SIEMENS EDA

Calibre® Rule-Based OPC User's and Reference Manual

Software Version 2021.2

SIEMENS

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

Table of Contents

List of Figures

List of Tables

Chapter 1
Rule-Based OPC Overview

Rule-based OPC performs optical corrections by moving target shape edges based on their lithographic impact.

You perform rule-based OPC using Calibre Standard Verification Rule Format (SVRF) operations. These operations enable edge classification according to a wide range of properties, and apply the corrections defined by the rules for those properties.

This section shows how to write an SVRF rule file to perform rule-based OPC. Examples demonstrate actual code snippets for correcting several real-world design problems.

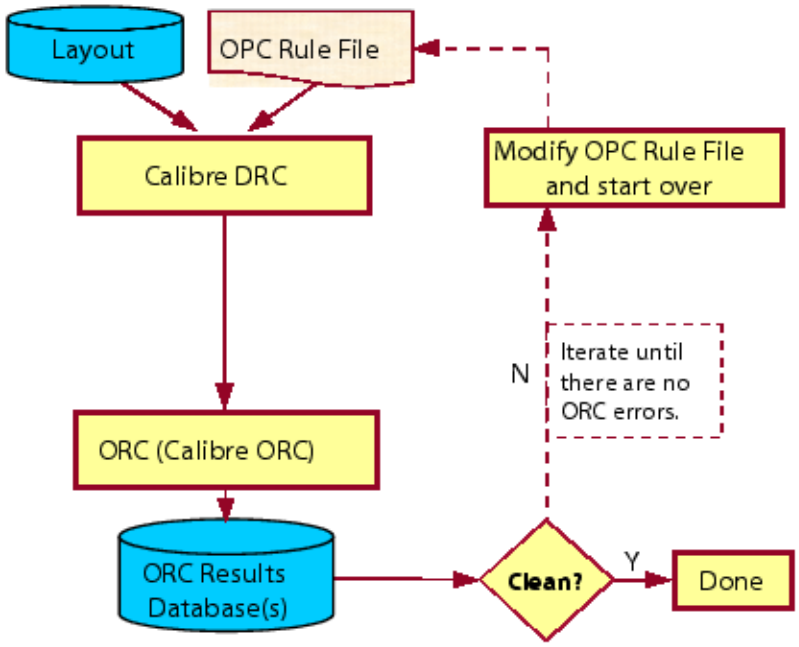
Calibre Solution

Calibre® RET solutions provide standard rule-based OPC, table-driven OPC, and model-based OPC. Calibre also provides a number of applications you can use for post-correction inspection. You may adopt one tool exclusively or mix and match tools as your process requires.

Calibre also provides a number of applications you can use for post-correction inspection. Calibre® RVE™ (results viewing environment), helps you step through DRC and manufacturing rule check (MRC) errors on any design layer. Calibre RVE is a tool for the visual inspection of design errors within a layout. This application runs concurrently with layout editors such as IC Station® or Calibre® WORKbench™. Errors hidden in a large layout can be difficult to find, and are displayed in list or table format with Calibre RVE. When you inspect an error with Calibre RVE, the viewing environment pans or zooms to the error of interest and highlights it. For information, refer to the *Calibre RVE User's Manual*.

Before invoking Calibre RVE, you must write rules files that identify the errors. One way to find errors is by using an SVRF rule file specifically designed to locate manufacturing errors. Another way is to use the Calibre® ORC™ batch tool to check the polygon edges in a layout to check for print-ability within a user-specified tolerance. Figure 1-1 shows a basic flow for performing rule-based optical rule checks (ORC) with the help of Calibre ORC.

Figure 1-1. Combining Rule-Based OPC With Calibre ORC



Related Topics

Calibre RVE User's Manual

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.
Italic	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.

Table 1-1. Syntax Conventions (cont.)

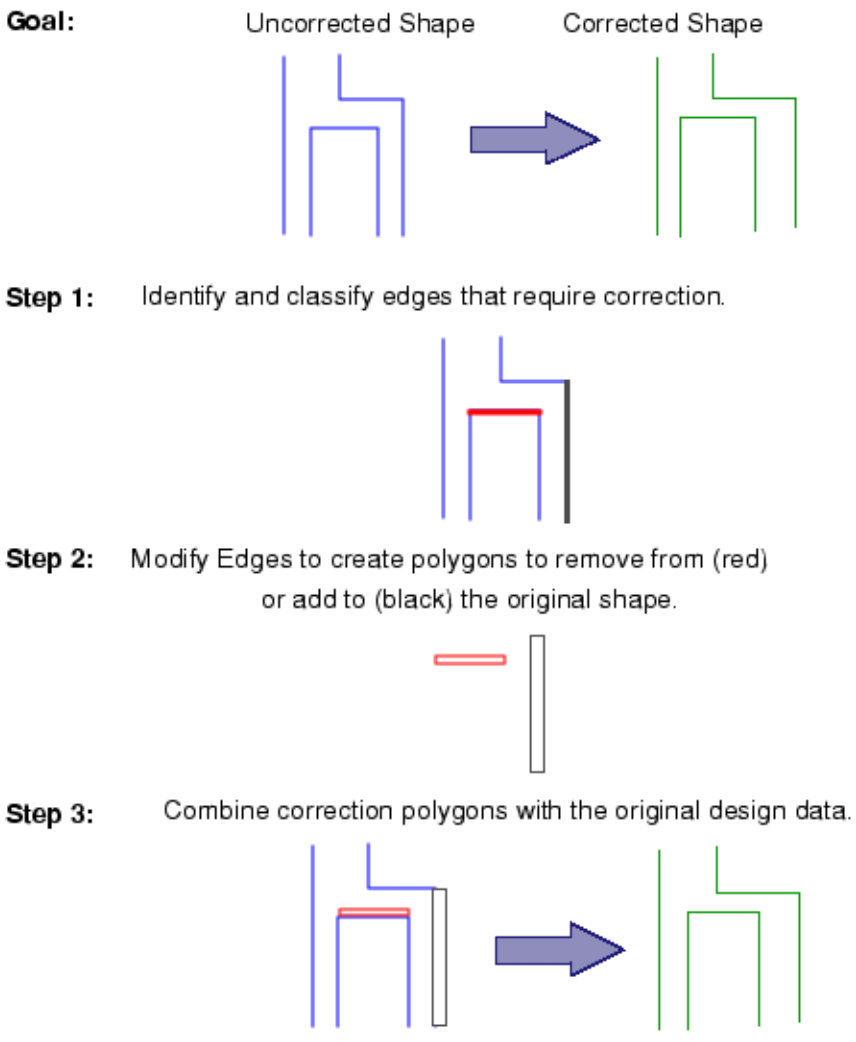
Underline	Underlining indicates either the default argument or the default value of an argument.
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[ ]	Brackets enclose optional 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.
‘ ’	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]	

Rule Writing for OPC

Adhering to systematic development methods result in accurate and consistent rule-based OPC recipes.

The following figure shows basic rule-based biasing steps.

Figure 1-2. Three Steps for Rule-Based Corrections



Classification of Edges that Require Correction

Classifying edges requiring correction is the first step to developing rule-based OPC. It can also serve as the first step in table-driven OPC and model-based OPC.

In this step, you classify edges based on physical characteristics. In particular, you look at:

- Distance to nearby features
- Feature width
- Interactions with other layers
- Topology (convex corner, concave corner, jogs, and so on)

Select edges or polygons from existing design or derived layers, using layer selector operations.

Dimension Checks as Layer Selectors

To find edges requiring correction via rule-based OPC, you often begin using the SVRF dimension check operations. SVRF provides three dimension checks:

- INTERNAL
- EXTERNAL
- ENCLOSURE

In their most generic form, these operations return error data, which is not strictly geometric data and which you cannot manipulate further using SVRF operations. However, you can override this generic behavior and direct the Calibre application to return actual edge or polygon data, thereby transforming a dimension check into a layer selector.

You control whether these operations return error data or return derived edge or polygon data through the layer specification method and the use of the Region keyword. The operations accept the following layer specifications:

- **Layer name alone —** *INT POLY <= 2*

  Operations return error data, which cannot be manipulated further.

- **Layer name in square brackets —** *INT [POLY] <= 2*

  Operations return edge data comprising those edges that satisfy the constraint. These data are also referred to as positive edge data.

- **Layer name in parenthesis —** *INT (POLY) <= 2*

  Operations return edge data comprising those edges that do not satisfy the constraint. These data are also referred to as negative edge data.

- **Operation plus the keyword REGION —** *INT POLY <= 2 REGION*

Operations return polygon data consisting of polygons created by joining opposing edges that satisfy the constraint. This keyword functions as a layer constructor rather than a layer selector.

Related Topics

Use of Metrics With Dimension Checks

Edge Classification

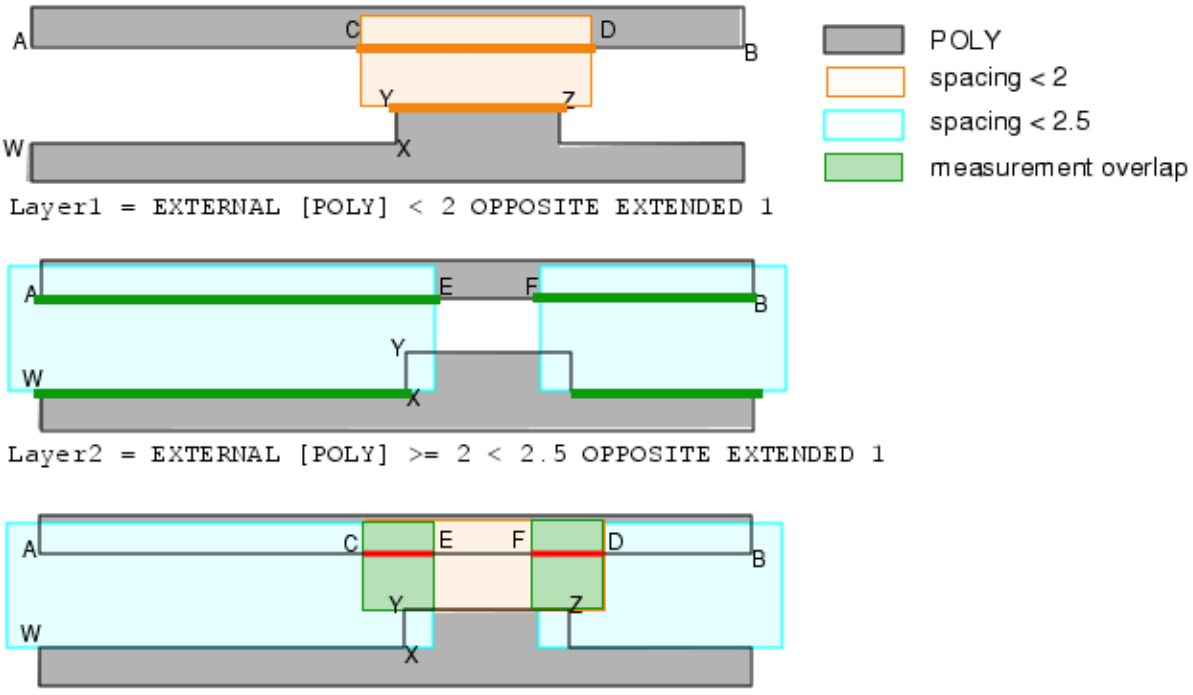
Use of Metrics With Dimension Checks

Edge selection can be performed incorrectly by selecting the same edge multiple times, selecting too many edges, and selecting edges to the side of a measurement region. Using metrics with dimension checks reduces incorrect edge selection.

Selection of the Same Edge Multiple Times

Selecting the same edge multiple times occurs when the measurement region for one edge overlaps the measurement region for another edge. In Figure 1-3, the two operations both return edges CE and FD. This could result in rule-based OPC attempting to apply two different biases to these edges.

Figure 1-3. Overlapping Measurement Regions



Shielding

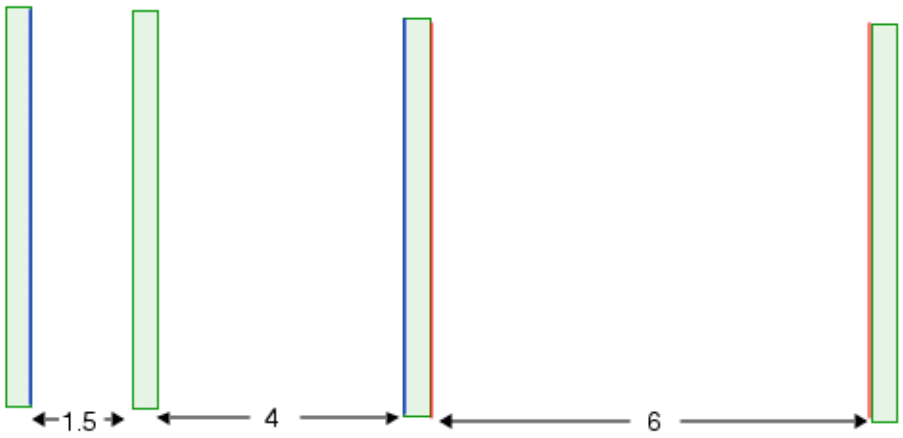
Unintended edges may be selected by the EXTERNAL operation when edges are unshielded. The operation may return a pair of edges separated by the specified space but not guarantee other edges are not between them.

For example, to find all edges of a POLY line layer separated from other POLY lines by 0.100, this operation may not return the intended edges:

```
EXTERNAL [POLY] <= 0.100 OPPOSITE
```

Figure 1-4. EXTERNAL Unshielded Edges

In a layout where the widths of each polygon are 0.50, the preceding operation would return both the pair of red lines (which are isolated) and the pair of blue lines (which are not isolated). To avoid selecting potentially unwanted edges, you can selectively return edges by ordering edge classification as seen in the following section.



Order of Classification

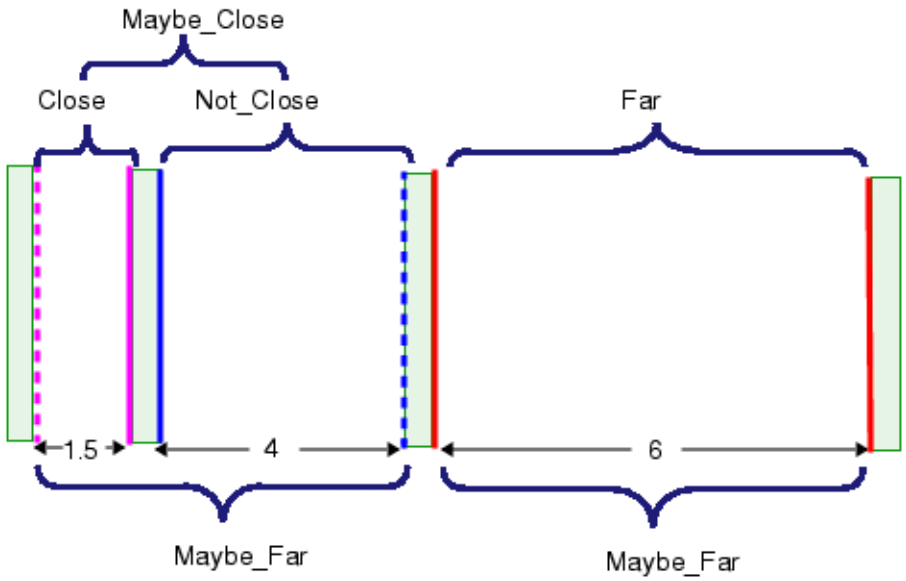
You can by classifying the most dense edges first, then removing the more dense lines from the results of the larger checks. This returns only the less dense lines. The following example shows how you can use a series of operations to classify edges into three groups:

- **Close —** less than 2 microns from the nearest polygon.
- **Not_Close —** between 2 microns and 5 microns from the nearest polygon.
- **Far —** 6 microns from the nearest polygon.

Figure 1-5. Avoiding Shielded Edge Selection

EXTERNAL [POLY] < = 6 OPPOSITE returns two sets of edges, those drawn in red, and those drawn as dotted lines. The first NOT COINCIDENT EDGE operation removes the blue edges,

creating the derived layer *far1*. The second NOT COINCIDENT EDGE operation also removes the magenta edges, leaving only the red edges on the derived layer *far*, which are truly isolated.

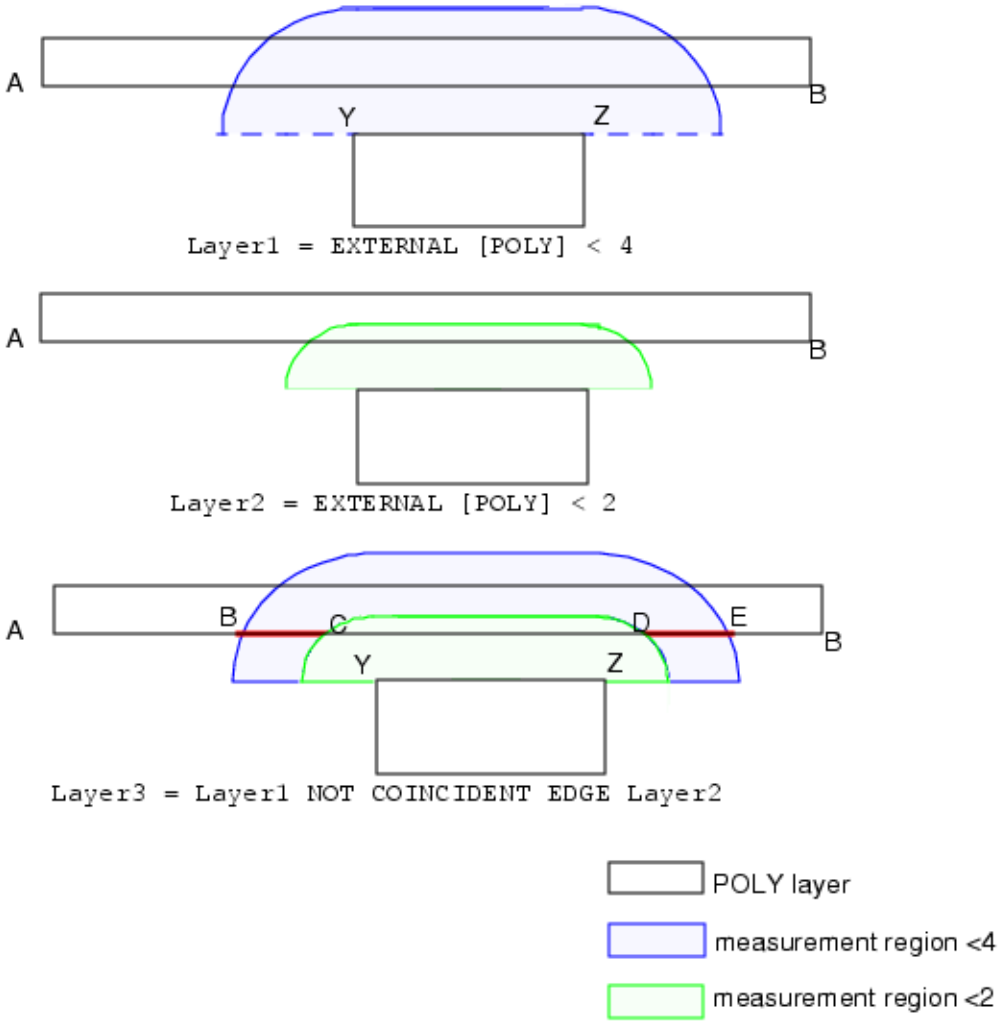



```
Close = EXTERNAL [POLY] < 2 OPPOSITE
Maybe_Close = EXTERNAL [POLY] < 5 OPPOSITE
Not_Close = Maybe_Close NOT COINCIDENT EDGE Close
Maybe_Far = EXTERNAL [POLY] <= 6 OPPOSITE
Far1 = Maybe_Far NOT COINCIDENT EDGE Not_Close
Far = Far1 NOT COINCIDENT EDGE Close
```

Metrics Assist With the Selection of Edges

Selecting adjacent edges (on the side) of a measurement region may occur when using either Euclidean or Square metrics, which do not allow control of the lateral portions of a measurement region. The preceding technique (avoiding the selection of unintended edges due to lack of shielding) can lead to selecting edges to the side of the measurement region when using the Euclidean metric.

Figure 1-6. Selecting Lateral Edges of a Measurement Region



Layer3 contains edges BC and DE. If your intention is to find edges that meet the spacing condition of between 2 and 4 microns from another edge, this code would return useless data. The same problem would result from using a single command with a range of values, for example, EXTERNAL POLY >= 2 < 4 SQUARE. You avoid this problem by using either the OPPOSITE or OPPOSITE EXTENDED metrics. With the OPPOSITE metric, the region does not extend to the side of the edge for which the region is created. With the EXTENDED metric you control how far it extends to the side, and can therefore define the same extension for both operations.

Use INTERNAL to Find Convex Corners

You can use INTERNAL to find convex corners:

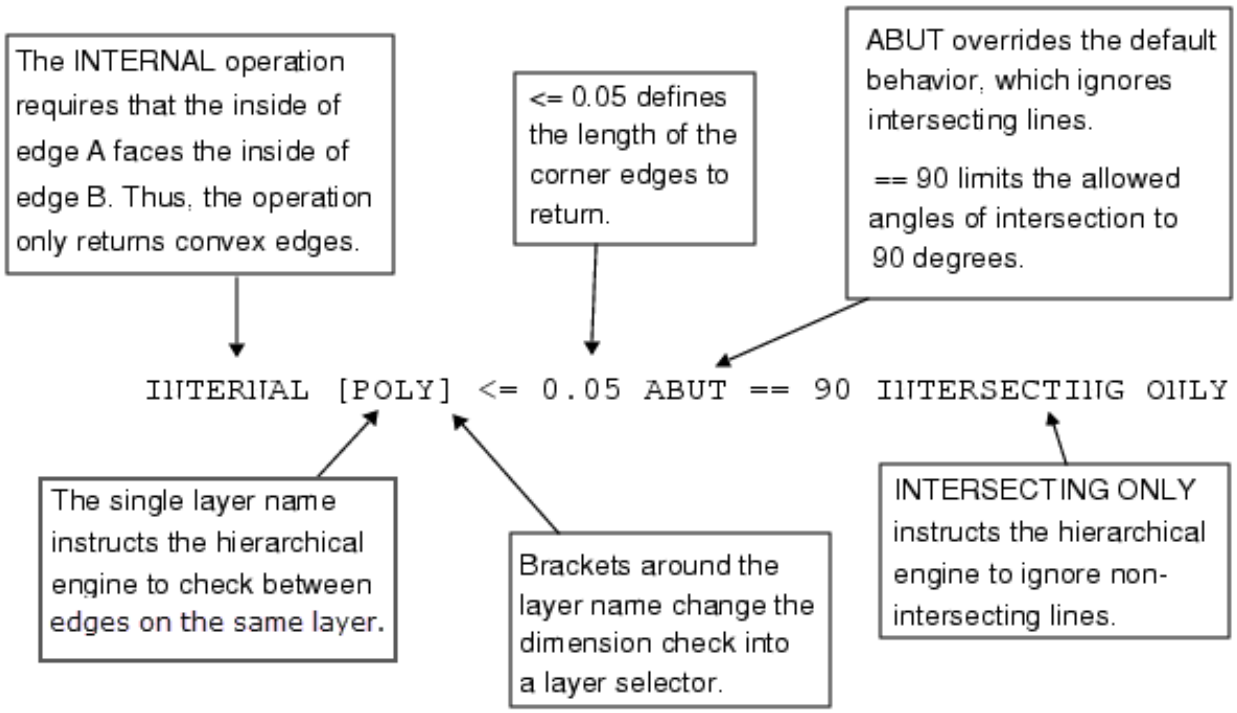
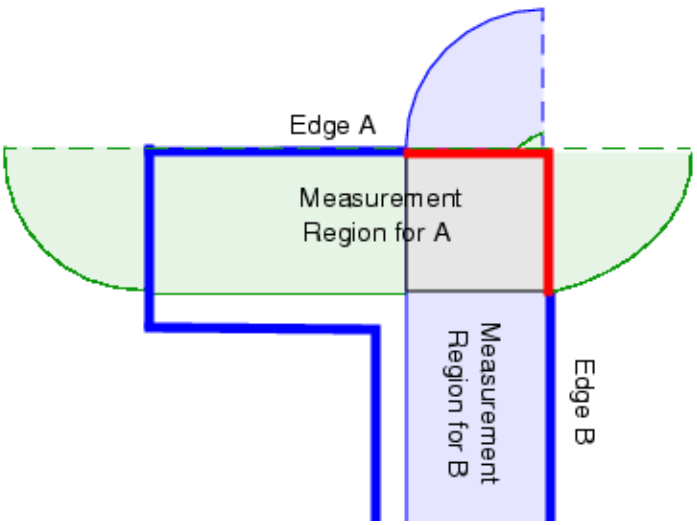


Figure 1-7. Finding Convex Corners

Figure 1-7 shows the measurement regions the Calibre® nmDRC™ hierarchical engine creates when you use the INTERNAL operation to find convex corners.



Use EXTERNAL to Find Concave Corners

You can use EXTERNAL to find concave corners:

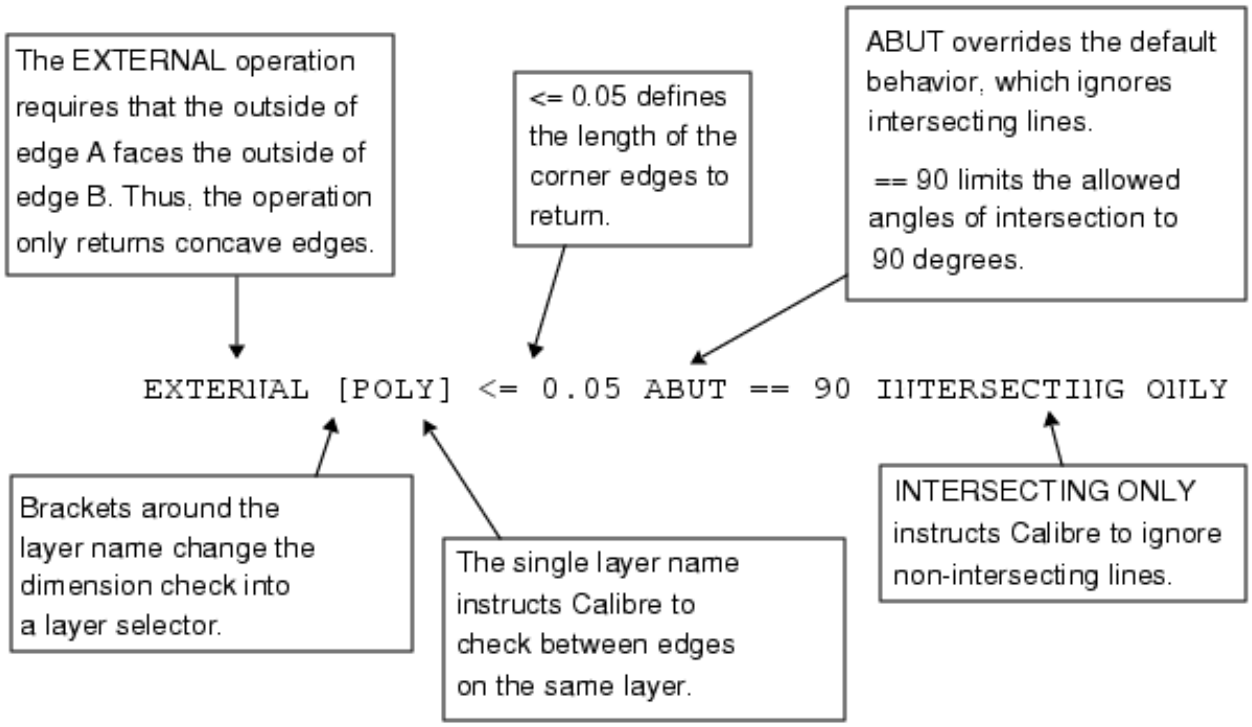
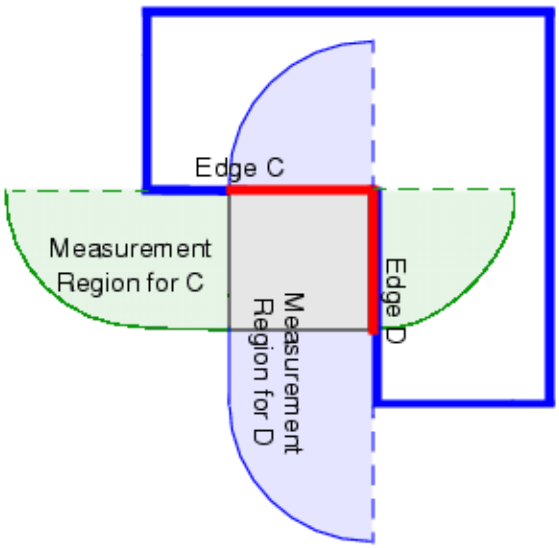


Figure 1-8. Finding Concave Corners

Figure 1-8 shows the measurement regions the Calibre nmDRC hierarchical engine creates when you use the EXTERNAL operation to find concave corners.



Related Topics

Dimension Checks

Edge Classification

Edges must be classified correctly to provide expected biasing results.

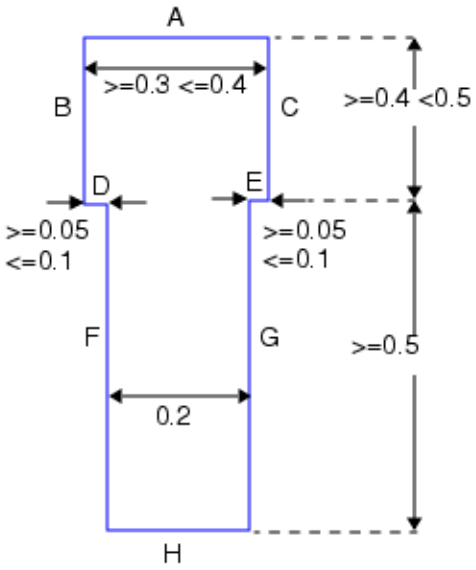
CONVEX EDGE

Use CONVEX EDGE to select edges based on line-end length, the angle at one or both ends, and the length of abutting edges.

Figure 1-9. Edge Selection With CONVEX EDGE

This operation uses layer selection, not dimensional checks, to perform edge selection. After finding the line-ends, you can then use dimensional checks to classify them based on their distance to other edges.

Figure 1-10. Line-End Definition



A: `CONVEX EDGE POLY ANGLE1==90 ANGLE2==90 LENGTH1>=0.4<0.5 LENGTH2>=0.4<0.5 WITH LENGTH>=0.3<=0.4`

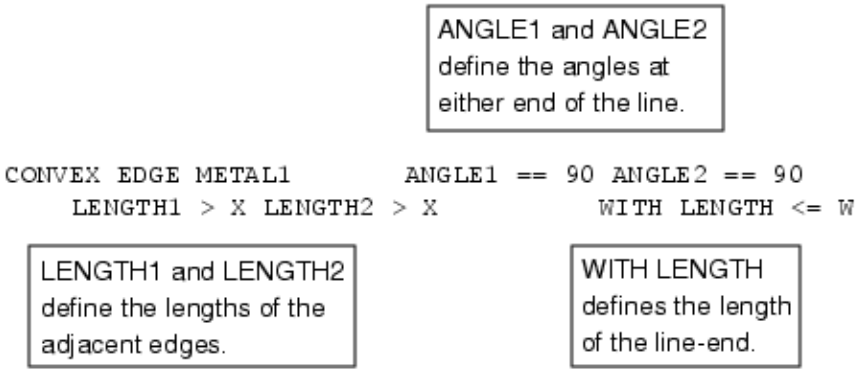
B, C: `CONVEX EDGE POLY ANGLE1==90 ANGLE2==90 LENGTH1>=0.05<=0. 1 LENGTH2>=0.3<=0.4 WITH LENGTH>=0.4<0.5`

D, E: `CONVEX EDGE POLY ANGLE1==90 ANGLE2==270 LENGTH1>=0.4<0.5 LENGTH2>=0.5 WITH LENGTH>=0.05<=0.1`

F, G: `CONVEX EDGE POLY ANGLE1==270 ANGLE2==90 LENGTH1>=0.05<=0.1 LENGTH2==0.2 WITH LENGTH>=0.5`

H: `CONVEX EDGE POLY ANGLE1==90 ANGLE2==90 LENGTH1>=0.5 LENGTH2>=0.5 WITH LENGTH==0.2`

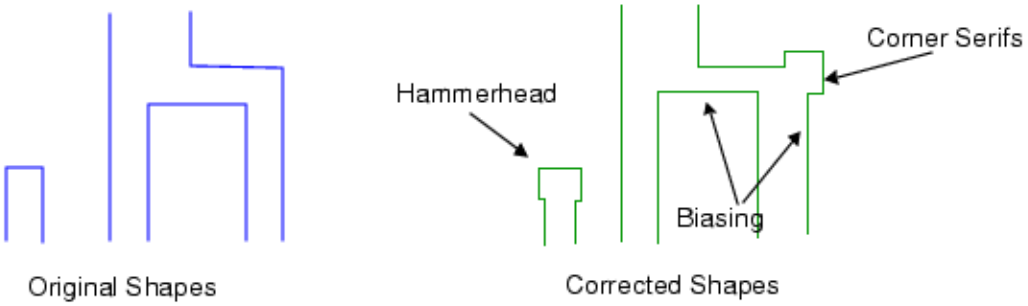
This line-end is defined having two 90 degree convex corners, a maximum length (W), and having adjacent edges greater than a specified length (X). This third criteria ensures no jogs or other small overhangs are counted as line-ends.



Edge Modification

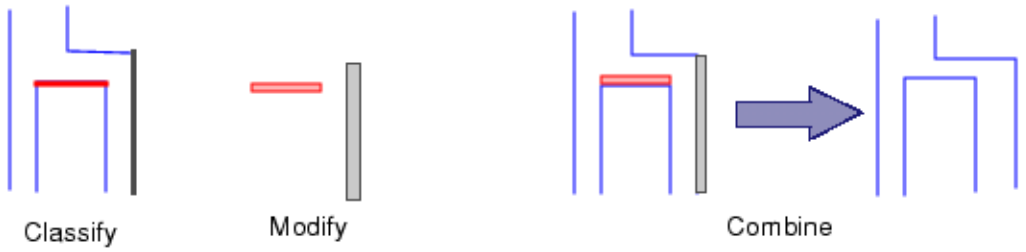
Modifying edges involves transforming the edges you classified in the previous step into polygons that, when added to or removed from the original polygons, result in the corrected shape. Before writing the rules that modify the edges, you must understand both the process you are correcting for and the options the Calibre nmDRC hierarchical engine provides you for modifying layout designs.

Figure 1-11. Typical Correction Plan



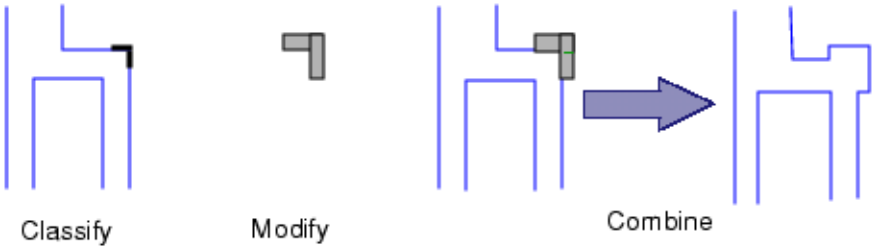
The figure illustrates three types of corrections required by a typical process: biasing, corner serifs, and hammerheads. Using SVRF, you implement these types of corrections that you can add to or remove from the target shapes.

Figure 1-12. Creating Edge Bias



The correction polygons you generate perform biasing, which increases or decreases the width of the polygon along specific edges. In this figure, the small red shape represents the portion of the original shape that must be removed. The large black polygon represents the portion that must be added.

Figure 1-13. Creating Corner Serifs



You generate correction polygons to add a corner serif. Each of the classified edges becomes a rectangle added to the target shape.

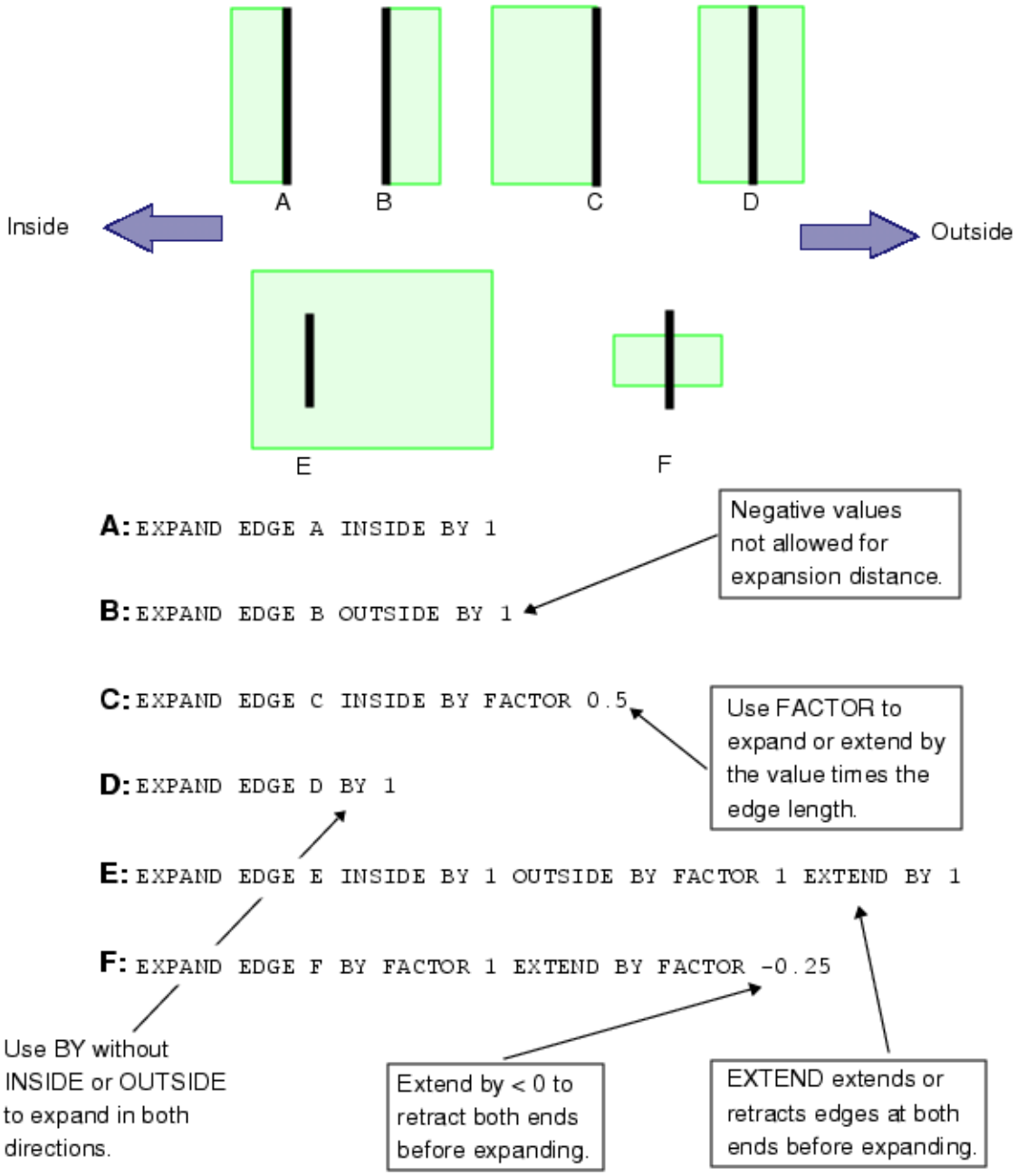
Figure 1-14. Creating Hammerheads

You generate a correction polygon to add a hammerhead to a line-end. The black polygon is the correction added to the target shape.

EXPAND EDGE

Use the EXPAND EDGE operation to create a polygon out of an edge. This operation takes an edge and expands it in the direction or directions specified. The result is a polygon.

