKEEPOUT 1003 //Metall User Stitch Keepout
...
drawn = DFM COPY M1 M1_FILL //collects all shapes to be decomposed
mask0_precio = M1_0 NOT M1_1
mask1_precio = M1_1 NOT M1_0
user_stitches = (M1_0 AND M1_1) OR M1USERSTITCH
keepout = DFM COPY M1_0 M1_1 M1USERSTITCHKEEPOUT

```

3. In the setup file for RET DPSTITCH, define suitable stitch conditions.

Some common methods of implementation are shown in the examples. For a complete list, see “[setlayer dpstitch options Commands](#)” on page 180.

4. Output the stitch candidate layer and possibly others from the RET DPSTITCH setup file.
 - a. In the options block, add -outLayer arguments to [dpStitchCandidates](#) for each layer you want available.

The -outLayer stitchCandidates argument outputs the stitch candidate layer. The -outLayer stitchCandidates argument must be present and be the first -outLayer argument in the dpStitchCandidates command. When developing the decomposition rules, you may also want to see the separators (-outLayer separators) or face classes (-outLayer <class_name>).

- b. Outside the options block but still within the LITHO FILE statement, add [setlayer dpstitch](#) commands to make the dpStitchCandidates layers available.

The order of layers specified in the options block and the setlayer dpstitch command must match. The names can be different, but their order determines which layer data is passed from the options block to the setlayer dpstitch command. These statements take the form:

```

setlayer <name> = dpstitch <drawn> <other_layers> \
    OPTIONS opt \
    MAP <outLayer_value>

```

- c. In the SVRF file, add derived layer statements to make the layers available to other SVRF calls.

These derived layer statements also invoke Calibre dpStitch.

```

<layer> = RET DPSTITCH <drawn> <other_layers>
FILE <setup_file>
MAP <name>

```

The <name> for MAP must match one of the setlayer <name> values. This determines which layer is being passed. Add a derived layer statement for each output layer you want to view or use in the RET NMDPC operation.

5. For the layers you want to view, add a rule check in the SVRF file to write them to a results database:

```
dpSC {COPY candidates} DRC CHECK MAP dpStitchCan 50
```

6. Add the calls to the SVRF file with [RET NMDPC](#).

This SVRF operation calls the setup file and outputs various types of layers. The output can be captured with either rule checks or derived layer statements, which you then save to the RDB. When a drawn layer cannot be separated into two masks satisfying the conditions, a conflict layer is created. The conflict layer contains only the shapes that cannot be resolved.

- a. (Optional) For easier resolution, also output the ring and warning layers, or the loop layer.

The different layers are indicated by the MAP keyword in the RET NMDPC statement.

```
conflict = RET NMDPC sCandidates drawn FILE setup.in MAP conflict
mask0   = RET NMDPC sCandidates drawn FILE setup.in MAP mask0
mask1   = RET NMDPC sCandidates drawn FILE setup.in MAP mask1
errors   = RET NMDPC sCandidates drawn FILE setup.in MAP ring
at_risk  = RET NMDPC sCandidates drawn FILE setup.in MAP warning
```

Conflicts caused by critical anchor layers are output separately, using the anchor_conflict and anchor_path MAP keywords.

- b. When using derived layers, write the layers to the RDB with DRC CHECK MAP:

```
conflict {COPY conflict} DRC CHECK MAP conflict 5
mask0   {COPY mask0}      DRC CHECK MAP mask0 10
mask1   {COPY mask1}      DRC CHECK MAP mask1 11
errors   {COPY errors}    DRC CHECK MAP errors 20
at_risk  {COPY at_risk}   DRC CHECK MAP at_risk 21
```

7. Run DRC checks to validate output.

Results

After the run completes, the results database contains the layers you have written, although either the mask layers or the conflict layers are empty.

Related Topics

[LITHO FILE](#)

[RET DPSTITCH](#)

[RET NMDPC](#)

Anchor Applications for Double-Patterning

The anchor0 and anchor1 keywords assign target shapes to either mask0 or mask1. They are properly assigned to either mask when they touch, overlap, or enclose the anchor shapes. Anchor assignment is flexible and suitable for many process types.

Anchor Applications

Anchors may be applied in a variety of styles to accommodate design constraints.

- **Seed** — An anchor shape is placed so that it merely intersects the target shape, either overlapping, coincident or enclosed by the target shape. The extended parts of the anchor do not add to the target shape.

Figure 5-1. Seed Anchors



- **Identical** — An anchor shape is drawn or derived to identically match the target shape to be assigned.

Figure 5-2. Identical Anchors



Because Calibre double-patterning permits all these methods of mask assignment, if you do not want one or more of these methods included, exercise care to prevent them from being included in the SVRF rules.

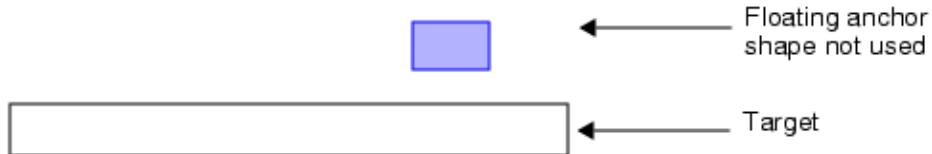
Incorrect Anchor Usage

This section describes incorrect methods of anchor usage given appropriate stitch candidates are not provided to nmDPC to resolve multiple and overlapping anchor usage. For more information, see “[Stitch Applications for Double-Patterning](#)” on page 311.

- **Floating anchor shapes** — Drawn anchor shapes that do not interact with target shapes are ignored. Although floating anchor shapes are not reported by default, you may use additional SVRF statements to report them as depicted in [Figure 5-3](#) on page 310. For example:

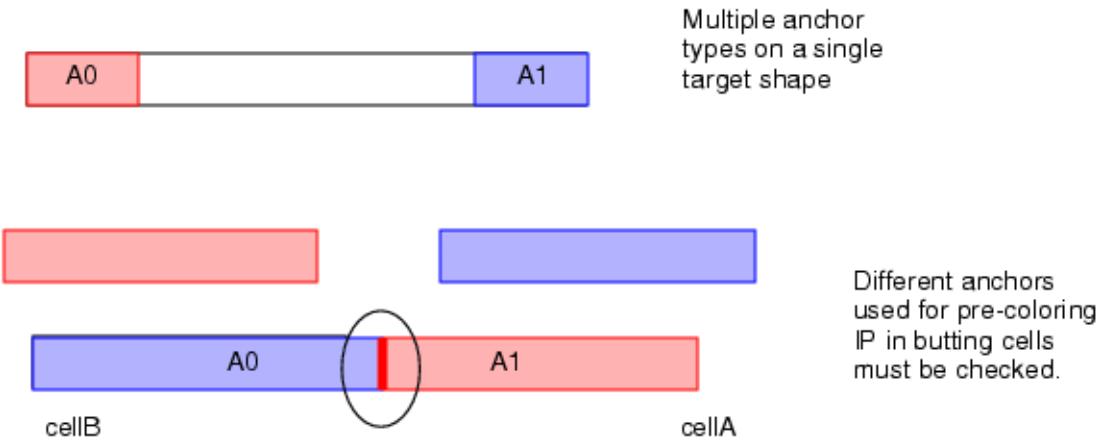
```
floating_anchor = input_anchor NOT INTERACT target
```

Figure 5-3. Floating Anchors



- **Multiple anchor shapes** — More than one anchor shape of different colors drawn on a single target shape. Target shapes with multiple anchor shapes are reported with the anchor_self_conflict error layer. In addition, the coloring engine arbitrarily picks one anchor and generates mask output. This also applies to interacting anchors stored at various hierarchical levels of the design.

Figure 5-4. Multiple Anchors



- **Overlapping anchors** — More than one anchor shape of different colors drawn on a single target shape that overlap. Target shapes with overlapping anchor shapes are reported with the anchor_self_conflict error layer. In addition, the coloring engine arbitrarily picks one anchor and generates mask output.

Figure 5-5. Overlapping Anchors



Stitch Applications for Double-Patterning

The NMDPC command invokes stitch candidate decomposition using the action stitch keyword in the LITHO FILE statement.

Stitch candidate polygons come from potentially four sources:

- User-drawn stitches with designated design layers
- Pre-colored stitches formed by anchor-overlaps
- Auto-generated stitches using custom SVRF
- Auto-generated stitches using dpStitch

Despite the stitch origin, the NMDPC command makes certain assumptions regarding the input stitches. If the stitches provided do not adhere to these assumptions, there may be unwanted results in the mask output from NMDPC. Nicety and mask settings dramatically alter the treatment of stitches. If stitch settings do not match the intention of the design application, then unwanted results in mask output may occur.

The NMDPC command has no inherent detection of the stitch sources mentioned previously, unless the shapes representing the sources are passed to NMDPC using the stitch keyword. This table describes the sources and action required by the tool.

This table shows these stitch methods as described above and their subsequent validation performed by the SVRF rules writer.

Table 5-2. Stitch Methods and Validation

Task	Description
Creating User-Drawn Stitches	User-drawn stitches can be helpful for controlling NMDPC.
Creating Pre-colored Stitches	Pre-colored stitches established with anchors retain predetermined colors for cell library and third-party IP for NMDPC.
Generating SVRF Custom Stitches	You can develop your own stitches that are input to NMDPC.
Auto-generating Stitches with DPSTITCH	Auto-generation of stitches using DPSTITCH is the least SVRF-intensive method.
Validation of Stitches	Calibre Multi-Patterning provides checks and operations during execution of dpStitch that ensure stitches adhere to the design assumptions described in this section. When generating stitches with methods other than dpStitch, you are responsible for ensuring adherence to these assumptions.

Table 5-2. Stitch Methods and Validation (cont.)

Task	Description
Control of Stitch Behavior with Priority and Mask Settings	Even though stitches may be user-drawn or generated by dpStitchCandidates, they are still considered candidates until appropriated by the accompanying priority and mask-specific settings. Generated stitches are dependent on these settings.
Stitch and Priority Interaction	Priority and mask settings specified for individual action stitch commands produce significant interactions between auto-stitches, user stitches and separators.

Creating User-Drawn Stitches

User-drawn stitches can be helpful for controlling NMDPC.

Prerequisites

- A [LITHO FILE](#) statement containing an options block.

Procedure

1. Read in the original stitch layers, for example:

```
LAYER m1_user_stitch 1002 //Metall1 User Stitches
```

2. Validate the stitch shapes, for example:

```
rect_user_stitch = RECTANGLE m1_user_stitch
user_stitch_min_space = EXT rect_user_stitch < 0.140
    OPPOSITE PARALLEL ONLY PROJ > 0
user_stitch_min_width = INT rect_user_stitch < 0.032
    REGION
user_stitch_min_space { DFM COPY user_stitch_min_space }
user_stitch_min_width { DFM COPY user_stitch_min_width }
```

3. Input stitches to action stitch command, for example:

```
LITHO FILE m1dpcfile [
    action rect_user_stitch :stitch 0 1;
    ...
]
```

Creating Pre-colored Stitches

Pre-colored stitches established with anchors retain predetermined colors for cell library and third-party IP for NMDPC.

Prerequisites

- A LITHO FILE statement containing an options block.

Procedure

1. Read in the original stitch layers, for example:

```
LAYER m1_user_stitch 1002 //Metall1 User Stitches
LAYER anchor_0 1003 //Metall1 User Stitches
LAYER anchor_1 1004 //Metall1 User Stitches
```

2. Derive overlapping anchors, remove overlapped anchors from original anchors and combine with original stitches, for example:

```
olap_anchors = AND anchor_0 anchor_1
anchor0 = NOT anchor_0 olap_anchors
anchor1 = NOT anchor_1 olap_anchors
all_user_stitch = COPY m1_user_stitch olap_anchors
```

3. Validate the stitch shapes and output errors, for example:

```
m1_stitch_candidates = RECTANGLE all_user_stitch
user_stitch_min_space = EXT m1_stitch_candidates < 0.140
    OPPOSITE PARALLEL ONLY PROJ > 0
user_stitch_min_width = INT m1_stitch_candidates < 0.032
    REGION
user_stitch_min_space { DFM COPY user_stitch_min_space }
user_stitch_min_width { DFM COPY user_stitch_min_width }
```

4. Input stitches to action :stitch command, for example:

```
LITHO FILE m1dpcfile [
    action m1_stitch_candidates :stitch 0 1;
    ...
```

Generating SVRF Custom Stitches

You can develop your own stitches that are input to NMDPC.

Prerequisites

- A LITHO FILE statement containing an options block.

Procedure

1. Develop custom stitches with SVRF, for example:

```
LAYER m1 101 //Metall1 Target Layer
LAYER U_STITCH 201 //User drawn stitches
```

2. Validate the stitch shapes, for example:

```
m1_user_stitch = U_STITCH
rect_user_stitch = RECTANGLE m1_user_stitch

user_stitch_min_space = EXT rect_user_stitch < 0.140
    OPPOSITE PARALLEL ONLY PROJ > 0
user_stitch_min_width = INT rect_user_stitch < 0.032
    REGION
...
...
```

3. Input stitches to action :stitch command, for example:

```
LITHO FILE m1dpcfile [
    action m1_dpstitch_candidates_all :stitch 0 1;
    ...
]
```

Auto-generating Stitches with DPSTITCH

Auto-generation of stitches using DPSTITCH is the least SVRF-intensive method.

Prerequisites

- A Design Rule Manual (DRM) providing process-specific dimensions.
- A [LITHO FILE](#) statement containing an options block.

Procedure

1. Read in the original stitch and anchor layers, for example:

```
LAYER m1_target 1001 //Metal1 Target Layer
LAYER m1_anchor0 1002 //Metal1 Anchor0 Layer
LAYER m1_anchor1 1003 //Metal1 Anchor1 Layer
```

2. Generate stitch candidates with dpStitch, for example:

```

options M1_stitch {
    version 1
    layer m1_target opc clear
    layer m1_anchor0 visible clear
    layer m1_anchor1 visible clear

    dpStitchSize \
        (0.04 <= length <= 0.05) for (0 < width <= 0.03): \
        (0.06 <= length <= 0.08) for (0.03 < width <= 0.06): \
        0.08 otherwise:

    dpStitchCandidates \
        criticalDistance 0.032 \
        anchor0 m1_anchor0 \
        anchor1 m1_anchor1 \
        -outLayer stitchcandidates \
        -outLayer separators \
    }

setlayer stitchcandidates = dpstitch m1_target \
    m1_anchor0 m1_anchor1 \
    MAP stitchcandidates OPTIONS M1_stitch

setlayer separators = dpstitch m1_target \
    m1_anchor0 m1_anchor1 \
    MAP separators OPTIONS M1_stitch

```

3. Input stitches, separators and anchors to action commands, for example:

```

LITHO FILE m1dpcfile [
    action stitchcandidates :stitch 0 1;
    anchor m1_anchor0 0 0;
    anchor m1_anchor1 0 1;
    action m1_separators 0 1;
    resolve;
]

m1_mask0 {RET NMDPC m1_separators m1_target m1_anchor0 m1_anchor1 \
    stitchcandidates FILE m1dpcfile MAP MASK0
}

m1_mask1 {RET NMDPC m1_separators m1_target m1_anchor0 m1_anchor1 \
    stitchcandidates FILE m1dpcfile MAP MASK1
}

```

Validation of Stitches

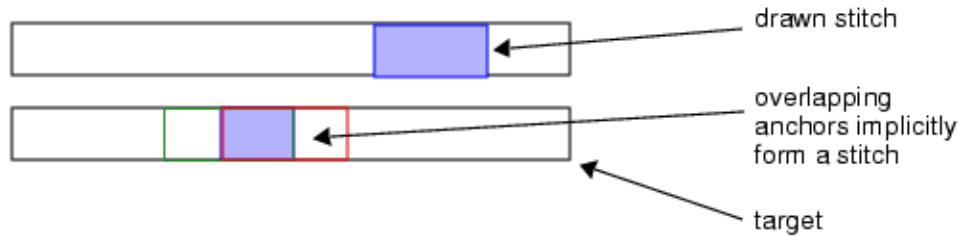
Calibre Multi-Patterning provides checks and operations during execution of dpStitch that ensure stitches adhere to the design assumptions described in this section. When generating stitches with methods other than dpStitch, you are responsible for ensuring adherence to these assumptions.

These same checks and operations must be applied to SVRF-derived stitches and separators by employing SVRF commands to ensure they are valid for input to the NMDPC command. This section contains a list of these required checks and operations as well as examples of their respective implementation.

The following series of example checks are based on a double-patterning derived stitch layer formed from drawn stitches combined with areas of overlapping anchors (forming implicit stitches). This operation relegates all stitches to be treated with the same nicety and mask settings. To treat them differently, do not combine them on the same layer.

```
anchor_formed_stitch = anchor1 AND anchor2
```

Figure 5-6. User-defined Stitch Layer

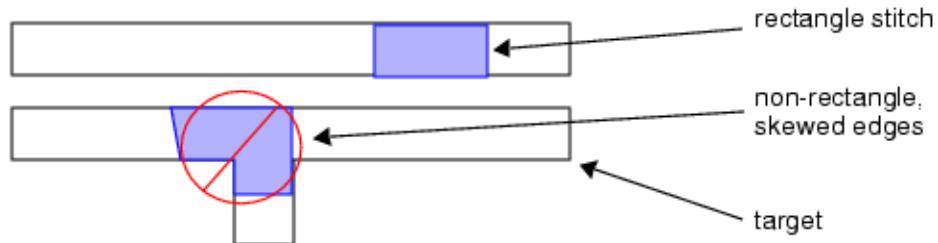


The following required checks and operations mimic those performed on dpStitch-generated stitches. They are applied to the previously derived user stitch layer and shown with their respective SVRF commands.

- Stitches must be polygonal, rectangle shapes:

```
rect_user_stitch = RECTANGLE all_user_stitch
nonrect_user_stitch = NOT RECTANGLE rect_user_stitch
```

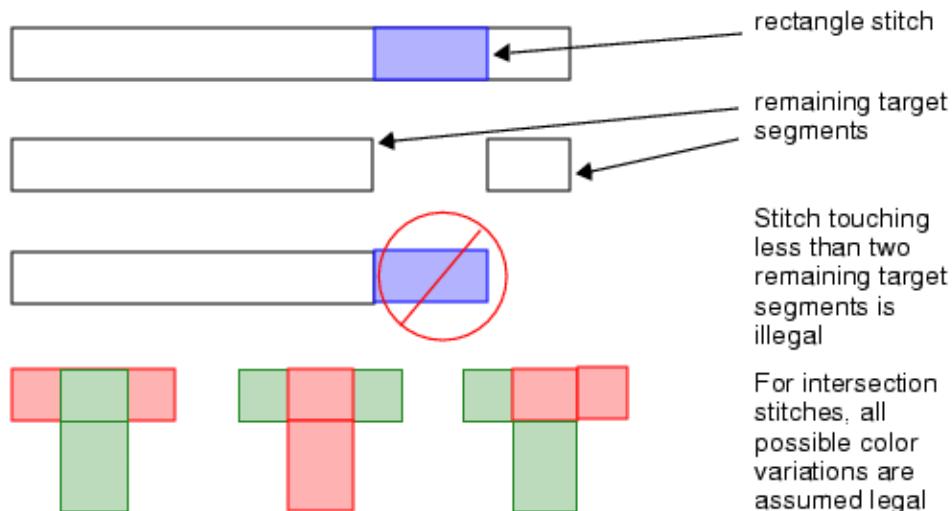
Figure 5-7. Stitch Layer Must be Polygonal Rectangles



- Stitches are subtracted (using a NOT boolean command) from the input layer, and must separate the target layer segments, touching more than one segment across the entire stitch edge. One possible SVRF solution:

```
target_layer = NOT target_rect_user_stitch
valid_user_stitch = DFM PROPERTY rect_user_stitch target_layer
OVERLAP ABUT ALSO MULTI [- = COUNT(target_layer)] >1
```

Figure 5-8. Stitch Shapes Removed from Target

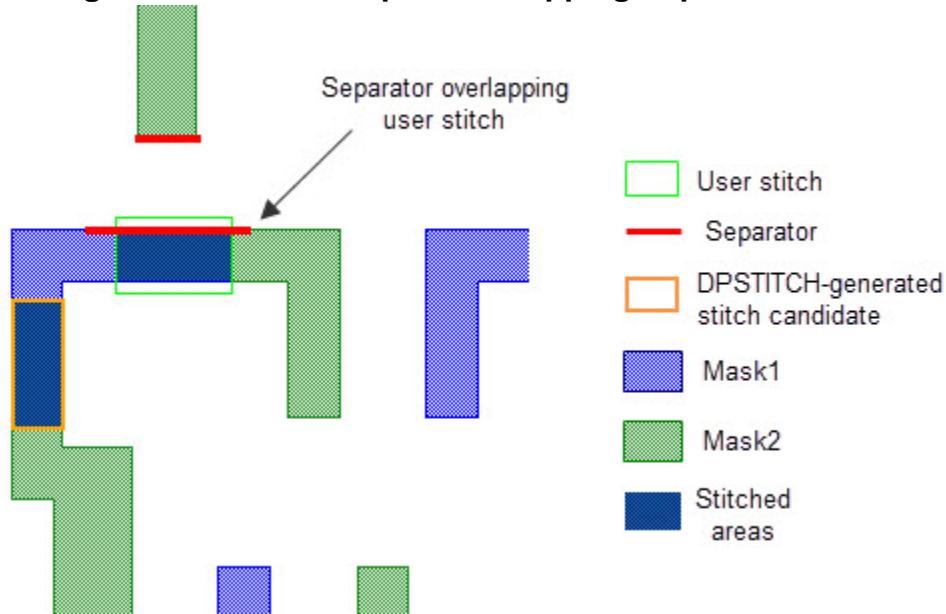


- Stitches require implementation decisions when inserted between shapes on the input layer.
- Stitches must not overlap or be edge-coincident with any separator (see [Figure 5-9](#) on page 318). By design, RET DPSTITCH provides the capability to check this condition and conform stitches to this requirement. However, if you introduce user-drawn or derived stitches, you are responsible to insure user stitch validity.



Caution A separator overlapping a stitch indicates there is no coloring solution using either mask, eliminating the stitch as a candidate. If this condition involving user stitches is not checked, lithographic resolution of the design may fail.

Figure 5-9. Stitch Shapes Overlapping Separators



To correctly handle user stitches, perform these preventative measures (these SVRF statements are examples only, and may vary depending on your specific process and implementation):

- Make sure all user stitches are carefully checked for separator-stitch overlap and all invalid stitches eliminated:

```
stitch_separator_conflict =
DFM PROPERTY user_stitch m1_separators
OVERLAP ABUT ALSO MULTI
[- = COUNT(m1_separators)] > 0

valid_user_stitch =
user_stitch NOT stitch_separator_conflict
```

- Make sure auto-stitch candidates are output by RET DPSTITCH:

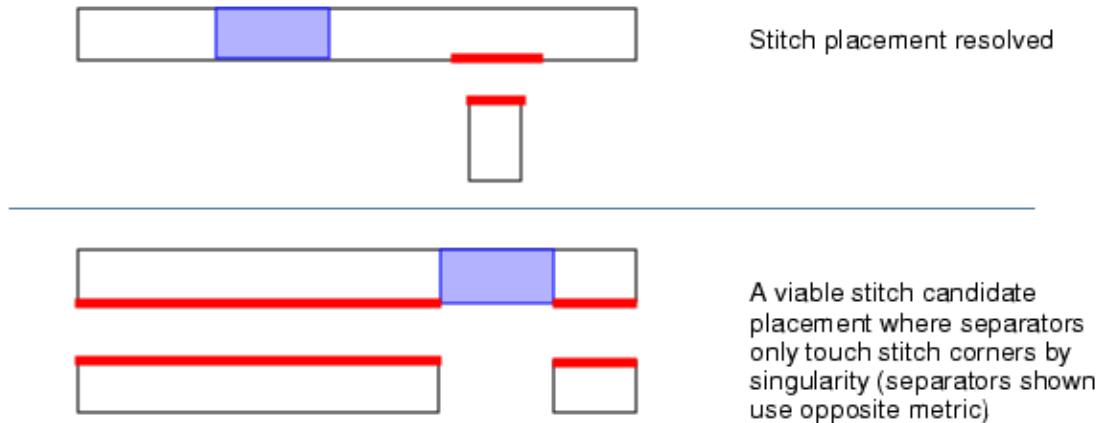
```
m1_dpstitch_candidates =
RET DPSTITCH m1_input mask1_anchor mask2_anchor
FILE dpstitch_setup MAP stitchcandidates
```

- User stitches can be declared in separate action commands to maintain varied priorities within the LITHO FILE statement input to the RET NMDPC command as shown in this example:

```
LITHO FILE m1_nmdpc_file [
    ...
    action m1_separators 0 1;
    action m1_dpstitch_candidates :stitch 1 1;
    action M1USERSTITCH :stitch 0 1;
]
```

Design-based alteration of stitch placement may also provide a solution:

Figure 5-10. Design-based Stitch Placement Resolution

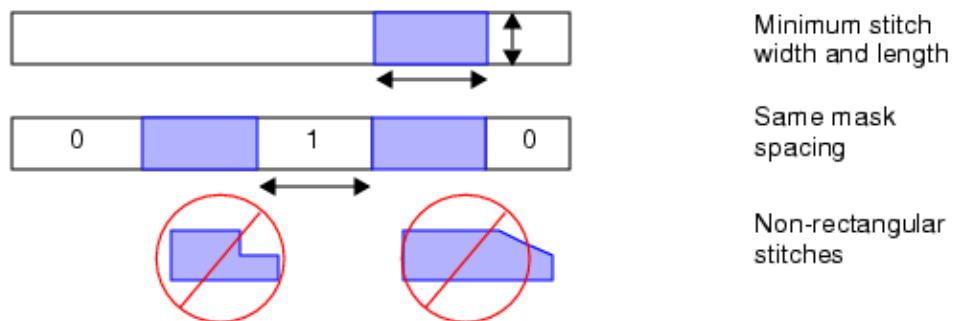


- All stitch candidates must be able to be used simultaneously without violating any DRC rules. This typically includes checks to insure each stitch is a rectangle, same-mask spacing, and length and width checks as shown in [Figure 5-11](#):

```
rect_user_stitch = RECTANGLE all_m1_user_stitch
nonrect_user_stitch = NOT RECTANGLE rect_user_stitch