Figure 1-15. Using EXPAND EDGE



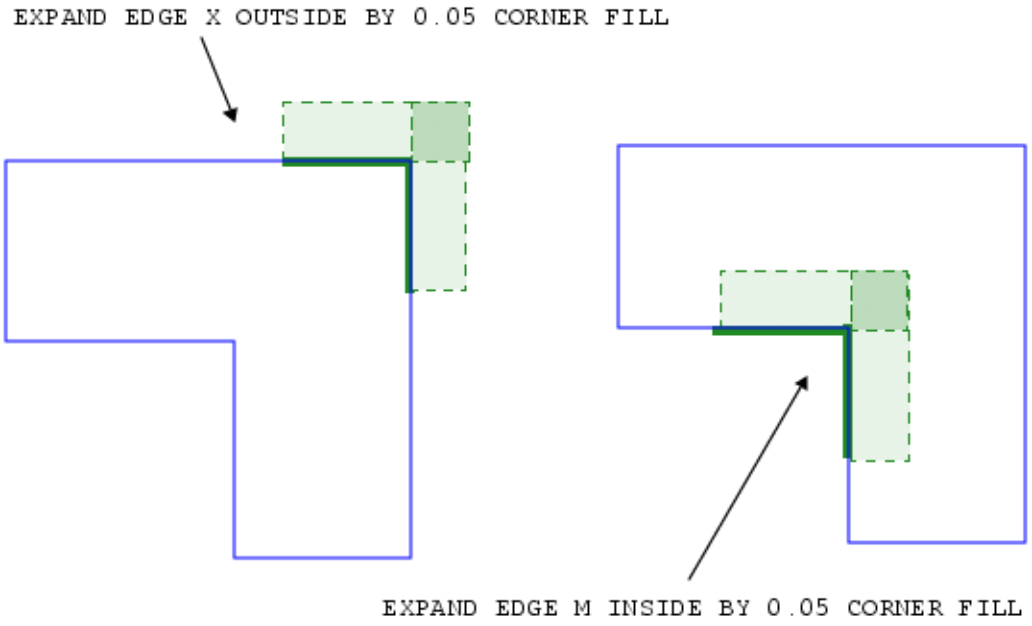
With EXPAND EDGE you grow the edge to form a polygon. The edge can be grown to the inside, the outside, or both. You can also change the edge length simultaneously.

Convex and Concave Corner Biasing

EXPAND EDGE and CORNER FILL provide quick and accurate biasing of convex and concave corners.

EXPAND EDGE biases corners by cutting notches in concave corners, and adding serifs to convex corners.

Figure 1-16. Convex and Concave Corner Biasing



EXPAND EDGE generates serifs or notches for convex and concave corners, respectively. CORNER FILL fills the square gap between rectangles formed by EXPAND EDGE at corners of the target layer.

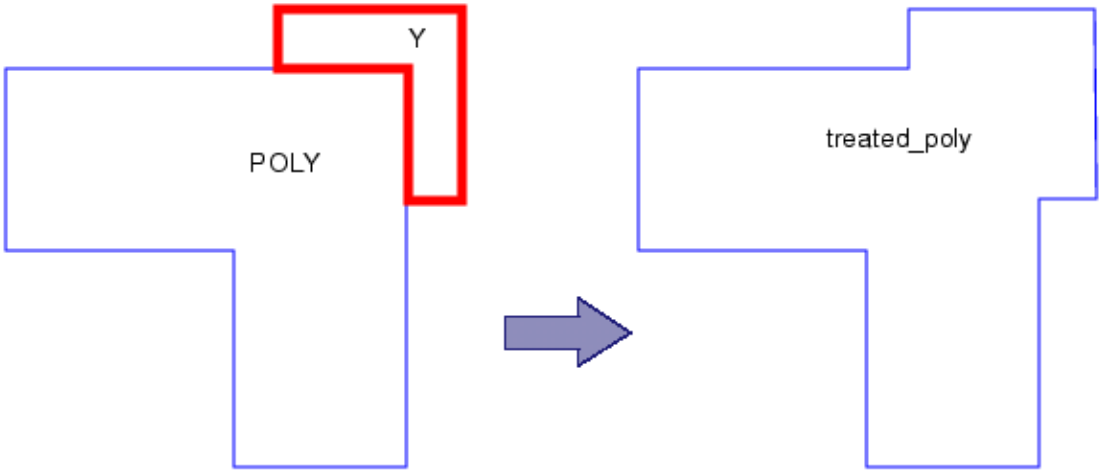
Corner Corrections

The modifications you make to edges are output to a distinct derived layer. The last step is to apply corrections to the design layer. Some derived layers contain polygonal data that must be added back to the target layer. Other derived layers may contain polygonal data to be removed from the target layer. You perform these operations using boolean operations such as AND, OR, NOT, and others.

Convex Corner Biasing

For convex corners, you form serifs that must be added to the target shape.

Figure 1-17. Merging Convex Corner Corrections



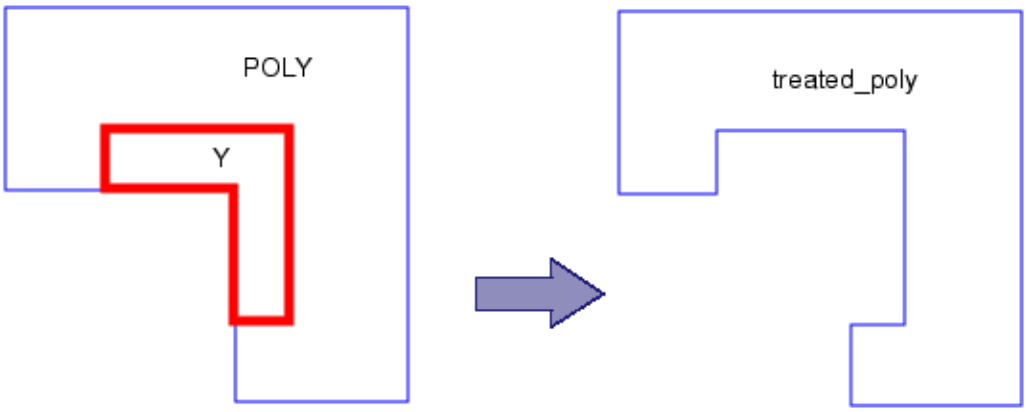

```
X = INTERNAL [POLY] <= 0.1 ABUT == 90 INTERSECTING ONLY
Y = EXPAND EDGE X OUTSIDE BY 0.05 CORNER FILL
treated_poly = POLY OR Y
```

You add the serifs and target shapes together using the OR operation. The OR operation merges polygons that touch. The figure is the target shape with the added serif.

Concave Corner Biasing

For concave corners, a notch representing negative biasing is removed from the target shape.

Figure 1-18. Merging Concave Corner Corrections




```
X = EXTERNAL [POLY] <= 0.1 ABUT == 90 INTERSECTING ONLY
Y = EXPAND EDGE X INSIDE BY 0.05 CORNER FILL
treated_poly = POLY NOT Y
```

You remove the notch from the original layer using the NOT operation.

Proper Data Storage

Combining correction and design data with layer operations requires that you store it in a usable form.

Data may be stored as a global derived layer, in the database, or both.

Table 1-2. Data Storage

If you want to ...	Do the following:
Pass data to any Calibre RET batch tool	Store the data as a global derived layer. This layer must be comprised of derived polygons because Calibre RET tools do not accept edge data as input. Calibre RET batch tools include Calibre® PRINTimage™, Calibre® ORC™, and Calibre® OPCpro™.
Send data to a mask shop	Use the rule check (signified by its enclosing braces { }) and DRC CHECK MAP statements to direct Calibre to write data to your database file using a unique layer number.
Perform additional rule-based checks on data	Store the data as both a global derived layer and in the results database (accessed by Calibre RVE). To do this, create the derived layer first, and then use the COPY operation within a rule file statement.

Examples

SVRF operations are required to implement rule-based OPC.

Via Biasing

Vias and contacts are often biased with respect to their density. Two SVRF operations can be used to increase the size of an isolated via.

Figure 1-19. Biasing an Isolated Via

The first operation finds vias with a separation from any other via of more than 200 nm. Each isolated via is then oversized by 10 nm.

Method 1

The EXTERNAL dimension check finds the isolated edges:

```
iso_via1 {
   x = EXT (via) < 2.0       // find via edges not closer than 2.0
   y = via WITH EDGE x == 4 // find the via that contains the edges
   z = SIZE y BY 0.05        // isolated via
   via OR z
}
```

Method 2

The SIZE command oversizes vias, then finds oversized vias not intersecting other vias:

```
iso_via2 {
   x = SIZE via BY 1        // grow each via by 1.0
   y = x INTERACT VIA == 1 // other via within 2.0
   z = via INSIDE y         // isolated via
   w = SIZE z BY 0.05
   via OR w
}
```

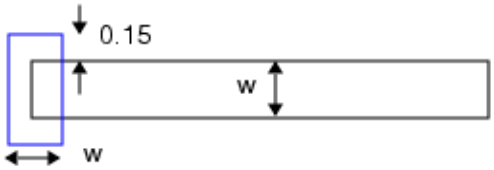
The two methods return the data in different forms. For the method outputting iso_via1, the intermediate geometries are edges; for the method outputting iso_via2, the intermediate geometries are polygons. They require different operations to isolate the vias. WITH EDGE is

used in the iso_via1 method because it finds polygons from edges; INSIDE is used for the iso_via2 method because it finds polygons from polygons.

Hammerhead Construction

Hammerheads are generally applied at the end of target geometries.

Figure 1-20. Line-End Specification Example



The figure shows a hammerhead added to each metal line that extends beyond the line of width W by at least 0.15, is centered on the end of the line, and extends to the side of the line-end by at least 0.15. The hammerhead treatment to the line-end is performed irrespective of line length.

SVRF Examples

The following example shows an SVRF sequence you may use to find the lines and add the hammerheads:

Figure 1-21. Adding Hammerheads

The variable W represents the length of the line-end. You can often use the same procedure with different values to implement a set of related rules.

```
metal_end_cap {
   x = CONVEX EDGE m1 == 2 // find metal edges at convex lines
   y = LENGTH x == W
   z = EXPAND EDGE y OUTSIDE BY W/2 INSIDE BY W/2 EXTEND BY W/2
   M1 OR y
}
```

Figure 1-22. Width-Dependent Hammerheads

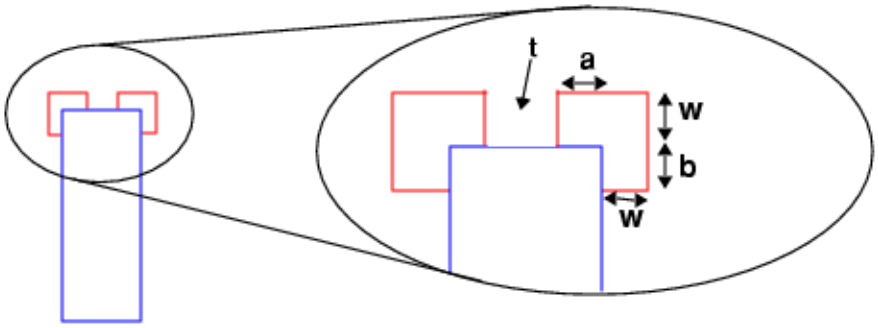
Assume that the size of hammerheads vary according to the width of the line, and that only lines with widths less than 0.9 um require hammerheads:

```
metal_end_cap {
   x = CONVEX EDGE m1 == 2 // find metal edges at convex lines
   w1 = LENGTH x <  0.3
   w2 = LENGTH x >= 0.3 < 0.5
   w3 = LENGTH x >= 0.5 < 0.7
   w4 = LENGTH x >= 0.7 < 0.9
   y1 = EXPAND EDGE w1 OUTSIDE BY 0.15 INSIDE BY 0.15 EXTEND BY 0.15
   y2 = EXPAND EDGE w2 OUTSIDE BY 0.25 INSIDE BY 0.25 EXTEND BY 0.25
   y3 = EXPAND EDGE w3 OUTSIDE BY 0.35 INSIDE BY 0.35 EXTEND BY 0.35
   y4 = EXPAND EDGE w4 OUTSIDE BY 0.45 INSIDE BY 0.45 EXTEND BY 0.45
   M1 OR ((y1 OR y2) OR (y3 OR y4))
}
```

Corner-Based Hammerheads

"Convex and Concave Corner Biasing" on page 27 shows how you can apply corner biasing to individual convex corners. Hammerheads are another method of biasing line-end corners.

Figure 1-23. Line-End Corner Biasing




```
line_end_corners {
  q = CONVEX EDGE m1 == 2 // find metal edges at convex lines
  r = LENGTH q < L
  s = EXPAND EDGE r INSIDE BY b OUTSIDE BY W EXTEND BY W
  t = EXPAND EDGE r OUTSIDE BY W EXTEND BY (-a)
  m1 OR (s NOT t)
}
```

Alternative approaches can be used when applying serifs as hammerheads to a pair of convex corners on line-ends of widths less than L.

Chapter 2
Calibre Table-Driven OPC

Table-Driven OPC (Calibre® TDopc™) consists of three SVRF operations, OPCBIAS, LITHO NMBIAS™ and OPCLINEEND.

These operations provide a fast and efficient rule-based approach to performing OPC that enables corrections to be applied to individual polygon edges based on properties of those edges. It is called table driven because you use a rules table as input to a batch tool that performs OPC on your layout.

This section provides a brief introduction to these tools and to the rest of this manual, which provides in-depth coverage of the OPCBIAS, LITHO NMBIAS and OPCLINEEND operations. The remaining sections are organized according to the operation you use.

The OPCBIAS topic covers variable isolated or dense line, and via biasing:

- OPCBIAS Command

The LITHO NMBIAS topic covers line and via biasing:

- LITHO NMBIAS Command

The OPCLINEEND topic covers line-end extension generation in the form of either simple line-end extensions or hammerheads with serifs:

- OPCLINEEND Command

Operation-specific sections assume your familiarity with SVRF and execution of Calibre programs. For more information of these subjects, refer to:

- Calibre nmDRC Hierarchical Engine
- SVRF Rule Files

Table Practices

Every table summarizes OPC rule practices. While practices may vary from one design or lithography style, many use the same table format.

The fact that two tables look identical does not guarantee that they translate into the same set of rules. If the two tables represent different practices (for example, there are different assumptions behind the tables), the resulting rules may be significantly different.

The discussions that follow can not assume any specific practice. They provide you with generic sets of steps you can use to translate your tables into rules. Before you begin to convert a rules table into a set of rules using either the OPCBIAS or OPCLINEEND operation, you must have an understanding of the technology you use and the tables that represent that practice.

Types of questions you should be able to answer are:

1. Which discrete table values actually represent ranges of values?
2. What is the minimum spacing at which biasing or extension is performed?
3. What is the maximum line width requiring correction?
4. What is the minimum spacing allowed between any two features?
5. Does the table assume that opposite edges are both biased and extended at the same time or that the corrections are sequential?
6. What is the minimum edge length to be corrected?
7. What is the minimum edge length to keep after correction?

Calibre TDopc Execution

OPCLINEEND, OPCBIAS, and LITHO NMBIAS involve writing SVRF commands including a TDopc operation, then invoking Calibre to process that SVRF file.

Running OPCLINEEND

OPCLINEEND performs biasing using width and height of line-ends (with no consideration for spacing violations) applied to the entire poly layer.

An exhaustive SVRF file would possibly include rules to correct spacing violations introduced by the default line-end treatment, and preprocessing the poly layer to specify shapes requiring correction.

Prerequisites

- Layout in GDS or OASIS®[1] format.
- A developmental SVRF file named *lineend.svrf* (see Examples at the end of this section).

Procedure

1. Write a rule file defining the lineend treatment required. The rule file must:
   a. Read in the target layer to receive line-end treatment.
   b. Write a rule for each width and height combination defining how line-ends meeting the criteria should be treated.
2. Run calibre using this rule file:

```
calibre -drc -hier lineend.svrf
```

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.

3. To view results, open *lineend_out.gds* in Calibre WORKbench. Figure 2-1 shows typical results:

Figure 2-1. OPCLINEEND Results

Examples

This SVRF file is a starting point for the development of OPCLINEEND rules:

```
//*************************************************************************
// Mentor Graphics (c) 2014
//*************************************************************************

// ********************************************************Input Section
LAYOUT SYSTEM GDSII
LAYOUT PATH "lineend.gds"
LAYOUT PRIMARY "top"
FLAG SKEW YES
LAYOUT ERROR ON INPUT YES
PRECISION 1000
RESOLUTION 1
LAYER poly  4
// *******************************************************Output Section
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "lineend_out.gds" GDSII
DRC SUMMARY REPORT "lineend.rep" HIER
DRC MAXIMUM VERTEX 199
DRC KEEP EMPTY YES
DRC CHECK MAP poly 4
DRC CHECK MAP lineend_opc 12

// *******************************************************Output Layers
poly {COPY poly}
lineend = OPCLINEEND poly
      WIDTH <= 0.2   HEIGHT >= 0.5       END 0.02  SERIF 0.02 0.1
      WIDTH <= 0.2   HEIGHT >= 0.1 < 0.5 END 0.02  SERIF 0    0

lineend_opc { lineend NOT poly }
```

The DRC CHECK MAP statements define the layer assignment for the output of this run.

Normally, you would output the layer lineend, which contains the polygons resulting from lineend treatment. To help you understand how OPCLINEEND modifies your data, this example returns a special layer containing only the modifications to your data.

Running OPCBIAS

OPCBIAS applies bias based on space and widths. Biasing can be positive or negative, adding to or reducing the width of a target shape, respectively.

Prerequisites

- Layout in GDS or OASIS format.
- A developmental SVRF file named *bias.svrf* (see Examples at the end of this section).

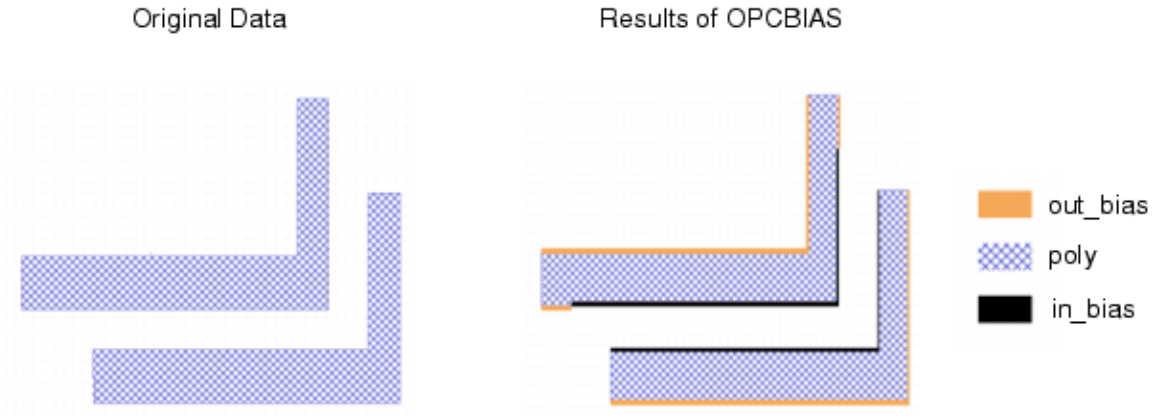
Procedure

1. Write a rule file defining the biasing required. The rule file must:
   a. Read in the target layer to receive biasing.
   b. Write a rule for each width and height combination defining how edges meeting the criteria should be treated.
2. Run calibre using this rule file:

```
calibre -drc -hier bias.svrf
```

3. To view results, open *bias_out.gds* in Calibre WORKbench. Figure 2-2 shows typical results:

Figure 2-2. OPCBIAS Results



Examples

This SVRF file is a starting point for the development of OPCBIAS rules:

```
//*************************************************************************
// Mentor Graphics (c) 2014
//*************************************************************************

// ********************************************************Input Section
LAYOUT SYSTEM GDSII
LAYOUT PATH "bias.gds"
LAYOUT PRIMARY "top"
FLAG SKEW YES
LAYOUT ERROR ON INPUT YES
PRECISION 1000
RESOLUTION 1
LAYER poly  4
// ********************************************************Output Section
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "bias_out.gds" GDSII
DRC SUMMARY REPORT "bias.rep" HIER
DRC MAXIMUM VERTEX 199
DRC KEEP EMPTY YES
DRC CHECK MAP poly 4
DRC CHECK MAP out_bias 12
DRC CHECK MAP in_bias 14

// ********************************************************Output Layers
poly {COPY poly}

bosd = OPCBIAS POLY
SPACE <=0.2 OPPOSITE EXTENDED 0.2 WIDTH <= 0.2 OPPOSITE EXTENDED 0.3 MOVE -0.01
SPACE <=0.2 OPPOSITE EXTENDED 0.2 WIDTH >0.2 <0.3 OPPOSITE EXTENDED 0.3 MOVE -0.02
SPACE >0.2 OPPOSITE EXTENDED 0.2 WIDTH <= 0.2 OPPOSITE EXTENDED 0.3 MOVE  0.01
SPACE >0.2 OPPOSITE EXTENDED 0.2 WIDTH >0.2 <0.3 OPPOSITE EXTENDED 0.3 MOVE  0.02

out_bias { bias NOT poly }
in_bias  { poly NOT bias }
```

The DRC CHECK MAP statements define the layer assignment for the output of this run.

Running LITHO NMBIAS

Calibre nmBIAS applies bias based on space and widths. Biasing can be positive or negative, adding to or reducing the width of a target shape, respectively. Either rule or table mode may be used.

For more information regarding Calibre nmBIAS, see “LITHO NMBIAS Command” on page 43.

Chapter 3
LITHO NMBIAS Command

LITHO NMBIAS-based commands perform biasing by calling a LITHO FILE statement that contains a setlayer nmbias command.

LITHO NMBIAS

SVRF command

Biases layers and calls setlayer nmbias.

Usage

LITHO [CL] **NMBIAS FILE** ***file targetLayer*** [*inLayer*] ... **MAP** ***outLayer***

Arguments

- **LITHO** [CL] **NMBIAS**

  A required pair of keywords that uses the Calibre® nmDRC™ hierarchical engine to perform biasing. The CL and curvilinear_bias keywords must be specified for curvilinear capability.

- **FILE** ***file***

  A required keyword followed by the pathname for the setup file. Alternatively, the setup file can be specified as an inline file definition.

- ***targetLayer***

  A required input layer that serves as the target layer. Additionally, this layer must be declared as type opc. The optional inLayers can be specified in any order.

- *inLayer*

  An optional list of supplementary input layers that provide additional simulation data and fragmentation assistance. All layers must be polygon layers. The types for these additional layers are specified in the LITHO FILE statement:

  - correction
  - external
  - hidden
  - island
  - reference
  - sraf
  - visible

  The required targetLayer and the optional inLayers can be specified in any order.

  Note

  Siemens EDA recommends using the reference layer:

  - Use the reference layer type for optimal control of edge fragmentation for LITHO nmBIAS. The reference layer can be passed to BIASRULE to perform various spacing constraints. The reference layer type is enabled when declared with the nmOPC fragment_interlayers command:

```
layer <reflayer> reference
   fragment_interlayers <targetlayer> <reflayer> ...
   fragment_inter ...
   BIASRULE <targetlayer> [<reflayer>] ...
```

- Where the reference layer type is used for multi-patterning applications, the layer must be marked with a pattern ID.

- **MAP *outLayer***

  A required keyword and argument specifying the name of the layer from the setup file. This layer is mapped to the layer in the rule file for output.

Description

The LITHO NMBIAS command biases polygons while respecting internal width and external space constraints. Use this command in conjunction with the setlayer nmbias command to derive output target layers for SVRF.

Examples

The following example uses the LITHO NMBIAS call to generate the AfterBiasrule layer using the nmbias.setup LITHO FILE. In the *nmbias.setup* file a layer command declares the input layer, BiasruleLyr, along with many other settings and options (not shown). The output is written to the postbias layer, which is returned to LITHO NMBIAS and mapped to the AfterBiasrule layer.

```
AfterBiasrule = LITHO NMBIAS FILE nmbias.setup BiasruleLyr MAP postbias
   LITHO FILE nmbias.setup [/*
   ...
   setlayer postbias = nmbias bias.in MAP bias.in OPTIONS first.pass
   ...
*/]
```

setlayer nmbias

setlayer command

Biases drawn layers.

Usage

setlayer ***output_layer_name*** = **nmbias** ***input_layer*** ...
MAP ***name***
OPTIONS ***opt_name***

Arguments

- ***output_layer_name***

  A required keyword specifying the name of the new output target layer. The new layer is created and the results of the operation are written to the layer. This layer must be a polygon layer.

- ***input_layer***

  A required layer name specifying the name of the input layer to be biased. Any number of subsequent input_layers may be specified. The layers specified must be polygon layers.

- **MAP** ***name***

  An required keyword followed by the associated argument indicating the layer or tag containing the output to be returned. This keyword is required even for non-concurrent runs.

- **OPTIONS** ***opt_name***

  A required argument that specifies the name of an options block containing any keyword to be used by the setlayer nmbias command. See options for more information.

Description

This command creates and derives layers, depending on the options used. You create output layers using the design layers you defined with the layer command, and you subsequently use one or more of them as output in your SVRF configuration file.

The setlayer command also uses concurrency in order to map different outputs to setlayer commands. The MAP keyword is required even for non-concurrent runs. The OPTIONS keyword must be declared last in the command line. The order of layers passed should match the definitions of layers declared using LITHO NMBIAS.

Examples

```
setlayer postbias = nmbias bias.in MAP bias.in OPTIONS first.pass
```

options

LITHO FILE command

Provides control and parameter specification for fragmentation and biasing operations.

Usage

options ***opt_name*** ‘**{**’
options_list
‘**}**’

Arguments

- ***opt_name***

  A required argument specifying the unique name of the options block. There is a 15 character limit on the name.

- ***options_list***

  The layers and keywords to be passed to setlayer nmbias. In this options block, you can specify Calibre nmBIAS Tcl Scripting Commands.

Description

Names the options block and lists all commands to be read by the setlayer nmbias command. The options block is contained by the LITHO FILE statement. The commands within the options block evaluate all variables before runtime. The options block is then read by the setlayer nmbias command. Any number of option blocks may be specified within the LITHO FILE statement. The options block commands include various functions:

- Input layers and their types
- Fragmentation and controls
- Tagging
- Biasing controls

Examples

This example demonstrates a simple options block implementation:

Figure 3-1. options Block Implementation

```
AfterBiasrule = LITHO NMBIAS FILE nmbias.setup BiasruleLyr MAP postbias

LITHO FILE nmbias.setup [/*
   tilemicrons 40 adjust
   layer bias.in hidden

   # options block is named 'first.pass'
   options first.pass {
      version 1
      layer bias.in opc
      ...
   }

# first.pass is passed to the setlayer command
setlayer postbias = nmbias bias.in MAP bias.in OPTIONS first.pass
```

Calibre nmBIAS Tcl Scripting Commands

LITHO NMBIAS uses Tcl Scripting Commands to perform biasing.

One command must be specified per line.

Table 3-1. Calibre nmBIAS Tcl Scripting Commands Summary

Keyword	Description
BIASRULE	Biases edges using a rule- or table-based method.
BIASSMOOTH	Biases edges using a table-based format.
curvilinear_bias	Enables curvilinear biasing in nmBIAS.
CURVILINEAR_SMOOTHING	Smooths curved fragment portions of curvilinear geometries.
displacement_limit_mode	Provides for tighter DISPLACEMENT limits to take precedence.
FRAGMENT_MOVE	Moves all fragments simultaneously while adhering to mask constraints.
curvilinear_bias	Smooths a subset of fragments towards a given reference.
movement_mode	Determines whether edge biasing is performed sequentially or in parallel between multiple BIASRULE or BIASSMOOTH commands.
NEWTAG locked	Returns a set of locked fragments.
NOTCHFILL	Fills notches in shapes of the designated layer.
PRESERVE_CENTERLINE	Biases qualifying shapes around their center line.
VIA_SHIFTING	Shifts via polygons on the biased layer by a specified distance away from the reference.
VIA_UPSIZE	Upsizes via polygons on the bias layer to a specified area.

BIASRULE

Calibre nmBIAS Tcl Scripting Commands

Biases edges using a rule- or table-based method.

Usage

BIASRULE
[pattern *mask_number*]
in_layer
[ref_layer]
[-enclosedlayer *enc_layer*]
[-tag *tagName*]
[-tag2 | -tags *tagName*]
[-tagw *tagName*]
-rule '**{**'
SPACE *range* [*metric*]
[WIDTH *range* [*metric*]]
[ENCLOSING *range* [*metric*]]
[LENGTH1 *range*]
{**MOVE *value*** |
GROW_UNTIL_SPACE *value* [max_disp *value*]
| **GROW_UNTIL_WIDTH *value*** [max_disp *value*]
| **GROW_UNTIL_ENCLOSING *value***
}
'**}**'
[-exclude_shielded]
[-interpolate *method grid*]
[-jog_ignore *value*]
[-lock]
[-lock_all]
[-projecting [*length*] [-ignore_singularity]]
[-projecting_space [*length*] [-ignore_singularity]]
[-two_pass_mode]
[-unmoved]

Alternatively, you can use the -table method of the BIASRULE command:

BIASRULE
[3D]
[pattern *mask_number*]
in_layer
[ref_layer]
[-overlapLayer *layer*]
[selector *constraint metric*]
[-tag *tagName*]
[-tag2 | -tags *tagName*]

[-tagw *tagName*]
-table [row *metric* col *metric*] ‘**{**’ ***bias_matrix*** ‘**}**’ [*options*]
[... -table [row *metric* col *metric*] ‘{’*bias_matrix* ‘}’ [*options*]]
[-exclude_shielded]
[-interpolate *method grid*]
[-jog_ignore *value*]
[-lock]
[-lock_all]
[-minimize_jogs]
[-projecting [*length*] [-ignore_singularity]]
[-projecting_space [*length*] [-ignore_singularity]]
[-colTag *tagname*]
[-outTag *tagname*]
[-rowTag *tagname*]
[-selectorTag *tagname*]
[-side_depth *value*]
[-side_measurement *value* bias]
[-unmoved]

Description

The BIASRULE command biases edge fragments. The command allows you to specify edges and their respective biasing values. The syntax for biasing uses either a method of rule statements or a table of values. For a complete syntactical example, refer to “Comprehensive BIASRULE SVRF Files” on page 130.

BIASRULE Modes

BIASRULE can be used in -rule or -table mode. Both modes apply corrections to individual target shape edges based on properties of those edges. BIASRULE supports large bias jog smoothing and fragment tagging to identify specific fragments to bias. Automated interpolation between space or width values is provided to make smoother transitions with fewer rules, and for -table mode only, 3D interpolation evaluates any three metrics simultaneously, two specified through the table, and one specified through the selector. To perform OPC on your layout using BIASRULE, you specify rules similar to OPCBIAS. To perform OPC using the table-based method, you specify any number of tables with dimensions as input.

Note

The BIASRULE command in either -rule or -table formats must not exceed 32767 characters in length. Exceeding this character length results in a compile-time error.

Arguments

- BIASRULE arguments

  The following BIASRULE arguments are grouped in this section and listed alphabetically within their respective groups.

- Layer and Tag Keywords
- Control and Constraint Keywords
- Biasing Methods

Examples

Example 1 — BIASRULE Command Organization

You place the BIASRULE command within an options block which is placed in a LITHO FILE statement for subsequent use by setlayer nmbias and LITHO NMBIAS, respectively.

```
LITHO FILE nmbias.setup [
   tilemicrons 40 adjust
   layer bias.in hidden

   options nmbias_options {
      version 1
      layer bias.in opc

      BIASRULE bias.in -tag all_frags \
         -rule {
            SPACE > 0.22 MOVE 0.025 OPPOSITE EXTENDED 0.075
            SPACE > 0.12 <= 0.22 MOVE 0.020 OPPOSITE EXTENDED 0.075
            ...
      }

      BIASRULE bias.in -tag all_frags \
         -table { \
               0.045 0.060 0.075 OPPOSITE OPPOSITE EXTENDED 0.020 :\
      0.050 0.003 0.005 0.007 \
      0.060 0.003 0.005 0.007 \
      ...
      }

         -table row length1 col width2 {
               0.040 0.065 0.80 OPPOSITE OPPOSITE EXTENDED 0.020 :\
      0.050 0.003 0.005 0.008 \
      0.060 0.003 0.005 0.008 \
      ...
      }
   }
   setlayer postbias  = nmbias bias.in MAP bias.in OPTIONS nmbias_options
]
biasedMX = LITHO NMBIAS FILE nmbias.setup MX MAP postbias
```

Example 2 — BIASRULE Metric, and Space and Width Relationships

Figure 3-2 demonstrates the available BIASRULE metrics.

Figure 3-2. BIASRULE Metrics Illustration

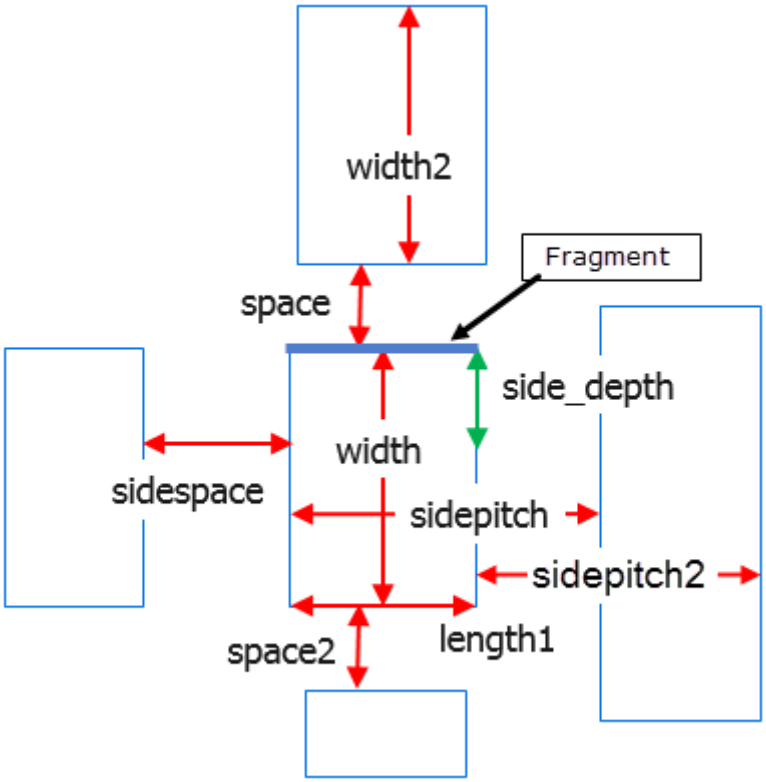


Figure 3-3 demonstrates the BIASRULE width and space relationships.

Figure 3-3. BIASRULE width and space Relationships

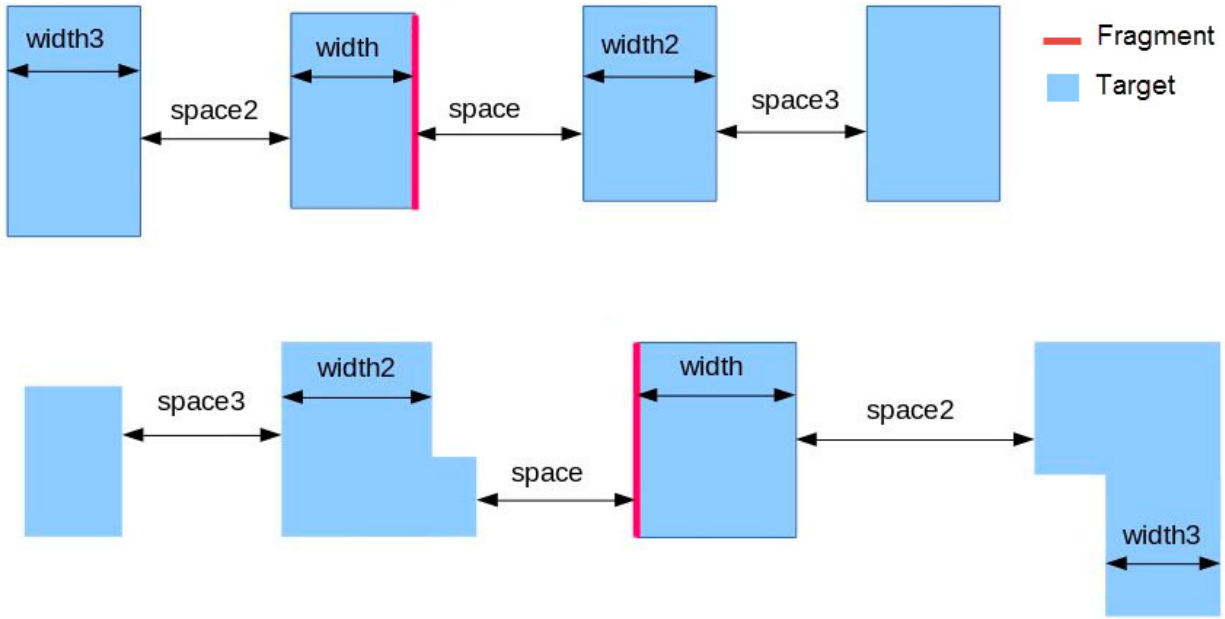
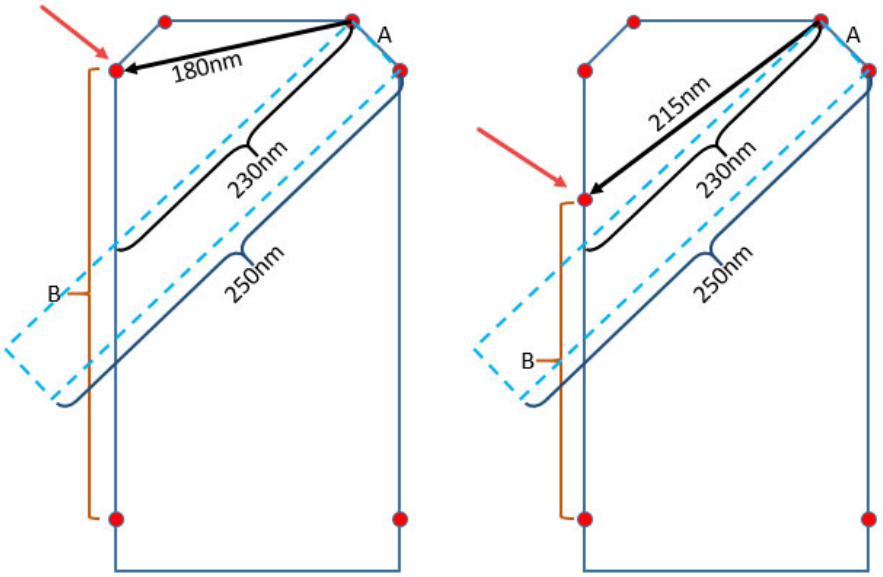


Figure 3-4. How BIASRULE Handles 45-Degree Biasing



The shortest distance between the two fragment points is 180 nm. Given the BIASRULE table, segment A is biased 1.0 nm.

The shortest distance between the two fragment points is 215 nm. Given the BIASRULE table, segment A is biased 3.0 nm.

```
BIASRULE target -table {
          0.060 0.100   :\
   0.170 0.001 0.001 \
   0.190 0.002 0.002 \
   0.210 0.003 0.003 \
   0.220 0.004 0.004 \
   0.250 0.005 0.005 \
}
```

Given the bias table shown and the two different fragment points shown at left:

1) BIASRULE first draws a bounding box perpendicular to fragment A which is equal to the largest width in the table, in this case, 250nm.

2) BIASRULE finds the nearest fragment, in this case fragment B.

3) BIASRULE finds the shortest distance between a fragment point on segment A and a fragment point on the opposite side of the shape.

Example 3 — 3D Interpolation and Selector Usage Within BIASRULE

```
BIASRULE 3D original  \
   -tag le_frags \
   -length1 0.1 opposite \
   -table row width col space { \
               0.05     0.07     0.09   : \
      0.05    0.001    0.002    0.003    \
      0.06    0.002    0.003    0.004    \
   }    \
   -length1 0.2 opposite \
   -table row width2 col space { \
               0.06     0.08     0.11   : \
      0.07    0.002    0.004    0.003    \
      0.08    0.003    0.005    0.004    \
   }
```

Example 4 — Non-3D Interpolation and Selector Usage Within BIASRULE

```
BIASRULE original  \
   -tag le_frags \
   -length1 > 0.004 opposite \
   -table row width col space { \
               0.05     0.07     0.09  : \
      0.05    0.001    0.002    0.003   \
      0.06    0.002    0.003    0.004   \
   } \
   -tag le_frags \
   -length1 > 0.2 opposite \
   -table row width2 col space { \
               0.06     0.08     0.11  : \
      0.07    0.002    0.004    0.003   \
      0.08    0.003    0.005    0.004   \
   }
```

BIASSMOOTH

Calibre nmBIAS Tcl Scripting Commands

Biases edges using a table-based format.

Usage

BIASSMOOTH *in_layer*
[-minfraglength *length*]
[-minjogheight *jogheight*]
[-smoothallcorners]
[-keepmaxbias]
[-keepaveragebias]

Arguments

- ***in_layer***

  A required argument specifying the name of the layer containing the fragments to smooth.

- -minFragLength *length*

  An optional argument and positive floating point number specifying the smallest fragment length that is smoothed with a neighbor on either side or a neighbor on both sides. By default, length is set to the minimum fragment length.

- -minjogheight *jogheight*

  An optional keyword and argument specifying the minimum jog height that is smoothed for unbiased fragments. Unbiased fragments with jogs less than the specified *jogheight* are not smoothed. Height is set to 0 by default. In this case, all jogs are smoothed.

- -smoothallcorners

  An optional keyword that smooths all jogs by ignoring the minjogheight value.

- -keepmaxbias

  An optional keyword that smooths the fragment to the maximum bias of its neighbors instead of the default smoothing behavior that smooths the fragment to the minimum bias. This applies when the fragment cannot be split, is collinear with both neighbors, and the difference between the bias of previous fragment and the bias of the next fragment is less than minjogheight.

- -keepaveragebias

  An optional keyword that smooths the fragment to the average bias of its neighbors instead of the default smoothing behavior that smooths the fragment to the minimum bias. This applies when the fragment cannot be split, is collinear with both neighbors, and the difference between the bias of previous fragment and the bias of the next fragment is less than minjogheight.

Description

BIASSMOOTH is a post-processing operation used to smooth those jogs, considered greater than maximum by the specified keywords, created by BIASRULE.

Examples

In this example, BIASSMOOTH is used to smooth jogs that are larger than 15nm.

```
BIASSMOOTH bias.in -minFragLength 0.060 -minJogHeight 0.015
```

curvilinear_bias

Calibre nmBIAS Tcl Scripting Commands

Enables curvilinear biasing in nmBIAS.

Usage

curvilinear_bias {**off** | **on**}

Arguments

- **off** | **on**

  One of two required arguments.

  - **off** — Disables curvilinear capability. This is the default.
  - **on** — Enables curvilinear capability. This setting requires that LITHO CL NMBIAS also be specified.

Description

Enables curvilinear biasing in nmBIAS. Requires that LITHO CL NMBIAS also be specified.

Note

An error results if curvilinear_bias is specified without LITHO CL NMBIAS.

Examples

```
curvilinear_bias on
...
BIASRULE L3 L1 -tag all_frags_L3 -rule {
  SPACE > 0.03 <= 0.04 WIDTH < 0.05 MOVE -0.01
  ...
}

LITHO CL NMBIAS ...
```

CURVILINEAR_SMOOTHING

Calibre nmBIAS Tcl Scripting Commands

Smooths curved fragment portions of curvilinear geometries.

Usage

CURVILINEAR_SMOOTHING -tag *tag_set*
[-dist *value*]
[-minlength *length*]
[-resolution avg | max | min]

Arguments

- **-tag *tag_set***

  A required argument specifying the name of the tagset to smooth.

- -dist *value*

  An optional argument specifying the distance of which to search for neighboring fragments, within the given tagset, that is used for biasing. The default value is 0.25 * *interaction distance*.

- -minlength *length*

  An optional argument specifying the minimum length of a fragment to break an extended fragment chain within the specified tagset. The default length is 2 * *fragment_min*. Specified values greater than or equal to the interaction distance result in an error.

- -resolution avg | max | min

  An optional argument specifying how to compute biasing of the fragments in the specified tagset based on linked fragments not belonging to the tagset.

  avg

  The average length of both linked fragments. This is the default.

  max

  The longest length of either linked fragment.

  min

  The shortest length of either linked fragment.

Description

Smooths bias fragments near tighter curves for curvilinear geometries. For each fragment (F) in the specified tagset, fragment links are traversed on either side of the fragment to find the first fragment (L1, L2) that does not belong to the tagset. The links are traversed up to a distance of 0.25 * interaction distance or the user specified distance from F. If L1 or L2 are not found, the first long neighbor (2 * fragment_min or user specified value) along the traversed links is isolated. The resolution setting determines how the bias of F is computed using L1 and L2.

Examples

```
CURVILINEAR_SMOOTHING -tag conv_conc_frags -resolution max
```

displacement_limit_mode

Calibre nmBIAS Tcl Scripting Commands

Tightens fragment bias displacement.

Usage

displacement_limit_mode last | strict

Arguments

- last | strict

  The strict option enables tighter DISPLACEMENT limits. The last option enables the limits established by the most recent biasing. The last option is default.

Description

Provides for tighter DISPLACEMENT limits to take precedence.

Examples

```
BIASRULE bias.in -tag tag1 ...
BIASRULE bias.in -tag tag2 ...
DISPLACEMENT tag3 fixedOffsetIncr <val>
displacement_limit_mode strict
...
```

FRAGMENT_MOVE

Calibre nmBIAS Tcl Scripting Commands

Moves fragments.

Usage

FRAGMENT_MOVE

Arguments

- -freeze_nonmoving

  Temporarily freeze all fragments with the desired displacement. This behavior is identical to DISPLACEMENT.

- -freeze_tag *tagname*

  Temporarily freeze all the fragments that belong to the required tag name.

Description

This command moves all fragments to their desired displacements simultaneously, while obeying mask constraints. This command only moves fragments whose desired displacement differs from the current actual displacement.

In other words, fragments that were moved using FRAGMENT_SET_DISPLACEMENT -force do not require a subsequent call to this command to move the fragments. There are two ways to set desired displacements:

- OPC_ITERATION
- FRAGMENT_SET_DISPLACEMENT (without the -force option)

The implication is that calling either of those commands would typically require a subsequent call to FRAGMENT_MOVE. The maximum movement per iteration is also enforced (the default is Nyquist/2), which can also result in a fragment moving less than expected.

Note that FRAGMENT_MOVE conforms to max_iter_movement when the command is used with OPC_ITERATION.

If two or more BIASRULE commands are specified sequentially, the first one has priority. However, specifying FRAGMENT_MOVE at the end of the second BIASRULE command turns on EDE optimization and the code behaves as if the two BIASRULE commands had the same priority. You need to manually freeze fragments that are not intended to move before executing FRAGMENT_MOVE for the results of the first BIASRULE command to remain unmoved.

A valid use of FRAGMENT_MOVE for such a case is demonstrated in the following example:

```
BIASRULE bias.in -tag tag1 ...
BIASRULE bias.in -tag tag2 ...
DISPLACEMENT tag3 fixedOffsetIncr <val>
TAG or tag1 tag2 -out tagBiased
FRAGMENT_MOVE freeze_tag tagBiased
```

Tcl Results

None

Examples

```
FRAGMENT_SET_DISPLACEMENT line_end 0.03
FRAGMENT_SET_DISPLACEMENT lea 0.02
FRAGMENT_SET_DISPLACEMENT convex_corner  0.015
FRAGMENT_SET_DISPLACEMENT concave_corner -0.015
# All goals set, now do the movement
FRAGMENT_MOVE
```

FRAGMENT_SMOOTHING

Calibre nmBIAS Tcl Scripting Commands

Smooths a subset of fragments towards a given reference.

Usage

FRAGMENT_SMOOTHING ***in_tag***
reference ***value***
maxstepsize_1 ***value***
maxstepsize_2 ***value***
-out ***value***
[limit *inner_smooth outer_smooth*]
[tagset_only]

Arguments

- ***in_tag***

  A required argument specifying the name of the layer containing the tagset to smooth.

- **reference** ***value***

  A required value (in microns) to indicate the direction of smoothing. Users could set the reference to a negative value to prevent bridging or a positive value to prevent pinching.

- **maxstepsize_1** ***value***

  A required value in microns, used for smoothing a fragment with a bias larger than the reference. The value must be a non-negative number.

- **maxstepsize_2** ***value***

  A required value in microns, used for smoothing a fragment with a bias smaller than the reference. The value must be a non-negative number.

- **-out** ***tagset***

  A required name specifying a tagset of input fragments that were smoothed by this operation.

- limit *inner_smooth outer_smooth*

  An optional keyword specifying the movement of fragments relative to movement performed by FRAGMENT_SMOOTHING. This method provides a greater level of fragment smoothing flexibility than using the DISPLACEMENT -limit keyword alone. See Figure 3-5 for more information.

  inner_smooth

  A negative value in user units providing movement inward relative to the target edge.

  outer_smooth

  A positive value in user units providing movement outward relative to the target edge.

- -tagset_only

  An optional model in which delta displacement is computed considering neighboring fragments that belong to the input tagset. By default, both collinear neighbors contribute to delta displacement.

Description

Smooths a subset of fragments towards a given reference. Only fragments with no corners or fragments with one corner are eligible for smoothing. FRAGMENT_SMOOTHING can be iteratively called to ensure there are no jogs having a height greater than maxstepsize on the output.

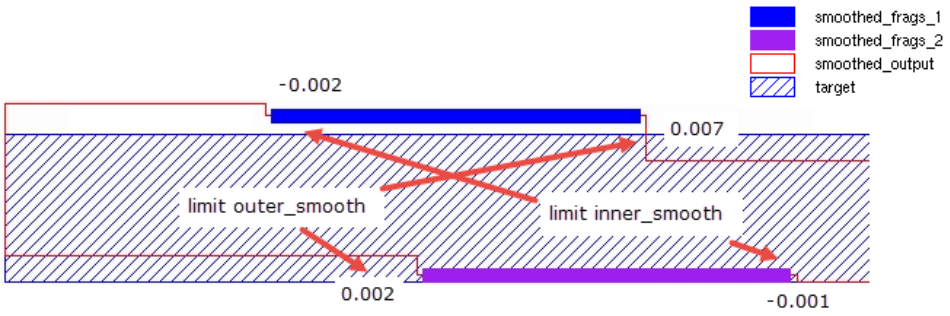
Examples

These examples demonstrate FRAGMENT_SMOOTHING.

```
FRAGMENT_SMOOTHING all_frags ref 0.005
   maxstepsize_1 0.007 maxstepsize_2 0.007 -out smoothed_frags

// Iterative calls:
FRAGMENT_SMOOTHING all_frags ref -0.005
   maxstepsize_1 0.007 maxstepsize_2 0.007 -out smoothed_frags

FRAGMENT_SMOOTHING all_frags ref -0.005
   maxstepsize_1 0.007 maxstepsize_2 0.007 -out smoothed_frags2

FRAGMENT_SMOOTHING all_frags ref -0.005
   maxstepsize_1 0.007 maxstepsize_2 0.007 -out smoothed_frags3
```

This example shows FRAGMENT_SMOOTHING limit values (keyword shown in red). The inner_smooth values are always negative and outer_smooth values are always positive.

Figure 3-5. FRAGMENT_SMOOTHING limit Values




```
NEWTAG fragment all_frags len < 0.062 mcorner1 concave mcorner2 \
   no_corner -out tg1
NEWTAG fragment all_frags len < 0.062 mcorner1 convex mcorner2 \
   no_corner -out tg2
FRAGMENT_SMOOTHING tg1 ref 0.005 maxstepsize_1 0.000 \
   maxstepsize_2 0.000 limit -0.002 0.007 -out smoothed_frags_1
FRAGMENT_SMOOTHING tg2 ref -0.005 maxstepsize_1 0.000 \
   maxstepsize_2 0.000 limit -0.001 0.002 -out smoothed_frags_2
```

movement_mode

Calibre nmBIAS Tcl Scripting Commands

Determines whether edge biasing is performed sequentially or in parallel between multiple BIASRULE or BIASSMOOTH commands.

Usage

movement_mode [incremental | delayed]

Arguments

- incremental

  An optional argument specifying that original data is input to the first BIASRULE or BIASSMOOTH command, and after processing, sequentially passed into the next BIASRULE or BIASSMOOTH command, and so on. This is the default mode.

- delayed

  An optional argument specifying that original data is input to all BIASRULE or BIASSMOOTH commands in parallel.

Description

The movement_mode command determines whether edge biasing is performed sequentially or in parallel between multiple BIASRULE or BIASSMOOTH commands. The incremental mode takes the biased edge or smoothed output and hands it sequentially to the next BIASRULE or BIASSMOOTH command, respectively. The delayed mode passes all original input edges for processing to all BIASRULE or BIASSMOOTH commands in parallel.

Note

The movement_mode command must be specified before any tag-scripting commands, such as NEWTAG.

Examples

Example 1

In this example, movement_mode is used to bias edges using input data passed to all three BIASRULE commands at one time.

```
movement_mode delayed
...
BIASRULE L2 -tag enclosed_L2 -rule {
   SPACE > 0 WIDTH >= 0.040 MOVE 0.007
}
BIASRULE L1 -enclosedLayer L2 -rule {
   SPACE > 0 ENCLOSING > 0.025 <= 0.040 MOVE 0.005
}
BIASRULE L3 L1 -tag all_frags_L3 -rule {
   SPACE > 0.03 <= 0.04 WIDTH < 0.05 MOVE -0.01
}
```

NEWTAG locked

Layer and Tag Keywords

Returns a set of locked fragments.

Usage

newtag locked *tag_in* -out *tag_out*

Arguments

- *tag_in*

  An argument specifying the name of the input tag.

- *tag_out*

  An argument specifying the name of the locked output tag.

Description

An optional keyword that returns a set of locked fragments. You can lock fragments with the BIASRULE command to avoid fragment alteration by subsequent BIASRULE commands.

Examples

Example 1 — newtag locked Command in Context with BIASRULE -lock Keywords.

```
NEWTAG fragment target -out all_frags

BIASRULE ...
   -tag t1 -table {...} -lock -lock_all
   -tag t2 -table {...}
   -tag t3 -table {...} -lock -lock_all
   ...

NEWTAG locked all_frags -out locked
```

NOTCHFILL

Calibre nmBIAS Tcl Scripting Commands

Fills notches in shapes of the designated layer.

Usage

NOTCHFILL ***layer***
[-maxNotchDepth *depth*]
[-maxNotchLength *length*]

Arguments

- ***layer***

  Required argument that specifies the layer on which notches are to be filled.

- -maxNotchDepth *depth*

  Optional argument that specifies the maximum depth of a notch. Only when the fragment depth (the length of both adjacent notch fragments) is less than or equal to the specified value is the notch filled. This option defaults to the value of fragment_min/4.

- -maxNotchLength *length*

  Optional argument that specifies the maximum length of a notch. Only when the fragment length is less than or equal to the specified value is the notch filled. This option defaults to the fragment_min value.

Description

This command fills notches in shapes of the designated layer in the design. Notches are detected using the current position of relevant fragments. A fragment is classified as a notch fragment if these criteria are met:

- Manhattan fragments are longer than the minimum notch length (defaults to the fragment_min value).
- Adjacent manhattan fragments are longer than the minimum notch depth (defaults to fragment_min value divided by 4).
- Concave corners are present on both ends of the fragments using the current positions of relevant fragments.

Examples

```
NOTCHFILL met1

NOTCHFILL met1 -maxNotchLength 0.020

NOTCHFILL met1 -maxNotchLength 0.020 -maxNotchDepth 0.045
```

PRESERVE_CENTERLINE

Calibre nmBIAS Tcl Scripting Commands

Biases qualifying shapes around their center line.

Usage

PRESERVE_CENTERLINE *layer* [-tag *tagname*] -maxPolyWidth *width* [min | max | avg]

Arguments

- *layer*

  A required name of a layer to preserve polygon centerlines while biasing.

- -tag *tagname*

  An optional keyword that specifies a tag set of edges to be biased.

- -maxPolyWidth *width*

  A required keyword specifying a maximum applicable polygon width. Only those polygons of widths smaller than the specified value are candidates to be biased symmetrically around their center line. All other polygons are biased without regard to their center line.

- min | max | avg

  An optional keyword determining whether facing fragments are biased to the minimum, maximum or average bias of the fragments. The default is avg.

Description

Biases fragments of polygons with qualifying widths symmetrically around their centerlines.

Examples

```
BIASRULE bias.in ...

PRESERVE_CENTERLINE bias.in -maxPolyWidth 0.075 max
```

VIA_SHIFTING

Calibre nmBIAS Tcl Scripting Commands

Shifts via polygons on the biased layer by a specified distance away from the reference layer.

Usage

VIA_SHIFTING ***bias_layer***
ref_layer [*ref_layer2*]
-minEnclosure ***enclosure***
[-bottom *ref_layer*]
[-maxWidth *width*]
[-maxShift *value*]
[-minEnclosure_top *enclosure*]
[-minEnclosure_bot *enclosure*]
[-minSpace *space*]
[-minSpace_bot *space*]
[-minSpace_top *space*]
[-runlength {-enclosure *value* | -enclosing *value*}]
[-shift *proplayer propname_x propname_y resolution*]
[-tag *name*]
[-top *ref_layer*]
[-outLayer currentCenter]
[-outLayer newCenter]

Arguments

- ***bias_layer***

  A required argument that specifies the OPC or correction layer whose polygons are to be shifted or centered.

- ***ref_layer*** [*ref_layer2*]

  A required reference layer (or optionally two layers) specified for shifting bias_layer polygons.

  If you specify one reference layer, it is used to shift vertical and horizontal layer fragments on the bias_layer.

  If you specify two reference layers, the first layer is used to shift the vertical fragments and the second layer is used to shift the horizontal layers.

- **-minEnclosure** ***enclosure***

  A required argument pair that specifies the minimum desired enclosure distance between bias_layer and ref_layer.

  If there is enough room to shift bias_layer fragments an enclosure distance away from ref_layer, then it is applied. Otherwise, the bias_layer fragments are centered within the given space.

- -bottom *ref_layer*

  An optional argument pair that specifies the bottom metal reference layer. Either a generic reference layer, or top and bottom reference layers may be specified.

- -maxShift *width*

  An optional argument pair that specifies the maximum shift distance of bias_layer fragments that need to be shifted. The default value is limited by the geometries of the bias_layer and ref_layer.

- -maxWidth *width*

  An optional argument pair that specifies the maximum width of bias_layer fragments that need to be shifted or centered. The default value is two times the enclosure value (2 * enclosure).

- -minEnclosure_bot *enclosure*

  An optional argument pair that specifies the minimum desired enclosure distance between bias_layer and bottom metal ref_layer.

  If there is enough room to shift bias_layer fragments an enclosure distance away from the bottom ref_layer, then it is applied. Otherwise, the bias_layer fragments are centered within the given space.

- -minEnclosure_top *enclosure*

  An optional argument pair that specifies the minimum desired enclosure distance between bias_layer and the top metal ref_layer.

  If there is enough room to shift bias_layer fragments an enclosure distance away from the top ref_layer, then it is applied. Otherwise, the bias_layer fragments are centered within the given space.

- -minSpace *spacing*

  Optionally specifies the minimum space between the bias layer and generic reference layer.

- -minSpace_bot *spacing*

  Optionally specifies the minimum space required between the bias layer and bottom metal reference layer. The bottom reference layer should already be defined using -bottom.

- -minSpace_top *spacing*

  Optionally specifies the minimum space required between the bias layer and top metal reference layer. The top reference layer should already be defined using -top.

- -runlength *length* {-enclosure *value* | -enclosing *value*}

  Optionally specifies the selection criteria for vias that must be shifted to comply with MRC rules. If the runlength is greater than or equal to the specified length and where enclosure measurements (metal enclosing via) are found to be less than its specified value, or where an enclosing measurement (vias partially outside metal) are found to be less than its specified value, the via is selected to be shifted (provided the vias belong to the given tag set). Two

-runlength keywords with one or both of -enclosure and -enclosing may be specified per VIA SHIFTING command.

- -shift *proplayer propname_x propname_y resolution*

  An optional keyword set that specifies the name of a layer containing properties. All terms within the set must be specified.

  - *proplayer*

    The name of the layer hosting the properties.

  - *propname_x, propname_y*

    The property names that define the x- and y-direction shifts (in microns), respectively. If multiple markers (or properties) interact with a given via, then the minimum property value is used by default.

  - *resolution*

    - average

      The average value of all properties on the layer is selected.

    - max

      Selects the maximum property value.

    - min

      Selects the minimum property value. This is the default.

  Note

  The maximum movement of any via fragment or edge is limited to 0.05 microns in either direction. To modify the maximum movement, specify a custom value using the max_opc_move keyword within the setup file.

- -tag *name*

  An optional argument pair that specifies a tag name representing one or more via edges identifying via shapes to be processed.

- -top *ref_layer*

  An optional argument pair that specifies the top metal reference layer. Either a generic reference layer, or top and bottom reference layers may be specified.

- -outLayer currentCenter

  An optional argument pair that specifies the output layer for outputting the current centers of polygons on bias_layer.

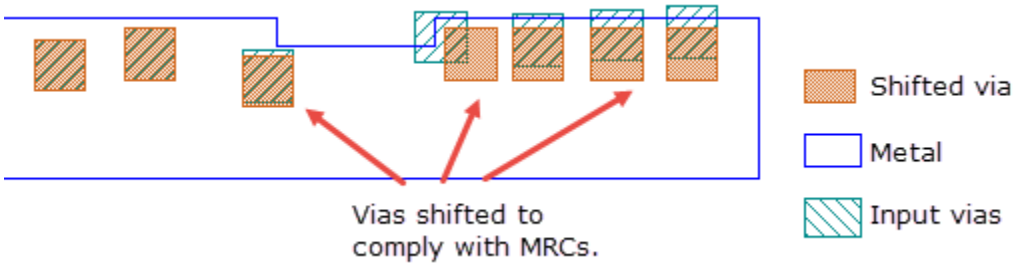
- -outLayer newCenter

  An optional argument pair that specifies the output layer for outputting the new centers of polygons on bias_layer.

Description

Shifts via polygons on the biased layer by a specified distance away from the reference layer. If the specified enclosure distance is achieved, then shifting is applied to maintain that distance. Otherwise polygons on bias_layer are centered with respect to ref_layer (or both layers, if specified). The bias_layer polygons must be 4-point shapes with Manhattan edges. Shifting or centering maintains the aspect ratio of bias_layer polygons. Figure 3-6 depicts typical VIA_SHIFTING results.

Figure 3-6. VIA_SHIFTING Results



Examples

Example 1

An implementation of VIA_SHIFTING for three layers is shown.

```
VIA_SHIFTING bias.in ref.in -minEnclosure 0.020

VIA_SHIFTING bias.in ref.in \
   -minEnclosure 0.020 \
   -outLayer currentCenter -outLayer newCenter

VIA_SHIFTING bias.in ref1.in ref2.in \
   -minEnclosure 0.020 -maxWidth 0.045 \
   -outLayer currentCenter -outLayer newCenter

VIA_SHIFTING via_in M1
   -runlength 0.010 -enclosure 0.008 \
   -runlength 0.010 -enclosure 0.008 \
   -minEnclosure 0.008 \
   -outLayer origCenter -outLayer newCenter
```

Example 2

Use of the -shift keyword with VIA_SHIFTING is shown in an abbreviated context of the setup file. The -shift keyword and associated arguments are shown in red.

```
tmp_marker = (GROW marker LEFT BY 0.009) NOT marker
final_marker = DFM PROPERTY tmp_marker [x_shift = PERIM(tmp_marker)/3]
   [y_shift = PERIM(tmp_marker)/4]
all_frags = LITHO NMBIAS FILE tdopc.setup VIA metal final_marker
   MAP all_frags
currentCenter = LITHO NMBIAS FILE tdopc.setup VIA metal final_marker
   MAP currentCenter
newCenter = LITHO NMBIAS FILE tdopc.setup VIA metal final_marker
   MAP newCenter


LITHO FILE tdopc.setup [/*
   tilemicrons 40 adjust
   layer bias.in hidden
   layer final_marker hidden
   options first.pass {
      version 1
      layer bias.in opc
      layer final_marker hidden
      fragment_min 0.030
      fragment_max 1.500
      fragment_inter off
      read_layer_properties final_marker x_shift y_shift
      NEWTAG all bias.in -out all_frags
      VIA_SHIFTING bias.in -shift final_marker x_shift y_shift average
         -outLayer currentCenter -outLayer newCenter
```

VIA_UPSIZE

Calibre nmBIAS Tcl Scripting Commands

Upsizes via polygons on the bias layer to a specified area.

Usage

VIA_UPSIZE *bias_layer ref_layer* [*ref_layer*] **-minEnclosure *enclosure*** [-minSpace *space*] **-toArea *area* -maxAspect *aspect_ratio***

Arguments

- ***bias_layer***

  Required argument that specifies the OPC or correction layer whose polygons are to be upsized.

- ***ref_layer*** [*ref_layer*]

  Required reference layer (or optionally two layers) specified for shifting ***bias_layer*** polygons.

  If you specify one reference layer, it is used to shift vertical and horizontal layer fragments on the ***bias_layer***.

  If you specify two reference layers, the first layer is used to shift the vertical fragments and the second layer is used to shift the horizontal layers.

- **-minEnclosure *enclosure***

  Required argument pair that specifies the minimum desired enclosure distance between ***bias_layer*** and *ref_layer*.

  If there is enough room to shift ***bias_layer*** fragments an ***enclosure*** distance away from ***ref_layer***, then it is applied.

- -minSpace *space*

  Optional argument pair that specifies the minimum space between ***bias_layer*** polygons.

- **-toArea *area***

  Required argument pair that specifies the desirable minimum area of ***bias_layer*** polygons.

- **-maxAspect *aspect_ratio***

  Required argument pair that specifies the maximum value for the aspect ratio of ***bias_layer*** polygons.

Description

Upsizes the polygons on ***bias_layer*** to a specified area while maintaining minimum space, enclosure, and aspect ratio constraints.

If the specified enclosure and minimum space distances are achievable, then upsizing is applied in order to maintain these distances and the maximum aspect ratio. The ***bias_layer*** polygons which are not Manhattan rectangles are ignored (not processed).

Examples

```
VIA_UPSIZE bias.in ref.in -minEnclosure 0.020 -toArea 0.0006 \
      -maxAspect 1.4
VIA_UPSIZE bias.in ref1.in ref2.in -minEnclosure 0.020 \
      -minSpace 0.015 -minEnclosure 0.020 -toArea 0.0006 -maxAspect 1.4
```

Layer and Tag Keywords

BIASRULE requires input layers and permits tagging edges of input layers.

Table 3-2. Layer and Tag Keywords

Keyword	Description
pattern	Declares the mask number to bias.
in_layer	Declares the name of an input layer to bias.
ref_layer	Declares the name of a reference layer used to perform biasing.
-enclosedlayer	Declares the name of an enclosure layer used to qualify biasing.
-overlapLayer	Declares the name of an overlap layer used to qualify biasing.
-colTag	Declares an annotated tagset containing the classification values for the col metric.
-outTag	Declares the output tagset with annotated bias values.
-rowTag	Declares an annotated tagset containing the classification values for the row metric.
-selectorTag	Declares the selector tagset.
-tag	Declares a name for a group of edge fragments.
-tag2 | -tags	Declares a name for a group of space-based edge fragments.
-tagw	Declares a name for a group of width-based edge fragments.

For organizational keyword information, refer to “BIASRULE” on page 50.

pattern

BIASRULE keyword: Layer and Tag Keywords

Declares the mask number to bias.

Usage

pattern *mask_number*

Arguments

- *mask_number*

  An argument specifying the number of the mask to process.

Description

An optional keyword and argument specifying the numbers of masks on which to perform simultaneous biasing for multiple layers. The pattern number must start with 0 and be specified consecutively. Use this option only for multi-patterning applications.

in_layer

BIASRULE layer: Layer and Tag Keywords

Declares the name of an input layer to bias.

Usage

BIASRULE *in_layer* {***rules***}

Arguments

None

Description

A required argument specifying the name of the layer to bias. The layer must be an OPC or correction layer. There is no default layer name.

Note

Visible layers types must never be input to Calibre nmBIAS. Visible layer type interaction with other layer types results in false edges and biasing failure. As of Calibre release 2016.3, all visible layer types are converted to hidden layer types.

ref_layer

BIASRULE layer: Layer and Tag Keywords

Declares the name of a reference layer used to perform biasing.

Usage

BIASRULE in_layer [*ref_layer*] {***rules***}

Arguments

None

Description

An optional correction layer (psm or ddl type) or a visible layer (sraf type) containing shapes that BIASRULE uses to measure spacing. At most, two reference layers can be specified. This layer is mutually exclusive with -tag2 | -tags.

When you specify a ref_layer, spacing is measured between the edge to be corrected and polygons on both the in_layer and on ref_layer. When you do not specify a ref_layer, spacing is measured between the edge to be corrected and other polygons on the in_layer. The ref_layer must not share any edges with the in_layer.

Note

Visible layers types must never be input to Calibre nmBIAS. Visible layer type interaction with other layer types results in false edges and biasing failure. As of Calibre release 2016.3, all visible layer types are converted to hidden layer types.

-enclosedlayer

BIASRULE keyword: Layer and Tag Keywords

Declares the name of an enclosure layer used to qualify biasing.

Usage

-enclosedlayer *enc_layer*

Arguments

- *enc_layer*

  An argument specifying the layer name that is used only in enclosing checks.

Description

An optional keyword and layer name specifying the layer name that is used only in enclosing checks. The layer under bias must enclose this layer and enclosing rules must be specified. Use this option only for BIASRULE -rule.

-overlapLayer

BIASRULE keyword: Layer and Tag Keywords

Declares the name of an overlap layer used to qualify biasing.

Usage

-overlapLayer *layer*

Arguments

- *layer*

  An argument specifying the layer name.

Description

An optional keyword and layer name specifying the overlap layer that is required only when overlap or enclosure metrics are specified. You use this to perform two-layer width checks (overlap) and enclosing checks (enclosure). You use this option only for BIASRULE -table.

-colTag

BIASRULE keyword: Layer and Tag Keywords

Declares an annotated tagset containing the classification values for the col metric.

Usage

-colTag *tagname*

Arguments

- *tagname*

  An argument specifying the name of the tag.

Description

An optional keyword and argument specifying an annotated tagset containing the classification values for the col metric. If a fragment is not present in the annotated tagset, it is considered to be isolated and therefore, the largest col rule applies to it. The NMBIAS_MEASURE_TAG command must be used to generate these annotated tagsets. For each rule table, two annotated tagsets must be provided.

-outTag

BIASRULE keyword: Layer and Tag Keywords

Declares the output tagset with annotated bias values.

Usage

-outTag *tagname*

Arguments

- *tagname*

  An argument specifying the name of the tag.

Description

An optional keyword and argument specifying the output tagset with incremental, annotated bias values. Only one output tagset per pattern may be specified. If -outTag is specified, BIASRULE does not perform biasing but only populates the -outTag tagset and attaches bias properties to each fragment from the input tagset or bias layer.

-rowTag

BIASRULE keyword: Layer and Tag Keywords

Declares an annotated tagset containing the classification values for the row metric.

Usage

-rowTag *tagname*

Arguments

- *tagname*

  An argument specifying the name of the tag.

Description

An optional keyword and argument specifying an annotated tagset containing the classification values for the row metric. If a fragment is not present in the annotated tagset, it is considered to be isolated and therefore, the largest row rule applies to the fragment. The NMBIAS_MEASURE_TAG command must be used to generate these annotated tagsets. For each rule table, two annotated tagsets must be specified.

-selectorTag

BIASRULE keyword: Layer and Tag Keywords

Declares the selector tagset.

Usage

-selectorTag *tagname*

Arguments

- *tagname*

  An argument specifying the name of the tag.

Description

An optional keyword and argument specifying the selector tagset. In 2D mode, the selector constraint can simply be translated to an input tagset (specifying the -selectorTag explicitly is not needed). However, in 3D mode, it is required and the same tagset must be specified for all the tables specified within a BIASRULE.

-tag

BIASRULE keyword: Layer and Tag Keywords

Declares a name for a group of edge fragments.

Usage

-tag *tagname*

Arguments

- *tagname*

  An argument specifying the name of the tag.

Description

An optional keyword and argument specifying a group of fragments to be biased.

-tag2 | -tags

BIASRULE keyword: Layer and Tag Keywords

Declares a name for a group of space-based edge fragments.

Usage

-tag2 *tagname*

-tags *tagname*

Arguments

- *tagname*

  An argument specifying the name of the tag.

Description

An optional keyword and argument specifying a group of fragments used to determine the space classification of fragments on the in_layer. If a fragment is unclassified, it follows the rule specifying the largest spacing value. These tags are mutually exclusive with ref_layer.

When using -tag2 and -tags, space is only measured from the in_layer to -tag2 or -tags fragments. Any fragments on the in_layer not included in -tag2 or -tags are not used for space measurement when -tag2 or -tags is defined.

-tagw

BIASRULE keyword: Layer and Tag Keywords

Declares a name for a group of width-based edge fragments.

Usage

-tagw *tagname*

Arguments

- *tagname*

  An argument specifying the name of the tag.

Description

An optional keyword and argument specifying a group of fragments used to determine the width classification of fragments on the in_layer. If a fragment is unclassified, it follows the rule specifying the largest width. The -tagw keyword is used only with -rule method.

When using -tagw, width is only measured from the in_layer to the -tagw fragments. Any fragments on the in_layer not included in -tagw are not used for width measurement when -tagw is defined.

Control and Constraint Keywords

BIASRULE uses keywords to control edge manipulation during biasing.

Table 3-3. BIASRULE Control and Constraint Keywords

Keyword	Description
3D	Enables 3D interpolation for -table mode.
-exclude_shielded	Enables shielding to assist with space or width measurements.
-interpolate	Enables interpolate mode.
-jog_ignore	Ignores qualifying jog lengths during contiguous edge length summation.
-length1	Classifies edges by length for subsequent biasing application. Used only with -table method.
-lock	Locks fragments from subsequent table biasing.
-lock_all	Locks fragments globally from subsequent table biasing.
-minimize_jogs	Removes 1 DBU jogs from BIASRULE output.
-projecting	Overrides metrics to bias all classified width fragments in projection.
-projecting_space	Considers length of space classifications to bias all classified width fragments in projection.
selector	Selects a table of constraints within a BIASRULE table.
-side_depth	Determines distance perpendicular to fragments when computing sidespace or sidepitch metrics.
-side_measurement	Determines the classification parameter when computing sidespace or sidepitch metrics.
-two_pass_mode	Forces GROW_UNTIL_SPACE or GROW_UNTIL_WIDTH irrespective of edge blockage.
-unmoved	Forces measurement from unmoved fragments.

For organizational keyword information, refer to “BIASRULE” on page 50.

3D

BIASRULE keyword: Control and Constraint Keywords

Enables 3D interpolation for -table mode.

Usage

3D

Arguments

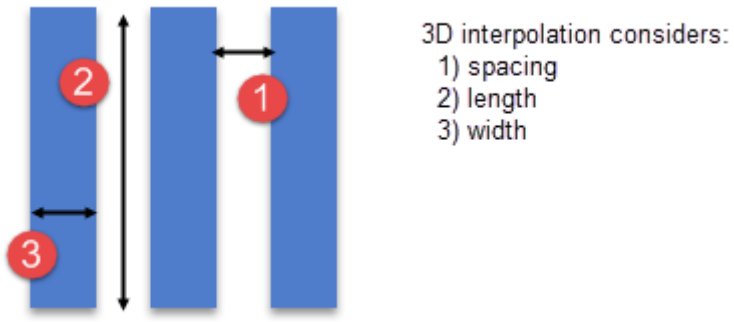
None

Description

An optional keyword specifying 3D interpolation method. 3D compound interpolation handles spacing, width, and lengths simultaneously across dense shapes, shown in Figure 3-7. Although additional -interpolation options are offered for 3D, use of the -interpolation keyword becomes optional, because 3D implicitly performs compound interpolation. 3D interpolation evaluates any three metrics simultaneously, two specified through the table, and one specified through the selector.

For more information regarding 3D interpolation options, see "-interpolate" on page 96. Although 3D increases accuracy across dense shapes, this capability is compute intensive and may result in longer run times.

Figure 3-7. 3D Interpolation Shape Consideration



In Figure 3-7, 3D is shown using the three metrics diagrammed.

Examples

Example 1 - 3D Usage with Three Tables

```
BIASRULE 3D bias.in reference.in                                \
                          -tag tagA                             \
                          -length1 0.0 opposite                 \
                          -table row space col width {          \
          0.040      0.060   OPPOSITE OPPOSITE :                \
 0.040    0.000      0.001                                      \
 0.060    0.001      0.002                                      \
 0.080    0.002      0.003                                      \
                          }                                     \
                          -length1 0.200 opposite               \
                          -table row space col width {          \
          0.040      0.060   OPPOSITE OPPOSITE :                \
 0.040    0.001      0.002                                      \
 0.060    0.002      0.003                                      \
 0.080    0.003      0.004                                      \
                          }                                     \
                          -length1 0.300 opposite               \
                          -table row space col width {          \
          0.040      0.060   OPPOSITE OPPOSITE :                \
 0.040    0.002      0.003                                      \
 0.060    0.003      0.004                                      \
 0.080    0.004      0.005                                      \
                          }
```

Example 2 - Two Table Usage With -interpolate

```
 BIASRULE 3D bias.in reference.in                          \
                          -tag tagA                        \
                          -length1 0.400 opposite          \
                          -table row space col width {     \
          0.040      0.060   OPPOSITE OPPOSITE :           \
 0.040    0.001      0.002                                 \
 0.060    0.002      0.003                                 \
 0.080    0.003      0.004                                 \
                          }                                \
                          -length1 0.600 opposite          \
                          -table row space col width {     \
          0.040      0.060   OPPOSITE OPPOSITE :           \
 0.040    0.002      0.003                                 \
 0.060    0.003      0.004                                 \
 0.080    0.004      0.005                                 \
                          }                                \
                          -interpolate implicit
```

-exclude_shielded

BIASRULE keyword: Control and Constraint Keywords

Enables shielding to assist with space or width measurements.

Usage

-exclude_shielded

Arguments

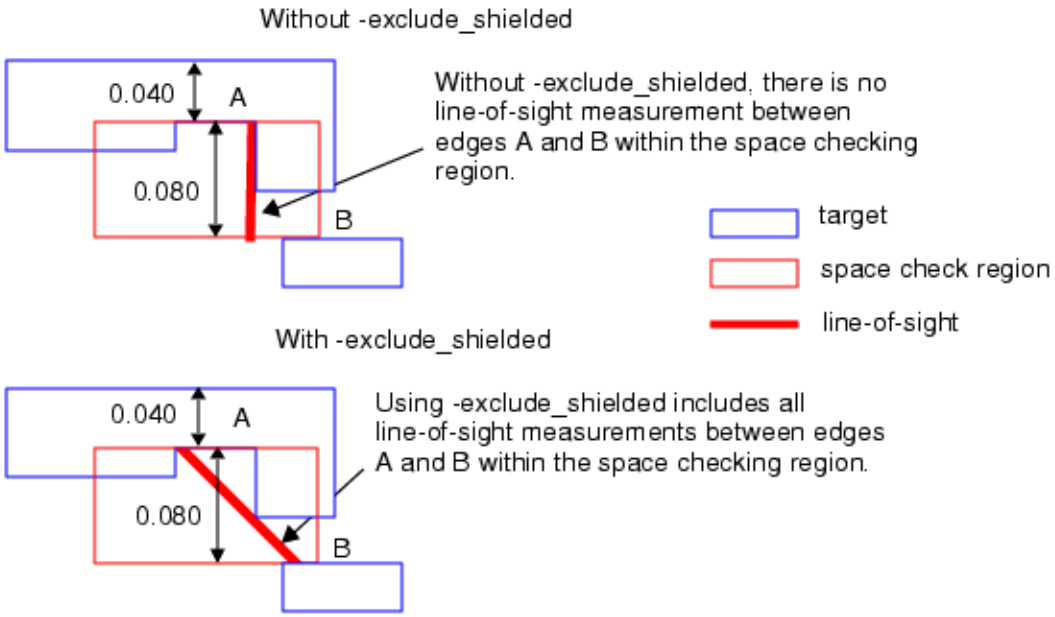
None

Description

An optional keyword that considers projecting fragments for WIDTH classification. The -exclude_shielded keyword may help exclude internal or external edges from measurement that would otherwise be included for biasing. BIASRULE shielding does not require the shielding edge to completely block the measurement region between edges. The -exclude_shielded keyword may consume compute resources. To reduce runtime, avoid using the EUCLIDEAN modifier with -exclude_shielded.