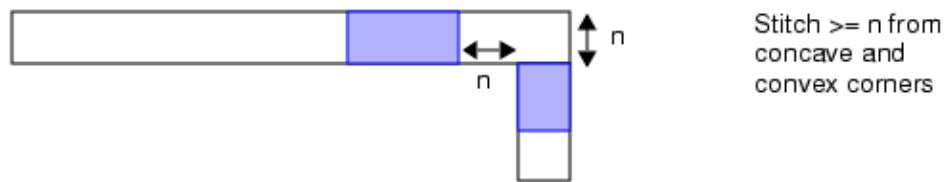
user_stitch_min_space = EXT rect_user_stitch < 0.140
    OPPOSITE PARALLEL ONLY PROJ > 0
user_stitch_min_space { DFM COPY user_stitch_min_space }
user_stitch_min_width = INT rect_user_stitch < 0.032 REGION
user_stitch_min_width { DFM COPY user_stitch_min_width }
```

Figure 5-11. Stitches Must be DRC-Clean



- Stitch candidates must comply with corner checks, where the stitch must remain a certain distance from either a convex or concave corner of the target shape:

Figure 5-12. Stitch Corner Rule Enforcement

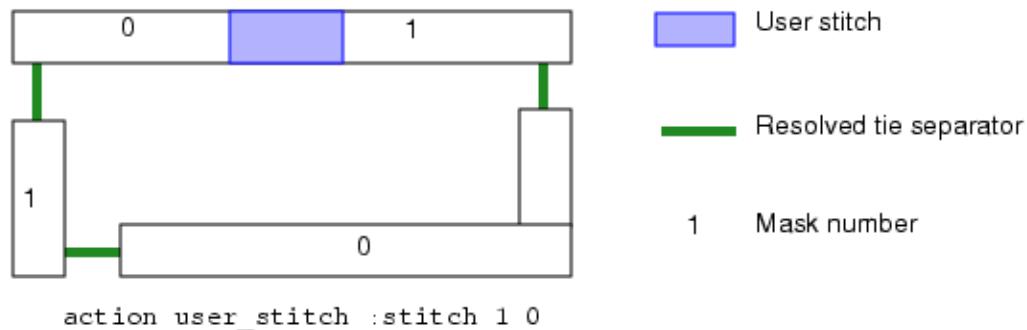
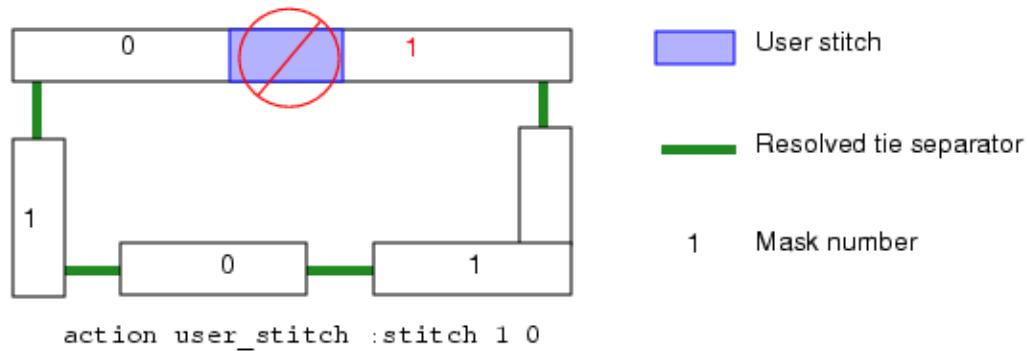


Control of Stitch Behavior with Priority and Mask Settings

Even though stitches may be user-drawn or generated by `dpStitchCandidates`, they are still considered candidates until appropriated by the accompanying priority and mask-specific settings. Generated stitches are dependent on these settings.

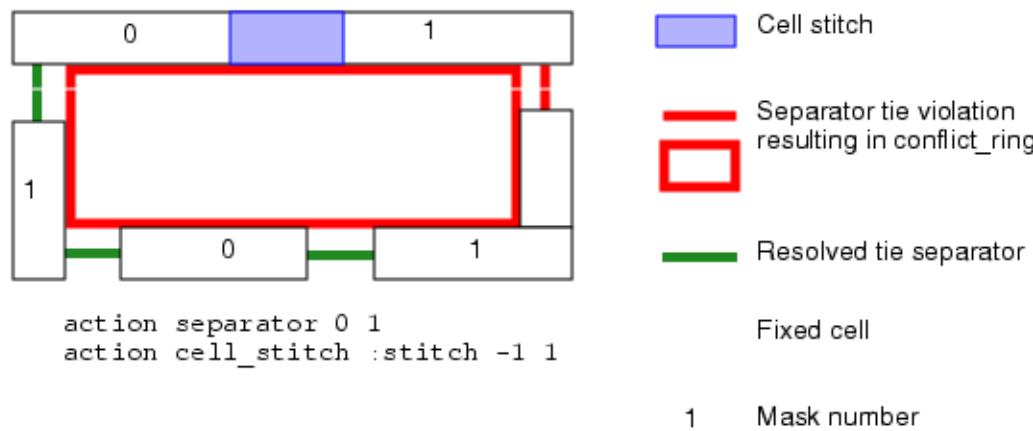
Figure 5-13 demonstrates `:stitch` assigning a priority of 1 (weak) and a mask setting of 0 (same mask). This minimizes the chances of the stitch being used and is set when the need for the stitch is optional. In this case, the stitch would force an odd cycle if used and is discarded, resolving all separators. In the second illustration, the stitch is used to resolve what would otherwise result in an odd cycle. In both cases the assumed separator priority is 0.

Figure 5-13. Minimizing and Controlling Use of Stitches



[Figure 5-14](#) demonstrates :stitch assigning a priority of -1 (strong) and a mask setting of 1 (different mask). This enforces use of the stitch and is set when the need for the stitch is required. The scenario shown is a case where the fixed cell is third-party IP or from a cell library where alteration of data is not permissible. In this case, the stitch is required and forces an odd cycle with data in a lower level of hierarchy. This scenario requires some data alteration.

Figure 5-14. Enforcing use of Stitches



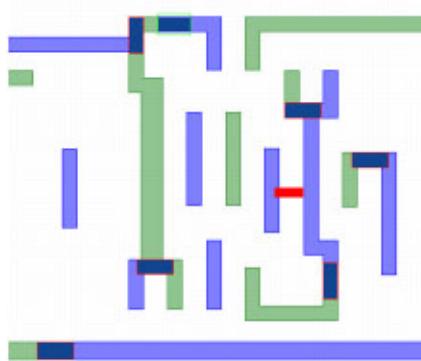
Stitch and Priority Interaction

Priority and mask settings specified for individual action stitch commands produce significant interactions between auto-stitches, user stitches and separators.

This section covers a few of the interactions in a small design area as an example.

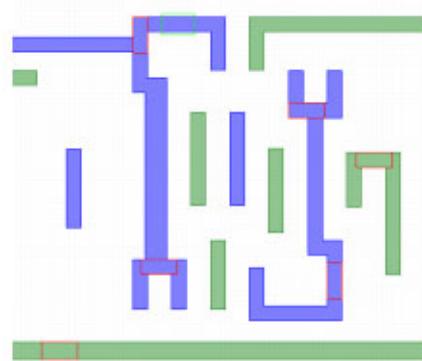
Figure 5-15 demonstrates priority and mask assignment settings that force, inhibit, maximize, and minimize stitches. The settings that force stitches may generate separator violations while forcing all stitches to be implemented. All separator niceties are assumed to be 0.

Figure 5-15. Stitch and Priority Interactions



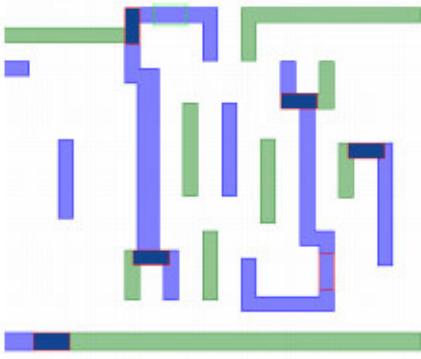
`action ml_stitch :stitch -1 1`

All stitches are inserted (including user stitch) due to strong priority and different mask specification. Strong priority forced a violated separator (in red).



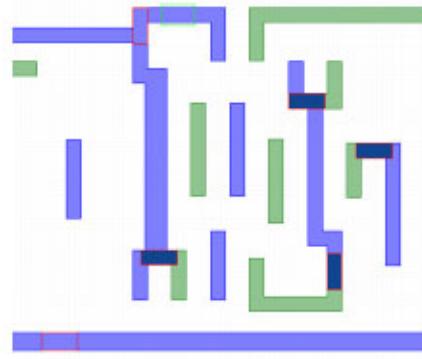
`action ml_stitch :stitch -1 0`

Stitches are inhibited due to strong priority and same mask specification.



`action ml_stitch :stitch 1 1`

Stitches are maximized with a weak priority and different mask specification. Maximizing stitches is performed without causing any coloring variations.



`action ml_stitch :stitch 1 0`

Stitches are minimized with a weak priority and same mask specification. The power rail and upper left stitches are not included.

Using SVRF Variables within a LITHO FILE Statement

When the LITHO FILE statement is contained within the SVRF file, you can use variables set in the main SVRF level within the LITHO FILE statement.

The methods described here work for Calibre nmDPC, but may be different for other tools.

Prerequisites

- A [LITHO FILE](#) statement containing an options block.

Procedure

1. Specify the variable value in the SVRF file.

- To read in an environment variable set from the shell:

```
VARIABLE variable_name ENVIRONMENT
```

- To set it within the SVRF file (variables are called in Step 3):

```
VARIABLE le_limit 0.03
VARIABLE le_leg 0.04
```

2. Inherit the variables within the LITHO FILE statement:

```
LITHO FILE stitch_file [
    svrf_var_import le_limit
    svrf_var_import le_leg
    ...
```

3. Call the variables within the options block by prepending them with a \$.

```
dpFaceClass name(lineend) : \
    true for (length <= $le_limit) and minAdjacent < $le_leg: \
    false otherwise:
```

4. (Optional) Pass a variable on the command line (requires TVF):

```
set variable_name [tvfarg::get_tvf_arg]
```

5. When you invoke Calibre, append “-tvfarg value” to the command line:

```
calibre -drc -hier -tvfarg value rules
```

Related Topics

[VARIABLE \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

DP Debugging Methodology

Debugging double-patterning errors is simplified with the Calibre RVE. MAP keywords in the SVRF rules file provide visual debugging for double-patterning errors resulting from non-conforming layout topologies.

The Results Viewing Environment (RVE) loads a Results Database file (RDB file) assisting the designer to resolve these errors. Using an optimal DP debugging methodology you resolve errors in this order:

1. [Standard Design Rule Checks](#)
2. [Spacing Checks for Opposite Mask Errors](#)
3. [Self-Conflict Errors](#)
4. [Anchor-based Self-Conflict Errors](#)
5. [Ring and Warning Errors Highlighting Odd-Cycle Errors](#)
6. [Anchor-Path Errors](#)

This order minimizes the work required to fix DP errors, and limits misleading results occurring in other checks subsequent in the list.

Standard Design Rule Checks

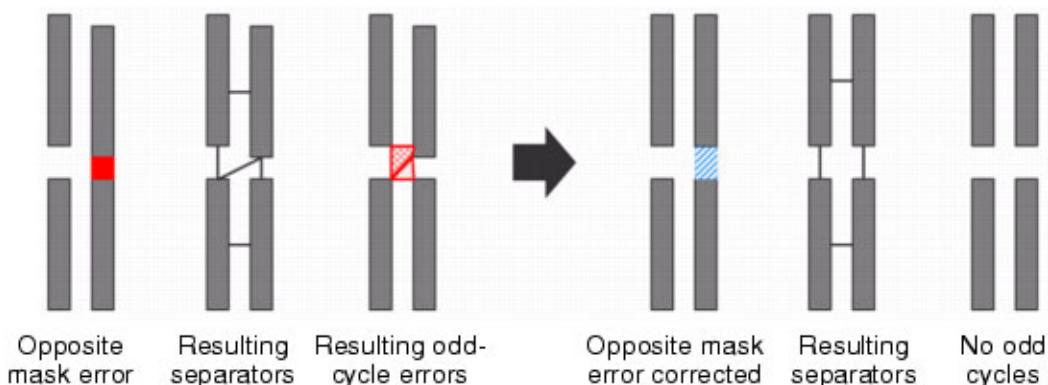
Implement standard DRC rules against your design first. This preliminary step ensures the bulk of processing and manufacturing problems are caught early during validation.

Spacing Checks for Opposite Mask Errors

Spacing checks involve two polygons isolated by one spacing constraint, with no possible coloring solution. When minimum spacing constraints are violated, other error types typically result.

The left side of [Figure 5-16](#) shows two polygons in violation of an opposite mask spacing constraint. The spacing violation generates a diagonal tip-to-tip separator in the layout, resulting in two odd-cycle errors between four polygons. Following indications of odd-cycle errors, it is possible to correct the odd-cycle error, but not correct the original minimum spacing violation. You may attempt two disparate fixes where one strategic correction repairs both errors. On the right side of the figure, the minimum spacing violation has been fixed; the odd cycles do not appear, both issues being addressed with one correction.

Figure 5-16. Spacing Checks for Opposite Mask Errors

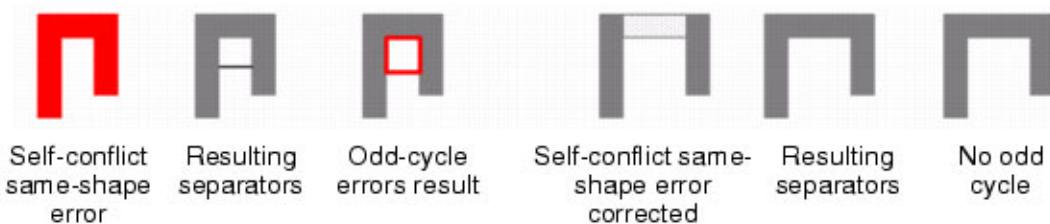


Self-Conflict Errors

Self-conflict errors are isolated to single polygons. They are characterized by notches formed by edges violating a spacing constraint between edges of the same shape. Standard DRC notch checks are similar except they do not consider mask color, where stitching may resolve the conflict. Polygonal interaction of self-conflict errors with other shapes may form extended errors of various types.

The left side of [Figure 5-17](#) shows a polygon with a red self-conflict error. These types of errors highlight the entire shape. The separator results in an odd-cycle error of one. The right side of the figure demonstrates correction of the self-conflict error (moving one leg of the shape further away), which also fixes the odd-cycle error.

Figure 5-17. Self-Conflict Shape Errors



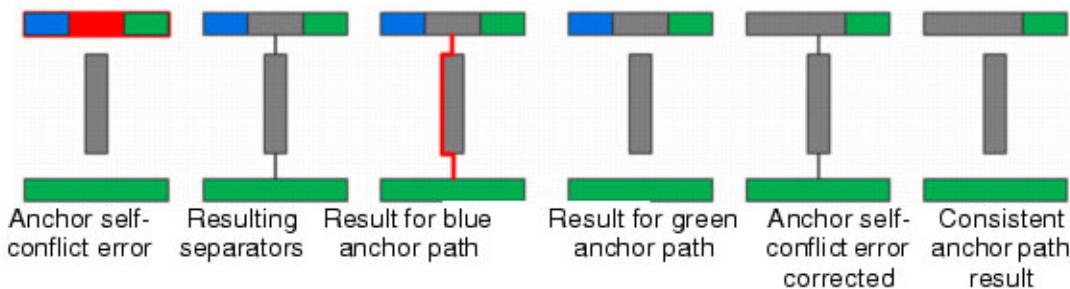
Anchor-based Self-Conflict Errors

The anchor-based self-conflict error type is isolated to a single polygon. This error results from single polygons having multiple conflicting anchors.

The left side of [Figure 5-18](#) shows two anchors of differing colors overlapping a single shape. The coloring algorithm assumes a random anchor color to trace. Given the separator interactions with other polygons, there is a path from one anchor to the other anchor. If the tool chooses starting the trace at the blue anchor, an anchor-path error to the green anchor at the bottom is created. If the tool selects the green anchor, no anchor-path error is shown. By fixing

the anchor self-conflict error, results of the anchor-path errors become deterministic, and not random.

Figure 5-18. Anchor-based Self-Conflict Errors

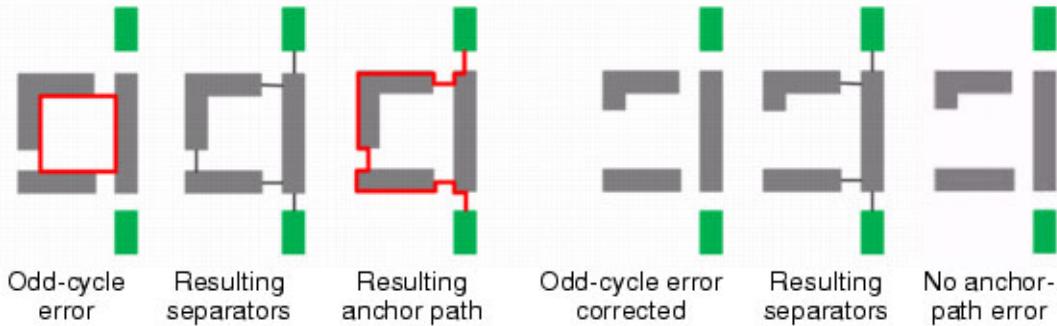


Ring and Warning Errors Highlighting Odd-Cycle Errors

Having resolved local-area error types, focus next on wider-range odd-cycle and anchor-path error types. You fix odd-cycle errors first to prevent interaction with anchor-path errors.

The left side of [Figure 5-19](#) shows an odd-cycle error. Due to separator interactions with other polygons, the shapes forming the odd cycle interact with two anchors at the top and bottom of the pattern, which use this odd-cycle error path to form an anchor-path error. Correcting the odd cycle requires moving a shape to correct one of the three spaces contributing to it. Regardless of which space is corrected (and associated separator removed), the anchor-path error is simultaneously corrected, as shown on the right of the figure.

Figure 5-19. Ring and Warning Errors Highlighting Odd-Cycle Errors

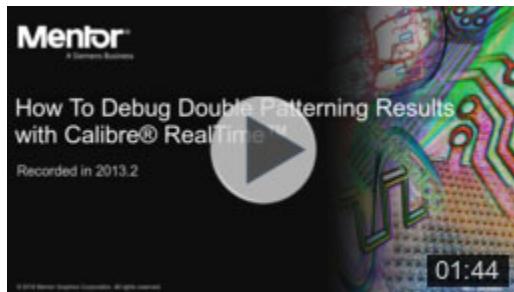


Anchor-Path Errors

The reverse is not true. If correction of the anchor-path error is attempted first, modifying the spaces between either anchor (removing their associated separators) successfully corrects the anchor-path error, but does not correct the odd-cycle error. For this reason, odd-cycle errors should be corrected first before attempting correction of anchor-paths errors.

Debug Double-Patterning Errors With Calibre RealTime

For extended help debugging DP errors, consider viewing one of the Siemens EDA online videos demonstrating Calibre RealTime use to correct DP errors in a double-patterned design.



Related Topics

[Debugging DP Layout Errors](#)

Debugging DP Layout Errors

After running Calibre double-patterning, you find DP-related errors revealed by the MAP keywords specified in the RET NMDPC command.

In this procedural example, the error type names are user-defined. Your names may be different. The names are derived from the SVRF rules file. This procedure only focuses on a subset of the available DP error resolution tools.

Prerequisites

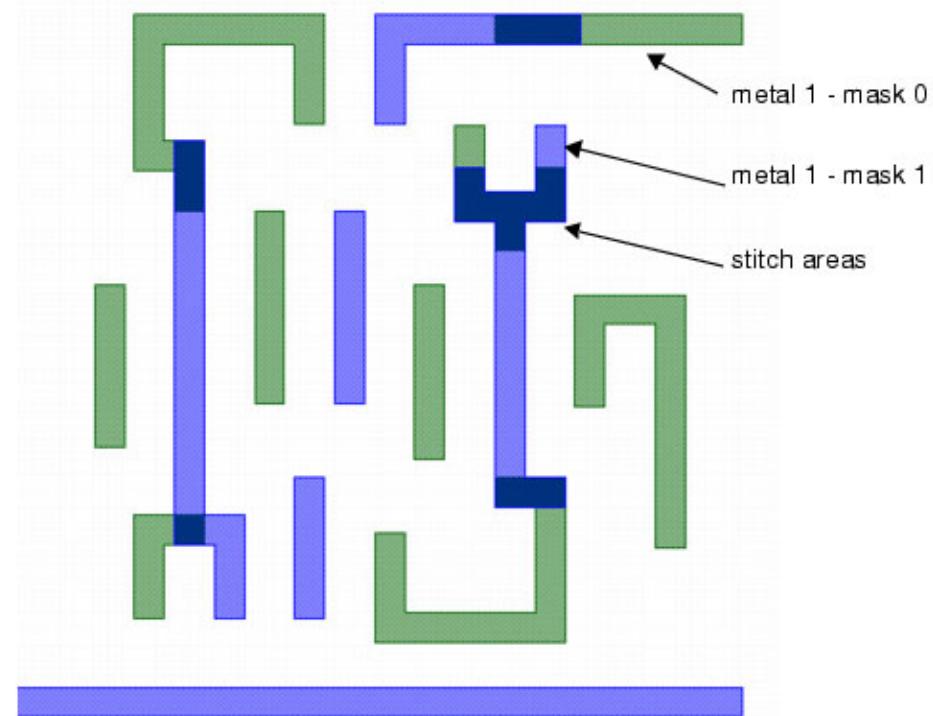
- Successful completion of an SVRF double-patterning run using RET NMDPC.
- Specification of MAP keywords using RET NMDPC.
- Prior successful DRC completion.

Procedure

1. Open the output layout:

```
calibredrv output_layout
```

Figure 5-20. Output Layout

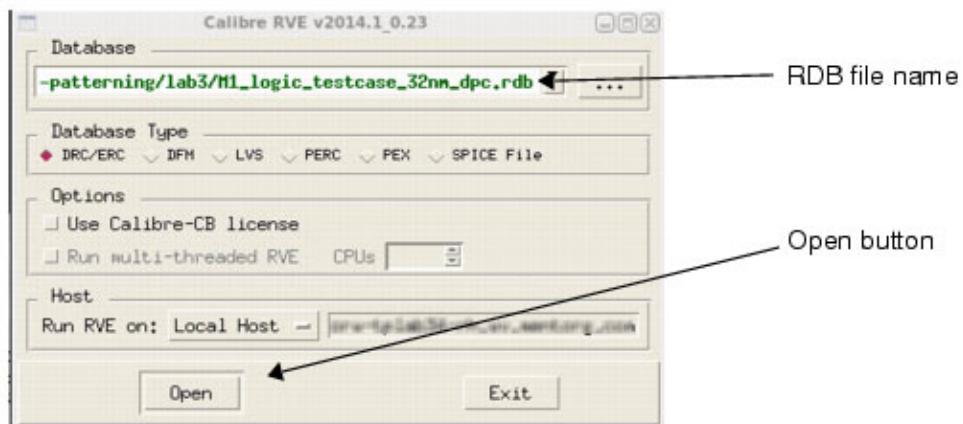


2. Open the results database using the calibredrv menu:

Verification — Start RVE

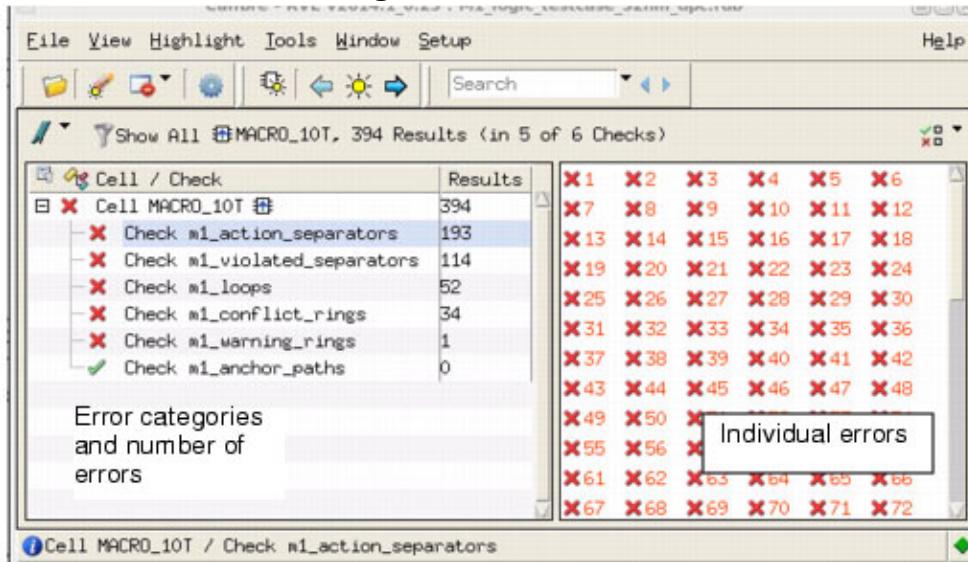
- a. Select an RDB filename.
- b. Click the **Open** button.

Figure 5-21. Starting RVE



The RVE form opens with various error types displayed in the RVE form. The error types are output by the SVRF and RET NMDPC commands in the Rules file:

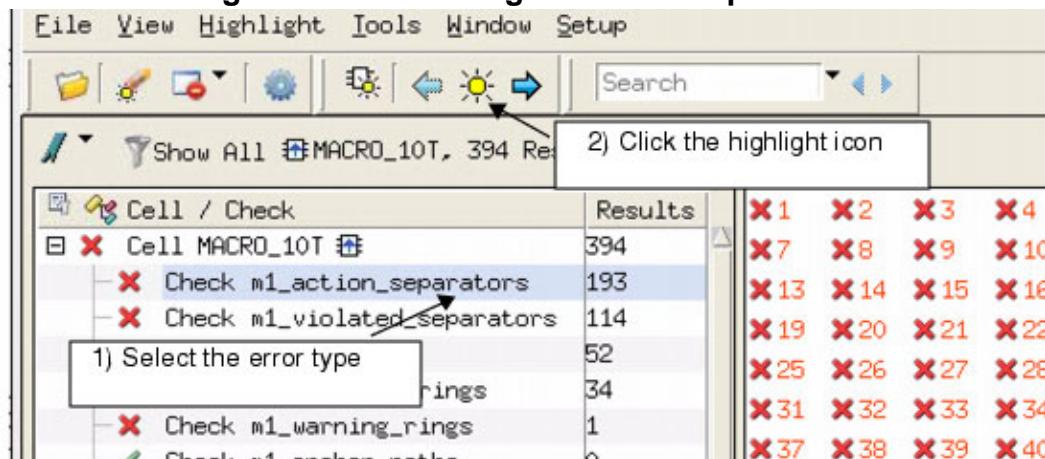
Figure 5-22. RVE Form



The next series of steps explore the general means of selecting error types and specific error types used for debugging.

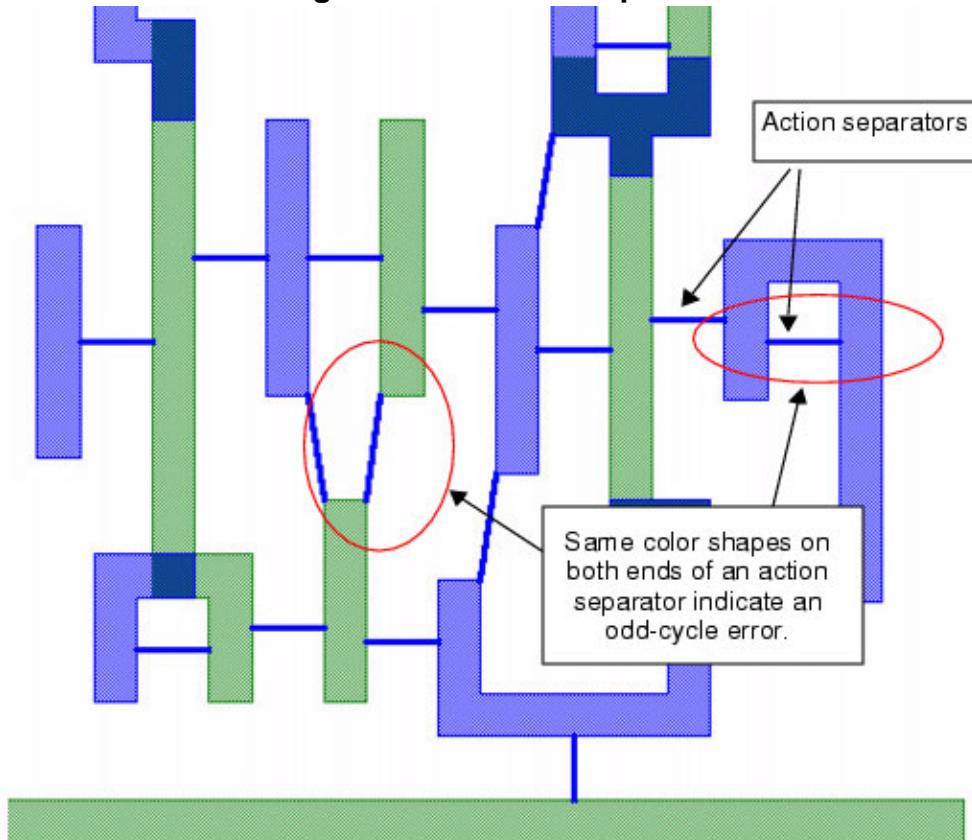
3. To select any error type (and specifically action separators) from the RVE form:
 - a. Select the action separator error type.
 - b. Click the highlight icon.

Figure 5-23. Selecting an Action Separator



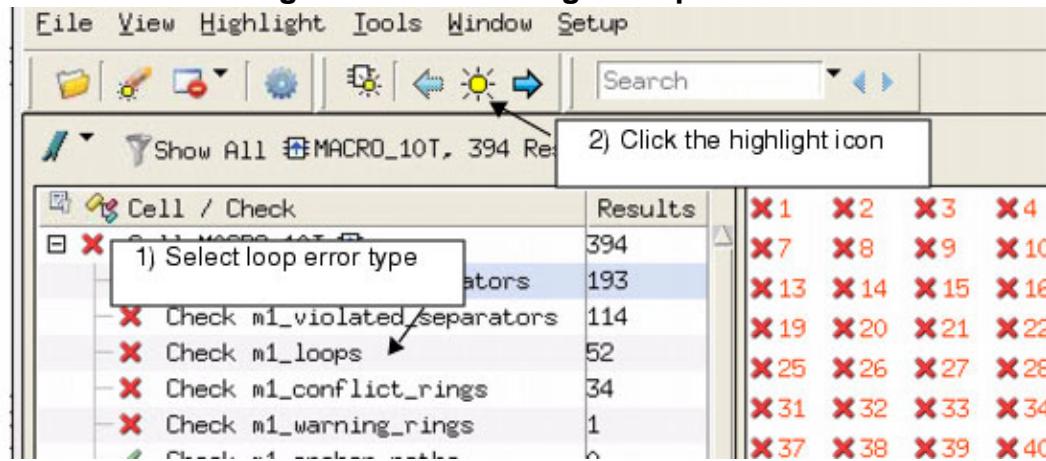
In Figure 5-24, RVE displays the action separators. Action separators indicate that two shapes the separator touches must be placed on different masks:

Figure 5-24. Action Separators



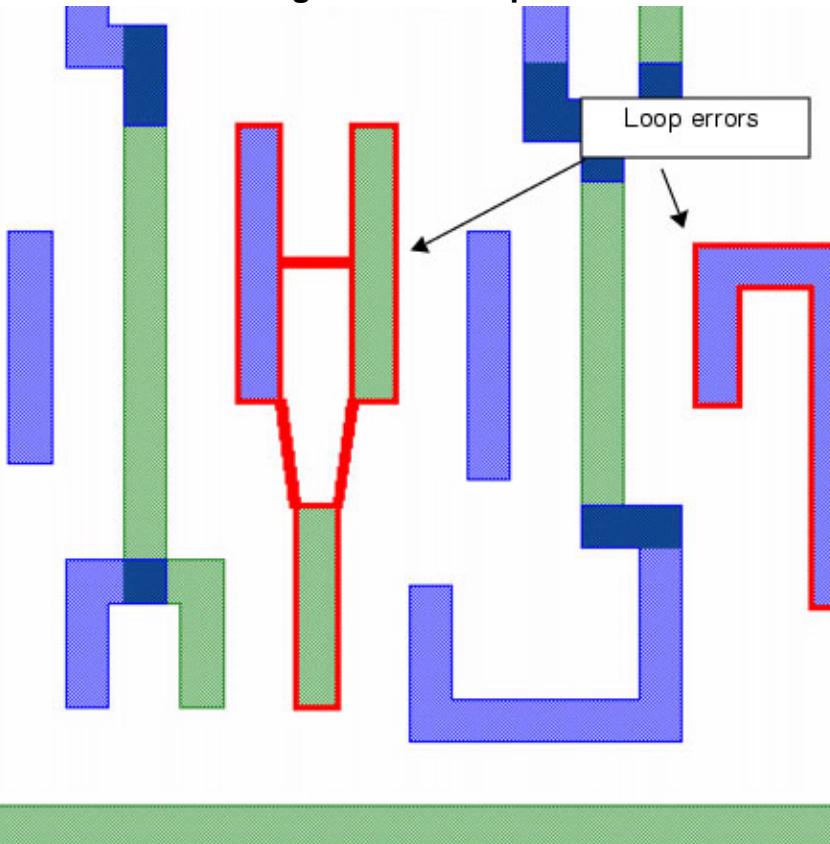
4. To select loop errors from the RVE form:
 - a. Select the loop error type.
 - b. Click the highlight icon.

Figure 5-25. Selecting a Loop Error



The loop errors are displayed on the output layout in [Figure 5-26](#). Loop errors highlight shapes and action separators that are grouped into odd-cycle errors:

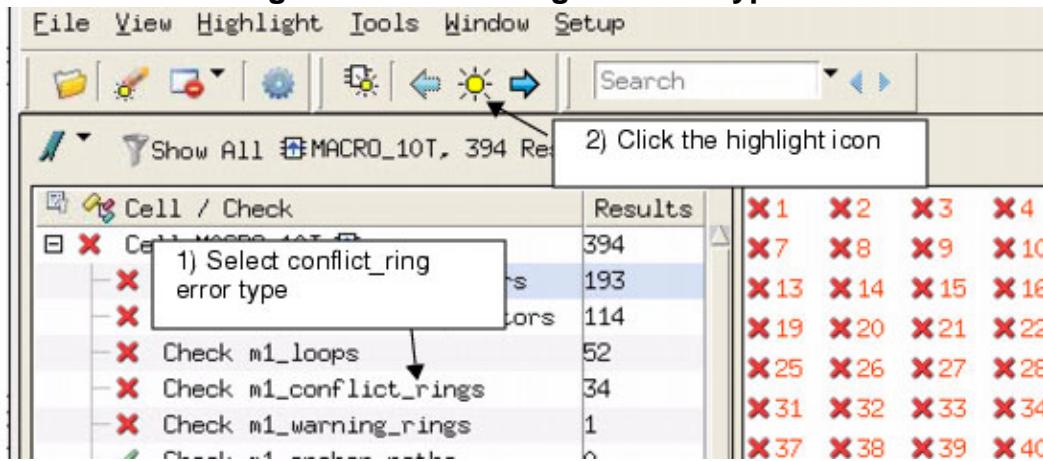
Figure 5-26. Loop Errors



5. To select conflict rings from the RVE form:

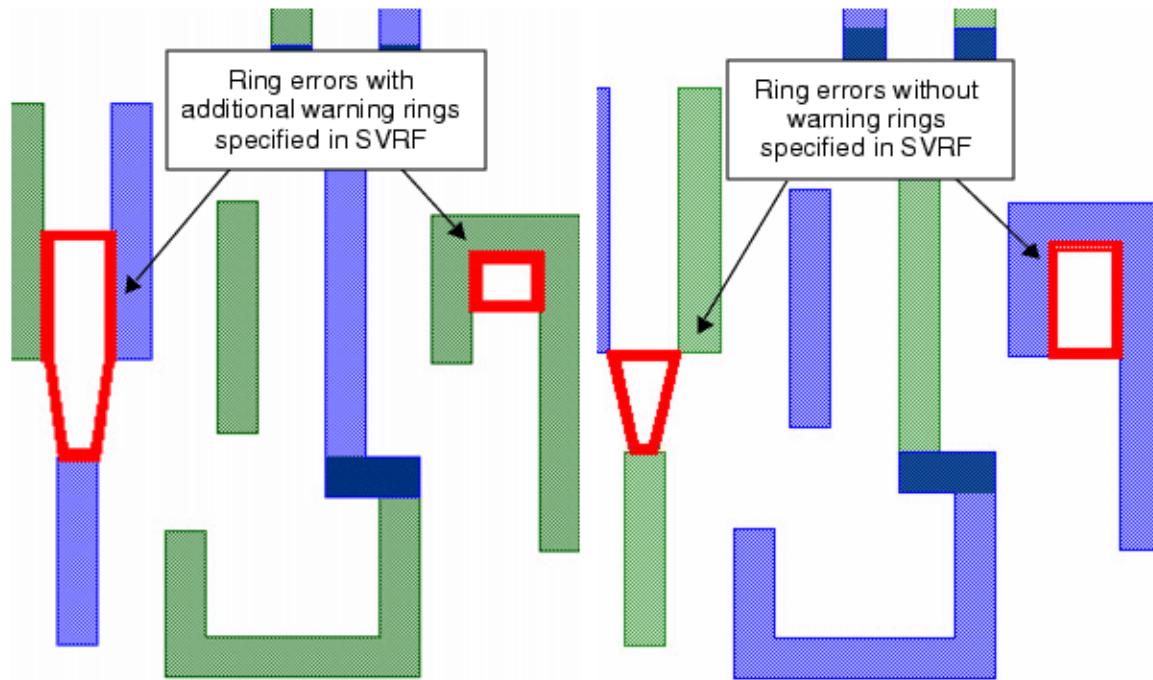
- Select the conflict ring error type.
- Click the highlight icon.

Figure 5-27. Selecting an Error Type



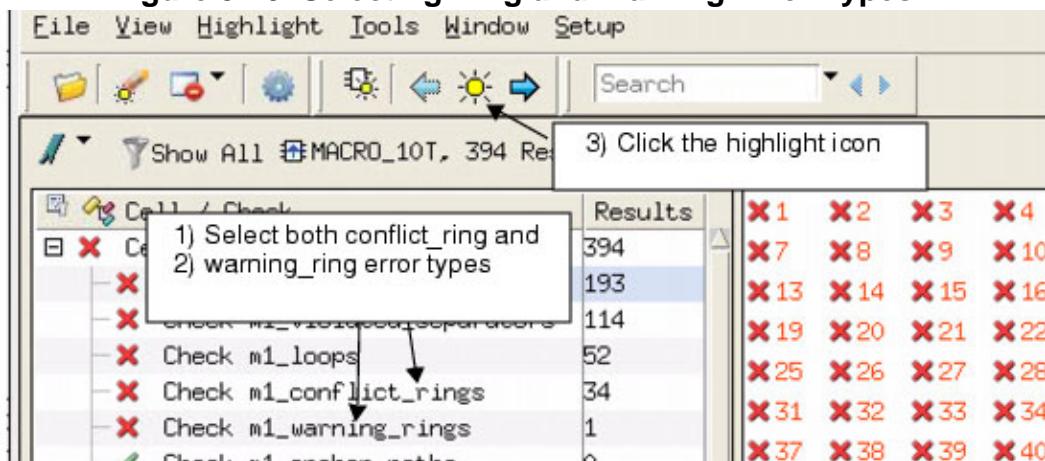
Conflict rings highlight odd-cycle errors. Conflict rings touch all shapes involved in an odd cycle. Conflict rings appear in two different ways depending on whether warning error types are additionally specified in the SVRF rules file. The differences are shown in Figure 5-28:

Figure 5-28. Conflict Ring Errors



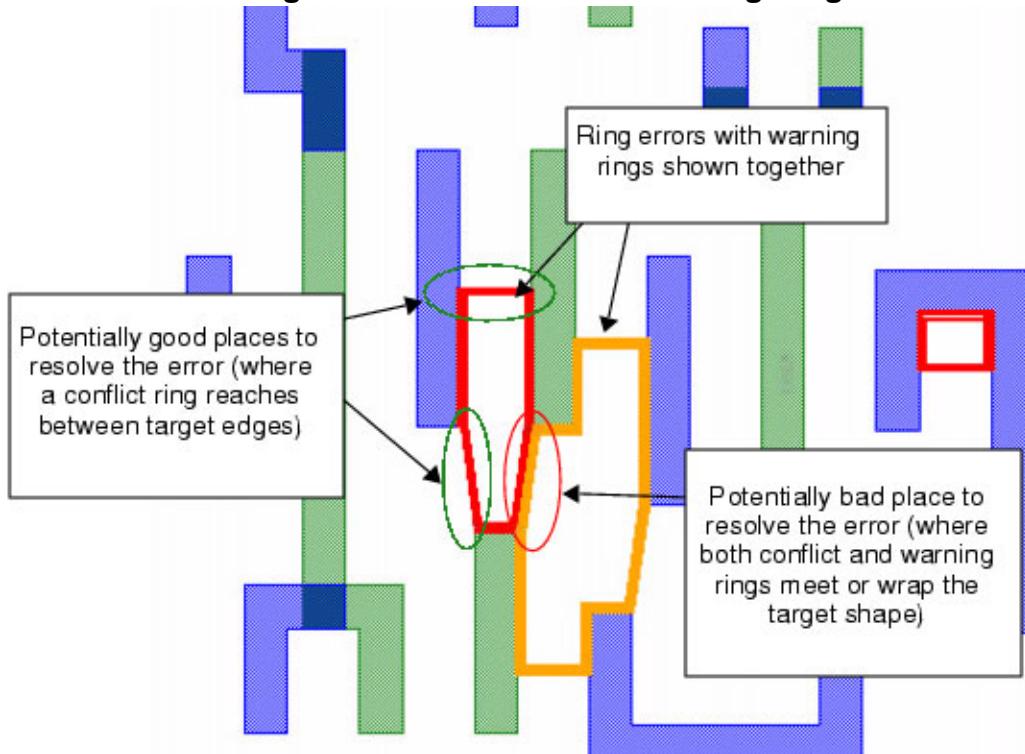
6. To simultaneously select conflict rings and warnings from the RVE form:
 - a. Select the conflict ring error type.
 - b. While holding the Shift key, select the warning ring error type.
Both error types in the RVE form are selected.
 - c. Click the highlight icon.

Figure 5-29. Selecting Ring and Warning Error Types



Both conflict and warning rings appear as shown in Figure 5-30. When used together, conflict and warning rings highlight odd-cycle errors and indicate where odd-cycle errors can be resolved. Conflict rings touch all shapes involved in an odd cycle.

Figure 5-30. Conflict and Warning Ring Errors



Fixing DP Layout Errors

Edit a layout to resolve errors found by Calibre DP error types.

This example demonstrates use of conflict and warning rings to fix odd-cycle errors in a layout.
This example does not cover use of Calibre® DESIGNrev™.

Prerequisites

- Successful completion of an SVRF double-patterning run using RET NMDPC.
- Display of conflict and warning ring output with RVE.

Procedure

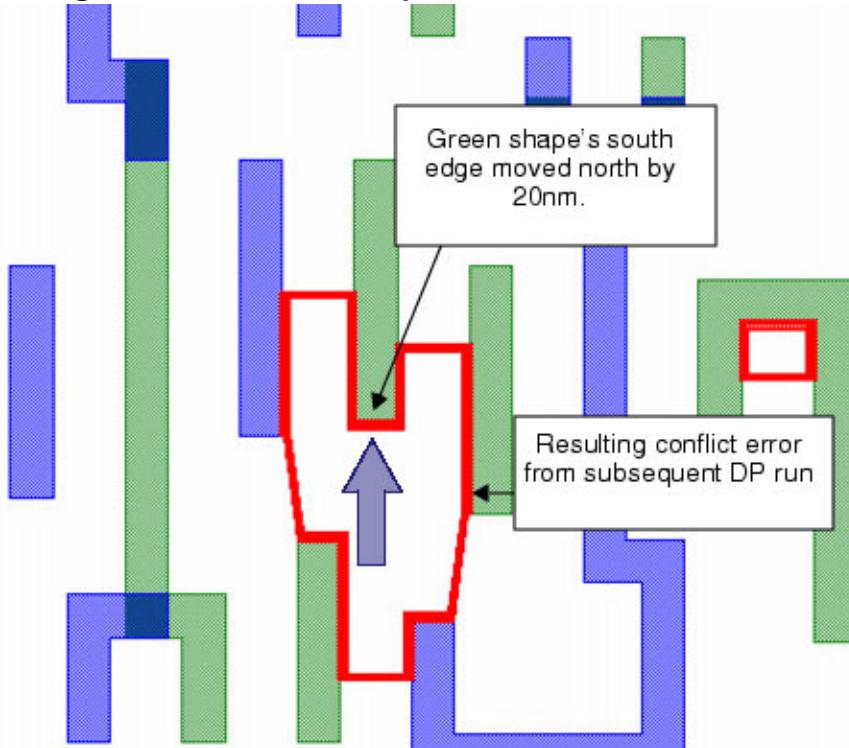
1. Open the input layout for editing:

```
calibredrv input_layout
```

2. Move the south edge of the target shape shown by 20 nm.

The first attempted edit was to move the south edge the indicated shape up 20 nm. As shown in [Figure 5-31](#), this propagated the error so that another, separate conflict error appeared after a subsequent DP run:

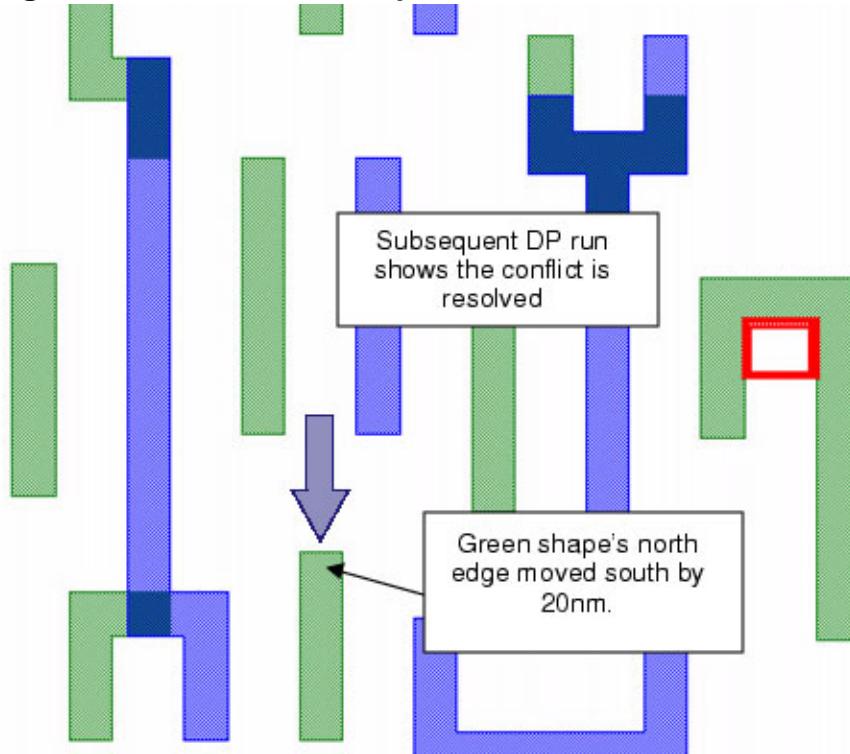
Figure 5-31. First Attempted Edit and Error Results



3. Move the north edge of the target shape shown south by 20 nm.

The second edit was to move the north edge of the indicated shape down 20 nm. As shown in [Figure 5-32](#), this resolved the error after a subsequent DP run:

Figure 5-32. Second Attempted Edit and Error Resolution

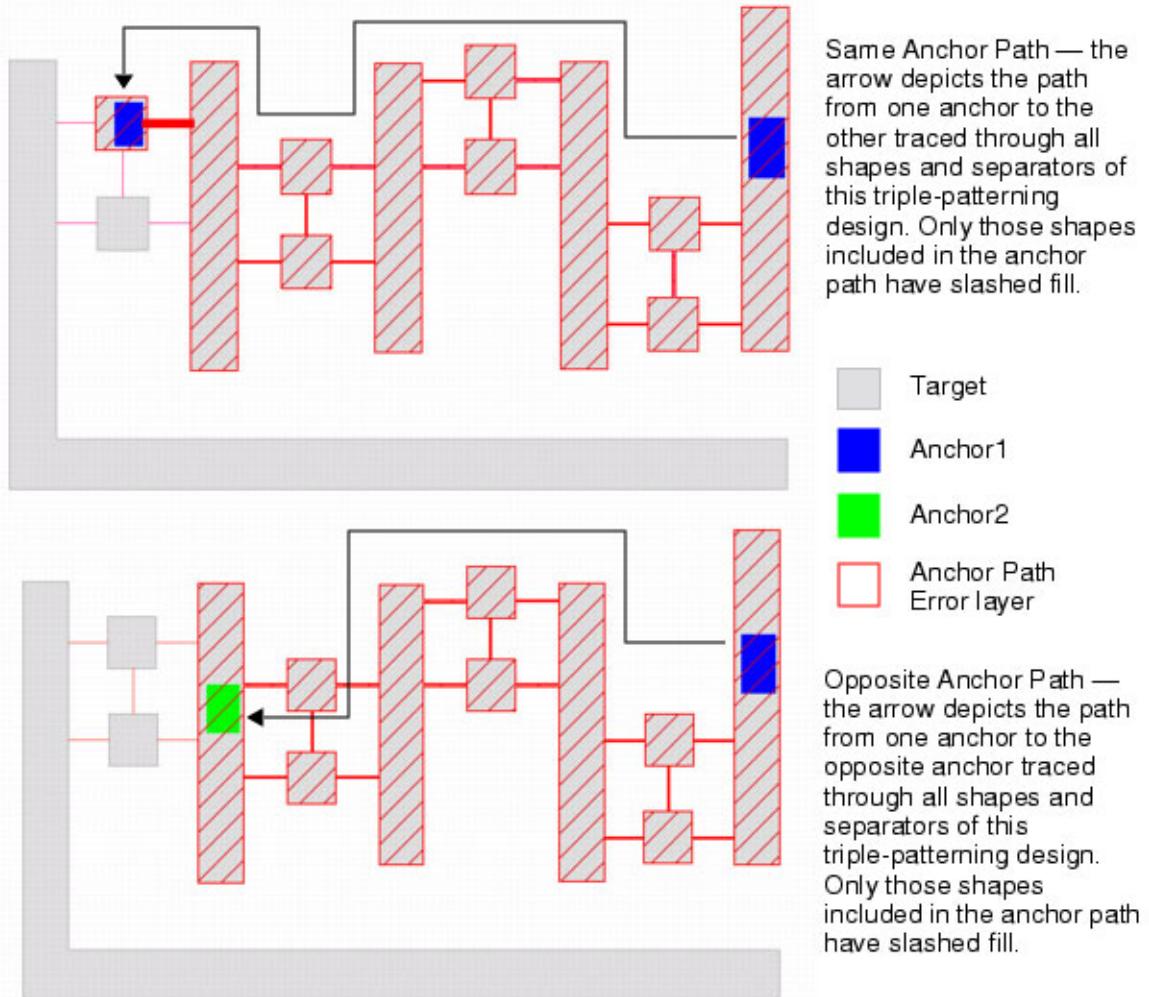


MP Visualization and Error Resolution

Visualization keywords assist the user to resolve multi-patterning-related decomposition failures. Three classes of visualization keywords are supported for various decomposition errors.

- **Anchor Paths** — Highlights the shapes and separators between two or more anchors in violation.

Figure 5-33. Anchor Path Examples



The SAME_ANCHOR_PATH and OPPOSITE_ANCHOR_PATH MAP keywords output all shapes and separators involved in anchor conflicts.

- **Self-Contained Violations** — Highlights all shapes having separators to all other colorable polygons. Isolates layout configurations that are inherently not colorable.

Figure 5-34. Self-Contained Violations

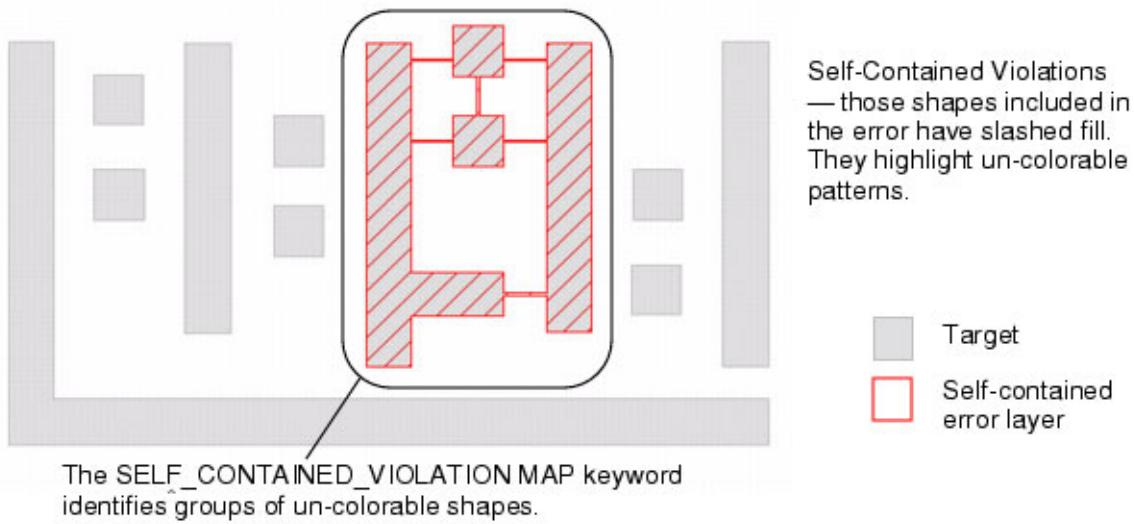
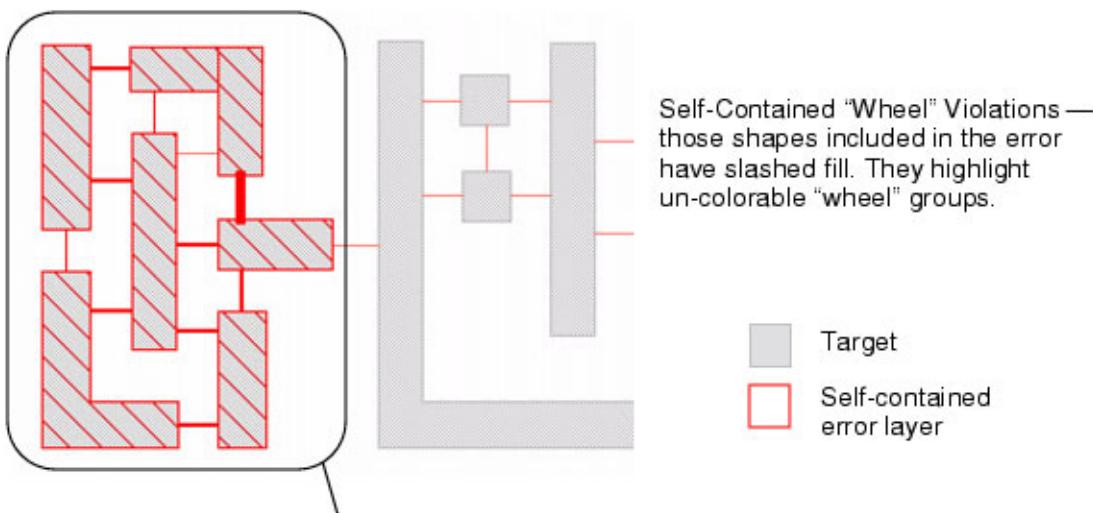
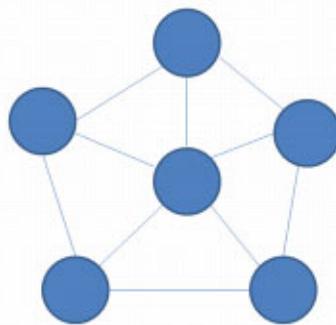


Figure 5-35. Self-Contained “Wheel” Violations



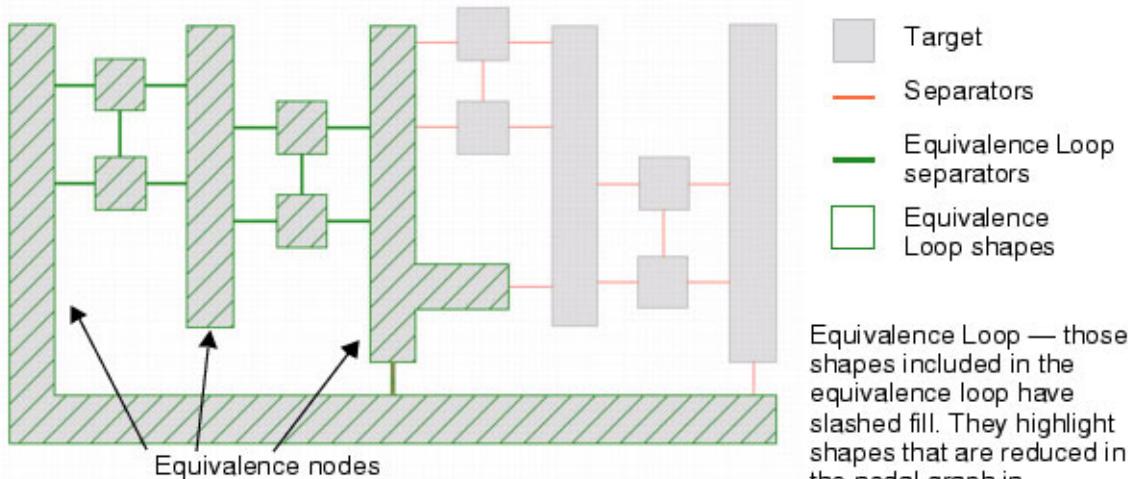
The SELF_CONTAINED_VIOLATION_MAP keyword also identifies “wheels”, groups of un-colorable shapes.

Symbolic “Wheel” Pattern — This graph is a depiction of the physical topology shown in Figure 2-9. An odd number of shapes all interacting with two neighbors and a center node comprises a wheel. This wheel is termed a “W6”, indicative of its six nodes.



- **Equivalence Loops** — Highlights the shapes that, when combined, form a nodal reduction. These reduced node maps can aid in the resolution of decomposition problems.

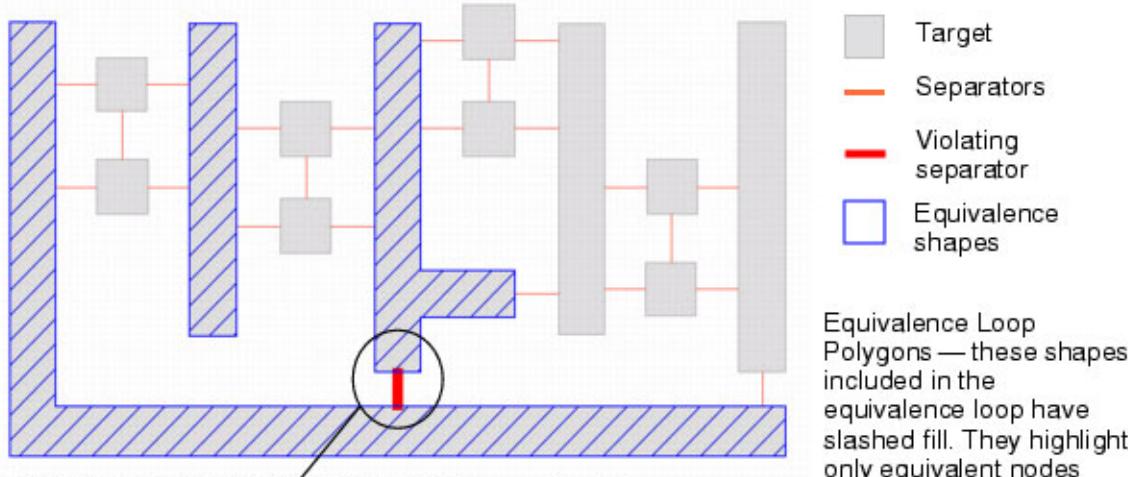
Figure 5-36. Equivalence Loop



The EQUIV_LOOP MAP keyword outputs all shapes and separators involved in patterns of layout that can be reduced.

Equivalence Loop — those shapes included in the equivalence loop have slashed fill. They highlight shapes that are reduced in the nodal graph in Figure 2-6.

Figure 5-37. Equivalence Loop Polygons

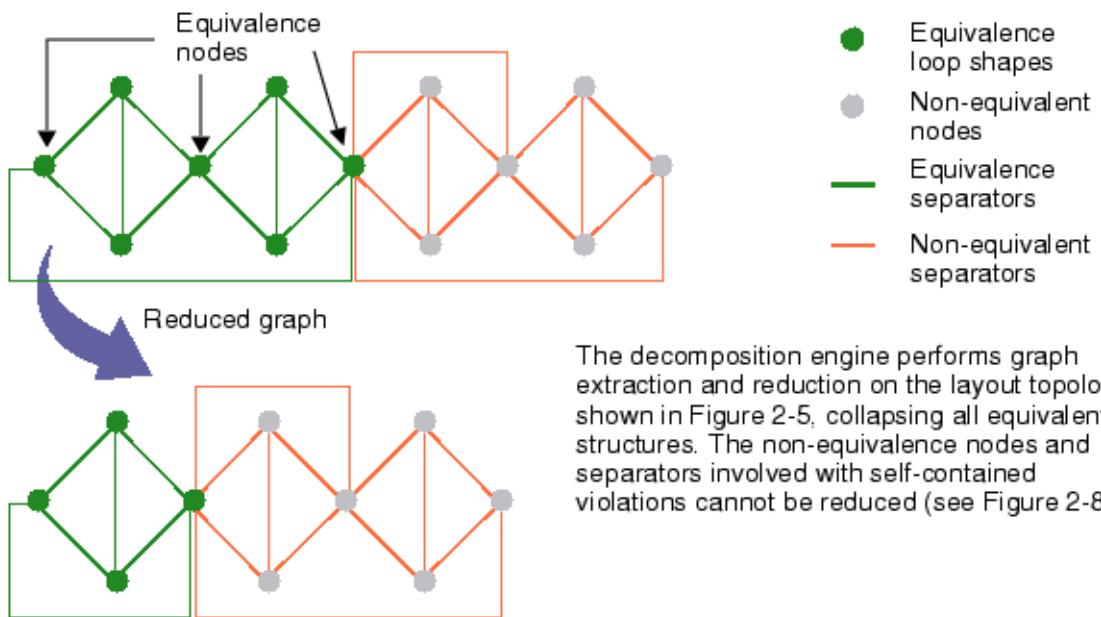


The EQUIV_LOOP_VIOLATION_SEP MAP keyword outputs separators that can ultimately identify color conflicts.

These violation shapes are output by the EQUIV_LOOP_POLYGONS MAP keyword.

Equivalence Loop Polygons — these shapes included in the equivalence loop have slashed fill. They highlight only equivalent nodes used to reduce the graph. Violation separators can be used in conjunction with equivalence shapes to locate coloring errors.

Figure 5-38. Equivalence Reduction Map



The decomposition engine performs graph extraction and reduction on the layout topology shown in Figure 2-5, collapsing all equivalent structures. The non-equivalence nodes and separators involved with self-contained violations cannot be reduced (see Figure 2-8).

Chapter 6

Example Files

Use example rule files to help speed your file development.

Whole-Shape Decomposition	341
Stitch Shape Decomposition	342

Whole-Shape Decomposition

The action keyword is used for whole shape decomposition. The action keyword processes layers containing polygon or edge-pair separators. These layers can then be derived with SVRF preceding the RET NMDPC command.

Figure 6-1. Whole-Shape Decomposition