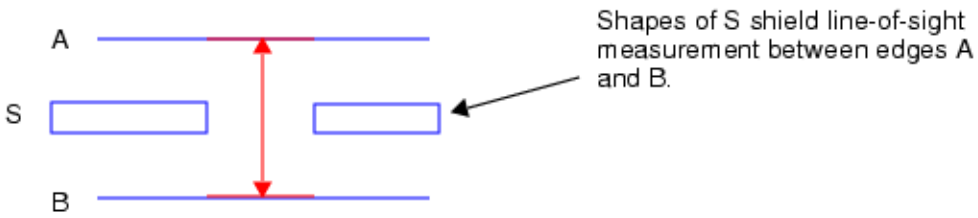
For a complete description of shielding see the “Edge shielding” section of the TDDRC command in the *Standard Verification Rule Format (SVRF) Manual.*

Figure 3-8. BIASRULE Shielding Applied to Space



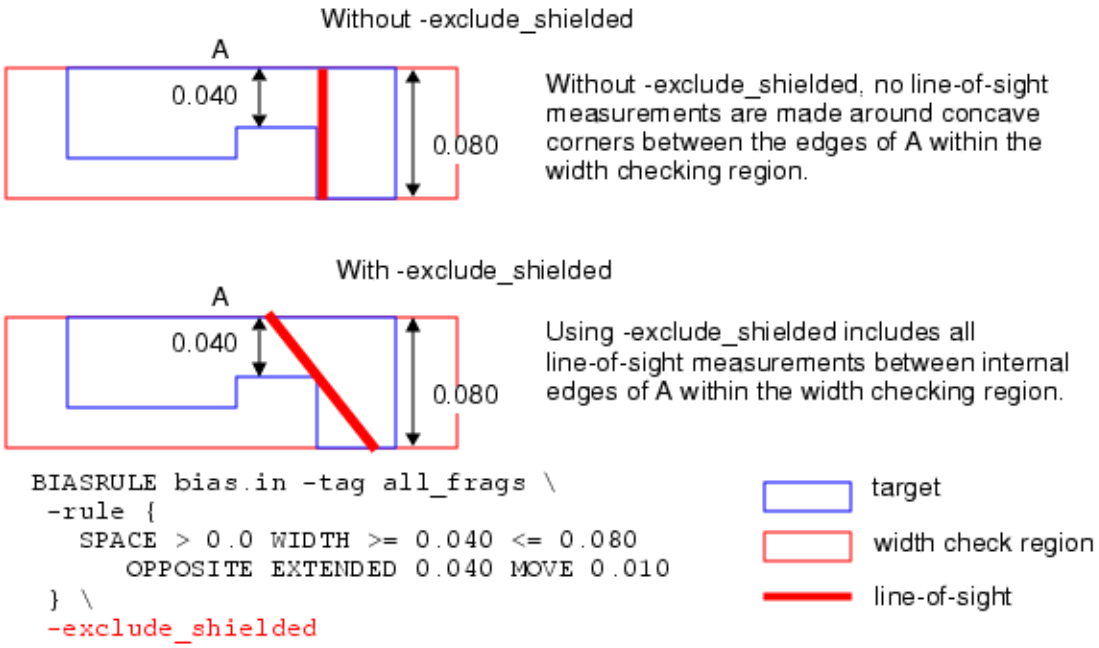

```
BIASRULE bias.in -tag all_frags \
 -rule {
   SPACE <= 0.080 OPPOSITE EXTENDED 0.050
      WIDTH > 0.0 <= 0.040 MOVE 0.010
 } \
 -exclude_shielded
```

Figure 3-9. Inter-layer BIASRULE Shielding Applied to Space



Only portions of edges A and B having line-of-sight exposure (those not blocked by shapes of S) are considered during measurement.

Figure 3-10. BIASRULE Shielding Applied to Width



-interpolate

BIASRULE keyword: Control and Constraint Keywords

Enables interpolate mode.

Usage

-interpolate *method grid*

Arguments

- *method*

  One of the following interpolation keywords may be specified:

  - NO — No interpolation. This is the default.
  - SPACE — Linear interpolation based on neighboring space rules.
  - WIDTH — Linear interpolation based on neighboring width rules.
  - SPACE_WIDTH | WIDTH_SPACE — Bilinear interpolation between neighboring space and width rules.
  - ROW — Linear interpolation based on the row metric.
  - COL — Linear interpolation based on the col metric.
  - ROW_COL | COL_ROW — Bilinear interpolation between neighboring row and column rules.
  - implicit — Provides interpolation between 0 to the lowest table value.
  - all | table_only {row | col | row_col} | selector_only — 3D interpolation usage only. In 3D interpolation mode, only one interpolate keyword can be specified for a single pattern or BIASRULE command:

    - all — Applies compound 3D interpolation. This is the default for 3D interpolation.
    - table_only {row | col | row_col} — Applies interpolation only to values within the table, either for each row, each column or both. One argument must be specified.
    - selector_only — Applies interpolation to the selector metric only. For more information, see “selector” on page 107.

- *grid*

  A specified grid to apply the interpolated value against. By default, -interpolate chooses the closest mid-point between two specified values in the bias table based on 1DBU.

Description

An optional keyword and argument specifying the interpolation method. Interpolation can reduce the number of rules needed and the runtime compared to OPCBIAS. Up to 15 interpolate values can be specified, one per table.

Examples

Example 1 — Default Usage

```
-interpolate no
```

Example 2 — Standard -interpolate Usage

```
BIASRULE bias.in \
   -overlapLayer bias.in \
   -tag full_sizing \
   -table row space col enclosure { \
            0.000 0.060 0.070 0.120 OPPOSITE EXTENDED 0.080 \
                                    OPPOSITE EXTENDED 0.080 :\
      0.000 0.000 0.005 0.007 0.007 \
      0.060 0.000 0.005 0.007 0.007 \
      0.061 0.000 0.004 0.006 0.006 \
      0.062 0.000 0.003 0.005 0.002 \
      0.063 0.000 0.002 0.004 0.002 \
      0.064 0.000 0.001 0.003 0.002 \
      0.065 0.000 0.000 0.002 0.004 \
      0.070 0.000 0.000 0.002 0.006 \
   } \
   -interpolate space 0.05 -projecting -exclude_shielded

BIASSMOOTH bias.in -minFragLength 0.06 -minJogHeight 0.015
```

Example 3 — 3D -interpolate Usages

```
-interpolate all

-interpolate table_only row_col

-interpolate selector_only
```

-jog_ignore

BIASRULE keyword: Control and Constraint Keywords

Ignores qualifying jog lengths during contiguous edge length summation.

Usage

-jog_ignore *value*

Arguments

- *value*

  A threshold value of the jog specified in microns.

Description

A threshold value, below which the jog in the edge segment is ignored. If the length of the jog is less than value specified, the length of the contiguous edge is summed without the portion of the jog.

Examples

Example 1

The -jogignore keyword is shown here in context.

```
BIASRULE bias.in \
   -overlapLayer bias.in \
   -tag full_sizing \
   -table row space col enclosure { \
            0.000 0.060 0.070 0.120 OPPOSITE EXTENDED 0.080 \
                                    OPPOSITE EXTENDED 0.080 :\
      0.000 0.000 0.005 0.007 0.007 \
      0.060 0.000 0.005 0.007 0.007 \
   } \
   -jogignore 0.005 -projecting -exclude_shielded
```

-length1

BIASRULE keyword: Control and Constraint Keywords

Classifies edges by length for subsequent biasing application. Used only with -table method.

Usage

-length1 *constraint*

Arguments

- *constraint*

  An optional argument specifying a range of positive floating-point values.

Description

An optional keyword and argument specifying the length condition an edge must meet in order for the rules in the table to be applied. Up to eight length1 constraints can be specified, one for each table. Only edges whose lengths are within the specified range are biased according to the specified -table value. If length1 is not specified, all of the fragments of the polygon get biased.

-lock

BIASRULE keyword: Control and Constraint Keywords

Locks fragments from subsequent table biasing.

Usage

-lock

Arguments

None

Description

Where multiple rule tables exist within a single BIASRULE command, each specifying various fragment biasing, the last rule table is applied. This optional keyword local to the containing BIASRULE command locks fragments, so subsequent rule tables cannot alter previously performed biasing.

-lock_all

BIASRULE keyword: Control and Constraint Keywords

Locks fragments globally from subsequent table biasing.

Usage

-lock_all

Arguments

None

Description

Where multiple BIASRULE commands exist within an SVRF file, the last BIASRULE command and its associated biasing is applied. This optional keyword local to the containing BIASRULE command locks all fragments across BIASRULE commands, so subsequent BIASRULE commands cannot alter previously performed biasing.

-minimize_jogs

BIASRULE keyword: Control and Constraint Keywords

Removes 1 DBU jogs from BIASRULE output.

Usage

-minimize_jogs

Arguments

None

Description

An optional keyword that removes 1 DBU jogs output by BIASRULE. This is enabled only for collinear, MRC restricted fragments with jogs of 1 DBU. This keyword does not guarantee ridding all jogs from BIASRULE output, but significantly reduces the number of jogs overall.

-projecting

BIASRULE keyword: Control and Constraint Keywords

Overrides metrics to bias all classified width fragments in projection.

Usage

-projecting [*length*] [-ignore_singularity]

Arguments

- *length*

  An optional argument setting the minimum distance a fragment must project onto the fragment being classified.

- -ignore_singularity

  An optional keyword that inhibits measurement of fragments projecting onto each other at a single point (also referred to as kissing corners). The effect of the -ignore_singularity keyword is shown in combination with the -projecting keyword in Figure 3-12.

Description

An optional keyword that affects projecting width fragments only. This keyword takes effect only when the OPPOSITE EXTENDED metric (specified with a value greater than 0) or the EUCLIDEAN metric is specified. The -projecting keyword applies bias, overriding potentially large values of OPPOSITE EXTENDED. BIASRULE projection is illustrated in Figure 3-11.

Figure 3-11. BIASRULE -projecting Usage

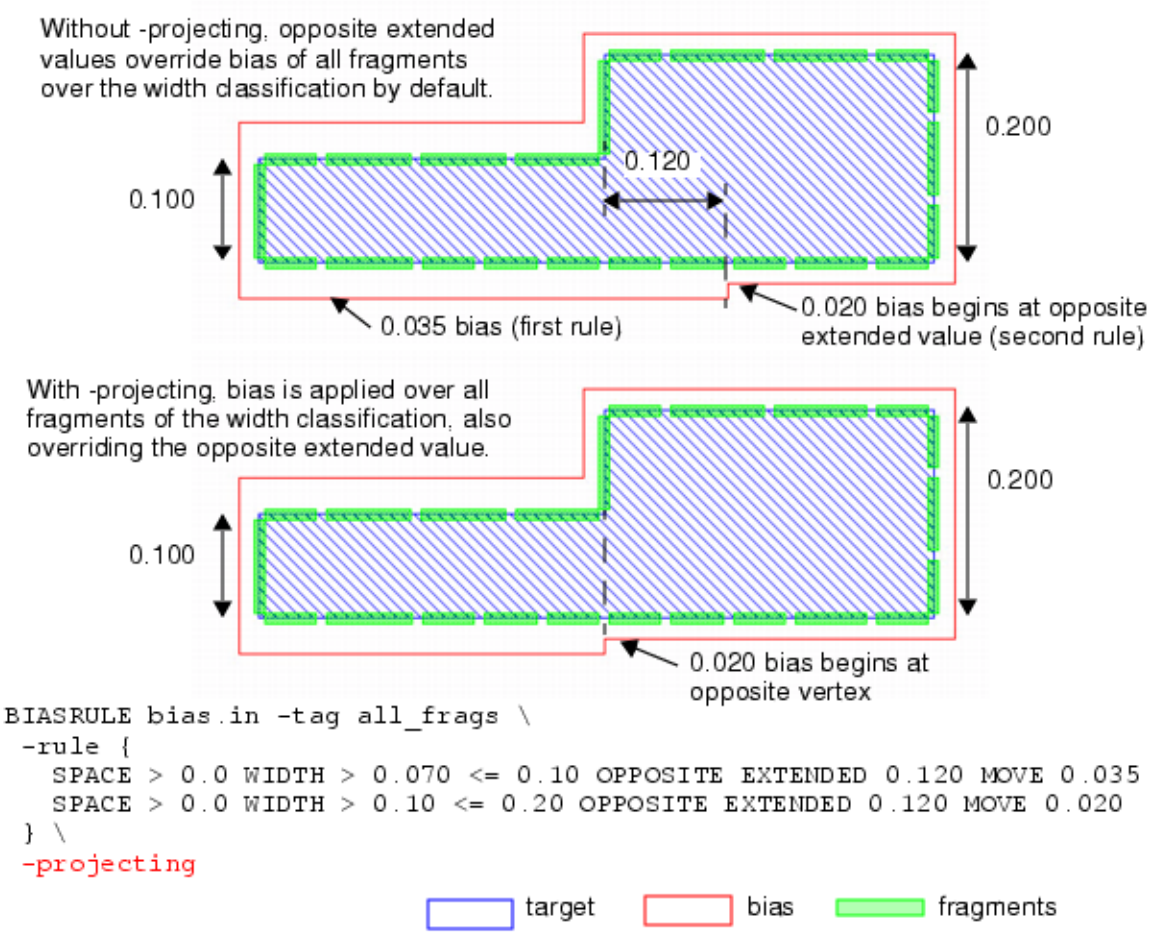
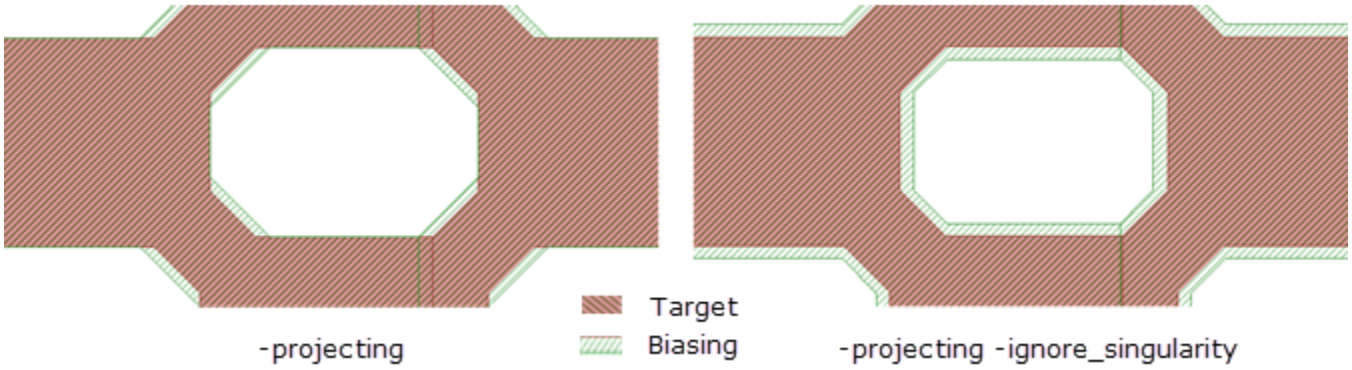


Figure 3-12. BIASRULE -projecting -ignore_singularity Usage



-projecting_space

BIASRULE keyword: Control and Constraint Keywords

Considers length of space classifications to bias all classified width fragments in projection.

Usage

-projecting_space [*length*] [-ignore_singularity]

Arguments

- *length*

  An optional argument setting the minimum length of space a fragment must maintain in projection to another fragment being classified. If a length value is specified, the fragment must project at least (greater than or equal to) the specified distance onto the fragment being classified.

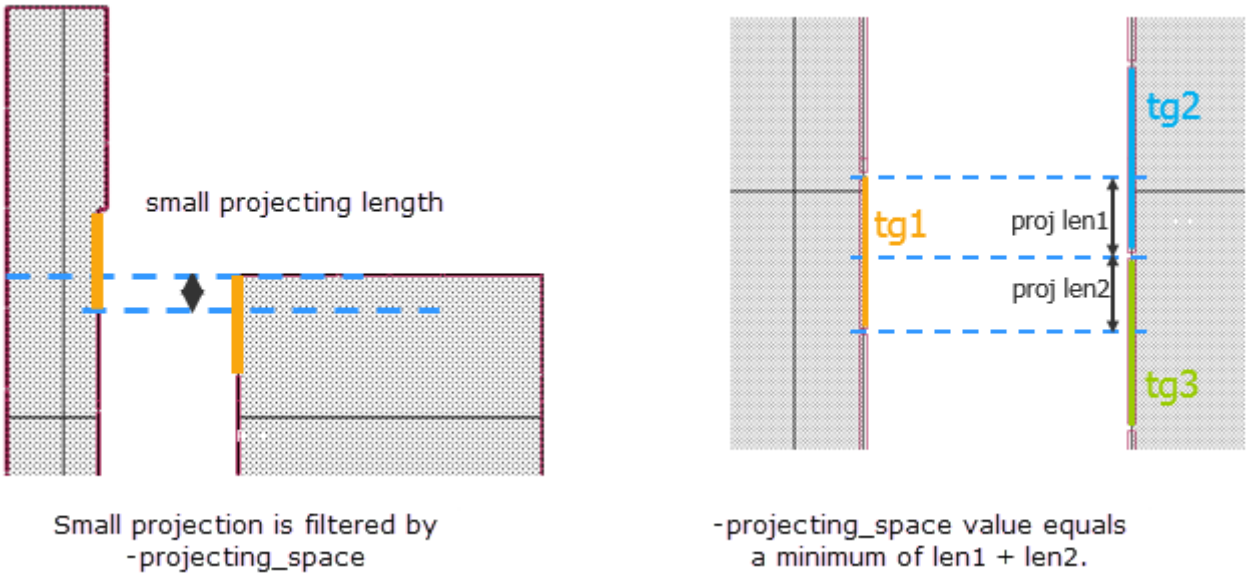
- -ignore_singularity

  An optional keyword that inhibits measurement of fragments projecting onto each other at a single point (also referred to as kissing corners). The effect of the -ignore_singularity keyword is shown in combination with the -projecting keyword in Figure 3-12.

Description

An optional keyword that affects projecting width fragments by space classification only. This keyword is applicable only with OPPOSITE EXTENDED metric (specified with a value greater than 0) or the EUCLIDEAN metric. The -projecting_space keyword applies bias only when the length of projected edges is satisfied and is similar to NEWTAG external -projecting in function. The -projecting_space keyword impacts space classification and produces results similar to -projecting for width classification. See Figure 3-13 for more information.

Figure 3-13. Usage of -projecting_space



Small projection is filtered by -projecting_space

-projecting_space value equals a minimum of len1 + len2.

selector

BIASRULE keyword: Biasing Methods

Selects a table of constraints within a BIASRULE table.

Usage

{*selector constraint metric*}

Arguments

- *selector constraint metric*

  An optional keyword specifying a selector, constraint and metric applied to the respective table constraints. The selector keyword is coupled with a floating-point constraint, and a metric. Allowed metrics include: space, width, length1, width2, space2, overlap, pitch, sidepitch, sidepitch2, sidespace and enclosure.

Description

An optional keyword used to select a table of constraints. The selector can be one of the following keywords:

- -width, -width2, -width3
- -space, -space2, -space3
- -length1
- -enclosure
- -enclosing
- -overlap
- -pitch, -sidepitch, -sidepitch2
- -sidespace

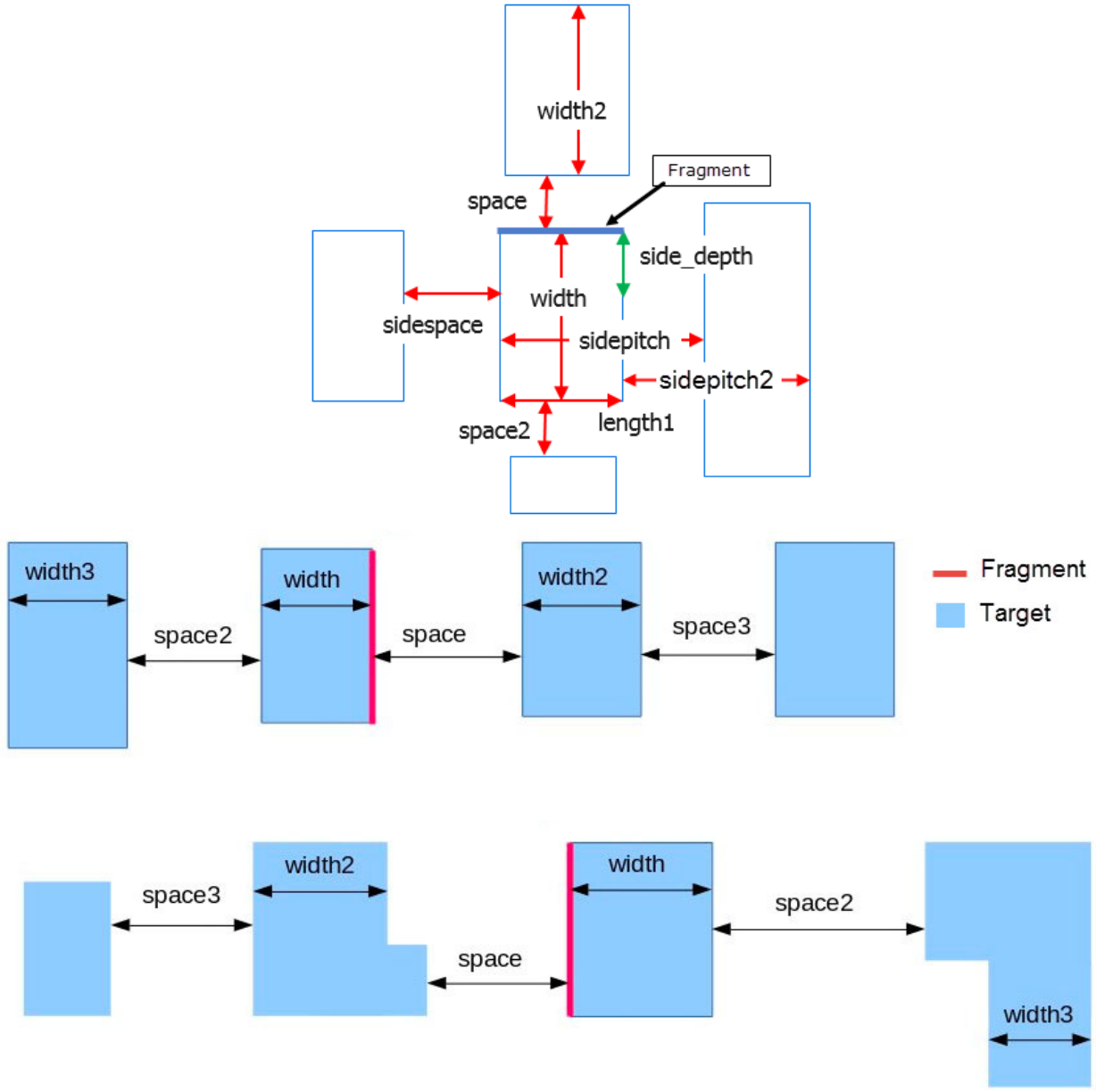
In 3D-interpolation mode, one selector constraint can be specified. In non-3D-interpolation mode, for each rule table, multiple selector constraints (up to 16) can be specified. In 3D-interpolation mode, the selector constraint must only be a number (for example, 0.001) rather than a range (for example, > 0.001).

Examples

Example 1

Figure 3-14 illustrates spatial and selector relationships.

Figure 3-14. BIASRULE selector Relationships



Example 2

An example of the selector used with the -table keyword for 3D interpolation mode is shown in Figure 3-15.

Figure 3-15. 3D Interpolation BIASRULE selector Usage

```
...  -length1 0.1 opposite -table { ... } \
     -length1 0.2 opposite -table { ... } \
     -length1 0.3 opposite -table { ... }
```

Example 3

An example of the selector used with the -table keyword for non-3D interpolation mode is shown in Figure 3-16. In non-3D interpolation usage, the tags must be specified for every -table iteration.

Figure 3-16. Non-3D Interpolation BIASRULE selector Usage

```
...  -tag le_frags \
     -length1 > 0.2 opposite -table { ... } \
     -tag le_frags \
     -length1 > 0.3 opposite -table { ... } \
     -tag le_frags \
     -length1 > 0.4 opposite -table { ... }
```

-side_depth

BIASRULE keyword: Control and Constraint Keywords

Determines distance perpendicular to fragments when computing sidespace or sidepitch metrics.

Usage

-side_depth *depth*

Arguments

- *depth*

  An argument specifying the value of side depth.

Description

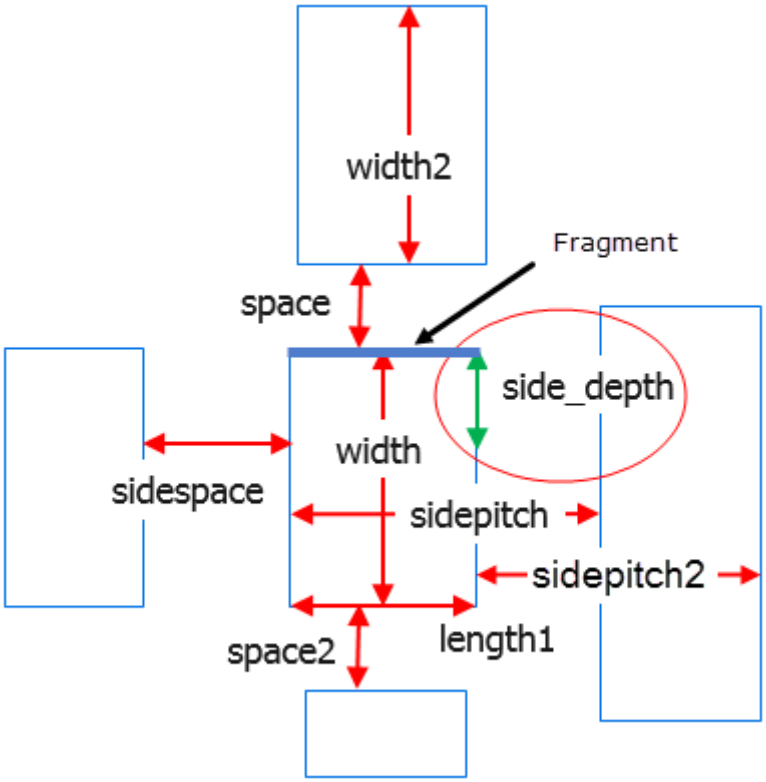
An optional parameter that specifies how much distance in a perpendicular direction to the fragment is considered when computing sidespace or sidepitch metrics. The default value is 0 microns.

Examples

Example 1 — Using -side_depth for sidespace or sidepitch Metrics.

```
-table row length1 col sidespace { \
   c1 c2 c3 ... cN :
}  -side_depth 0.020

-table row width col sidepitch { \
   c1 c2 c3 ... cN  :
} -side_depth 0.020
```

Figure 3-17. side_depth Parameter



-side_measurement

BIASRULE keyword: Control and Constraint Keywords

Determines the classification parameter when computing sidespace or sidepitch metrics.

Usage

-side_measurement {min | max}

-side_measurement {min | max | avg} bias

Arguments

- min

  An argument specifying the minimum value of the classification.

- max

  An argument specifying the maximum value of the classification.

- avg

  An argument specifying the average value of the classification.

- bias

  An argument specifying minimum, maximum or average bias value.

Description

An optional argument that determines which parameter to use for classification - the minimum or maximum value. You only use this argument where two sidespace or sidepitch metrics are present (one on either side of the target shape).

The second usage provides a method to consider the maximum, minimum or average bias corresponding to the two sidespace or sidepitch values.

Examples

Example 1 — Using -side_measurement for sidespace or sidepitch Metrics.

```
-table row length1 col sidespace { \
   c1 c2 c3 ... cN :
}  -side_measurement avg bias

-table row width col sidepitch { \
   c1 c2 c3 ... cN  :
} -side_measurement max
```

-two_pass_mode

BIASRULE keyword: Control and Constraint Keywords

Forces GROW_UNTIL_SPACE or GROW_UNTIL_WIDTH irrespective of edge blockage.

Usage

-two_pass_mode

Arguments

None

Description

An optional argument used to force GROW_UNTIL_SPACE or GROW_UNTIL_WIDTH to reach their specified values, enabling asymmetric biasing where required. Without -two_pass_mode, complete biasing of edges may not be achieved where MRC constraints block edge movement.

-unmoved

BIASRULE keyword: Control and Constraint Keywords

Forces measurement from unmoved fragments.

Usage

-unmoved

Arguments

None

Description

When more than one BIASRULE command is specified, the second and subsequent BIASRULE command measures from the moved position of any fragment. This optional keyword forces measurement from unmoved fragment positions instead.

Biasing Methods

BIASRULE uses two methods to organize biasing commands and values.

Table 3-4. BIASRULE Biasing Methods

Keyword	Description
-rule	Specifies the biasing of edges with a rule-based method per provided space and other ranges.
-table	Specifies the biasing of edges with a table-based method per provided space and width ranges.

For organizational keyword information, refer to “BIASRULE” on page 50.

-rule

BIASRULE keyword: Biasing Methods

Specifies the biasing of edges with a rule-based method per provided space and other ranges.

Usage

-rule '**{**'
SPACE *range* [*metric*]
SPACE2 *range* [*metric*]]
WIDTH *range* [*metric*]]
ENCLOSING *range* [*metric*]]
LENGTH1 *range*]

{MOVE *value* |
GROW_UNTIL_SPACE *value* [max_disp *value*] |
GROW_UNTIL_WIDTH *value* [max_disp *value*] |
GROW_UNTIL_ENCLOSING *value*}
'**}**' [-two_pass_mode]

Arguments

- **SPACE *range*** [*metric*]

  A required keyword and required range of floating point numbers specified in microns defining distance between edges for the corresponding biasing to apply. You must specify at least one SPACE keyword for each -rule statement. An unbounded interval is interpreted as the bounded complement of the unbounded interval. For example, > a is interpreted as everything not in the <= a interval.

  A ***range*** cannot be a strict equality (a == b). For example:

  - > a < b
  - <= a < b
  - > a

  metric — One of three optional keywords may be specified with the SPACE keyword:

  EUCLIDEAN *value* — Specifies the value in distance from the outer edge, using an arc around corners.

  OPPOSITE — The default metric, trimmed to facing edges of different shapes.

  OPPOSITE EXTENDED *value* — Specifies a lateral extension to the OPPOSITE metric.

- SPACE2 *range* [*metric*]

  An optional keyword and range of positive floating point numbers specified in microns that specifies the secondary distance between edges of adjacent shapes for the corresponding biasing to apply.

metric — Any of the aforementioned SPACE metrics may be used with WIDTH.

- WIDTH *range* [*metric*]

  An optional keyword and range of positive floating point numbers specified in microns that specifies the width a shape must meet in order for biasing to apply to an edge on that shape.

  metric — Any of the aforementioned SPACE metrics may be used with WIDTH.

- ENCLOSING *range* [*metric*]

  An optional keyword and range of positive floating point numbers specified in microns that specify the enclosing condition a shape must meet in order for the corresponding biasing to apply.

  metric — Any of the aforementioned SPACE metrics may be used with ENCLOSING.

- LENGTH1 *range*

  An optional keyword and range of positive floating point numbers specified in microns that specify the length an edge of a shape must meet in order for the corresponding biasing to apply to that edge.

- MOVE *value*

  An optional keyword and floating point number specified in microns that moves biased edges satisfying the spatial criteria by the specified *value*. A negative value causes biasing toward the interior of polygons. A positive value causes outward biasing.

- GROW_UNTIL_SPACE *value* [max_disp *value*]

  An optional keyword and positive floating point number specified in microns that displaces edges of shapes on the ***in_layer*** to conform to spaces of *value* within the stated mrc_rules.

  max_disp *value* — An optional keyword and argument pair specifying the maximum displacement an edge is allowed, but not exceeding the values specified in the mrc_rule command.

  The bias is determined by the equation (space-*value*)/2 where *value* is the value specified and space is the measured amount for the fragment. If the fragment is opposite from multiple edges, the smallest space is used. If the fragments on both sides of the space use the same GROW_UNTIL_SPACE value, then the space is that *value*.

- GROW_UNTIL_WIDTH *value* [max_disp *value*]

  An optional keyword and positive floating point number specified in microns that displaces edges of shapes on the ***in_layer*** to conform to widths of *value* within the stated mrc_rules.

  max_disp *value* — An optional keyword and argument pair specifying the maximum displacement an edge is allowed, but not exceeding the values specified in the mrc_rule command.

- GROW_UNTIL_ENCLOSING *value*

  An optional keyword and floating point number specified in microns that biases fragments on the ***in_layer*** when the enclosure range is met and the specified movement is larger than the range value. Fragments are moved on the ***in_layer*** until the fragment is the specified

distance from the edge of the enclosed layer. GROW_UNTIL_ENCLOSING can only increase the coverage, otherwise the fragment is not moved. This option can be specified once per -rule keyword. This option can only be used when an enclosing layer is specified with -enclosedlayer.

- -two_pass_mode

  An optional argument, -two_pass_mode, allows GROW_UNTIL_SPACE or GROW_UNTIL_WIDTH to reach their specified values, enabling asymmetric biasing where required. Without -two_pass_mode, complete biasing of edges may not be achieved where MRC constraints block edge movement.

Description

A required keyword and argument list specifying rules for biasing fragments. The syntax for rules is similar to the syntax used in OPCBIAS. This command retargets by specifying a group of fragments to bias and various rules for biasing them. For a full example, refer to "Comprehensive BIASRULE SVRF Files."

Examples

Example 1

The following example shows correction parameters specified by the rule method. These rules are equivalent to the values specified in the -table of Example 3-2 on page 128.

Example 3-1. Rule Method Parameter Specification

```
-rule { \
   SPACE    >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
   WIDTH    >= 0.050 < 0.060 OPPOSITE EXTENDED 0.0200
   LENGTH1 >  0.062          MOVE 0.0030 \
   \

   SPACE    >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
   WIDTH    >= 0.060 < 0.061 OPPOSITE EXTENDED 0.0200
   LENGTH1 >  0.062          MOVE 0.0030 \
   \

   SPACE    >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
   WIDTH    >= 0.061 < 0.062 OPPOSITE EXTENDED 0.0200
   LENGTH1 >  0.062          MOVE 0.0020 \
   \

   SPACE    >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
   WIDTH    >= 0.062 < 0.063 OPPOSITE EXTENDED 0.0200
   LENGTH1 >  0.062          MOVE 0.0010 \
   \

   SPACE    >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
   WIDTH    >= 0.063 < 0.064 OPPOSITE EXTENDED 0.0200
   LENGTH1 >  0.062          MOVE 0.0000 \
   \
```

```
SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.064 < 0.065 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0000 \
\

SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.065         OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0000 \
\

SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.050 < 0.060 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0050 \
\

SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.060 < 0.061 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0050 \
\

SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.061 < 0.062 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0040 \
\

SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.062 < 0.063 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0030 \
\

SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.063 < 0.064 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0020 \
\

SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.064 < 0.065 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0010 \
\

SPACE  >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.065         OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0000 \
\

SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.050 < 0.060 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0070 \
\

SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.060 < 0.061 OPPOSITE EXTENDED 0.0200
LENGTH1 >  0.062         MOVE 0.0070 \
\
```

```
    SPACE   >= 0.075          OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.061 < 0.062 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062          MOVE 0.0060 \
    \

    SPACE   >= 0.075          OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.062 < 0.063 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062          MOVE 0.0050 \
    \

    SPACE   >= 0.075          OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.063 < 0.064 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062          MOVE 0.0040 \
    \

    SPACE   >= 0.075          OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.064 < 0.065 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062          MOVE 0.0030 \
    \

    SPACE   >= 0.075          OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.065          OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062          MOVE 0.0020 \
}
```

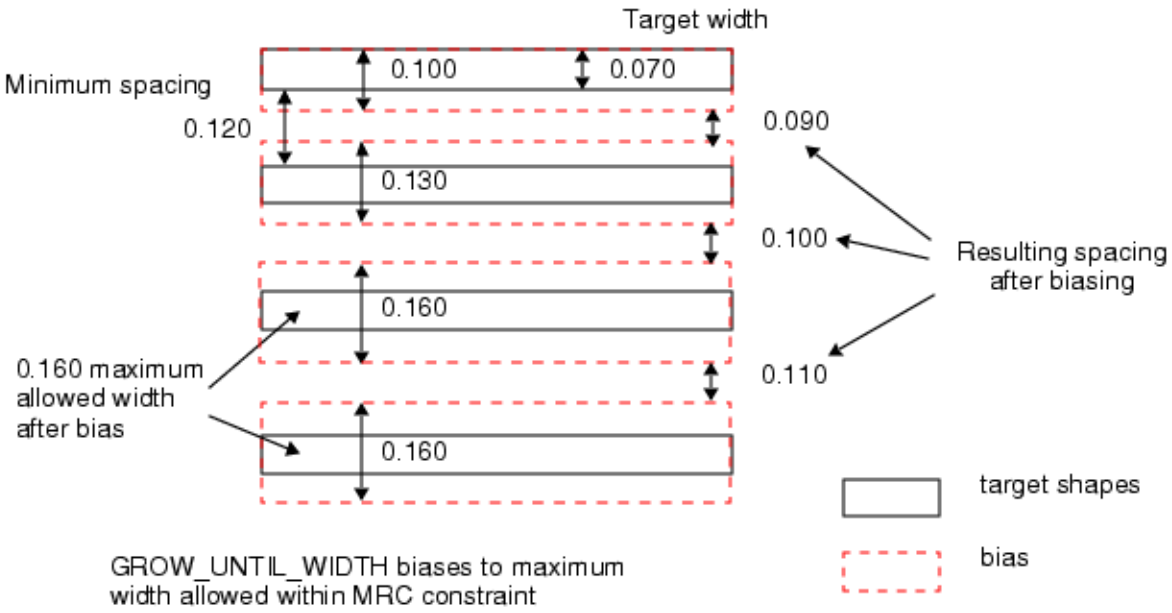
Example 2

Example 3-18 uses GROW_UNTIL_WIDTH to grow the polygons to a ceiling width of 160nm. This is illustrated in Figure 3-19.

Figure 3-18. GROW_UNTIL_WIDTH Code

```
SPACE   >= 0.120          OPPOSITE EXTENDED 0.0795
WIDTH   >= 0.070 <= 0.075 OPPOSITE EXTENDED 0.2000
LENGTH1 >  0.062          GROW_UNTIL_WIDTH  0.1600
```

Figure 3-19. GROW_UNTIL_WIDTH Example



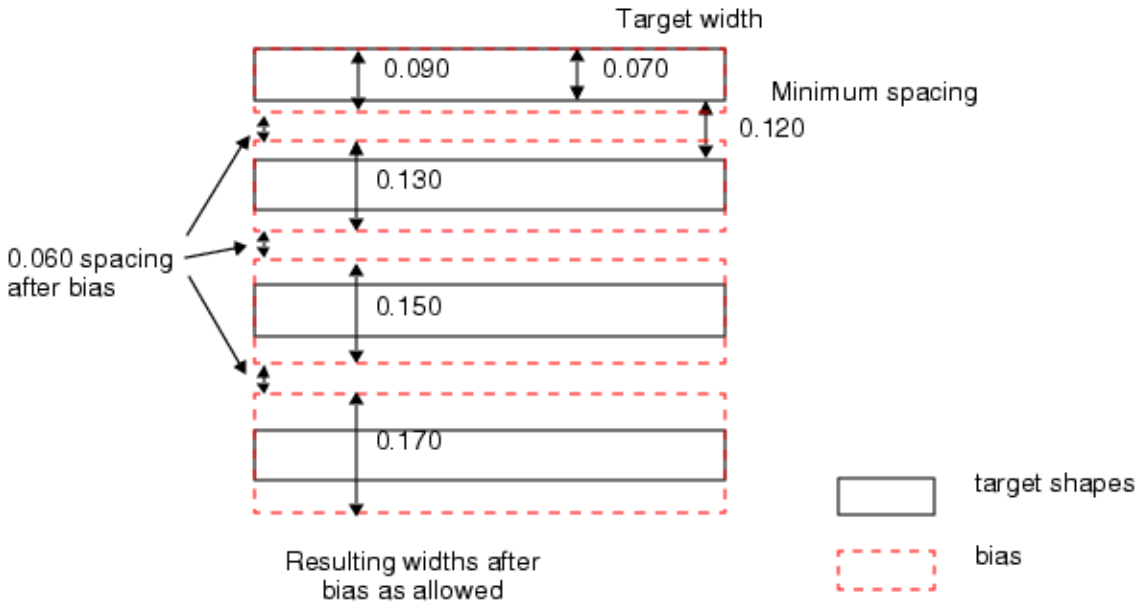
Example 3

Example 3-20 uses GROW_UNTIL_SPACE to grow the polygons until they are separated by 60nm. This is illustrated in Figure 3-21.