```
//The first 14 lines are standard DRC setup.  
LAYOUT SYSTEM OASIS  
LAYOUT PATH "contact_layer.oas"  
LAYOUT PRIMARY "*"  
  
FLAG ACUTE      YES  
FLAG NONSIMPLE YES  
FLAG OFFGRID    YES  
FLAG SKEW       YES  
  
PRECISION 1000  
RESOLUTION 1  
  
DRC RESULTS DATABASE "result.oas" OASIS PSEUDO  
DRC MAXIMUM RESULTS ALL  
DRC SUMMARY REPORT "result.report" HIER  
  
LAYER input_layer 122  
LAYER separators 115  
  
// Separators can be passed in (above) or derived (below)  
  
// Layer Definitions  
critical_action_layer = INTERNAL separators< 0.150 OPPOSITE REGION  
optional_action_layer = separators NOT critical_action_layer
```

```
// Copy layers
input_layer {COPY input_layer} DRC CHECK MAP input_layer 1
separators {COPY separators } DRC CHECK MAP separators 3
critical_action_layer {COPY critical_action_layer} DRC CHECK MAP
    critical_action_layer 4
optional_action_layer {COPY optional_action_layer} DRC CHECK MAP
    optional_action_layer 5

LITHO FILE action_file [
    resolve;
    action critical_action_layer 0 1;
    action optional_action_layer 1 0;
]

conflict = RET NMDPC critical_action_layer input_layer
    optional_action_layer FILE action_file MAP conflict
mask0      = RET NMDPC critical_action_layer input_layer
    optional_action_layer FILE action_file MAP mask0
mask1      = RET NMDPC critical_action_layer input_layer
    optional_action_layer FILE action_file MAP mask1

mask0      {COPY mask0      } DRC CHECK MAP mask0      100
mask1      {COPY mask1      } DRC CHECK MAP mask1      101
conflict   {COPY conflict} DRC CHECK MAP conflict 102
```

Stitch Shape Decomposition

The dpStitchCandidates command generates stitches over regions of overlapping shapes on previously decomposed masks.

Some foundries permit polygons to be split across masks. The dpStitchCandidates LITHO FILE shown here is one possible way to automate locating stitch placements. It returns two layers, the stitch candidates and the separators. Example 6-2 shows the setup file, which would be placed inside an SVRF file and called by RET DPSTITCH.

Figure 6-2. Stitched Shape Decomposition

```
LITHO FILE dpstitch_setup_M1 [ /*
    layer m1_input visible clear
    layer m1_stitch_keepout visible clear
    layer m1_m0_precolor visible clear
    layer m1_m1_precolor visible clear
    layer usersep1 hidden clear
    tilemicrons 35 adjust

    options M1OPTS {
        layer m1_input opc clear
        layer m1_stitch_keepout visible clear
        layer m1_m0_precolor visible clear
        layer m1_m1_precolor visible clear
        layer usersep1 hidden clear
```

```

dpStitchSize (0.04 <= length <= 0.05) for (0 < width <= 0.03): \
(0.06 <= length <= 0.08) for (0.03 < width <= 0.06): \
0.08 otherwise:

dpFaceClass name(tip): \
    true for length <= 0.03 and minAdjacent >= 0.03: \
    false otherwise:

dpSpacing (tip,tip) metric(opposite): 0.06 otherwise:
dpSpacing (tip,side) metric(opposite): 0.06 otherwise:
dpSpacing (side,side) metric(opposite): 0.06 otherwise:

minIntersectionArmLength 0.04 otherwise:
intersectionStitch true for numArms >= 3: \
    false otherwise:
minConcaveCornerOverlap \
    0.06 for (0.04 <= width < 0.06):
minConcaveCornerSpacing \
    0.06 for width < 0.06: \
    0.04 otherwise:
minConvexCornerSpacing 0.06 for width < 0.06: \
    0.04 otherwise:
dpStitchCandidates criticalDistance 0.03 minArea 0.0036 \
    stitchKeepout m1_stitch_keepout \
    anchor0 m1_m0_precolor anchor1 m1_m1_precolor maxStitchWidth 0.5 \
    -outLayer stitchcandidates \
    -outLayer side \
    -outLayer separators -outLayer tip
}
setlayer stitchcandidates = dpstitch m1_input m1_stitch_keepout \
    m1_m0_precolor m1_m1_precolor usersep1 \
    MAP stitchcandidates OPTIONS M1OPTS

setlayer separators = dpstitch m1_input m1_stitch_keepout \
    m1_m0_precolor m1_m1_precolor usersep1 \
    MAP separators OPTIONS M1OPTS