Figure 3-20. GROW_UNTIL_SPACE Code

```
SPACE    >= 0.120           OPPOSITE EXTENDED 0.0795
WIDTH    >= 0.070 <= 0.075  OPPOSITE EXTENDED 0.2000
LENGTH1 >  0.062            GROW_UNTIL_SPACE  0.0600
```

Figure 3-21. GROW_UNTIL_SPACE Example



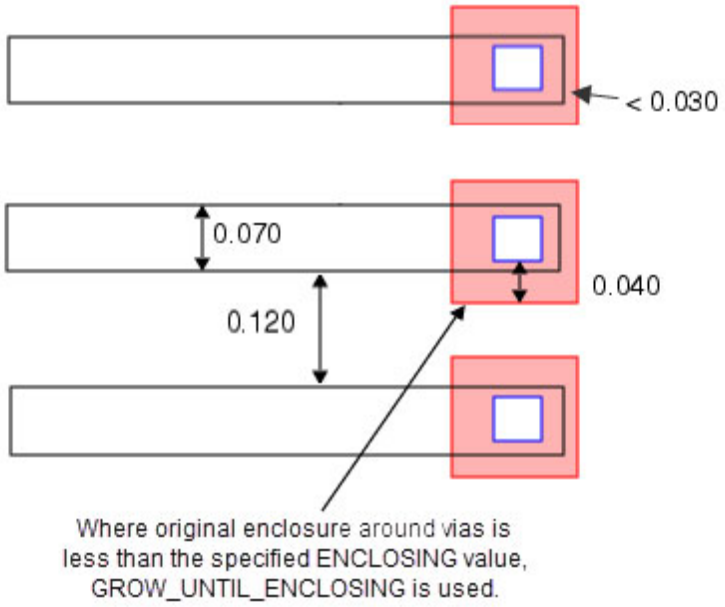
Example 4

Example 3-22 uses GROW_UNTIL_ENCLOSING to grow all conforming polygons until they enclose the vias (or any other conforming shapes) by 40nm. This is illustrated in Figure 3-23.

Figure 3-22. GROW_UNTIL_ENCLOSING

```
SPACE     >= 0.120 WIDTH >= 0.070
ENCLOSING <  0.030 OPPOSITE EXTENDED 0.001
LENGTH1   >  0.059 GROW_UNTIL_ENCLOSING 0.040
```

Figure 3-23. GROW_UNTIL_ENCLOSING Example



-table

BIASRULE keyword: Biasing Methods

Specifies the biasing of edges with a table-based method per provided space and width ranges.

Usage

-table [row *metric* col *metric*]
‘**{**’

	col1	***col2***	***colN***	[*col_proj* [*angle*] *row_proj* [*angle*]] ‘**:**’
row1	***c1_r1_bias***	***c2_r1_bias***	***cN_r1_bias***	
row2	***c1_r2_bias***	***c2_r2_bias***	***cN_r2_bias***	
rowN	***c1_rN_bias***	***c2_rN_bias***	***cN_rN_bias***	

‘**}**’

Arguments

- row *metric* col *metric*

  An optional pair of metrics applied to the respective row and column that constrain specified values to a matrix of measurements. Row and column metrics provide measurement between all edge relationships and determine their assignment to either row or column within the table matrix. The default metrics are width and space for row and col, respectively. The row and column metrics provide all possible measurement relationships and are depicted in Figure 3-24 on page 126.

  enclosing

  A measurement between the inside edge of the bias shape and outside edge of the overlapping shape The overlapping layer is declared by -overlapLayer.

  enclosure

  A measurement between the outside edge of the bias shape and inside edge of the overlapping shape. The overlapping layer is declared by -overlapLayer.

  length1

  A measurement of the fragment edge length.

  overlap

  Selects bias shapes overlapping any shape of a layer declared with -overlapLayer.

  pitch

  A measurement between the edge fragment of the bias shape to the inside edge of the fragment-facing shape. The pitch metric is measured perpendicular to the fragment edge.

  sidepitch

  A measurement between the inside edge of the bias shape side to the outside edge of an adjacent shape. The sidepitch metric is measured parallel to the fragment edge. The sidepitch metric is dependent on the value of -side_depth.

sidepitch2

A measurement between the outside edge of the bias shape side to the inside edge of an adjacent shape. The sidepitch2 metric is measured parallel to the fragment edge. The sidepitch2 metric is dependent on the value of -side_depth.

sidespace

A measurement between the outside edges of the bias and secondary shapes. The sidespace metric is dependent on the value of -side_depth.

space

A measurement between the edge fragment of the bias shape and outside edge of the fragment-facing shape. This is the default column metric.

space2

A measurement between the outside edges of the bias and secondary shapes.

space3

A measurement between the outside edges of the fragment-facing and tertiary shapes.

width

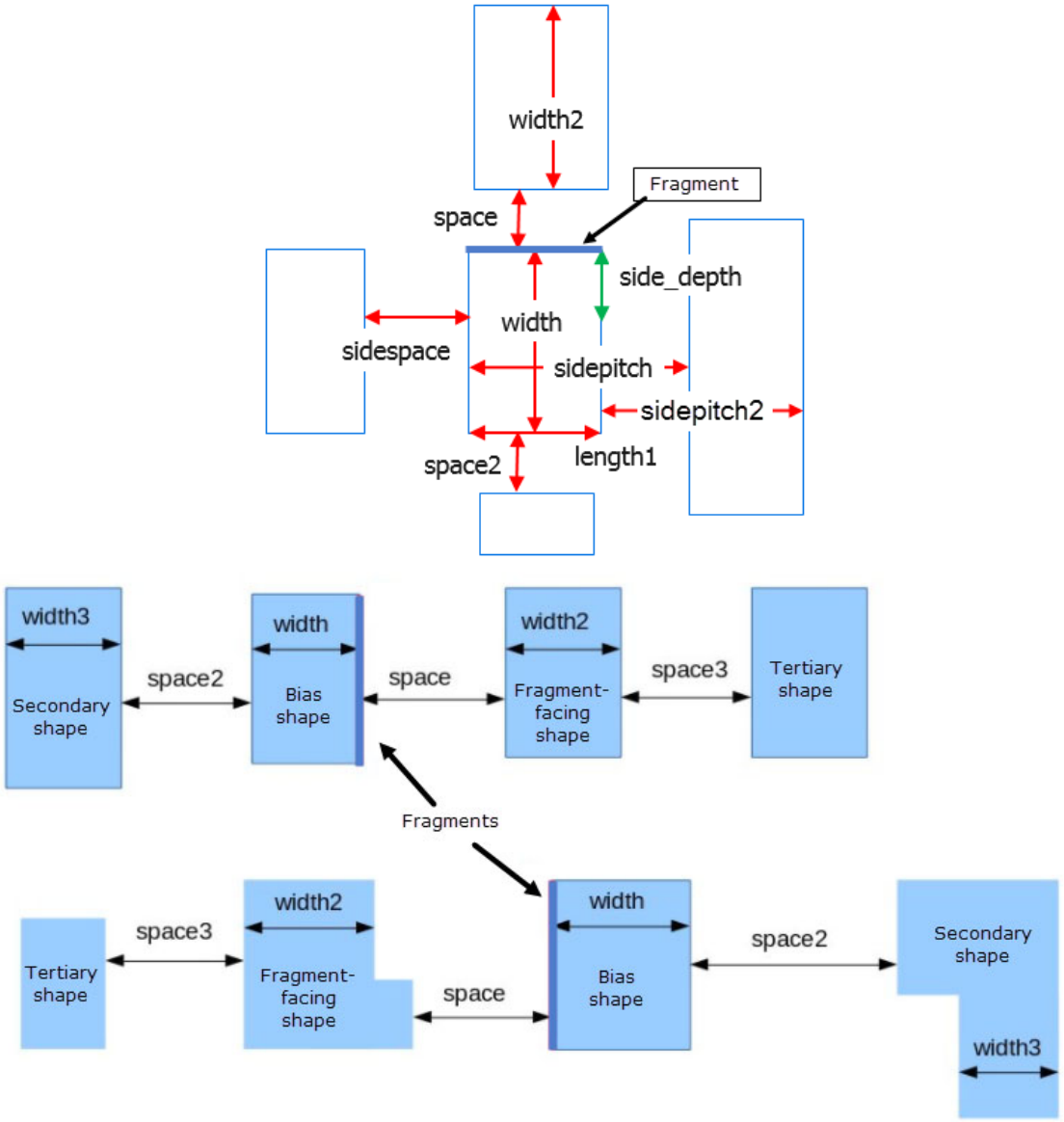
A measurement between the edge fragment of the bias shape and the facing inside edge of the same primary shape. The width metric is measured perpendicular to the fragment edge. This is the default row metric.

width2

A measurement of the width of the fragment-facing shape. The width2 metric is measured perpendicular to the fragment edge.

The metric measurement relationships are shown in the following figure:

Figure 3-24. Measurement Relationships



- ***col1 ... colN***

  A series of required floating-point values in user units specifying boundaries of column measurements. These values must be unique and specified in ascending order. You must specify at least one column value. Where no column metric is specified, column defaults to the space metric.

- ***row1 ... rowN***

  A series of required floating-point values in user units specifying boundaries of row measurements. These values must be unique and specified in ascending order. You must

specify at least one row value. Where no row metric is specified, row defaults to the width metric.

- ***c1_r1_bias ... cN_rN_bias***

  A series of required floating-point values in user units specifying biasing of the ***in_layer*** at the corresponding metric matrix of the table. Negative numbers cause biasing toward the interior of shapes. Positive numbers cause outward biasing.

- *col_proj* [angle *constraint*] *row_proj* [angle *constraint*]

  An optional group of arguments specifying the column and row projection type, respectively, with their associated optional angle constraints.

  col_proj, *row_proj* — Three projection types are provided:

  OPPOSITE — Direct projection from edge to edge. This is default.

  OPPOSITE EXTENDED *value* — Like OPPOSITE but with an additional lateral extension value. This permits the area of edge projection to be modified. The value may be either positive or negative.

  EUCLIDEAN *value* — Distance determined by spherical radius from the shape edge.

  angle *constraint* — The angle keyword and constraint pair provides a constraint to filter skewed edges to be excluded from measurement, and can only be specified in conjunction with row and column metrics. The angle keyword only applies to space and width metrics. The angle keyword and constraint must be specified after the projection keyword, the constraint comprising an operator separated from the angle value(s) by a space:

  ```
  OPPOSITE angle > 15 OPPOSITE EXTENDED 0.1 angle >= 0 < 75
  ```

Description

A required keyword and matrix specifying space and width rules with their respective biasing values. For an example of a table specification and further detail how biasing is applied, refer to Example 3-2.

If no tag set is specified for -table, then only one table can be specified in which case the table applies biasing to all the fragments on in_layer.

No more than eight tables may be specified. In the first row of the table, a space and colon must appear at the end of the line or a parsing error results.

Examples

Example 1

This example of the table method demonstrates how to specify parameters. The values specified in the table are equivalent to the rules specified with the -rule method seen in Example 3-1 on page 118.

Example 3-2. BIASRULE -table Usage

```
-table { \
         0.045 0.060 0.075 OPPOSITE OPPOSITE EXTENDED 0.020 :\
   0.050 0.003 0.003 0.007 \
   0.060 0.003 0.004 0.008 \
   0.070 0.003 0.005 0.009 \
   ...
}
```

The boundaries of the space measurement are 0.045, 0.060, and 0.075. The boundaries of the width measurement are 0.050, 0.060, and 0.070.

If space is less than the smallest space measurement boundary, or if width is less than the smallest width measurement boundary, no bias is applied. Given Example 3-2, no bias is applied if space is less than 0.045 or if width is less than 0.050.

If a space measurement is equal to a space measurement boundary, the bias corresponding to that boundary is applied. If a space measurement lies between two boundaries, the bias of the lower boundary is applied. Given Example 3-2, if space is equal to 0.060 and width is equal to 0.0515, a bias of 0.004 is applied.

If space is greater than or equal to the largest space measurement boundary, or if width is greater than or equal to the largest width space measurement, the bias corresponding to the largest space and width boundary is applied. Given Example 3-2, if space is equal to 0.077 and width is equal to 0.0735, then a bias of 0.009 is applied.

Example 2

Example 3-3 demonstrates how to specify row and col metrics for table method. The metrics specified are not default, and require specification to be applied to the table:

Example 3-3. BIASRULE -table row and col Usage

```
-table row width2 col space2 { \
         0.050 0.070 0.09  : \
   0.050 0.001 0.002 0.003   \
   ...
}
```

Smoothing Methodology

Smoothing is performed by incremental and then decremental passes.

1. Incremental smoothing — Fragments can be smoothed with a fragment neighbor on either side, or with a fragment neighbor on both sides of the fragment in question.
2. Decremental smoothing — After the incremental smoothing pass, the decremental smoothing pass tracks the fragments that could not be moved to the desired location due

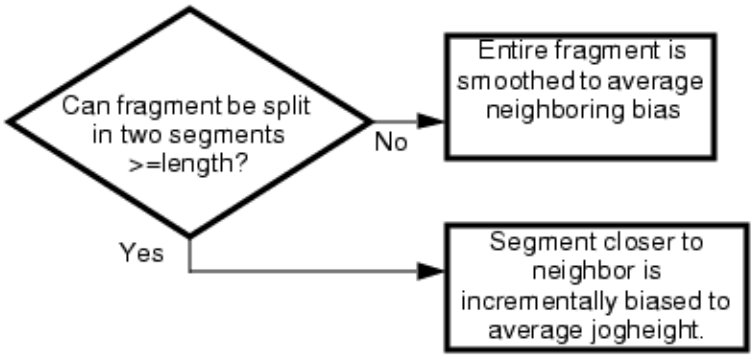
to MRC restrictions. In this smoothing pass, the neighbors of the restricted fragments are decrementally smoothed so the fragments can be moved without violating MRC.

Incremental smoothing of fragments can be performed in two ways:

- Smooth fragments with a fragment neighbor on either side.
  - If the fragment can be split into two pieces greater than or equal to length, then the segment closer to the neighbor is incrementally biased to the average jogheight.
  - If the fragment cannot be split into two pieces greater than or equal to length, then the entire fragment is smoothed to the average bias of its neighbors.

The process of smoothing fragments with a neighbor on either side is shown in Figure 3-25.

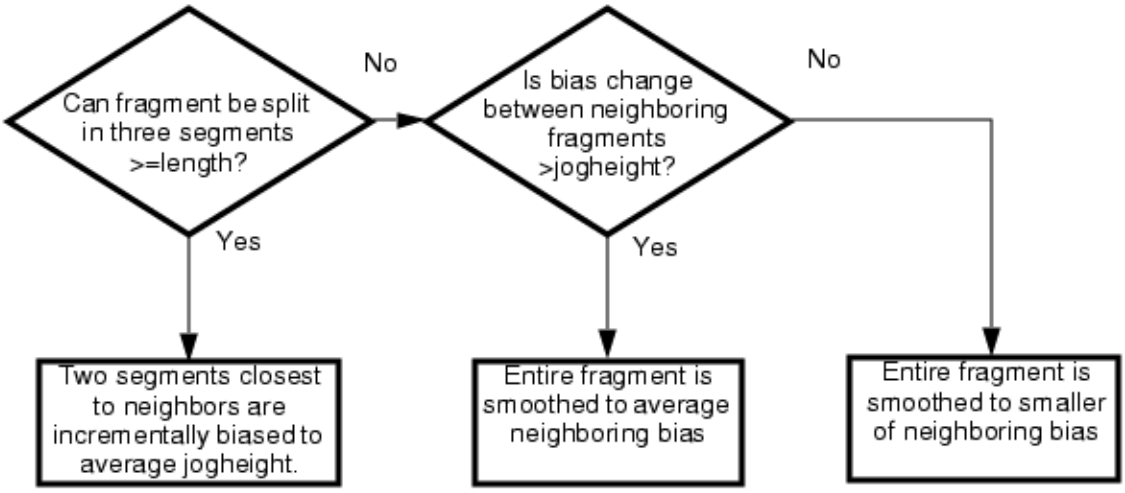
Figure 3-25. Smoothing Fragments With a Neighbor on Either Side



- Smooth fragments with a fragment neighbor on both sides.
  - If the fragment can be split into three pieces greater than or equal to length, then the two pieces closest to the neighbors are incrementally biased to the average jogheight.
  - If the fragment cannot be split into three pieces greater than or equal to length, then the smoothing process depends on the difference between the biases of the previous fragment and the next fragment.
  - If the bias difference between the previous fragment and the next fragment is greater than the specified jogheight, then the fragment is smoothed to the average bias of its neighbors.
  - If the bias difference is less than or equal to the specified jogheight, then the fragment is smoothed to the smaller bias of its neighbors.

The process of smoothing fragments with a neighbor on both sides is shown in Figure 3-26.

Figure 3-26. Smoothing Fragments With Neighbors on Both Sides



Comprehensive BIASRULE SVRF Files

The -rule and -table writing methods differ but provide identical biasing.

Rule Method

Example 3-27 contains a complete SVRF file demonstrating the BIASRULE -rule method.

Figure 3-27. BIASRULE for Rule Method

```
////////////////////////////////////////
         CONTROL STATEMENTS          //
////////////////////////////////////////

LAYOUT SYSTEM
OASISLAYOUT PATH
"$GDSin" // variable substitution with $
LAYOUT PRIMARY
"*"LAYOUT PROCESS BOX RECORD YES
LAYOUT ERROR ON INPUT     NO
FLAG SKEW    YES
FLAG ACUTE   YES
FLAG OFFGRID YES
DRC MAXIMUM RESULTS ALL
DRC MAXIMUM VERTEX 199
DRC KEEP EMPTY NO
DRC RESULTS DATABASE "$GDSout" OASIS PSEUDO
DRC SUMMARY REPORT   "./log/rule_interpolate.sum" HIER
UNIT LENGTH U
PRECISION   4000
RESOLUTION  1
LAYER BiasruleLyr 15
```

```
//--------------------------- Bias Operation --------------------------//
AfterBiasrule = LITHO NMBIAS FILE nmbias.setup BiasruleLyr MAP postbias
//--------------------- layer and tile definitions ---------------------//

LITHO FILE nmbias.setup [/*
   tilemicrons 40 adjust
   layer bias.in hidden
   options first.pass {
      version 1
      layer bias.in opc

//--------------------------- default fragmentation -------------------//
   fragment_min 2*frgu
   fragment_inter -interdistance (largest_space) -ripplelen 2*frgu \
      -num 2 -shield 1 -externalOnly -adjust
   fragment_inter -interdistance (largest_width) -ripplelen 2*frgu \
      -num 2 -shield 1 -internalOnly -adjust
   fragment_corner convex fragment 2*frgu 2*frgu
   fragment_corner concave fragment 2*frgu 2*frgu
   fragment_max 4*frgu

//--------------------------- fragmentation ---------------------------//
   fragment_inter -interdistance 0.64 -num 1 -ripplelen 0.0625 \
      -shield 1 -shift 0.0795
   fragment_min            0.0600
   fragment_corner convex  0.0625
   fragment_corner concave 0.0625
   fragment_max            5.0

//---------------------- mask rule checks ----------------------//
   mrc_rule external bias.in {
      use 0.005 euclidean
      not_projecting
      use 0.070 euclidean
   }

//------------------------------ tagging -----------------------------//
   NEWTAG all bias.in -out all_frags
   NEWTAG fragment all_frags len < 0.1 corner1 convex corner2 convex \
      len1 > 0.05 len2 > 0.05 -out le_frags
   NEWTAG neighbor both le_frags len <= 0.2 corner convex \
      -out le_adj_frags

   TAG or le_frags le_adj_frags -out all_le_frags
   TAG subtract all_frags le_frags -out full_sizing1
   NEWTAG external le_frags full_sizing1 opposite \
      -out tight_le range <= 0.1
   NEWTAG neighbor both tight_le len <= 0.2 \
      corner convex -out tight_le_adj_frags
   NEWTAG external full_sizing1 le_frags opposite extended 0.0795 \
      -out tight_lin range <= 0.1
   NEWTAG neighbor both tight_lin len <= 0.2 corner no_corner \
      -out tight_lin_adj_frags_1
   TAG subtract tight_lin_adj_frags_1 tight_lin -out tight_lin_adj_frags
   TAG or tight_lin_adj_frags tight_le_adj_frags -out reduced_sizing
   TAG subtract full_sizing1 reduced_sizing -out full_sizing
```

```
   NEWTAG edge all_frags len >= 0.55 -out long_frags
   NEWTAG internal all_frags all_frags opposite extended 0.0795 \
      -out narrow_lin range <= 0.1

//----------------------------- BIASRULE -----------------------------//
   BIASRULE bias.in -tag full_sizing \
   -rule { \SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.050 < 0.060 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0030 \
      \

      SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.060 < 0.061 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0030 \
      \

      SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.061 < 0.062 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0020 \
      \

      SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.062 < 0.063 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0010 \
      \

      SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.063 < 0.064 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0000 \
      \

      SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.064 < 0.065 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0000 \
      \

      SPACE   >= 0.045 < 0.060 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.065         OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0000 \
      \

      SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.050 < 0.060 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0050 \
      \

      SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.060 < 0.061 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0050 \
      \

      SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
      WIDTH   >= 0.061 < 0.062 OPPOSITE EXTENDED 0.0200
      LENGTH1 >  0.062         MOVE 0.0040 \
      \
```

```
    SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.062 < 0.063 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0030 \
    \

    SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.063 < 0.064 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0020 \
    \

    SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.064 < 0.065 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0010 \
    \

    SPACE   >= 0.060 < 0.075 OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.065         OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0000 \
    \

    SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.050 < 0.060 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0070 \
    \

    SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.060 < 0.061 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0070 \
    \

    SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.061 < 0.062 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0060 \
    \

    SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.062 < 0.063 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0050 \
    \

    SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.063 < 0.064 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0040 \
    \

    SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.064 < 0.065 OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0030 \
    \

    SPACE   >= 0.075         OPPOSITE EXTENDED 0.0795
    WIDTH   >= 0.065         OPPOSITE EXTENDED 0.0200
    LENGTH1 >  0.062         MOVE 0.0020 \
} \
```

```
      -interpolate width
      BIASSMOOTH bias.in -minFragLength 0.06 -minJogHeight 0.015
   }
   setlayer postbias = nmbias bias.in MAP bias.in OPTIONS first.pass
*/]

// ----------------------------- Output ------------------------------ //
BiasruleLyr   { COPY BiasruleLyr   } DRC CHECK MAP BiasruleLyr   15
AfterBiasrule { COPY AfterBiasrule } DRC CHECK MAP AfterBiasrule 115
```

Table Method

Example 3-28 contains a complete SVRF file demonstrating the BIASRULE -table method.

Figure 3-28. BIASRULE for Table Method

```
////////////////////////////////////////
//          CONTROL STATEMENTS          //
////////////////////////////////////////
LAYOUT SYSTEM  OASIS
LAYOUT PATH    "$GDSin" // The dollar sign
LAYOUT PRIMARY "*"
LAYOUT PROCESS BOX RECORD YES
LAYOUT ERROR ON INPUT     NO/GROW_UNTIL_E
FLAG SKEW    YES
FLAG ACUTE   YES
FLAG OFFGRID YES
DRC MAXIMUM RESULTS ALL
DRC MAXIMUM VERTEX 199
DRC KEEP EMPTY NO
DRC RESULTS DATABASE "$GDSout" OASIS PSEUDO
DRC SUMMARY REPORT   "./log/table_interpolate.sum" HIER
UNIT LENGTH U
PRECISION   4000
RESOLUTION  1
LAYER BiasruleLyr 15

//--------------------------- Bias Operation --------------------------//
AfterBiasrule = LITHO NMBIAS FILE nmbias.setup BiasruleLyr MAP postbias
//--------------------- layer and tile definitions ---------------------//
LITHO FILE nmbias.setup [/*
tilemicrons 40 adjust
layer bias.in hidden
options first.pass {
   version 1
   layer bias.in opc

//--------------------------- fragmentation ---------------------------//
   fragment_inter -interdistance 0.64 -num 1 -ripplelen 0.0625 \
      -shield 1 -shift 0.0795
   fragment_min            0.0600
   fragment_corner convex  0.0625
   fragment_corner concave 0.0625
   fragment_max            5.0
```

```
//---------------------- mask rule checks ----------------------//
   mrc_rule external bias.in {
      use 0.005 euclidean
      not_projecting
      use 0.070 euclidean
}

//----------------------------- tagging -----------------------------//
NEWTAG all bias.in -out all_frags
NEWTAG fragment all_frags len < 0.1 corner1 convex corner2 convex \
      len1 > 0.05 len2 > 0.05 -out le_frags
NEWTAG neighbor both le_frags len <= 0.2 corner convex \
      -out le_adj_frags
TAG or le_frags le_adj_frags -out all_le_frags
TAG subtract all_frags le_frags -out full_sizing1
NEWTAG external le_frags full_sizing1 opposite \
      -out tight_le range <= 0.1
NEWTAG neighbor both tight_le len <= 0.2 \
      corner convex -out tight_le_adj_frags
NEWTAG external full_sizing1 le_frags opposite extended 0.0795 \
      -out tight_lin range <= 0.1
NEWTAG neighbor both tight_lin len <= 0.2 corner no_corner \
      -out tight_lin_adj_frags_1
TAG subtract tight_lin_adj_frags_1 tight_lin -out tight_lin_adj_frags
TAG or tight_lin_adj_frags tight_le_adj_frags -out reduced_sizing
TAG subtract full_sizing1 reduced_sizing -out full_sizing
NEWTAG edge all_frags len >= 0.55 -out long_frags
NEWTAG internal all_frags all_frags opposite extended 0.0795 \
      -out narrow_lin range <= 0.1

//----------------------------- BIASRULE -----------------------------//
BIASRULE bias.in \
   -tag full_sizing \
   -table { \
      0.045 0.060 0.075 OPPOSITE EXTENDED 0.0795 OPPOSITE EXTENDED 0.020 :\
0.050 0.003 0.005 0.007 \
0.060 0.003 0.005 0.007 \
      0.061 0.002 0.004 0.006 \0.062 0.001 0.003 0.005 \
0.063 0.000 0.002 0.004 \
0.064 0.000 0.001 0.003 \
      0.065 0.000 0.000 0.002 \
} \

   -interpolate width
   BIASSMOOTH bias.in -minFragLength 0.06 -minJogHeight 0.015
}
   setlayer postbias = nmbias bias.in MAP bias.in OPTIONS first.pass
*/]

//----------------------------- Output -------------------------------
BiasruleLyr   { COPY BiasruleLyr   } DRC CHECK MAP BiasruleLyr   15
AfterBiasrule { COPY AfterBiasrule } DRC CHECK MAP AfterBiasrule 115
```

Chapter 4
OPCBIAS Command

The OPCBIAS command performs edge biasing on target shapes.

OPCBIAS

SVRF Commands

Performs edge biasing on input target shapes.

Usage

OPCBIAS
[*subset_layer*]
in_layer
[*ref_layer*]
{**SPACE *constraint*** [*metric*]
MOVE *value*
[WIDTH *constraint* [*metric*]] [WIDTH2 *constraint* [*metric*]]
[LENGTH1 *constraint*] [LENGTH2 *constraint*]}
| {**SPACELAYER *layer* MOVE *value***}
[CLOSEREDGE [V1 | V2]]
[CLOSERSYMMETRIC]
[GRID *grid_size*]
[IGNORE *jog_length*]
[IMPLICITBIAS]
[MINBIASLENGTH *value*
[*min_corner_frag* | {*min_convex_corner min_concave_corner*}]
[ANGLED
[*angle_value*] [*min_corner_frag* | {*min_convex_corner min_concave_corner*}]]
[OPTION ACUTE_ANGLE_ABUT {NO | YES}]
[OPTION CAREFUL_MEASURE {NO | YES}]
[OPTION CAREFUL_SKEW {NO | YES}]
[OPTION CORNER_FILL {NO | YES}]
[OPTION ENHANCED_SMOOTH {NO | YES}]
[OPTION FAST_CLASSIFY {NO | YES}]
[OPTION IMPLICIT_METRIC {NO | YES}]
[OPTION PROCESSING_MODE {HIERARCHICAL | FLAT}]
[OPTION PROJECTING_ONLY {NO | YES}]
[OPTION SPIKE_CLEANUP {NO | YES}]
[VERSION2 | V2]

Note

The keyword VERSION2 has been added as an alternative to V2, resolving potential layer confusion errors during processing.

Description

The OPCBIAS command biases edges of shapes on the input layer you specify. Edges are selected using various constraints and then biased by various keywords provided by the OPCBIAS command.

Arguments

- *subset_layer*

  An optional subset layer, derived from the in_layer, containing edges intended for correction. The subset_layer can be used when applying OPC to a subset of the input layer edges. The subset_layer must be derived from the input polygon layer. For an example using a subset_layer, refer to "Subset Layers and Biasing" on page 171.

- ***in_layer***

  Specifies a required input layer comprised of polygons on which to apply correction. When a subset_layer is specified, only those edges that coincide with edges on the subset_layer are corrected.

- *ref_layer*

  An optional reference layer comprised of polygon or edge data containing geometries which OPCBIAS uses to measure spacing. The ref_layer is used only for spacing measurement.

  When you specify a ref_layer, spacing is measured between the edge to be corrected and polygons on both the in_layer and on ref_layer. When you do not specify a ref_layer, spacing is measured between the edge to be corrected and other polygons on the in_layer.

  Polygons on the in_layer should not be completely enclosed by polygons on the ref_layer. Typically, the ref_layer is a superset of the in_layer. All edges on the in_layer are coincident with edges on the ref_layer. Since the ref_layer is a superset of the in_layer, the ref_layer usually contains additional shapes that cannot be biased.

- **SPACE *range*** [*metric*] **MOVE *value*** [LENGTH1 *range*] [LENGTH2 *range*] [WIDTH *range* [*metric*]] [WIDTH2 *range* [*metric*]]

  A required keyword and range defining the distance needed between edges for any rule to apply. The SPACE keyword defines a constraining spatial range between edges to qualify and bias target shapes. Either SPACE or SPACELAYER must be used.

  OPCBIAS measures regions of a SPACE or WIDTH range and isolates edges that qualify within these ranges. OPCBIAS performs measurements extending between adjacent shapes for the SPACE range. For the WIDTH range, the measurements are performed within the shape.

  Edges are evaluated relative to the edges of adjacent polygons on either the in_layer or ref_layer. Any portion of an edge that lies within a measurement region containing an adjacent edge meets the SPACE range.

  Edges that fall within the SPACE range undergo biasing if they satisfy the optional WIDTH and LENGTH range. If these optional ranges are not specified, then only SPACE is used as qualifying criteria.

  SPACE *range* [*metric*] — A required keyword and constraint defining distance between edges for which the rule is applied.

  The range is a required range of floating point numbers specified in user units. A range cannot be a strict equality ($a == b$). An unbounded interval is interpreted as its

bounded complement. For example, > *a* is interpreted as everything not in the <= *a* interval. For example, valid ranges include:

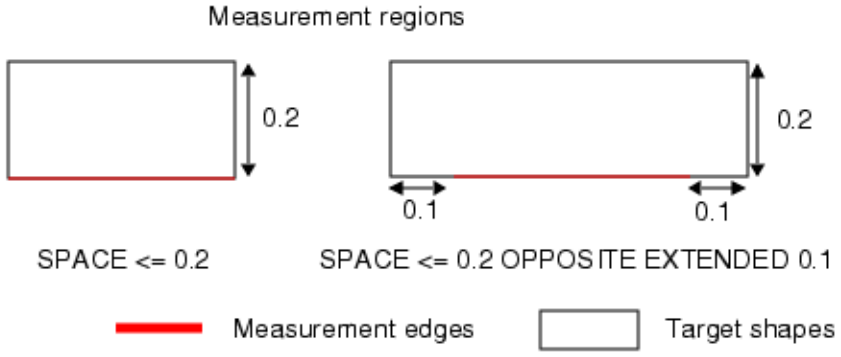
- > a < b
- <= a < b
- > a

The metric is an optional argument that determines how OPCBIAS measures a SPACE or WIDTH range. You may specify one of five metrics with the SPACE keyword:

- EUCLIDEAN — Measures a consistent, radial distance from any point on the edge of the in_layer using the specified constraint. Use the EUCLIDEAN metric to classify angled edges that may be missed by OPPOSITE.
- OPPOSITE SYMMETRIC — Measures within an area formed by perpendicular edge extension from the in_layer. This is the default metric.
- OPPOSITE — Measures facing edge length perpendicular from the in_layer.
- OPPOSITE EXTENDED *value* — Measures within an area created by extending the edge in both directions by the specified value then measuring perpendicular from the extended edge. When using OPPOSITE EXTENDED, you must specify the value by which the edge is extended. Specifying OPPOSITE EXTENDED 0 results in a compiler error.
- OPPOSITE EXTENDED SYMMETRIC *value* — Measures mutual facing edge length.
- OPPOSITE EXTENDED FSYMMETRIC *value* — Measures mutual facing edge length and fills missing edge segments.

Figure 4-1 shows how the measurement region varies depending on the metric used. The measurement region on the left uses the OPPOSITE metric with the SPACE constraint. The measurement region on the right uses the OPPOSITE EXTENDED metric.

Figure 4-1. Measurement Regions for the Two Most Common Metrics



More precision with SPACE or WIDTH constraints for the OPCBIAS operation can be obtained with other metrics. They provide all possible configurations of edges by further qualifying OPPOSITE metrics. For a complete description of advanced metrics, refer to the "Metrics" section of the *Calibre Verification User's Manual*.

Each metric applies only to the constraint directly preceding it. For example, in the rule excerpt below, OPPOSITE EXTENDED applies only to the WIDTH constraint.

```
SPACE > 0.3 <= 0.4 WIDTH <= 0.2 OPPOSITE EXTENDED 0.2
```

You must therefore specify separate metrics for SPACE and WIDTH constraints.

MOVE *value* — Specifies the distance to move biased edges. The value is a required floating-point number in user units. A negative value biases edges of the target shapes inward. A positive value biases edges outward.

WIDTH *range* [*metric*] — An optional keyword that qualifies the width of a shape for rules to apply to its edges. The WIDTH range is specified in microns and determines qualifying widths. Any of the aforementioned SPACE metrics may be used with WIDTH.

The WIDTH keyword measures opposing edges of the same shape on the in_layer or subset_layer. For each opposing edge, a measurement region is evaluated based upon the metric. The portion of an edge that lies within the measurement region of an opposing edge meets the WIDTH constraint. If an edge meets the WIDTH condition, it can be biased.

If WIDTH is not specified, biasing is not qualified by the width of shapes.

WIDTH2 *range* [*metric*] — An optional keyword specifying the width of an adjacent shape to be considered for the rule to apply to an edge of a shape on the in_layer. The WIDTH2 keyword controls which edges are biased according to the widths of opposing shapes. The WIDTH2 range is specified in microns and determines qualifying widths. Any of the aforementioned SPACE metrics may be used with WIDTH2.

The interval range behavior is the same as the SPACE constraint: if edge A projects upon edge B so that the SPACE measurement can be made between A and B, edge B is subject to biasing if edge B satisfies the WIDTH constraint or edge A satisfies the WIDTH2 constraint, given that one is specified.

If WIDTH2 is not specified, biasing is not determined by the width of opposite shapes.

LENGTH1 *range* — An optional keyword and range that controls biasing of edges according to their length. Edges that satisfy the constraint are candidates for biasing. Edges that do not meet this condition are not considered in SPACE or WIDTH measurements. The LENGTH1 range is specified in microns and determines qualifying edge lengths. The range can be bounded or unbounded.

Edges that satisfy the range are subsequently tested against the SPACE and WIDTH conditions. For example, in the case of gates, you can use this condition to ensure that undesired edges on the boundary of gate and field poly are not subject to OPC

biasing. To do this, use a constraint that classifies edges that are greater than gate length. This method filters out the undesired poly edges.

To refine edge classification by length, use CLOSEREDGE. It defines the minimum acceptable length for an edge after the initial edge classification (including those edges that receive a bias of zero). You use MINBIASLENGTH to establish a minimum length to all rules, whereas the LENGTH1 constraint applies only to the rule for which it is supplied.

If LENGTH1 is not specified, biasing is not determined by the length of edges.

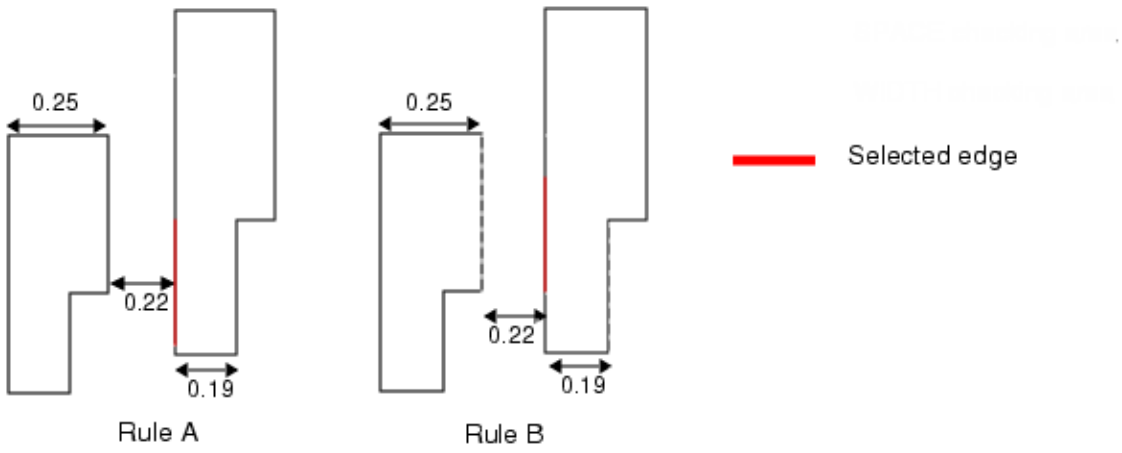
LENGTH2 *range* — An optional keyword and range that controls biasing of edges according to the lengths of opposite edges. The LENGTH2 range is specified in microns and determines qualifying edge lengths. The range can be bounded or unbounded.

The interval range behavior is the same as the SPACE constraint: if edge A projects upon edge B so that the SPACE measurement can be made between A and B, edge B is subject to biasing if edge B satisfies the LENGTH1 constraint or edge A satisfies the LENGTH2 constraint, given that one is specified. If LENGTH2 is not specified, biasing is not determined by the length of opposite edges.

This example shows the critical nature of SPACE argument ordering. Rule A with the OPPOSITE EXTENDED metric is used prior to the WIDTH condition and rule B with the OPPOSITE EXTENDED metric is used after the WIDTH condition. Figure 4-2 shows the results of the different ordering.

```
SPACE <= 0.25 OPPOSITE EXTENDED 0.2 WIDTH <= 0.2 //A after metric
SPACE <= 0.25 WIDTH <= 0.2 OPPOSITE EXTENDED 0.2 //B before metric
```

Figure 4-2. Ordering Impact of Metrics and Constraints

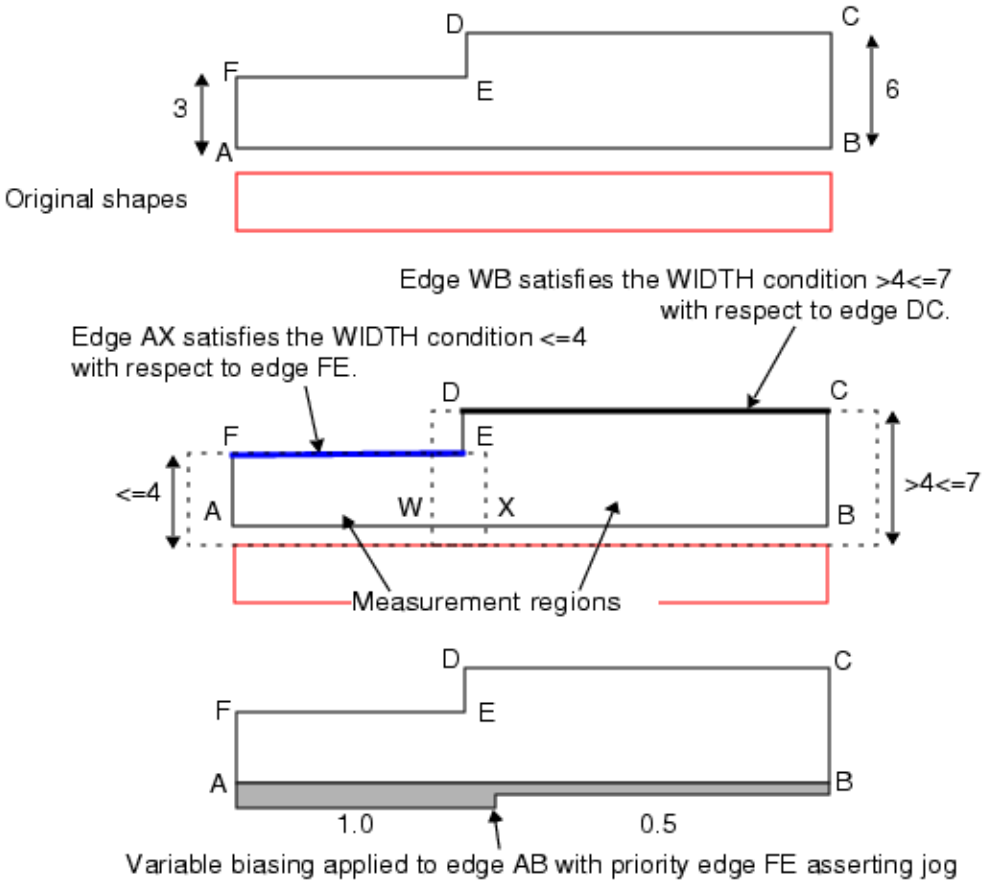


Using OPPOSITE EXTENDED may result in an edge (or portion of an edge) meeting more than one SPACE condition or more than one WIDTH condition. To resolve these conflicts, OPCBIAS prioritizes the edge that meets the closest condition.

Figure 4-3 demonstrates two rules in conflict that are resolved by OPCBIAS:

```
OPCBIAS POLY1
   SPACE <= 3 WIDTH <= 4
   OPPOSITE EXTENDED 1 MOVE 1
   SPACE <= 3 WIDTH > 4 <= 7 OPPOSITE EXTENDED 1 MOVE 0.5
```

Figure 4-3. Variable Biasing and Priority of Closest Edge



The measurement region for FE overlaps the measurement region for DC (overlap shown as shaded). On edge AB, the section between W and X lies in both measurement regions. The overlap, WX, is moved with the rest of AX (based on meeting the condition for the closer edge). Edge XB is moved based on meeting the condition for edge DC.

- **SPACELAYER *layer* MOVE *value*** — A required keyword and layer defining the bias for each layer ordered by priority.

  Since SPACELAYER rules have no edge-to-edge spacing values specified, the priority of the bias is determined by the order in which the rules are listed. Either SPACE or SPACELAYER must be used.

  SPACELAYER *layer* — A required keyword and layer that specifies the layer of which to bias edges. The layer input to SPACELAYER must be an edge.

MOVE *value* — Specifies the distance to move biased edges. The value is a required floating-point number in user units. A negative value biases edges of the target shapes inward. A positive value biases edges outward.

- CLOSEREDGE {V1 | V2}

  An optional keyword that prioritizes biasing by finding the closest edge. Edges receive treatment based on their priority and the lengths of neighboring edges as described in "Calculation of Treatment for Edges Selected by Priority" on page 165. CLOSEREDGE V2 mode supports a smoothing algorithm that handles flat designs, rotated data, and asymmetric data more efficiently. This is the default mode. The V1 mode specifies the original CLOSEREDGE behavior.

  If an edge is shorter than MINBIASLENGTH, the operation generally widens that edge into the fragment whose bias is closest to that short fragment. See "Treatment of Short Edges Between Two Long Edges" on page 165 for more information.

  Note

  CLOSEREDGE V2 is used if CLOSERSYMMETRIC is not specified.

- CLOSERSYMMETRIC — An optional keyword that enforces biasing symmetry (overriding CLOSEREDGE) under these constraints:
  - The edge is shorter than MINBIASLENGTH.
  - The adjacent fragments are longer than MINBIASLENGTH.
  - The short fragment has the higher priority.

  If not specified, symmetric widening only applies when adjacent edges have the same bias.

  Note

  To use CLOSERSYMMETRIC, the V1 processing mode must be enabled. The V1 processing mode is not the same as the V1 option for CLOSEREDGE.

  Symmetric widening involves extending the short fragment by an equal amount in each direction, which shortens the adjacent fragments by equal amounts. This is the default behavior when the adjacent fragments have the same bias. This behavior is performed even when the fragments on either side of the short fragment have different biases. To ensure proper biasing, use CLOSERSYMMETRIC only with reference layers that are a superset of the target layer. See "Treatment of Short Edges Between Two Long Edges" on page 165 for more information.

  Tip

  Siemens EDA recommends using CLOSEREDGE V2 instead of CLOSERSYMMETRIC. The smoothing with CLOSEREDGE V2 is typically closer to results obtained with flat processing.

- GRID *grid_size* — An optional keyword and integer value that specifies a reference grid when biasing 45-degree edges. The GRID keyword adjusts the bias of 45-degree edges so that they are on the specified grid.

  The *grid_size* must be a positive integer and must be identical to the grid specified by any SNAP command in the SVRF file.

  No grid adjustment is performed by default.

- IGNORE *jog_length* — An optional keyword used to skip biasing of edges shorter than the specified value. If the length of an edge is less than or equal to jog_length, then the edge is not biased and does not affect biasing of other edges.

  The jog_length is a positive floating-point number in microns specifying the maximum length of an edge to not bias. Edge lengths greater than jog_length are eligible for biasing.

  This keyword is only available when using the VERSION2 processing mode.

  Note

  To use the IGNORE keyword, you must use the VERSION2 processing mode. This is not the same function as the V2 argument of CLOSEREDGE.

- IMPLICITBIAS

  An optional keyword that controls priority of unbiased edges. When IMPLICITBIAS is specified, unbiased edges have the same priority as if bias rules for these edges were specified explicitly.By default, IMPLICITBIAS is off. When no rule for classification of an edge is specified, the edge remains unmoved. Additionally, these edges have the lowest priority during the cleanup step for enforcing MINBIASLENGTH.

- MINBIASLENGTH *value* [*frag_corner* | {*convex_corner concave_corner*}] [ANGLED [*angled_value*] [*min_corner_frag* | {*min_convex_corner min_concave_corner*}]]

  An optional keyword specifying the minimum length of edges after biasing in microns. You can optionally specify the minimum length of fragments at corners (separately for concave and convex corners if needed) and the minimum lengths for fragments at angled edges (separately for concave, convex, or both fragments, if needed).

  If possible, OPCBIAS merges edges with their neighbors before biasing.

  MINBIASLENGTH *value* — A positive floating-point number that specifies the minimum length for an edge after biasing. If not specified, any length edge is allowed.

  frag_corner — An optional floating-point number that specifies the minimum length of fragments at corners. You cannot specify both frag_corner and {convex_corner concave_corner}.

  {*convex_corner concave_corner*} — Two optional floating-point numbers that specify the minimum length of fragments at convex corners and concave corners, respectively.

  ANGLED *angled_value* — An optional keyword and floating-point number that specifies a separate minimum length for angled edges after biasing. When ANGLED

is specified, MINBIASLENGTH applies only to orthogonal edges. The default for angled_value is the value for MINBIASLENGTH.

angled_frag_corner — An optional floating-point number that, for angled edges, specifies the minimum length of fragments at corners. You cannot specify both angled_frag_corner and {angled_convex_corner angled_concave_corner}.

{*angled_convex_corner angled_concave_corner*} — Two optional floating-point numbers that specify the minimum length of fragments for angled edges at convex and concave corners, respectively.

In simplest form, MINBIASLENGTH specifies the minimum length for orthogonal edges after biasing:

```
MINBIASLENGTH 0.200
```

Using two arguments, the first (0.200) specifies the minimum acceptable length for horizontal and vertical edges after biasing; the second (0.210) specifies the minimum length of fragments at convex and concave corners:

```
MINBIASLENGTH 0.200 0.210
```

Because a third argument is specified in this next example, the second argument (0.210) now specifies the minimum length of fragments at convex corners only; the third argument (0.220) specifies the minimum length of fragments at concave corners:

```
MINBIASLENGTH 0.200 0.210 0.220
```

Including the ANGLED keyword allows you to specify an argument for the minimum length of angled edges after biasing (0.230); the second argument for the minimum length of angled fragments at convex corners (0.240); the third argument for the minimum length of angled fragments at concave corners (0.250):

```
MINBIASLENGTH 0.200 0.210 0.220 ANGLED 0.230 0.240 0.250
```

- OPTION ACUTE_ANGLE_ABUT {NO | YES} — An optional keyword that enables biasing of edges of abutting acute angles. Without enabling OPTION ACUTE_ANGLE_ABUT, no acute angles are biased.

  YES enables bias to acute angled edges. NO disables bias to acute angled edges. NO is the default argument.

- OPTION CAREFUL_MEASURE {NO | YES} — An optional keyword that reduces runtime when large values of the OPPOSITE EXTENDED metric are used. Siemens EDA recommends using this keyword if values of the OPPOSITE EXTENDED metric exceed ten times the minimum feature size.

  YES enables runtime reduction when large values of OPPOSITE EXTENDED are specified. NO disables runtime reduction when large values of OPPOSITE EXTENDED are specified. NO is the default argument.

- OPTION CAREFUL_SKEW {NO | YES}

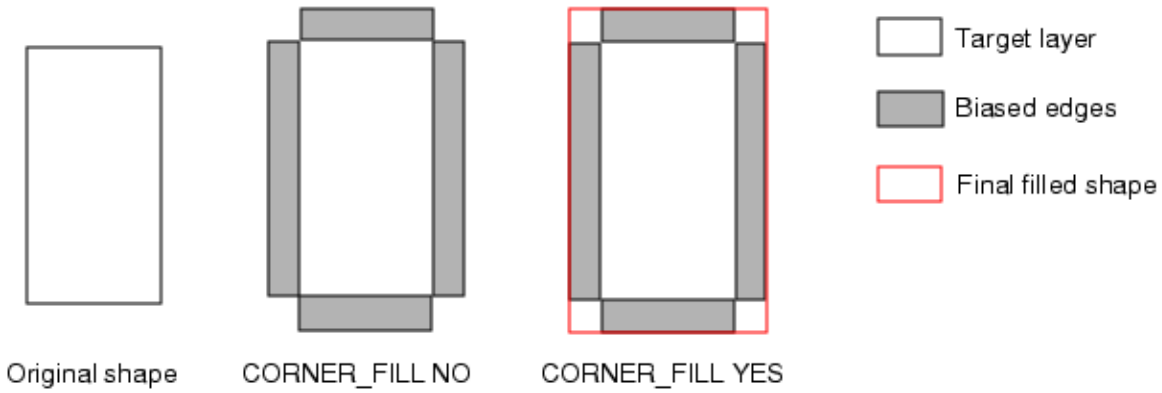
  An optional keyword that implements an algorithm that adds fill to gaps generated by biasing of skewed edges. For more information on skewed edges, see “Snap 45 Degree

Angles (Post-Bias)" on page 162.YES enables addition of fill to gaps. NO disables addition of fill to gaps. NO is the default argument.

- OPTION CORNER_FILL {NO | YES}

  An optional keyword that controls corner filling of shapes after biasing.YES enables addition of fill to corners. NO Disables addition of fill to corners. YES is the default argument. Figure 4-4 illustrates the use of the CORNER_FILL keyword.

Figure 4-4. CORNER_FILL



- OPTION ENHANCED_SMOOTH {NO | YES} — An optional keyword that improves matching of MINBIASLENGTH enforcement between flat and hierarchical output.

  YES enables improved flat and hierarchical matching. NO disables improved flat and hierarchical matching. YES is the default argument.

  Note
  OPTION ENHANCED_SMOOTH requires VERSION2 processing mode to be enabled.

- OPTION FAST_CLASSIFY {NO | YES} — An optional keyword that enables an alternate edge classification algorithm to reduce run time for flat runs only (specifically useful for section-based processing) or for large rule tables (those consisting of hundreds of rules). Enabling this option does not change the output of OPCBIAS. This option is enabled by the VERSION2 processing mode and is not compatible with rules using LENGTH1, LENGTH2, or WIDTH2.

  YES enables edge classification for flat runs. NO disables edge classification. NO is the default argument.

  Note
  OPTION FAST_CLASSIFY requires VERSION2 processing mode to be enabled.

- OPTION IMPLICIT_METRIC {NO | YES} — An optional keyword that inherits the metric of interpolated rules from a neighboring rule's metric.

  YES enables inheriting the interpolated rule metric from a neighbor rule. NO disables inheriting the interpolated rule metric from a neighbor and instead applies the OPPOSITE metric. YES is the default argument.

  Note

  The biasing values for MOVE are always set to 0.000 for implicit rules and no biasing is performed.

  Given this original rule set:

```
SPACE > 0.043 <= 0.044 OPPOSITE EXTENDED 0.060 WIDTH > 0.0 < 0.044 \
   OPPOSITE EXTENDED 0.120 MOVE 0.0001

SPACE > 0.045 <  0.046 OPPOSITE EXTENDED 0.060 WIDTH > 0.0 < 0.044 \
   OPPOSITE EXTENDED 0.120 MOVE 0.0005
```

  OPTION IMPLICIT_METRIC NO (implicit metric in red):

```
SPACE > 0.043 <= 0.044 OPPOSITE EXTENDED 0.060 WIDTH > 0.0 < 0.044 \
   OPPOSITE EXTENDED 0.120 MOVE 0.0001

SPACE > 0.044 <= 0.045 OPPOSITE          0.060 WIDTH > 0.0 < 0.044 \
   OPPOSITE          0.120 MOVE 0.0000

SPACE > 0.045 <  0.046 OPPOSITE EXTENDED 0.060 WIDTH > 0.0 < 0.044 \
   OPPOSITE EXTENDED 0.120 MOVE 0.0005
```

  OPTION IMPLICIT_METRIC YES (implicit metric in red):

```
SPACE > 0.043 <= 0.044 OPPOSITE EXTENDED 0.060 WIDTH > 0.0 < 0.044 \
   OPPOSITE EXTENDED 0.120 MOVE 0.0001

SPACE > 0.044 <= 0.045 OPPOSITE EXTENDED 0.060 WIDTH > 0.0 < 0.044 \
   OPPOSITE EXTENDED 0.120 MOVE 0.0000

SPACE > 0.045 <  0.046 OPPOSITE EXTENDED 0.060 WIDTH > 0.0 < 0.044 \
   OPPOSITE EXTENDED 0.120 MOVE 0.0005
```

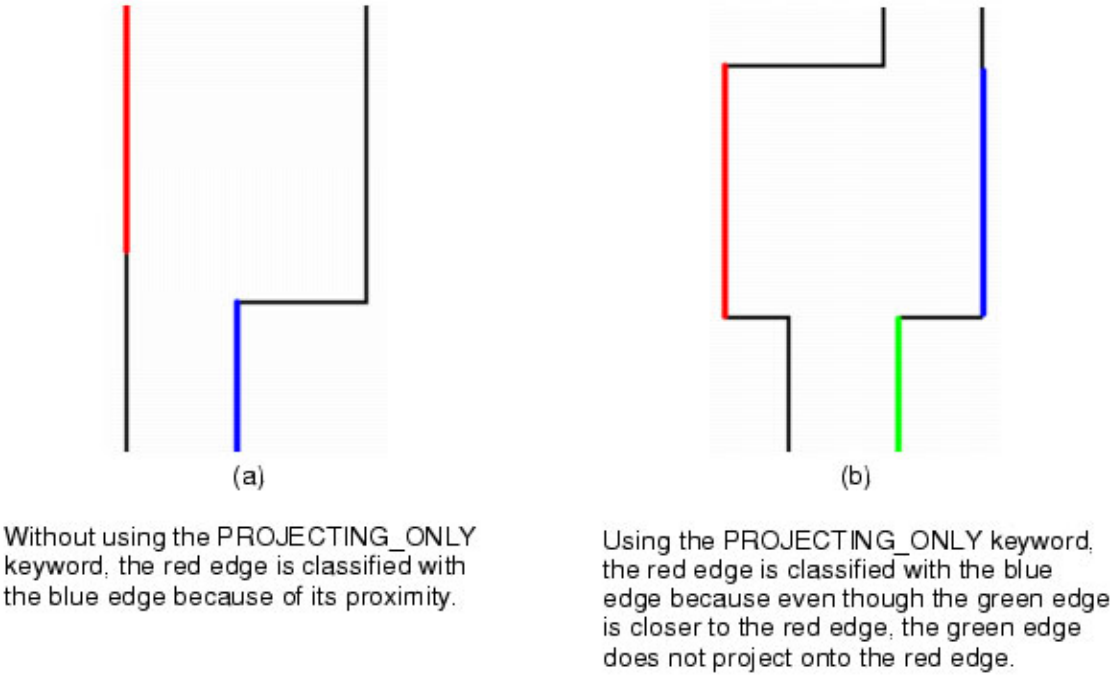
- OPTION PROCESSING_MODE {FLAT | HIERARCHICAL} — An optional keyword that determines the processing mode for OPCBIAS VERSION2.

  HIERARCHICAL processing mode is the default. FLAT processing mode automatically invokes ULTRAflex.

- OPTION PROJECTING_ONLY {NO | YES} — An optional keyword that is used with the OPPOSITE EXTENDED metric within the SPACE command to prioritize edges having projection instead of a closer edge that is not projected. This is performed prior to biasing.

  NO disables classification of projecting edges. YES enables classification of projecting edges. NO is the default argument. Figure 4-5 illustrates the use of the PROJECTING_ONLY keyword.

Figure 4-5. PROJECTING_ONLY Used with OPPOSITE EXTENDED



- OPTION SPIKE_CLEANUP {NO | YES}

  An optional keyword that repairs skewed edges after biasing by snapping the skewed edges (those angled edges that are non-45 degrees) to 45 degrees. This keyword is only available when using the VERSION2 processing mode. For more information on skewed edges, see "Snap 45 Degree Angles (Post-Bias)" on page 162.

  NO disables skew edge cleanup. YES enables skew edge cleanup. YES is the default argument.

  Note

  OPTION SPIKE_CLEANUP requires VERSION2 processing mode to be enabled.

- VERSION2 | V2

  An optional keyword that enables an updated version of hierarchical processing mode. V1 is the default processing mode. VERSION2 (or alternatively, V2) provides better scalability than V1 and uses the same streamlined hierarchical processing shared by Calibre® nmOPC™ and Calibre® OPCverify™. VERSION2 must be enabled to use IGNORE, OPTION FAST_CLASSIFY, or OPTION SPIKE_CLEANUP.

  In addition, VERSION2 enables a new transcript format.

Note

The keyword VERSION2 has been added as an alternative to V2, resolving potential layer confusion errors during processing.

OPCBIAS Command

The OPCBIAS command moves individual target shape edges by predefined amounts depending on target widths and spaces between target shapes. You can also define edge constraints by width and space. Most tables specify corrections as a function of these two ranges.

You define the biasing for edges based on geometric relationships defined by SPACE or SPACELAYER statements and their arguments. The specifications set up table-driven OPC corrections. To ensure proper correction, each statement must be specified with a unique constraint.

Table 4-1 shows a table defining biasing applied based on line width and spacing ranges.

Table 4-1. Sample Table of Rules

Spacing \ Line Width	<=0.1	>0.1<=0.2	>0.2<=0.3	>0.3<=0.4	>0.4<=0.5	>0.5
0.0	0	0	0	0	0	0
>0<=0.1	0	0	0	0	-0.10	-0.15
>0.1<=0.2	0	0	0	-0.05	-0.10	-0.10
>0.2<=0.3	0	0	0	-0.05	-0.05	-0.10
>0.3<=0.4	0	0.05	0	0	-0.05	-0.05
>0.4<=0.5	0	0.10	0.05	0.05	0	0
>0.5<=0.6	0	0.10	0.05	0.05	0	0.05
>0.6	0	0.10	0.10	0.05	0	0.05

Since each cell in an edge bias table represents a space, width, and associated movement, you create one rule for each cell in the table. The syntax for OPCBIAS consists of keyword-constraint pairs:

```
SPACE <range> WIDTH <range> MOVE <distance>
```

The constraints for SPACE and WIDTH are ranges, not discrete values. In many cases the range limits are implicit:

```
SPACE <= 0.150 WIDTH <= 0.150 MOVE 0
```

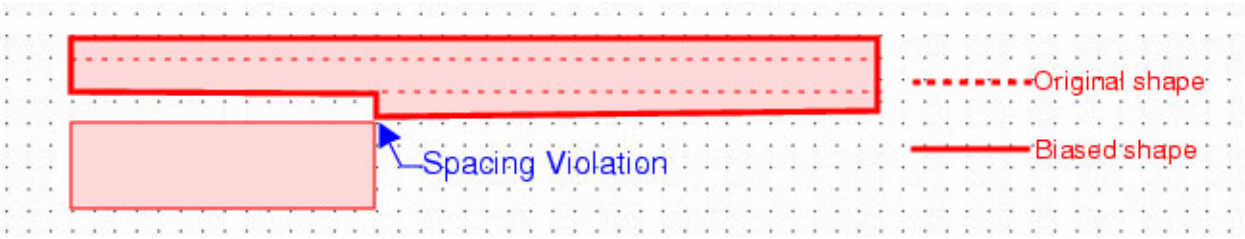
Note

OPCBIAS default behavior pushes output results down the hierarchy at process completion. This often improves the performance of subsequent SVRF operations. To disable this behavior, set the environment variable OPCBIAS_DISABLE_AUTOPUSH_V1 to 1 in the shell you run OPCBIAS from.

Spacing Offset Definition

Biasing exactly those portions of an edge that meet the SPACE constraint can result in spacing violations where an edge moves closer to an adjacent polygon.

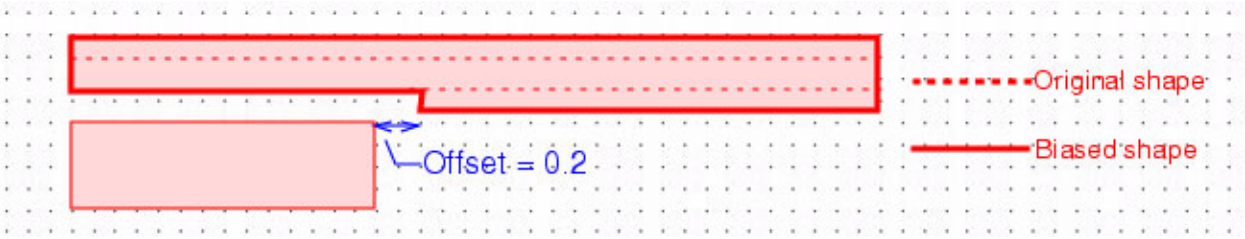
Figure 4-6. Biases Resulting in Spacing Violations



Use the Opposite Extended metric with SPACE to define an offset relative to change in the proximity environment. When the OPCBIAS operation breaks an edge into two or more edges, each meeting different spacing conditions, this constraint causes the breakpoint to be offset from the point at which the spacing changes. Because of the closer-edge-wins principle, the offset is added to the edge that meets the smallest spacing constraint. The following code sample and figure illustrate this usage.

```
SPACE <= 0.25 OPPOSITE EXTENDED 0.2 WIDTH <= 0.15 MOVE 0
SPACE >  0.25 OPPOSITE EXTENDED 0.2 WIDTH <= 0.15 MOVE 0.1
```

Figure 4-7. Using OPPOSITE EXTENDED to Avoid Violations



Width Offset Definition

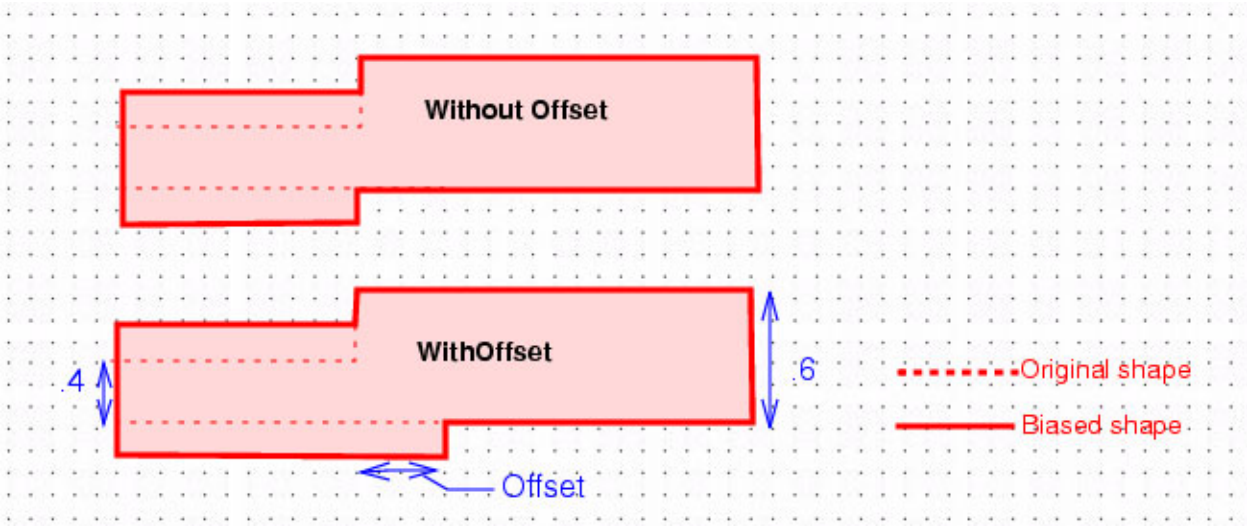
Biasing exactly those portions of an edge that meet the WIDTH constraint can result in polygons with irregular shapes.

Use the Opposite Extended metric with WIDTH to define an offset relative to change in width. When the OPCBIAS operation breaks an edge into two or more edges, each meeting different width conditions, this constraint causes the breakpoint to be offset from the point at which the

width changes. Because of the closer-edge-wins principle, the offset is added to the edge that meets the narrower width constraint.

```
SPACE > 0.15 WIDTH <= 0.4 OPPOSITE EXTENDED 0.3 MOVE 0.15
SPACE > 0.15 WIDTH >  0.4 OPPOSITE EXTENDED 0.3 MOVE 0
```

Figure 4-8. WIDTH Offset Definition With OPPOSITE EXTENDED



Use the Opposite Extended metric when you have an L-shaped polygon and want to ensure that biasing extends to the corner. In this situation, you must set the offset” equal to the maximum line width that could be encountered.

```
SPACE > 0.15 WIDTH <= 0.4 OPPOSITE EXTENDED 0.5 MOVE 0.15
SPACE > 0.15 WIDTH >  0.4 OPPOSITE EXTENDED 0.5 MOVE 0
```

Figure 4-9. Bias Extension to Corners With OPPOSITE EXTENDED

Length Condition Definition

The OPCBIAS operation provides two methods for controlling the lengths of edges after biasing:

- Within rules, you place constraints on the lengths of edges subject to biasing on a rule by rule basis using LENGTH1. Only those edges that satisfy this condition are corrected.

  For example, in the case of gates, you use this option to ensure that false edges on the boundary of gate and field POLY are not subject to OPC biasing. To do this, use a constraint that is greater than the channel length (channel length is equivalent to the gate width). When LENGTH1 is specified, this constraint is applied before edges are classified using the WIDTH and SPACE constraints.

- For the entire operation, you can define the minimum acceptable length for an edge after biasing (including edges with a bias of zero) using CLOSEREDGE. If needed, you can define two separate minbiaslength values, one for angled edges and one for orthogonal edges.

  MINBIASLENGTH applies to all rules processed by the operation whereas the LENGTH1 constraint applies only to the rule for which it is supplied. The OPCBIAS operation enforces compliance with MINBIASLENGTH after it fragments the edges but before it applies the biases.

Condition Validation

The OPCBIAS operation uses the SPACE, WIDTH, WIDTH2, LENGTH1, and LENGTH2 conditions to calculate the biasing to apply to each edge. To ensure proper correction, the rules must be mutually exclusive; every rule must differ from all other rules in at least one constraint.

In addition, SPACE, WIDTH, WIDTH2, LENGTH1, and LENGTH2 conditions must be written in terms of a consistent partitioning of spaces, widths, and lengths.

- SPACE constraints partition external physical space relative to an edge (the distance between objects).
- WIDTH constraints organize edges relative to the width of the polygon directly inside an edge.
- WIDTH2 constraints organize edges relative to the width of the polygon directly inside opposing edges.
- LENGTH1 constraints organize edges relative to length.
- LENGTH2 constraints organize opposing edges relative to length.

Within any set of rules, the partitioning must be consistent. To understand this concept, consider Table 4-2. The table itself is valid for edges of a specific length. The rows partition external physical space. The columns partition internal space. Cell values in the table represent the biasing to apply to any object that meets the constraints for that row and that column. All rules written to represent this table define treatment based on the same partitioning of space.

Table 4-2. Original Rules Table

Spacing \ Line Width	<=0.1	>0.1<=0.2	>0.2<=0.3	>0.3<=0.4	>0.4<=0.5	>0.5
0.0	0	0	0	0	0	0
>0<=0.1	0	0	0	0	-0.10	-0.15
>0.1<=0.2	0	0	0	-0.05	-0.10	-0.10
>0.2<=0.3	0	0	0	-0.05	-0.05	-0.10
>0.3<=0.4	0	0.05	0	0	-0.05	-0.05
>0.4	0	0.10	0.05	0.05	0	0

For a literal translation of the rules table into OPCBIAS rules, you would write one rule for each cell in the table. However, the OPCBIAS operation enables writing single rules with constraints spanning multiple partitions, which is analogous to merging adjacent cells in a rules table. For example, in Table 4-3, the four highlighted cells contain the same bias value. The single rule shown below represents all four cells:

```
SPACE >0.3 WIDTH>0.2<=0.4 MOVE = 0.05
```

Table 4-3. Translating Rules Tables

Spacing \ Line Width	<=0.1	>0.1<=0.2	>0.2<=0.3	>0.3<=0.4	>0.4<=0.5	>0.5
0.0	0	0	0	0	0	0
>0<=0.1	0	0	0	0	-0.10	-0.15
>0.1<=0.2	0	0	0	-0.05	-0.10	-0.10
>0.2<=0.3	0	0	0	-0.05	-0.05	-0.10
>0.3<=0.4	0	0.05	0.05	0.05	-0.05	-0.05
>0.4	0	0.10	0.05	0.05	0	0

The OPCBIAS operation does not allow you to split a partition one way for one rule and another way for another rule. For example, in the following two rules, the width partitions are split differently and cannot be used within the same OPCBIAS operation.

```
SPACE > 0   <= 0.1 OPPOSITE EXTENDED 0.1 WIDTH > 0.3 <  0.4 MOVE  0
SPACE > 0.1 <= 0.2 OPPOSITE EXTENDED 0   WIDTH > 0.3 <= 0.4 MOVE -0.05
```

Because the metric you use with either the SPACE or the WIDTH constraint plays a part in the partitioning, you must also make sure that the metric you use is consistent for the entire

partition. For example, if you use one extension value for a particular WIDTH or SPACE constraint, you must use the same extension value for all other occurrences of that constraint. The following two rules violate this rule and cannot be used within the same OPCBIAS operation:

```
SPACE > 0   <= 0.1 WIDTH > 0.3 <= 0.4 OPPOSITE EXTENDED 0.1 MOVE  0
SPACE > 0.1 <= 0.2 WIDTH > 0.3 <= 0.4 OPPOSITE EXTENDED 0.2 MOVE -0.05
```

Conversion of Table Values to OPCBIAS Rules

Before you convert a table of rules into OPCBIAS syntax, understand how to interpret the table data. Be sure you know how to translate the discrete table values for width and spacing into ranges of values and whether the rule reflects one- or two-edge movement.

Prerequisites

- Existing table of rules

Procedure

1. Convert table values into user units (in this case, microns). The initial table of rules for biasing contains indexes:

Spacing \ Line Width	1	2	3	4	5	6
0	0	0	0	0	0	0
1	0	0	0	0	-1.0	-1.5
2	0	0	0	-0.5	-1.0	-1.0
3	0	0	0	-0.5	-0.5	-1.0
4	0	0.5	0	0	-0.5	-0.5
5	0	1.0	0.5	0.5	0	0
6	0	1.0	0.5	0.5	0	0.5
7	0	1.0	1.0	0.5	0	0.5

   The table now contains usable values:

Spacing \ Line Width	0.1	0.2	0.3	0.4	0.5	0.6
0	0	0	0	0	0	0
0.1	0	0	0	0	-0.10	-0.15
0.2	0	0	0	-0.05	-0.10	-0.15
0.3	0	0	0	-0.05	-0.05	-0.10
0.4	0	0.05	0	0	-0.05	-0.05
0.5	0	0.10	0.05	0.05	0	0
0.6	0	0.10	0.05	0.05	0	0.05
0.7	0	0.10	0.10	0.05	0	0.05

2. Convert spacing and width constraints into ranges of values:

Spacing \ Line Width	<=0.1	>0.1<=0.2	>0.2<=0.3	>0.3<=0.4	>0.4<=0.5	>0.5
0	0	0	0	0	0	0
>0<=0.1	0	0	0	0	-0.10	-0.15
>0.1<=0.2	0	0	0	-0.05	-0.10	-0.15
>0.2<=0.3	0	0	0	-0.05	-0.05	-0.10
>0.3<=0.4	0	0.05	0	0	-0.05	-0.05
>0.4<=0.5	0	0.10	0.05	0.05	0	0
>0.5<=0.6	0	0.10	0.05	0.05	0	0.05
>0.6	0	0.10	0.10	0.05	0	0.05

3. Create rectangular groupings of adjacent cells having the same bias amount. The resulting table shows that groupings must be rectangular to ensure that all the cells in the grouping can be described by only one WIDTH and one SPACE constraint:

Spacing \ Line Width	<=0.1	>0.1<=0.2	>0.2<=0.3	>0.3<=0.4	>0.4<=0.5	>0.5
0	0	0	0	0	0	0
>0<=0.1	0	0	0	0	-0.10	-0.15
>0.1<=0.2	0	0	0	-0.05	-0.10	-0.10
>0.2<=0.3	0	0	0	-0.05	-0.05	-0.10
>0.3<=0.4	0	0.05	0	0	-0.05	-0.05
>0.4<=0.5	0	0.10	0.05	0.05	0	0
>0.5<=0.6	0	0.10	0.05	0.05	0	0.05
>0.6	0	0.10	0.10	0.05	0	0.05

4. For each table group:
   a. Define the SPACE constraint to reflect the space constraints for the group.
   b. Define the WIDTH constraints to reflect the width constraints for the group
   c. Define the MOVE distance for the value of the group.
5. For each ungrouped cell within the table:
   a. Define the SPACE constraint to reflect the space constraints for the cell.
   b. Define the WIDTH constraints to reflect the width constraints for the cell.

c. Define the MOVE distance to be the value in the cell.

These are the rules for the grouped table:

```
SPACE > 0   <= 0.2 WIDTH > 0.4 <= 0.5 MOVE -0.10
SPACE > 0.1 <= 0.3 WIDTH > 0.5        MOVE -0.10
SPACE > 0.2 <= 0.3 WIDTH > 0.3 <= 0.5 MOVE -0.05
SPACE > 0.3 <= 0.4 WIDTH > 0.4        MOVE -0.05
SPACE > 0.4 <= 0.6 WIDTH > 0.1 <= 0.2 MOVE  0.10
SPACE > 0.4 <= 0.6 WIDTH > 0.2 <= 0.3 MOVE  0.05
SPACE > 0.5        WIDTH > 0.5        MOVE  0.05
SPACE > 0.6        WIDTH > 0.1 <= 0.3 MOVE  0.10
SPACE > 0   <= 0.1 WIDTH > 0.5        MOVE -0.15
SPACE > 0.3 <= 0.4 WIDTH > 0.1 <= 0.2 MOVE  0.05
SPACE > 0.6        WIDTH > 0.3 <= 0.4 MOVE  0.05
```

The preceding rules contain only the specified table information. In most cases, rules must reflect process constraints in the table.

6. To bias an outer edge to extend to the corner of an L-shaped line, calculate the width offset as maximum line width:

```
WIDTH OPPOSITE EXTENDED 0.30
```

These rules include biasing to the corner extent:

```
SPACE >0  <=0.2 WIDTH >0.4<=0.5 OPPOSITE EXTENDED 0.30 MOVE -0.10
SPACE >0.1<=0.3 WIDTH >0.5      OPPOSITE EXTENDED 0.30 MOVE -0.10
SPACE >0.2<=0.3 WIDTH >0.3<=0.5 OPPOSITE EXTENDED 0.30 MOVE -0.05
SPACE >0.3<=0.4 WIDTH >0.4      OPPOSITE EXTENDED 0.30 MOVE -0.05
SPACE >0.4<=0.6 WIDTH >0.1<=0.2 OPPOSITE EXTENDED 0.30 MOVE  0.10
SPACE >0.4<=0.6 WIDTH >0.2<=0.3 OPPOSITE EXTENDED 0.30 MOVE  0.05
SPACE >0.5      WIDTH >0.5      OPPOSITE EXTENDED 0.30 MOVE  0.05
SPACE >0.6      WIDTH >0.1<=0.3 OPPOSITE EXTENDED 0.30 MOVE  0.10
SPACE >0  <=0.1 WIDTH >0.5      OPPOSITE EXTENDED 0.30 MOVE -0.15
SPACE >0.3<=0.4 WIDTH >0.1<=0.2 OPPOSITE EXTENDED 0.30 MOVE  0.05
SPACE >0.6      WIDTH >0.3<=0.4 OPPOSITE EXTENDED 0.30 MOVE  0.05
```

7. If you want biasing to be offset relative to changes in proximity, calculate the spacing offset based on design specs:

```
SPACE OPPOSITE EXTENDED 0.1
```

These rules include space-based biasing:

```
SPACE >0<=0.2
   OPPOSITE EXTENDED 0.1 WIDTH >0.4<=0.5
   OPPOSITE EXTENDED 0.30 MOVE -0.10 SPACE >0.1<=0.3
   OPPOSITE EXTENDED 0.1 WIDTH >0.5
   OPPOSITE EXTENDED 0.30 MOVE -0.10
SPACE >0.2<=0.3
   OPPOSITE EXTENDED 0.1 WIDTH >0.3<=0.5
   OPPOSITE EXTENDED 0.30 MOVE -0.05
SPACE >0.3<=0.4
   OPPOSITE EXTENDED 0.1 WIDTH >0.4
   OPPOSITE EXTENDED 0.30 MOVE -0.05
SPACE >0.4<=0.6
   OPPOSITE EXTENDED 0.1 WIDTH >0.1<=0.2
   OPPOSITE EXTENDED 0.30 MOVE 0.10
SPACE >0.4<=0.6
   OPPOSITE EXTENDED 0.1 WIDTH >0.2<=0.3
   OPPOSITE EXTENDED 0.30 MOVE 0.05

SPACE >0.5
   OPPOSITE EXTENDED 0.1 WIDTH >0.5
   OPPOSITE EXTENDED 0.30 MOVE 0.05
SPACE >0.6
   OPPOSITE EXTENDED 0.1 WIDTH >0.1<=0.3
   OPPOSITE EXTENDED 0.30 MOVE 0.10
SPACE >0<=0.1
   OPPOSITE EXTENDED 0.1 WIDTH >0.5
   OPPOSITE EXTENDED 0.30 MOVE -0.15
SPACE >0.3<=0.4
   OPPOSITE EXTENDED 0.1 WIDTH >0.1<=0.2
   OPPOSITE EXTENDED 0.30 MOVE 0.05
SPACE >0.6
   OPPOSITE EXTENDED 0.1 WIDTH >0.3<=0.4
   OPPOSITE EXTENDED 0.30 MOVE 0.05
```

8. Add the OPCBIAS operation just before the first rule. Use the MINBIASLENGTH option to specify the smallest correction to keep:

```
output_layer = OPCBIAS m1 MINBIASLENGTH <0.25
   SPACE >0<=0.2
      OPPOSITE EXTENDED 0.1 WIDTH >0.4<=0.5
      OPPOSITE EXTENDED 0.30 MOVE -0.10
   SPACE >0.1<=0.3
      OPPOSITE EXTENDED 0.1 WIDTH >0.5
      OPPOSITE EXTENDED 0.30 MOVE -0.10
   SPACE >0.2<=0.3
      OPPOSITE EXTENDED 0.1 WIDTH >0.3<=0.5
      OPPOSITE EXTENDED 0.30 MOVE -0.05
   SPACE >0.3<=0.4
      OPPOSITE EXTENDED 0.1 WIDTH >0.4
      OPPOSITE EXTENDED 0.30 MOVE -0.05
   SPACE >0.4<=0.6
      OPPOSITE EXTENDED 0.1 WIDTH >0.1<=0.2
      OPPOSITE EXTENDED 0.30 MOVE  0.10
```

```
SPACE >0.4<=0.6
   OPPOSITE EXTENDED 0.1 WIDTH >0.2<=0.3
   OPPOSITE EXTENDED 0.30 MOVE  0.05
SPACE >0.5
   OPPOSITE EXTENDED 0.1 WIDTH >0.5
   OPPOSITE EXTENDED 0.30 MOVE  0.05
SPACE >0.6
   OPPOSITE EXTENDED 0.1 WIDTH >0.1<=0.3
   OPPOSITE EXTENDED 0.30 MOVE  0.10
SPACE >0  <=0.1
   OPPOSITE EXTENDED 0.1 WIDTH >0.5
   OPPOSITE EXTENDED 0.30 MOVE -0.15
SPACE >0.3<=0.4
   OPPOSITE EXTENDED 0.1 WIDTH >0.1<=0.2
   OPPOSITE EXTENDED 0.30 MOVE  0.05
SPACE >0.6
   OPPOSITE EXTENDED 0.1 WIDTH >0.3<=0.4
   OPPOSITE EXTENDED 0.30 MOVE  0.05
```

OPCBIAS Cleanup Tasks

Strict interpretation of rules by OPCBIAS during biasing may lead to shapes that are not manufacturable, requiring OPCBIAS to perform clean up tasks.