*/]

```


Chapter 7

Calibre Multi-Patterning Best Practices

Best practices expedite Calibre Multi-Patterning recipe development.

Edge Versus Polygon Separators	345
Guidelines for Deriving Good Separator and Action Layers	346
Tile Size Guidelines	349
Summary of Command Settings.....	349

Edge Versus Polygon Separators

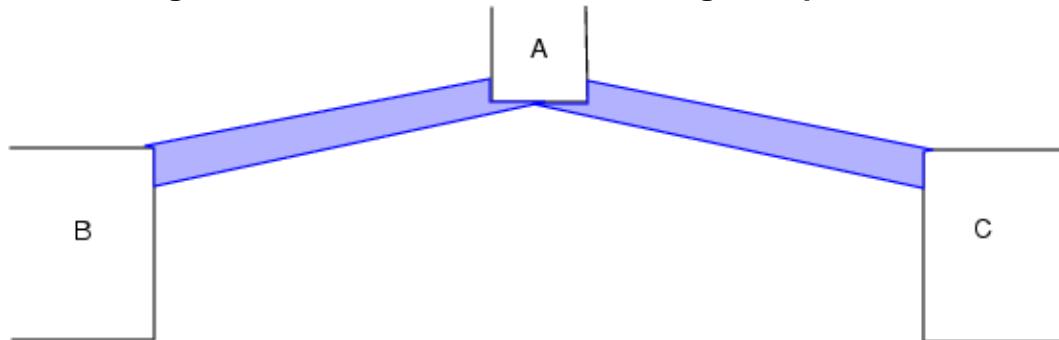
Separators and the action command handle edges or polygons differently.

The [action](#) command requires an input layer that may be a polygon or edge pair; however, it is best to use edge pairs for several reasons:

- To output warning loops and violation rings from [RET NMDPC](#) for fixing violations, the separators must be edge pairs.
- The Calibre core engine flattens hierarchy when shapes, but not edge pairs, at different levels overlap. This can lead to differences in flat and hierarchical output.
- Overlapping polygons on the same layer are merged, but edge pairs are not.

If polygon separators touch each other and are merged, it is likely to create false violations. For example, in [Figure 7-1](#), A needs to be on a different mask than B or C, but they can be on the same mask. Because the separators are polygons and touch, they merge. Calibre Multi-Patterning is directed that A, B, and C must all be separated from each other.

Figure 7-1. False Violations from Merged Separators



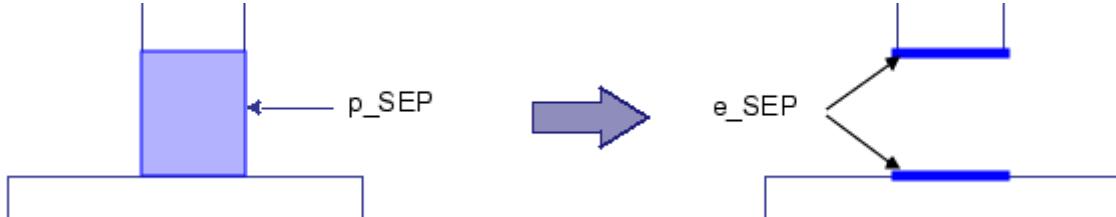
The `setlayer dpstitch` command, and thus the RET DPSTITCH operation, outputs separators as polygons. Use code similar to the following to convert the polygon separators into edge pair separators:

Figure 7-2. Converting Separators from Polygons to Edge Pairs

```
// First derive the separator layer
LITHO FILE dpSC_setup [
    layer m1      visible clear
    layer m1_ko   visible clear
    ...
    options OPTS {
        ...
        # Generates separator layer
        dpStitchCandidates ... -outLayer separators ...
    }
    // Passes separator layer out as "p_sep"
    setlayer p_sep      = dpstitch m1 m1_ko MAP separators           OPTIONS OPTS
    ...
]

p_SEP = RET DPSTITCH met1 keepOut FILE dpSC_setup MAP p_sep

// Then convert
VARIABLE pitch ...
temp = p_SEP COINCIDENT OUTSIDE EDGE met1
e_SEP = INTERNAL temp <= pitch/2
```



Guidelines for Deriving Good Separator and Action Layers

Separator and action layers are used to convey color relationships between one or more polygons and require careful assignment.

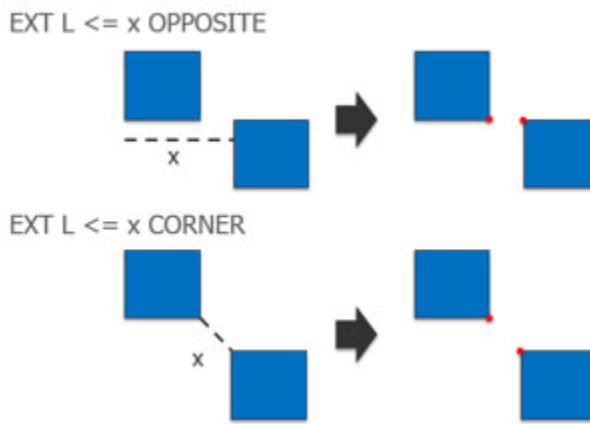
Separator and action layers (as termed by DFM MP and RET NMDPC, respectively) are type-3 edge pairs in which one edge from each edge pair touches a portion of a polygon on the target layer. Separator and action layers are employed by most of the Calibre Multi-Patterning commands. Careful derivation of these separator and action layers is needed to ensure that clear and accurate color requirements are being communicated to the tools. It is possible to generate edge pairs that the Calibre Multi-Patterning may ignore or misinterpret. The following sections

describe potential problems that can be avoided or accounted for when deriving separator and action layers.

Edge Cluster (Pair) Contents

- Edge clusters should comprise a pair of type-3 edges (they must contain exactly two edges).
- None of the edges within the edge cluster should be a singularity (also referred to as a trivial point).
- Some types of EXT, INT, or ENC measurements may generate edge pairs containing one or more singularities. In the following example, the EXT commands on the left generate results on the right containing singularities, single points marking the measurement on each shape:

Figure 7-3. Singularity Results



This table details singularity limitations of specific Calibre Multi-Patterning commands.

Table 7-1. Command Support of Singularity

Command	Singularity Support
RET NMDPC	Supports all types of singular or coincident interactions.
RET TP	None.
RET QP	None.
DFM MP	None.
DFM BALANCE	None.

Interaction Area

Make sure that each edge of an edge pair interacts with polygons by a length greater than a singularity.

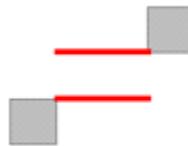
- In this example, the edge pair on the left touches no polygons and the edge pair on the right touches only one polygon. Both of these edge pairs are ignored by Calibre Multi-Patterning commands.

Figure 7-4. Polygon Interaction with Less Than Two Edges



- In this example, each edge in the edge pair touches a polygon but only as a singularity. The Calibre Multi-Patterning commands ignore this edge pair as well.

Figure 7-5. Polygon Interaction with Singularity



Derivation Operation Limitations

Most Calibre multi-patterning commands ignore or potentially misinterpret these edge cluster interactions. Some SVRF layer derivation operations used for processing these edge clusters during multi-patterning may also have similar issues. This table shows these commands and their behavioral limitations.

Table 7-2. Command Behavior with Singularities

Command	Operational Behavior with Singularities
DFM PROPERTY	Ignores edge pairs that contain singularities alone. Supports edge pairs containing both singularities and standard edges.
DFM COPY EDGE	Ignores singularities when copying edges.
DFM COPY REGION	Ignores dual-singularity edge pairs when copying regions.
DFM EXPAND EDGE	Does not expand singularities.
DFM SHIFT EDGE	Does not shift singularities.

Tile Size Guidelines

The selection of tile size is critical for optimum processing.

If the tile size is too large or too small, primary or remote memory consumption may go into swap and slow performance. The optimum tile size settings correlate to the technology node and balance the size of data per tile. Siemens EDA recommends you set tilemicrons to the following values based on node listed in [Table 7-3](#). For double-patterning, use the smaller values listed in the table.

Table 7-3. Hardware Requirements and tilemicrons Settings

Technology Node	Hardware Requirement	tilemicrons Setting
>=45 nm	2 GB/core	45 to 55
32 nm	3 GB/core	35 to 40 (30 for M1)
22 nm	4 GB/core	25 to 30 (20 for M1)

With regard to each value in [Table 7-3](#), double the memory per core on Simultaneous Multi-Processors (SMT) to support HyperThreading. For example, use an 8 GB/core at 22 nm, which is 64 GB per host on 8-core remotes and 96 GB per host on 12-core remotes.

Summary of Command Settings

Best practices result in better processing performance for Calibre Multi-Patterning.

Table 7-4. Recommended Settings Summary

Command	Default Setting	Recommended Setting	Notes
action	none	Use edge layers, not polygon layers	Polygon layers can lead to ambiguity. See “ Edge Versus Polygon Separators ”.
tilemicrons	100 um	Set size based on technology node, typically less than 35 for multi-patterning.	See “ Tile Size Guidelines ”.

Glossary

1D patterns

Sections of layout consisting of lines patterned in the same direction. Also referred to as unidirectional patterns.

2D patterns

Designs consisting of lines patterned in two dimensions. Also referred to as bidirectional patterns. Random logic designs often employ masks using 2D patterns.

coloring

The process of designating a mask for each target shape. An informal term for decomposition.

critical

A stitch or separator specification that must be achieved. If a critical layer cannot be adhered to, Calibre Multi-Patterning generates a violation.

decomposition

The process of separating the shapes on a single target layer into two or more mask layers for optical exposure that are re-combined during manufacture to form a single physical layer.

freeze

In the context of the litho-freeze-litho-etch process, a stage that stabilizes the initial exposure of resist so that the resist can better withstand the second exposure.

mandrel

An intermediate feature that occurs during self-aligned double-patterning (SADP). The mandrel assists in creating the intended features but is removed during final etch.

nicety

Synonymous with priority.

stitch

A rectangle shape that splits a target shape into overlapping shapes, which are output to separate masks. Stitching is one method of resolving coloring conflicts.

whole-shape decomposition

The style of mask decomposition that uses only complete shapes from the input layer, in contrast to splitting a single input shape to multiple masks. See also decomposition and stitch.

Index

— Symbols —

[]²²
{}²³
#¹⁷¹
|²³

— A —

action command
 best practice, [349](#)
Action layers, [20](#)
anchor command, [164](#)

— B —

Backslash (\), [157](#)
Best practices, [349](#)
Bold words, [22](#)

— C —

Calibre double patterning
 Calibre nmDPC, [15](#)
 requirements, [22](#)
Calibre dpStitch, [15](#)
 example file, [342](#)
 output, [307](#)
Calibre Multi-Patterning
 requirements, [22](#)
 SVRF commands, [37](#)
Calibre nmDP
 output, [308](#)
Calibre nmDPC, [15](#)
 example file, [342](#)
 output, [305](#)
Calibre tools
 LITHO FILE, [269](#)
 setup files, [157](#)
Coloring
 see Decompose
Command reference, [22](#)
Comments, [171](#)
Concurrency, [305](#)

Conflicts

 pitch, [160](#)

Consistency, [306, 345](#)

Contacts, [19](#)

Courier font, [22](#)

— D —

Debugging, [346](#)

Decompose

 methods, [17](#)
 overview, [16](#)
 with cuts, [21](#)

Define stitches, [307](#)

Dollar sign (\$), [323](#)

Double pipes, [23](#)

— E —

Edge types, [183](#)

Environment variables, [323](#)

Error rings, [144](#)

— F —

Face classes, [183](#)

— G —

Group shapes, [305](#)

— H —

Heavy font, [22](#)

— I —

Italic font, [22](#)

— L —

Layers

 names, [170](#)

 types, [170](#)

Line continuation, [157](#)

LITHO FILE

 multi-patterning command list, [156](#)

 SADP command list, [288](#)

LITHO FILE statements

RET NMDPC, [158](#)
Loops
 output, [345](#)

— M —

Minimum keyword, [22](#)

— P —

Parentheses, [23](#)
Pipes, [23](#)
Pitch
 critical, [160](#)
Pound sign (#), [171](#)
Precolor shapes, [305](#)
Program structure, [17](#)

— Q —

Quadruple-Patterning, [147](#)
Quotation marks, [23](#)

— R —

Recommendations, [349](#)
RET QP, [147](#)
RET TP, [151](#)
Rings
 violation, [345](#)

— S —

SADP-Specific LITHO FILE Statement and
 Commands, [288](#)
Separate shapes, [304](#)
separator command
 best practice, [349](#)
Separators
 converting, [349](#)
 troubleshooting, [346](#)
Setup files
 purpose, [16](#)
side face class, [22](#)
Slanted words, [22](#)
Square parentheses, [22](#)
Stitch
 automating, [342](#)
 defining, [307](#)
 overview, [21](#)

— T —

Tile size, [349](#)
tilemicrons command
 best practice, [349](#)

— U —

Underlined words, [22](#)

— V —

Variables, [323](#)
Vias, [19](#)
Violation rings, [144, 345](#)

— W —

Warning loops, [144, 345](#)

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the `<your_software_installation_location>/legal` directory.