Short Edge Handling (Pre-Bias)

The MINBIASLENGTH defines the minimum acceptable length for an edge after biasing. To avoid creating edges shorter than MINBIASLENGTH, the OPCBIAS operation attempts to combine the short edge with a neighboring edge using one of the following techniques:

- **Merging —** Combining a short edge with a neighboring edge, resulting in one long edge.
- **Widening —** Sliding the vertex point between the two edges until both edges are longer than MINBIASLENGTH.

The short edge handling process occurs after the OPCBIAS operation fragments the edges but before it applies the biases. Figure 4-10 shows a short edge in green with a neighboring edge in red. The fact that they are separate edges indicates that the OPCBIAS operation has calculated that they require different biases.

Figure 4-10. Short Edge Handling Treatments

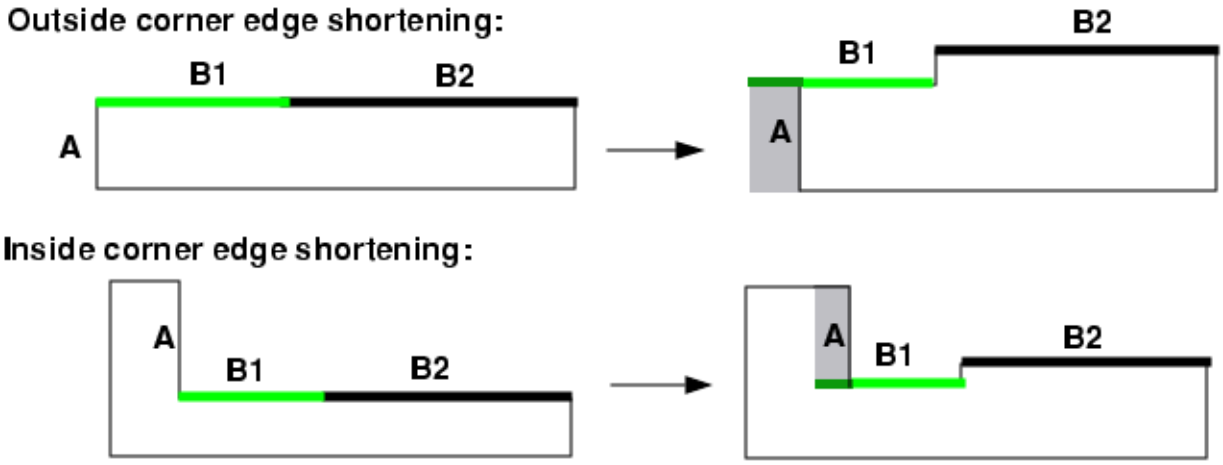
Note

When an edge has no neighboring edges with which it can combine to become longer, the OPCBIAS operation overrides the MINBIASLENGTH rule and biases the edge anyway. This might happen at a line-end, for example, when the MINBIASLENGTH value is longer than the line-end. If you do not want to bias short lines, you can direct OPCBIAS to ignore edges shorter than MINBIASLENGTH using the LENGTH1 constraint.

Corners Affecting Short Edges (Pre-Bias)

The OPCBIAS operation defines a short edge to be any edge shorter than MINBIASLENGTH. Because the OPCBIAS operation performs short edge handling before it applies the biases, corner edges present a special case. Figure 4-11shows how biasing can shorten corner edges.

Figure 4-11. Shorten Corner Edges by Biasing



If B1 is an outer edge, applying a negative bias to A shortens B1 by the size of the negative bias. If B1 is an inner edge, applying a positive bias to A shortens B1 by the size of the positive bias. Because of this shortening, if the OPCBIAS operation widened B1 to MINBIASLENGTH before applying the bias to A, the final length of B1 would be less than MINBIASLENGTH.

Short edge handling avoids this problem by using a different definition of “short” for corner edges than other edges.

- For outside corner edges, short is defined as < (MINBIASLENGTH + abs(maximum possible negative bias))[1].
- For inside corner edges, short is defined as < (MINBIASLENGTH + (maximum possible positive bias)).

1. Use the absolute value of the negative bias to ensure that you are adding a positive number to MINBIASLENGTH.

- For all other edges, short is defined as < MINBIASLENGTH.

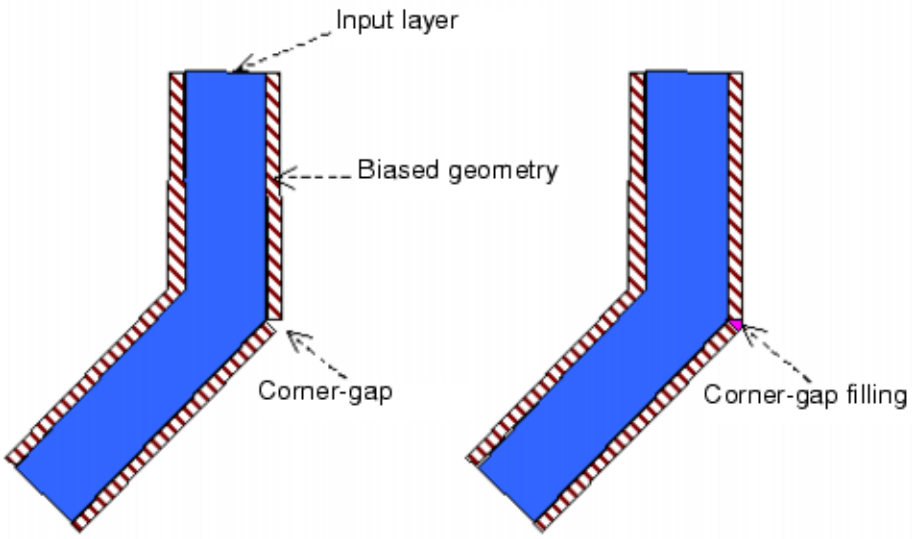
Treatment of Short Edges (Pre-Bias)

When an edge fragment is shorter than MINBIASLENGTH, the OPCBIAS operation must either merge the short fragment with an adjacent fragment or lengthen the short edge until its length equals MINBIASLENGTH. The OPCBIAS operation bases the decision to widen versus lengthen based on priorities it assigns to the bias values for the fragments. These priorities also impact the decision as to which bias to apply to the fragments that remain after clean up.

Corner Smoothing (Post-Bias)

If the edge shifting process causes discontinuity at the corners of abutting edges, then the system performs additional smoothing operation to form contiguous expanded edge geometry. Figure 4-12 shows corner gap filling for a 45-degree corner on a bent geometry.

Figure 4-12. Corner Gap Filling



Snap 45 Degree Angles (Post-Bias)

The GRID keyword directs OPCBIAS to adjust the bias of 45-degree edges so edges are placed on a 45-degree grid. In addition, you use the OPTION SPIKE_CLEANUP keyword to snap skewed edges to 45 degrees.

This feature is designed to be used when the results from the OPCBIAS operation serve as input to the SNAP operation. Without this special biasing, it is possible that the SNAP operation produces skew edges that are 1 grid unit off from being a 45-degree angle. With this special biasing, endpoints of 45-degree edges tend to snap in a consistent direction, and the SNAP operation expediently preserves 45-degree angles.

When using this feature, you must define a snap grid equal to the one used with the SNAP operation; if you attempt to perform snapping using the SNAP operation to place OPCBIAS

output on grid, it is possible that some biased edges can have their endpoints snapped on grid, when the original edges are skew. Using the GRID option avoids this problem, and snaps the OPCBIAS output for most edges to the same grid the SNAP operation uses. There are still some edges from an OPCBIAS GRID operation that are off grid due to internal edge breaking; however, any such off-grid edges are then placed on grid by SNAP. Figure 4-13 shows an example of grid-correction for angled edges.

Figure 4-13. OPCBIAS GRID Option

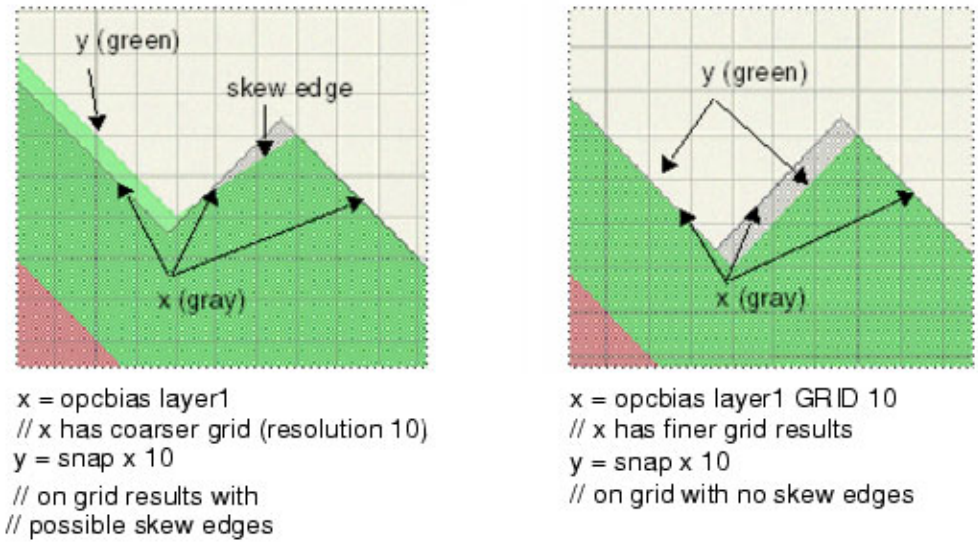


Figure 4-14 illustrates an example where slivers are created from snapping. This sliver can be resolved with spike edge clean up.

Figure 4-14. Skew Edge Clean Up

Related Topics

Calculation of Bias Priorities

Calculation of Bias Priorities

When OPCBIAS identifies a short edge and a neighbor it can merge, it calculates the bias to use for the result. OPCBIAS determines priority of bias by comparing all biases, and space and width categories.

The OPCBIAS operation chooses the prioritized edge as follows:

1. If both biases are positive:
    a. Consider the space category. The edge with the closest space category takes priority.
    b. If the space category is the same, the edge with the closest width category takes priority.
    c. If the width category is also the same, the longest edge takes priority.
2. If both biases are negative:
    a. Consider the width category. The edge with the closest width category takes priority.
    b. If the width category is the same, the edge with the closest space category takes priority.
    c. If the space category is also the same, the longest edge takes priority.
3. If biases are mixed (one bias is < 0, the other is > 0):
    a. Consider the space category. The edge with the closest space category takes priority.
    b. If priorities are the same, the positive bias takes priority.

Tables 4-4 through 4-6 summarize these three situations:

Table 4-4. Prioritized Bias — Both Positive — Evaluate SPACE First

Edge Bias	SPACE	WIDTH	LENGTH
Positive Positive	Closest edge takes priority	Not considered	Not considered
Positive Positive	Arbitrary	Closest edge takes priority	Not considered
Positive Positive	Arbitrary	Arbitrary	Longest edge takes priority

Table 4-5. Prioritized Bias — Both Negative — Evaluate WIDTH First

Edge Bias	WIDTH	SPACE	LENGTH
Negative Negative	Closest edge takes priority	Not Considered	Not Considered
Negative Negative	Arbitrary	Closest edge takes priority	Not Considered
Negative Negative	Arbitrary	Arbitrary	Longest edge takes priority

Table 4-6. Prioritized Bias — Mixed Bias — Evaluate SPACE First

Edge Bias	SPACE	Positive/Negative
Positive Negative	Closest edge takes priority	Not considered
Positive Negative	Arbitrary	Positive edge takes priority

Calculation of Treatment for Edges Selected by Priority

After selecting the prioritized edge, the OPCBIAS operation calculates the treatment the edge should receive by examining the selected bias and the lengths of the two edges:

- The OPCBIAS operation widens the short edge if the short edge has the selected bias, and the edge next to it is long enough that, after widening, both are greater than MINBIASLENGTH.
- The OPCBIAS operation merges the two edges and selects the resulting edge if the short edge is not selected or the short edge is selected, but the other edge is also considered short.

Note

Generally, edges are combined with one neighbor, but there are exceptions in places where a short edge is widened in both directions, where doing so preserves symmetry. This process is order-dependent and the results may vary from the order the edges are encountered.

Treatment of Short Edges Between Two Long Edges

When a short fragment is between two long fragments, the heuristics described in “Treatment of Short Edges (Pre-Bias)” on page 162 may not be clear regarding when and how to widen.

Consider the following situation, typically caused by a line-end that is shorter than MINBIASLENGTH influencing bias of an adjacent edge:

- There is a short fragment with long fragments on either side.
- The short fragment has the highest priority of the three.
- The long fragments both have the same bias.

These situations are resolved by widening the short fragment by an equal amount in each direction.

Figure 4-15. Treating Short Edges Between Long Edges with Same Bias

By default, this process applies only when the long fragments on either side of the short fragment have the same bias. When the two fragments on either side of the short fragment have different biases, the application widens the short fragment into the fragment whose bias is closest to that of the short fragment.

If you want to widen short fragments with long fragments on either side, regardless of whether the biases of the two long fragments is the same, you can override the default behavior shown in Figure 4-15 using the keyword CLOSERSYMMETRIC:

```
OPCBIAS poly MINBIASLENGTH 0.2 CLOSERSYMMETRIC
```

Specifying this keyword widens all short edges that meet the following two conditions:

- There is a short fragment with long fragments on either side.
- The short fragment has the highest priority of the three.

OPCBIAS Examples

The OPCBIAS command provides flexibility for various biasing applications.

- SPACE-Based Biasing
- SPACE- and WIDTH-Based Biasing
- Subset Layers and Biasing
- Double-Bias in One SVRF Run

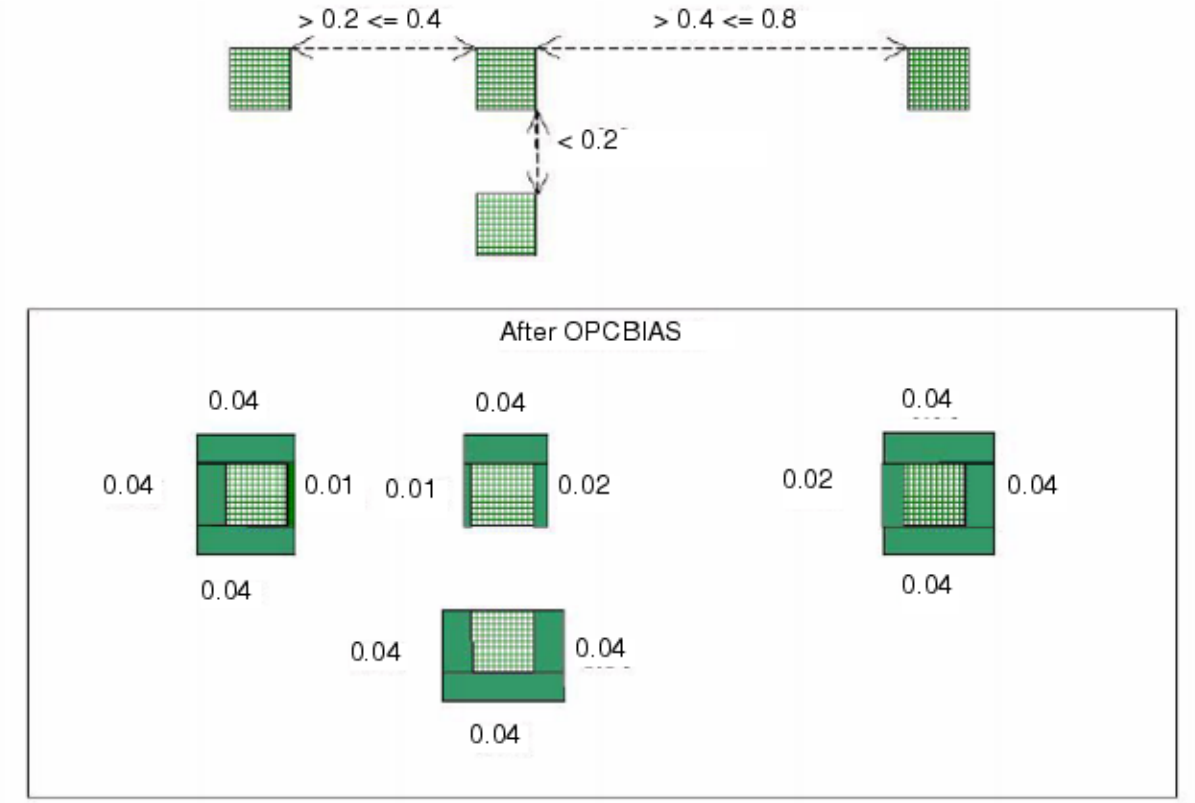
SPACE-Based Biasing

This example shows OPC performed based on the SPACE condition alone. Notice that four rules are specified here. Edges are checked if they meet the conditions specified and then moved accordingly.

Figure 4-16. SPACE-Based Biasing Example Code

```
OPCBIAS VIA1
   SPACE <=0.2        MOVE 0
   SPACE >0.2 <= 0.4 MOVE 0.01
   SPACE >0.4 <= 0.8 MOVE 0.02
   SPACE >0.8         MOVE 0.04
```

Figure 4-17. SPACE-Based Biasing



Explanation of SPACE-Based Biasing

The letters in the explanation refer to those in Figure 4-18.

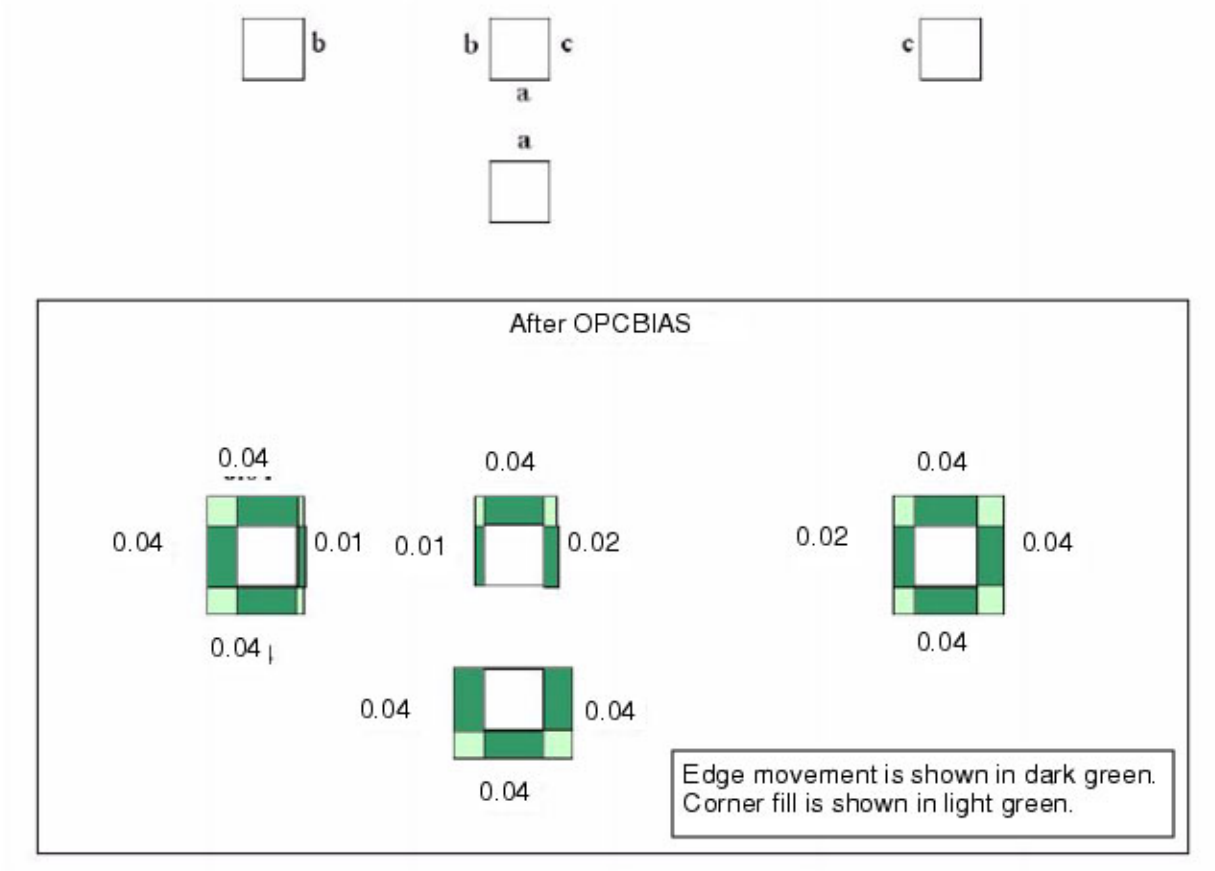
a — Meets SPACE <= 0.2, so edges do not move.

b — Meets SPACE >0.2<= 0.4, so edges move 0.01.

c — Meets SPACE >0.4<= 0.8, so edges move 0.02.

All other (unlabeled) edges meet SPACE >0.8, so edges move 0.04.

Figure 4-18. Explanation of SPACE-Based Biasing



SPACE- and WIDTH-Based Biasing

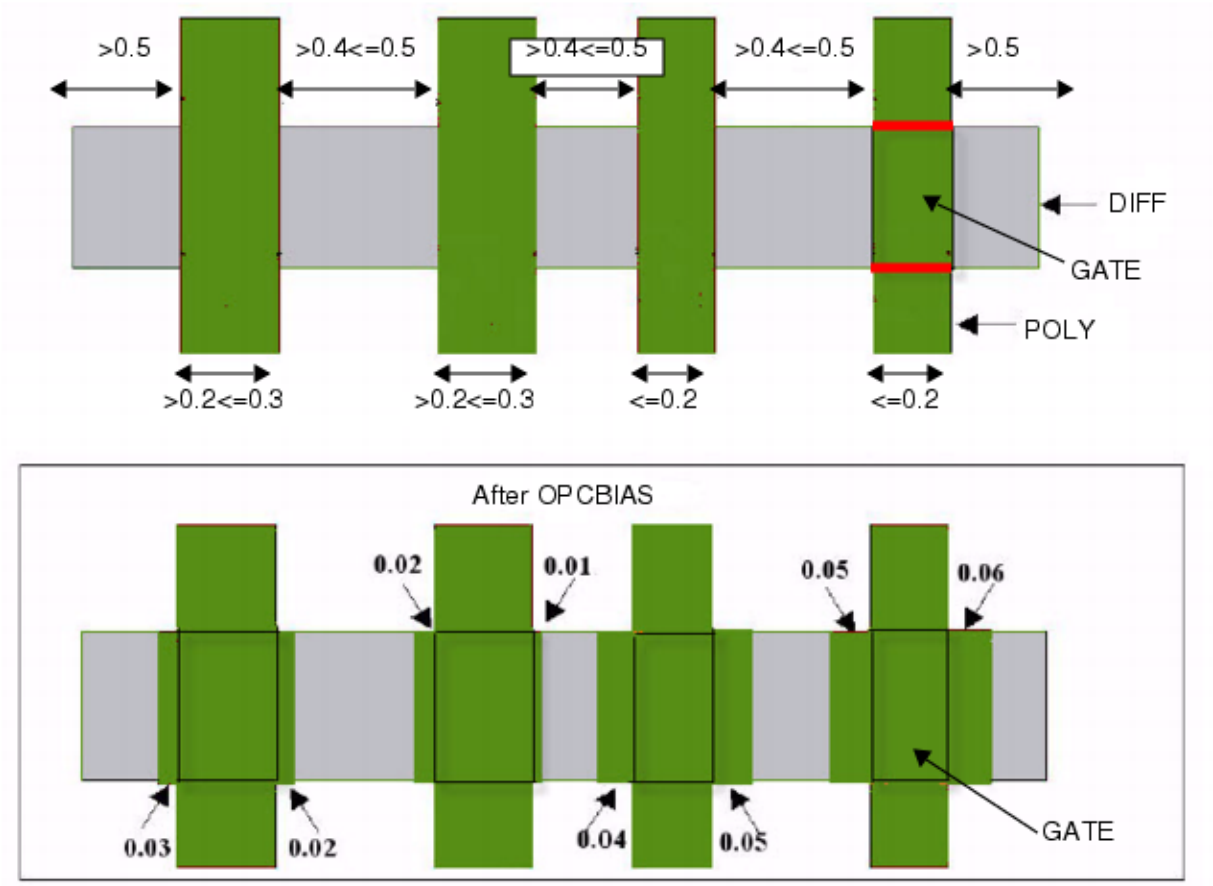
This example shows OPCBIAS with SPACE and WIDTH conditions.

The first statement generates GATE, which is the input layer. LENGTH1 ensures that the unwanted edges of GATE (generated by ANDing the POLY and DIFF layers shown in red in Figure 4-20) are not biased.

Figure 4-19. SPACE- and WIDTH-Based Biasing Example Code

```
GATE = POLY AND DIFF
OPC {OPCBIAS GATE
   SPACE >0.3 <= 0.4 WIDTH <=0.2 MOVE 0.04 LENGTH1 >0.2
   SPACE >0.4 <= 0.5 WIDTH <=0.2 MOVE 0.05 LENGTH1 >0.2
   SPACE >0.5        WIDTH <=0.2 MOVE 0.06 LENGTH1 >0.2
   SPACE >0.3 <= 0.4 WIDTH >0.2  MOVE 0.01 LENGTH1 >0.3
   SPACE >0.4 <= 0.5 WIDTH >0.2  MOVE 0.02 LENGTH1 >0.3
   SPACE >0.5        WIDTH >0.2  MOVE 0.03 LENGTH1 >0.3
}
```

Figure 4-20. SPACE- and WIDTH-Based Biasing

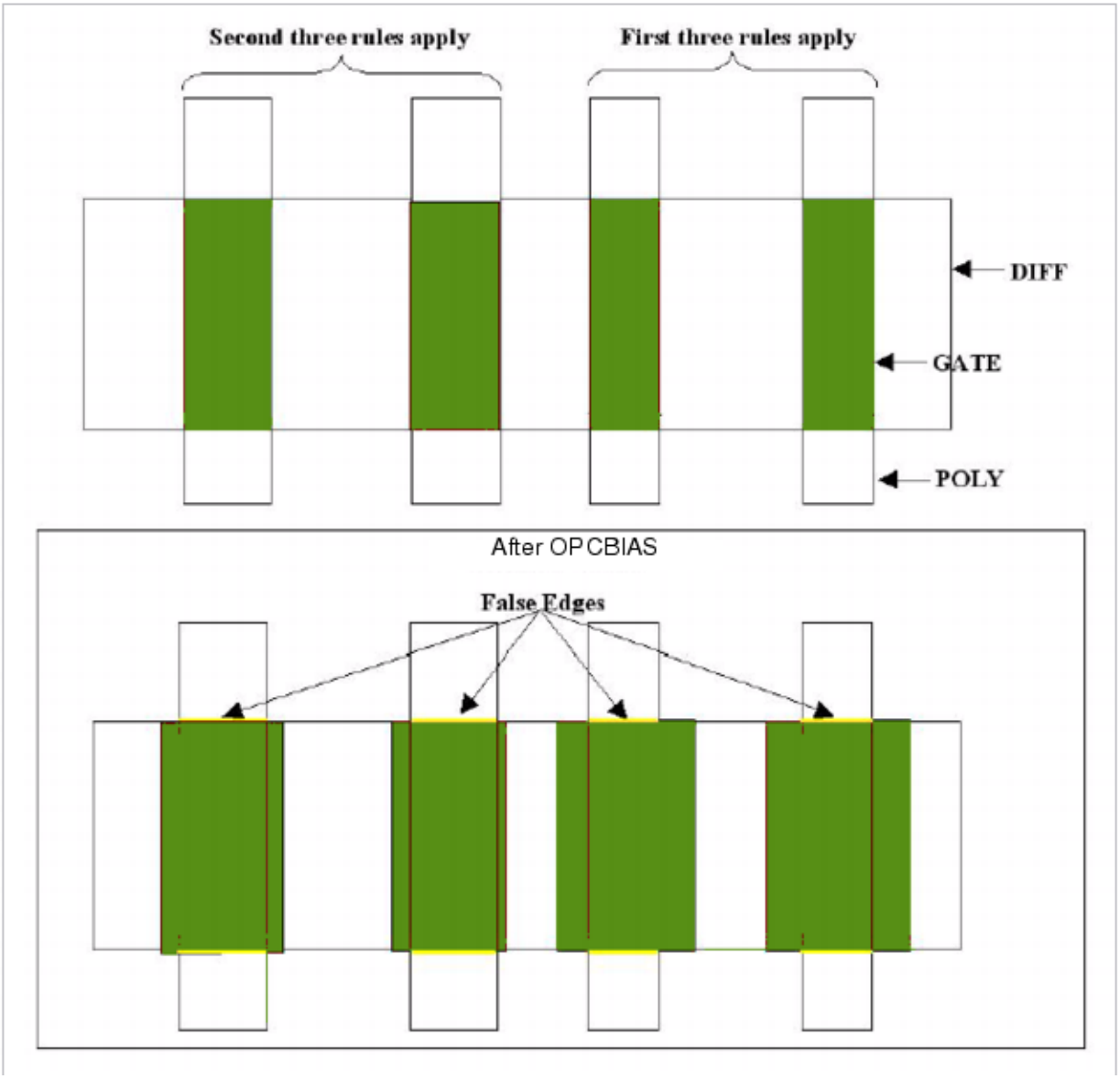


Explanation of SPACE- and WIDTH-Based Biasing

Each layer name in the explanation below refers to the names shown in Figure 4-21.

- Only edges on the new layer, GATE, formed by a geometric “and” of POLY and DIFF, receive correction.
- The two rectangles on the right satisfy the WIDTH condition for the first three rules. The two rectangles on the left satisfy the WIDTH condition for the second three rules.
- False Edges do not satisfy the LENGTH1 constraint, so they cannot be moved.

Figure 4-21. Explanation of SPACE- and WIDTH-Based Biasing



Subset Layers and Biasing

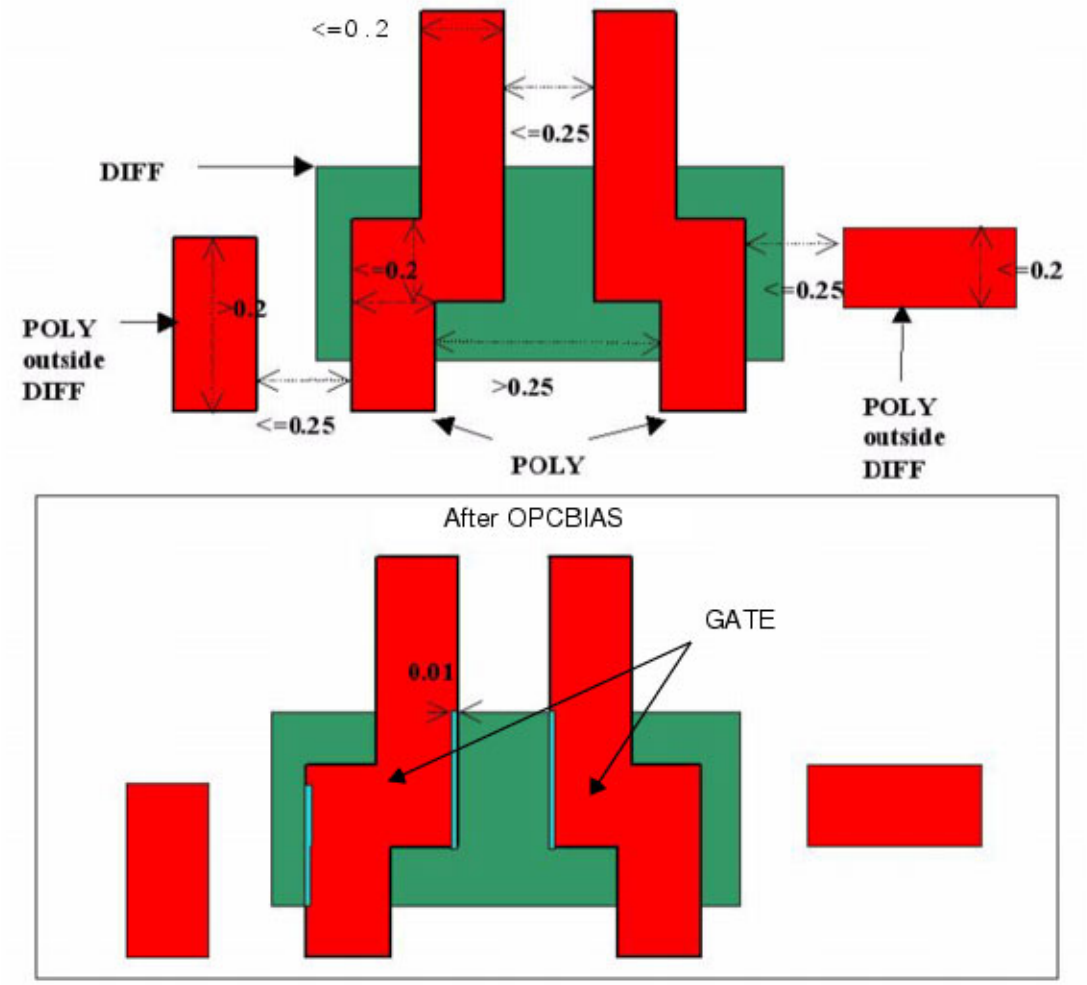
This example shows another way to avoid moving false edges: creating a subset_layer containing only inside edges. SVRF statements are used to create the subset_layer (GATE_EDGE) and the in_layer(GATE). The OPPOSITE EXTENDED metric used with the WIDTH condition causes OPC to be performed on jogs (defined as <=0.2) on the POLY layer.

Figure 4-22. Subset Layers and Biasing Example Code

```
GATE = POLY AND DIFF
GATE_EDGE = GATE INSIDE EDGE DIFF
```

```
OPC { OPCBIAS GATE_EDGE GATE POLY
  SPACE <=0.25 WIDTH <=0.2
  OPPOSITE EXTENDED 0.2 MOVE -0.01 LENGTH2 > 0.2
}
```

Figure 4-23. Subset Layers and Biasing



Explanation of Subset Layers and Biasing

The letters in the explanation refer to those in Figure 4-24.

a — Does not move because it is not inside DIFF.

b — Does not move because there is no nearby polygon.

c — Does not move because it extends beyond the polygon to the left, and space is not opposite extended.

d — Meets the SPACE and would not meet the WIDTH constraint, except that width is OPPOSITE EXTENDED.

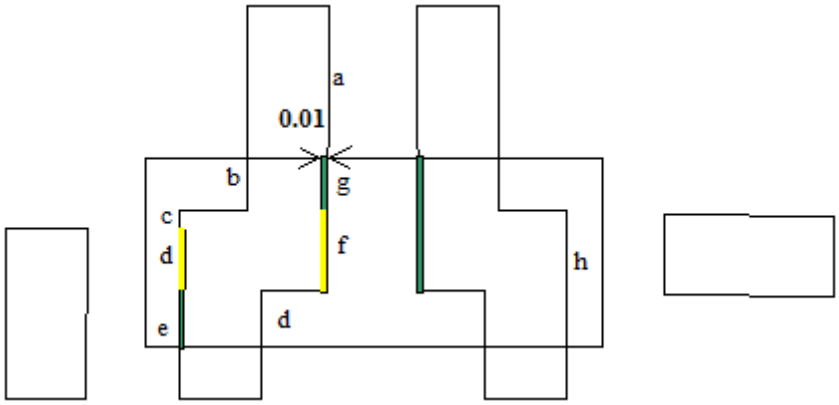
e — Meets both WIDTH and SPACE constraints.

f — Meets the SPACE, but would not meet the WIDTH constraint, except that width is OPPOSITE EXTENDED.

g — Meets both WIDTH and SPACE constraints.

h — Meets the SPACE constraint, and would be included in the WIDTH OPPOSITE EXTENDED; however, it does not move because the polygon to the right does not satisfy the LENGTH2 constraint.

Figure 4-24. Explanation of Subset Layers and Biasing



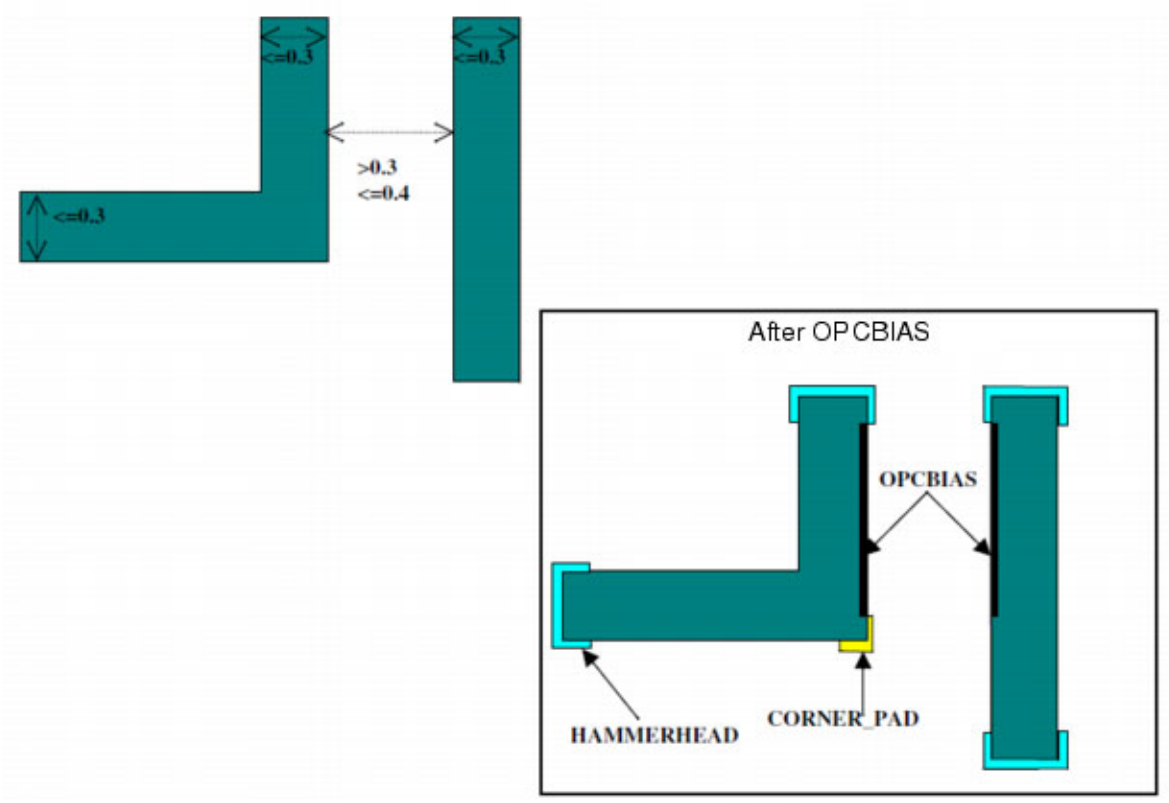
Double-Bias in One SVRF Run

This example shows the OPCBIAS operation being used twice within an SVRF rule file. The first occurrence creates the corner pads and the second biases the edges.

Figure 4-25. Double-Bias in One SVRF Run Example Code

```
HAMMERHEAD = OPCLINEEND MET1  // OPCLINEEND Command
  WIDTH<= 0.3 HEIGHT > 0.1 END 0.02 SERIF 0.02 0.01
MET1_EDGE1 = MET1 OUTSIDE EDGE HAMMERHEAD
CORNER_OUT = INT [MET1_EDGE1] <= 0.08 ABUT == 90 INTERSECTING ONLY

CORNER_PAD = (OPCBIAS CORNER_OUT MET1
  SPACE > 0.3 <= 0.4 MOVE 0.01
  SPACE > 0.4 MOVE 0.02) NOT MET1
MET1_EDGE2 = MET1_EDGE1 NOT COINCIDENT EDGE CORNER_OUT

OPC { OPCBIAS MET1_EDGE2 MET1
   SPACE > 0.3 <= 0.4 WIDTH <= 0.3 OPPOSITE EXTENDED 0.3 MOVE -0.01 }
```

Figure 4-26. Double-Bias in One SVRF Run



Explanation of Double-Bias in One SVRF Run

The letters in the explanation refer to those in Figure 4-27.

a — Meets both WIDTH and SPACE constraints.

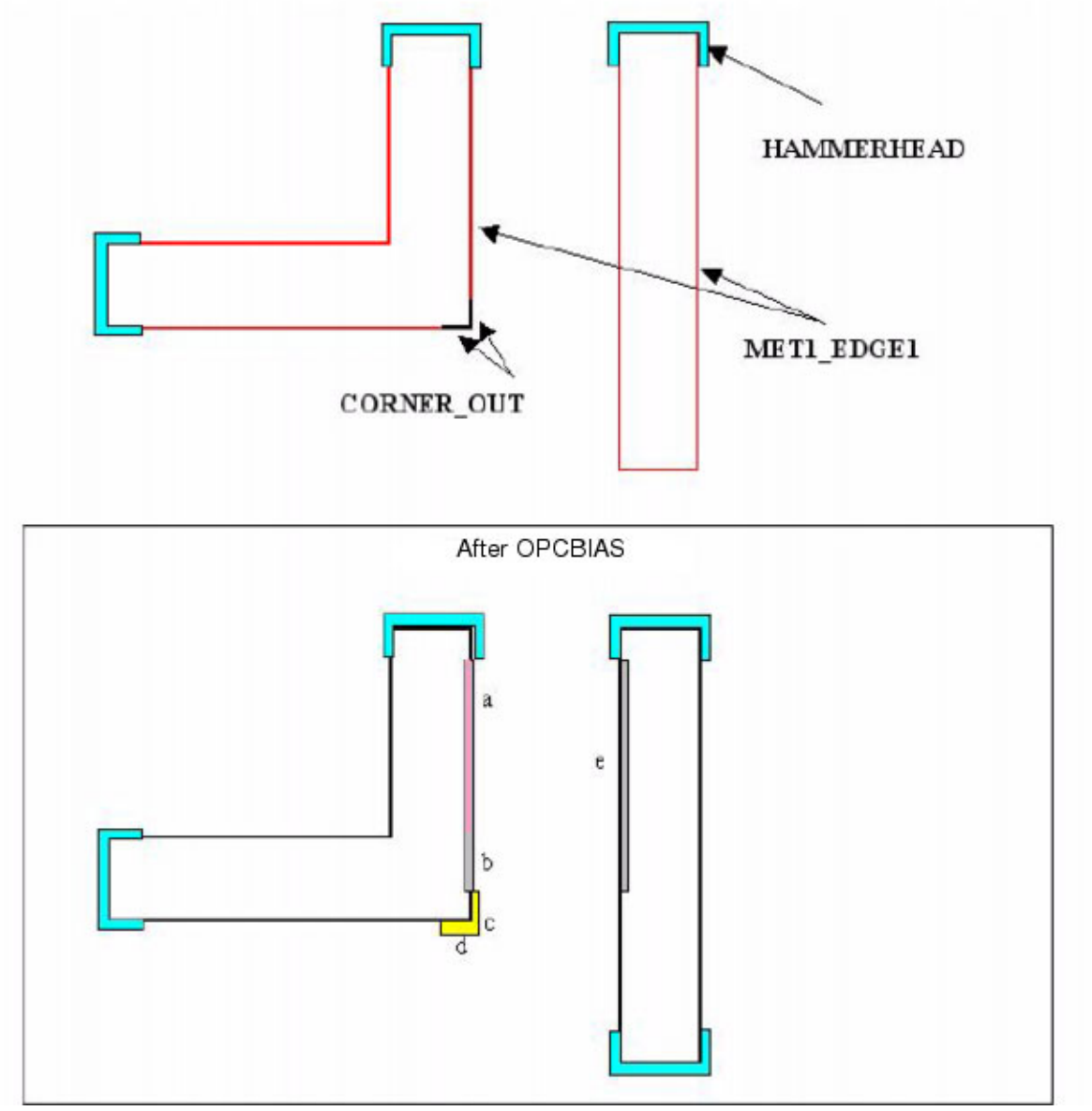
b — Meets the SPACE and the WIDTH constraint, because width is OPPOSITE EXTENDED.

c — On layer CORNER_OUT, therefore receives corner biasing.

d — On layer CORNER_OUT, therefore receives corner biasing.

e — Meets both WIDTH and SPACE constraints.

Figure 4-27. Explanation of Double-Bias in One SVRF Run



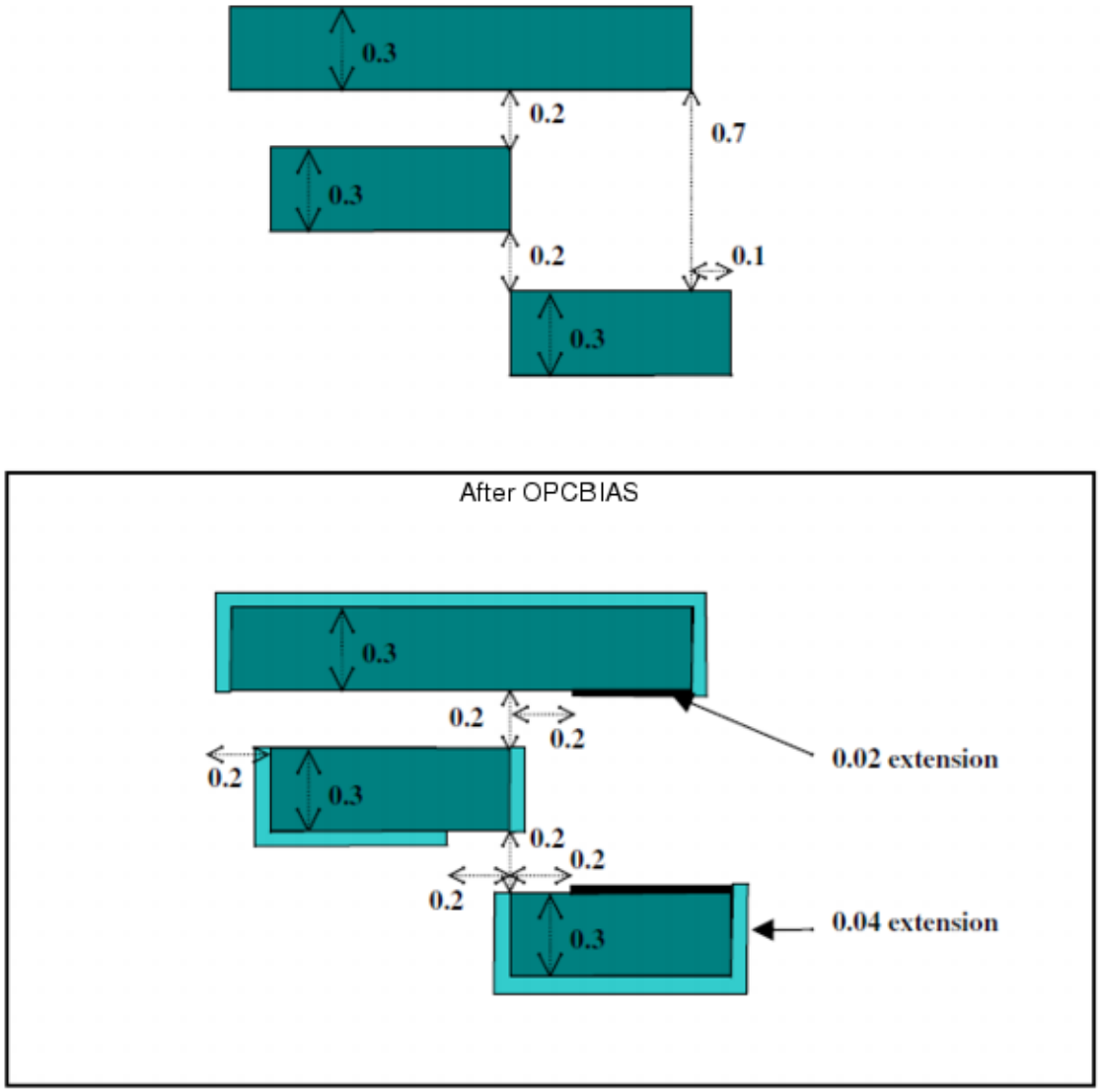
Closest Edge as a Priority

This example shows how the OPCBIAS operation uses the closest edge as a priority to resolve conflicts caused by edges meeting more than one SPACE condition when using the OPPOSITE EXTENDED metric.

Figure 4-28. Closest Edge as a Priority Example Code

```
OPCBIAS MET1
  SPACE <= 0.4        OPPOSITE EXTENDED 0.2 MOVE 0
  SPACE >  0.4 <=0.7 OPPOSITE EXTENDED 0.1 MOVE 0.02
  SPACE >  0.7                              MOVE 0.04
```

Figure 4-29. Closest Edge as a Priority



Explanation of Closest Edge as a Priority

The letters in the explanation refer to those in Figure 4-30.

a — Meets SPACE constraint <= 0.4, and does not move.

b — Conflict caused by OPPOSITE EXTENDED: Meets SPACE constraint <= 0.4 and SPACE constraint > 0.4 <= 0.7. Closer edges win, so does not move.

c — Meets SPACE constraint > 0.4 <= 0.7, and moves 0.02.

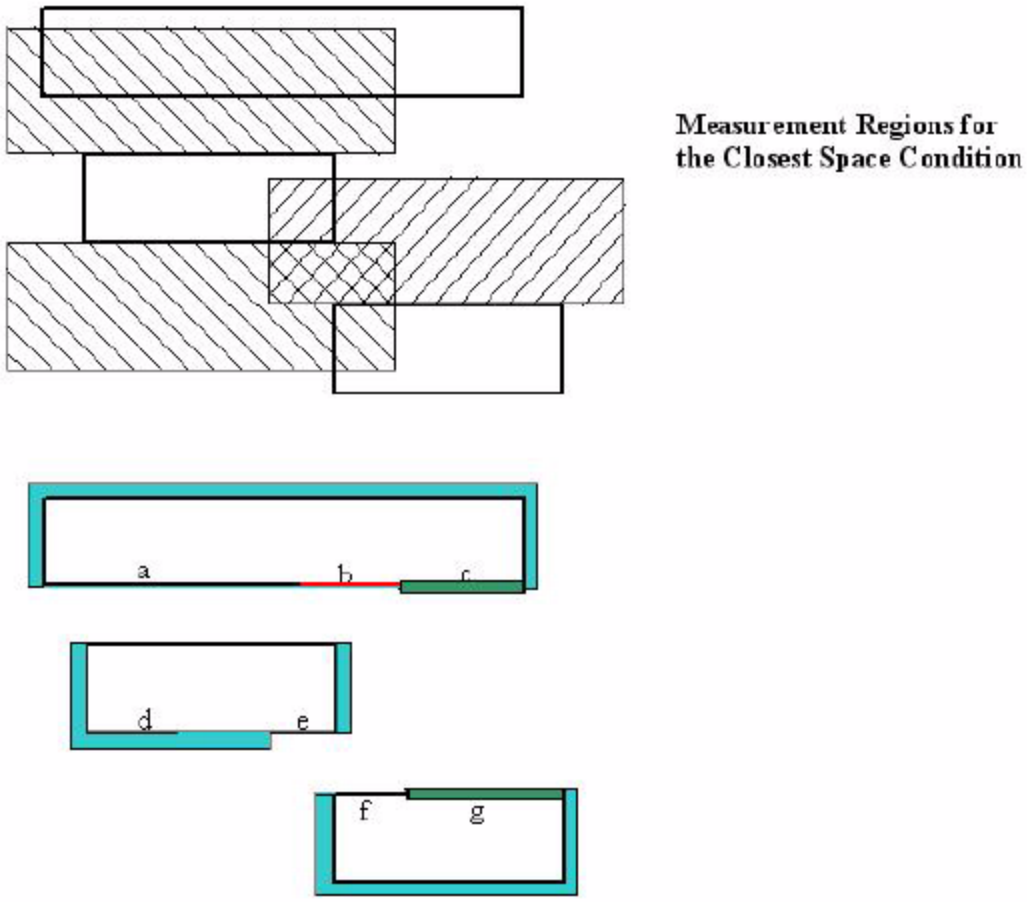
d — Meets SPACE constraint > 0.7, and moves 0.04.

e — Conflict caused by OPPOSITE EXTENDED: Meets SPACE constraint <= 0.4 and SPACE constraint > 0.7. Closer edges win, and does not move.

f — Conflict caused by OPPOSITE EXTENDED: Meets SPACE constraint <= 0.4 and SPACE constraint > 0.4 <= 0.7. Closer edges win, and does not move.

g — Meets SPACE constraint > 0.4 <= 0.7, and moves 0.02.

Figure 4-30. Explanation of Closest Edge as a Priority



Feature Smoothing With MINBIASLENGTH

This example shows how you can smooth features that fall below the minimum MINBIASLENGTH value you specify.

Classifying edges based on the SPACE constraint results in a small edge at the top of the geometry on the left. MINBIASLENGTH 0.14 ensures that this edge is smoothed out.

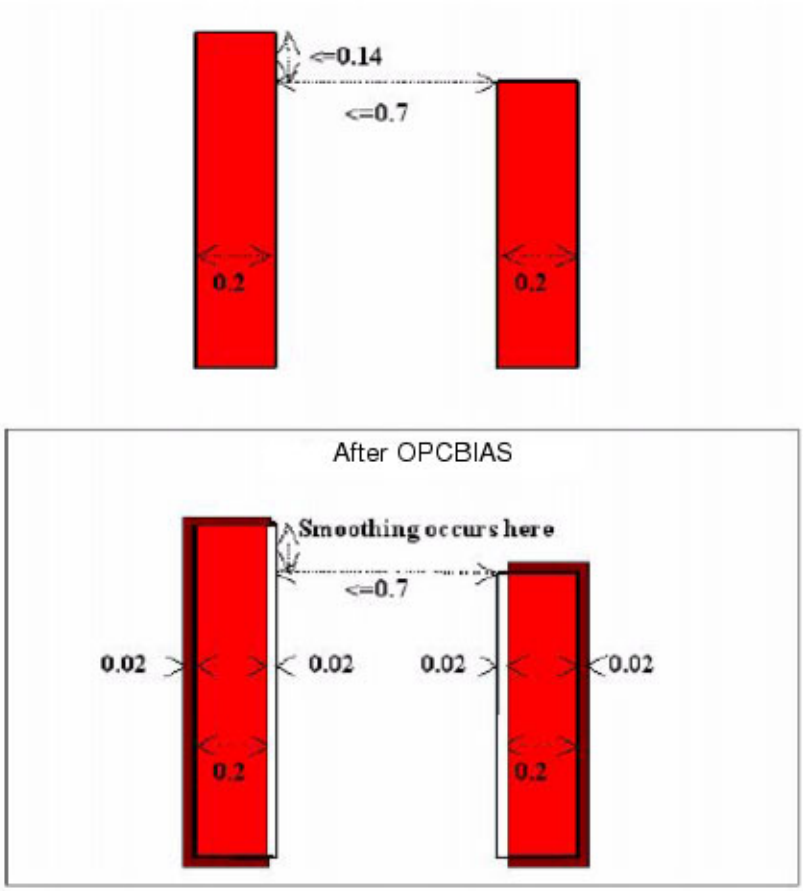
Notice that MINBIASLENGTH is applied AFTER edges are classified using the SPACE constraint whereas LENGTH1 would have been applied before such classification. Also note

that the MINBIASLENGTH applies to all rules processed by the operation whereas the LENGTH1 constraint applies only to the rule for which it is supplied.

Figure 4-31. Feature Smoothing With MINBIASLENGTH Example Code

```
OPCBIAS GATE  MINBIASLENGTH 0.14
  SPACE <= 0.7 MOVE -0.02
  SPACE >  0.7 MOVE  0.02
```

Figure 4-32. Feature Smoothing With MINBIASLENGTH



Explanation of Feature Smoothing With MINBIASLENGTH

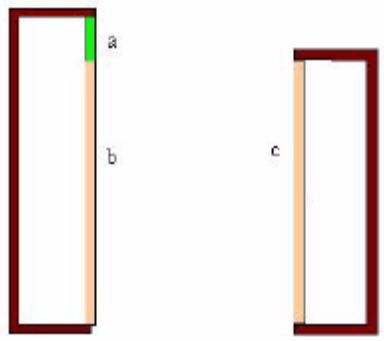
The letters in the explanation refer to those in Figure 4-33.

a — Meets the SPACE constraint > 0.7, and would move 0.02, but this violates MINBIASLENGTH. Because of this, smoothing applies, and this short edge gets added to b.

b — Meets SPACE constraint <= 0.7, and moves -0.02.

c — Meets SPACE constraint <= 0.7, and moves -0.02.

Figure 4-33. Explanation of Feature Smoothing With MINBIASLENGTH



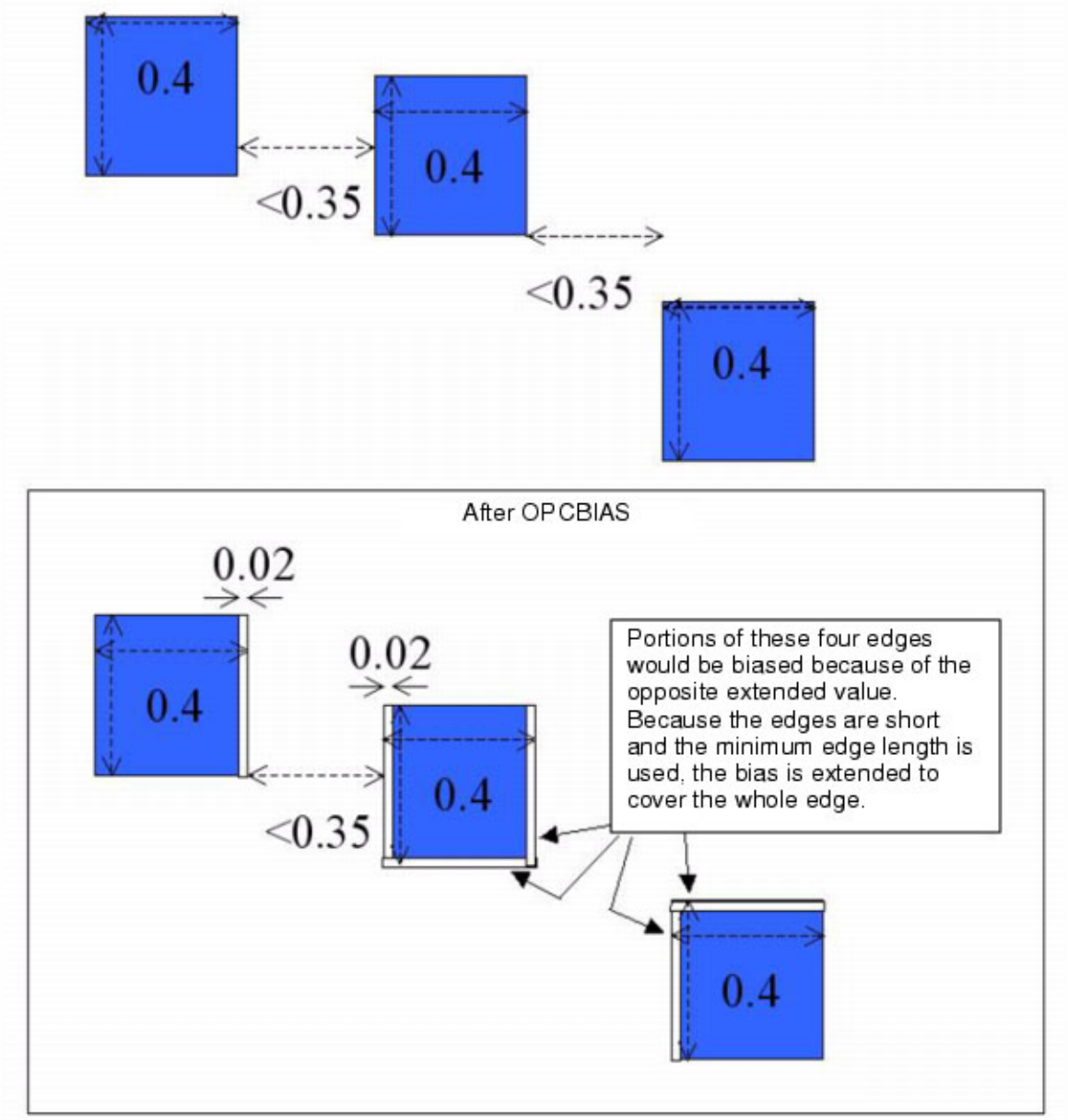
MINBIASLENGTH and OPPOSITE EXTENDED With SPACE

This example code shows how you can use the OPPOSITE EXTENDED metric with the SPACE condition to have an extended coverage of the measurement and how you can use MINBIASLENGTH to avoid jog creation on the CONTACT edges.

Figure 4-34. Code for MINBIASLENGTH and OPPOSITE EXTENDED With SPACE

```
OPCBIAS CONTACT MINBIASLENGTH 0.4
   SPACE < 0.35 OPPOSITE EXTENDED 0.4 MOVE -0.02
```

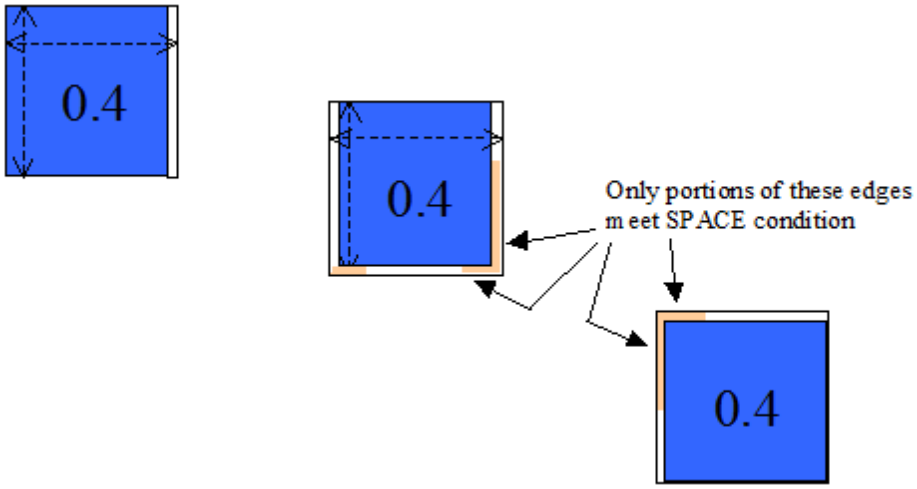
Figure 4-35. Results for MINBIASLENGTH and OPPOSITE EXTENDED With SPACE



Explanation of MINBIASLENGTH and OPPOSITE EXTENDED With SPACE

- Only the portions of the four edges indicated would be biased because of the OPPOSITE EXTENDED value.
- The rule specifies MINBIASLENGTH 0.4, so these are considered short edges, and smoothing occurs.
- Because the bias is negative, the short edges take priority and merge with the remaining edges, and the whole sides are biased.

Figure 4-36. Explanation for MINBIASLENGTH, OPPOSITE EXTENDED With SPACE



OPPOSITE EXTENDED With SPACE

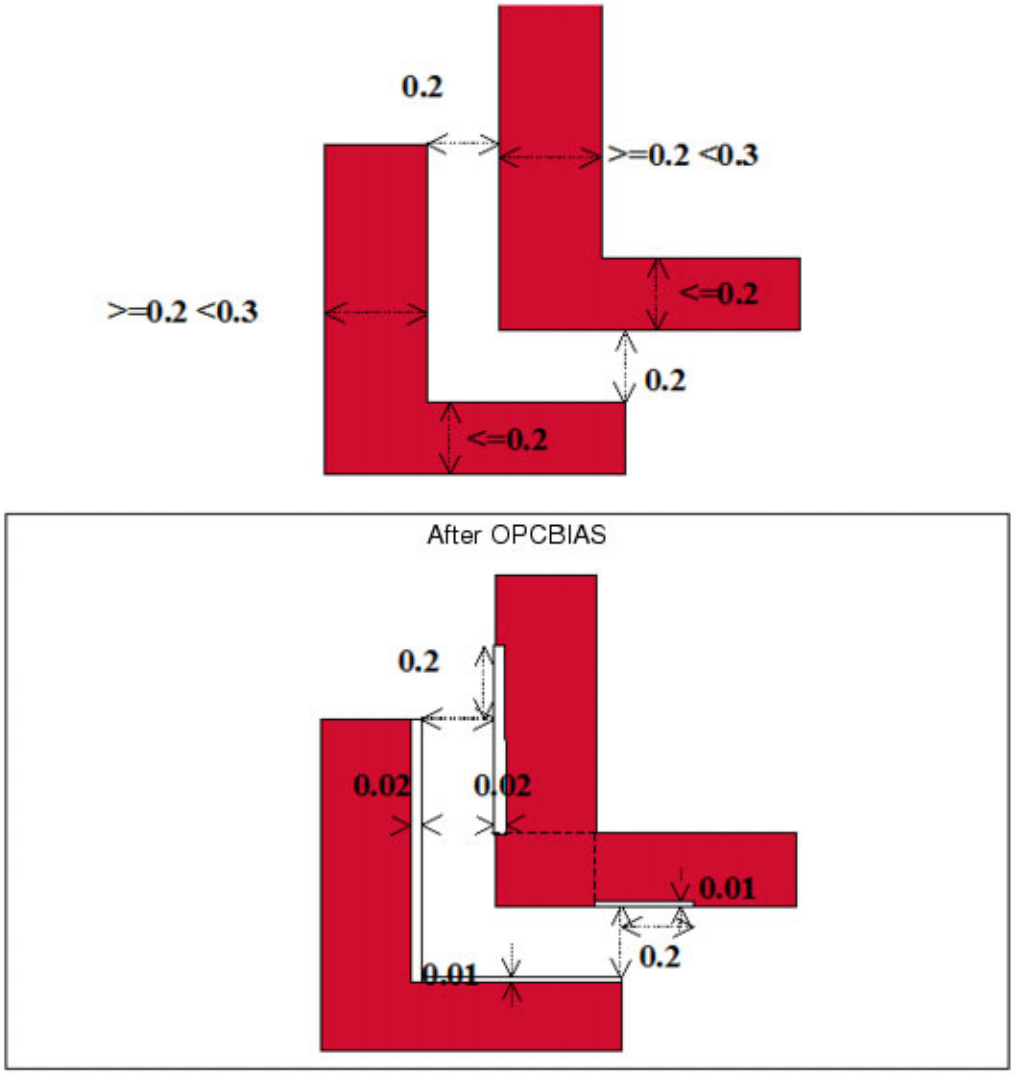
This example shows the effect of using the OPPOSITE EXTENDED metric with the SPACE condition alone.

Compare the results in this example with those in OPPOSITE EXTENDED With SPACE where the OPPOSITE EXTENDED metric is also used for the WIDTH condition.

Figure 4-37. OPPOSITE EXTENDED With SPACE Example Code

```
OPCBIAS MET1
  SPACE <=0.2 OPPOSITE EXTENDED 0.2 WIDTH <= 0.2       MOVE -0.01
  SPACE <=0.2 OPPOSITE EXTENDED 0.2 WIDTH >  0.2 < 0.3 MOVE -0.02
```

Figure 4-38. OPPOSITE EXTENDED With SPACE



Explanation of OPPOSITE EXTENDED With SPACE

The letters in the explanation refer to those in Figure 4-39.

a — Meets both WIDTH and SPACE constraints for second rule, moves -0.02.

b — Meets both WIDTH and SPACE constraints for first rule, moves -0.01.

c — Does not meet SPACE constraint for either rule.

d — Meets both WIDTH and SPACE constraints for second rule, moves -0.02.

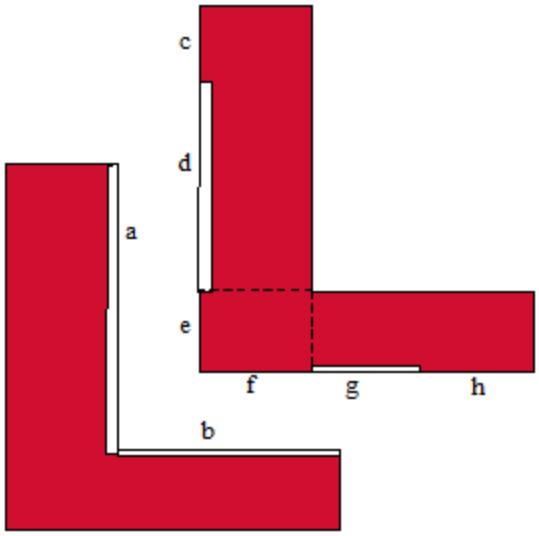
e — Meets the SPACE constraint for second rule but does not meet the WIDTH constraint.

f — Meets the SPACE constraint for first rule but does not meet the WIDTH constraint.

g — Meets both WIDTH and SPACE constraints for first rule, moves -0.01.

h — Does not meet SPACE constraint for either rule.

Figure 4-39. Explanation of OPPOSITE EXTENDED With SPACE



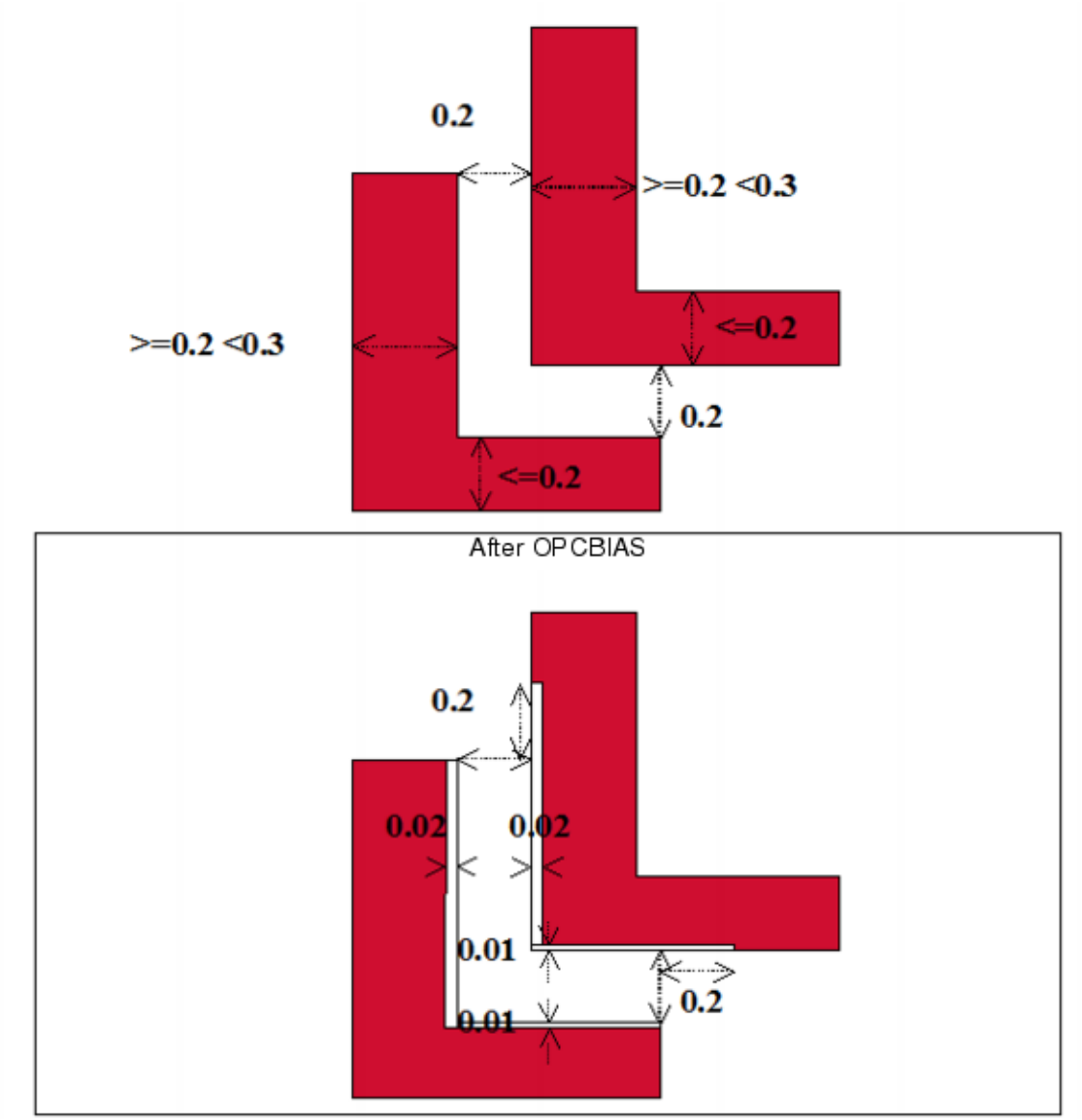
OPPOSITE EXTENDED With WIDTH and SPACE

This example shows the effect of using the OPPOSITE EXTENDED metric with the WIDTH condition as well as the SPACE condition.

Figure 4-40. OPPOSITE EXTENDED With WIDTH and SPACE Example Code

```
OPCBIAS MET1
   SPACE <=0.2 OPPOSITE EXTENDED 0.2 WIDTH <=0.2
      OPPOSITE EXTENDED 0.3 MOVE -0.01
   SPACE <=0.2 OPPOSITE EXTENDED 0.2 WIDTH > 0.2<0.3
      OPPOSITE EXTENDED 0.3 MOVE -0.02
```

Figure 4-41. OPPOSITE EXTENDED With WIDTH and SPACE



Explanation of OPPOSITE EXTENDED With WIDTH and SPACE

The letters in the explanation refer to those in Figure 4-42.

a — Meets both WIDTH and SPACE constraints for second rule, moves -0.02.

b — Meets both WIDTH and SPACE constraints for first rule, moves -0.01.

c — Does not meet SPACE constraint for either rule.

d — Meets both WIDTH and SPACE constraints for second rule, moves -0.02.

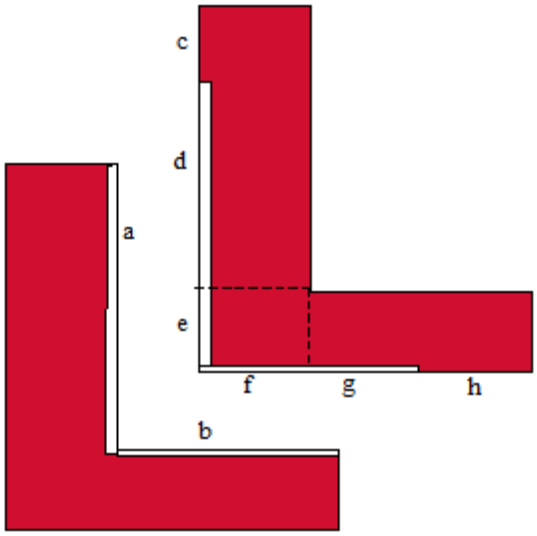
e — Meets the SPACE for second rule and also meets WIDTH constraint, because WIDTH is OPPOSITE EXTENDED. Moves -0.02.

f — Meets the SPACE for first rule and also meets WIDTH constraint, because WIDTH is OPPOSITE EXTENDED. Moves -0.01.

g — Meets both WIDTH and SPACE constraints for first rule, moves -0.01.

h — Does not meet SPACE constraint for either rule.

Figure 4-42. Explanation of OPPOSITE EXTENDED With WIDTH and SPACE



Basic Biasing

This example shows a basic rule applied to a polygon.

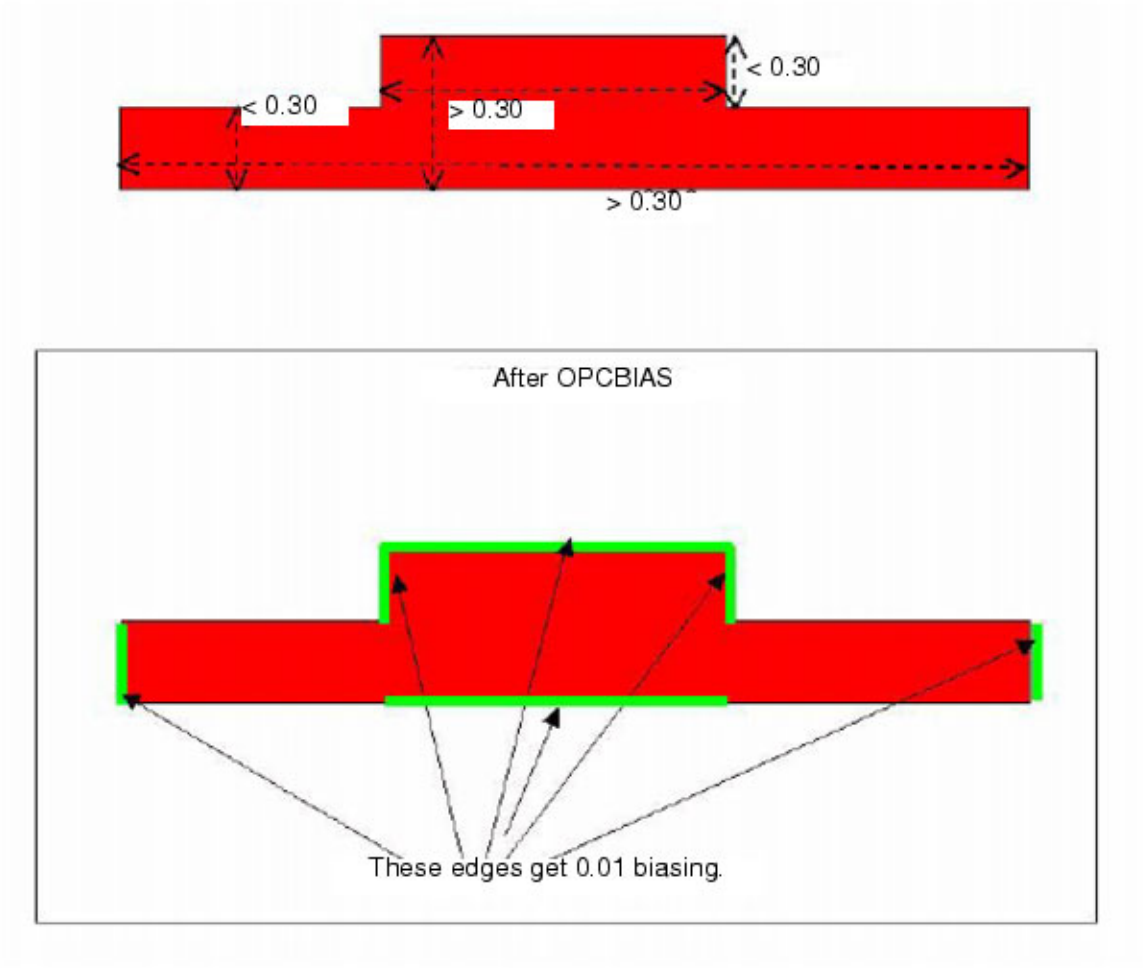
Compare the results in this example with those in LENGTH1 Biasing, which is identical except that it imposes a LENGTH1 constraint on the edges that receive OPC.

Note that in both this example and LENGTH1 Biasing, the SPACE condition is specified but not relevant to the example. This is because the SPACE condition is required.

Figure 4-43. Basic Biasing Example Code

```
OPCBIAS MET1
   SPACE >= 0.5 WIDTH >= 0.3 MOVE 0.01
```

Figure 4-44. Basic Biasing



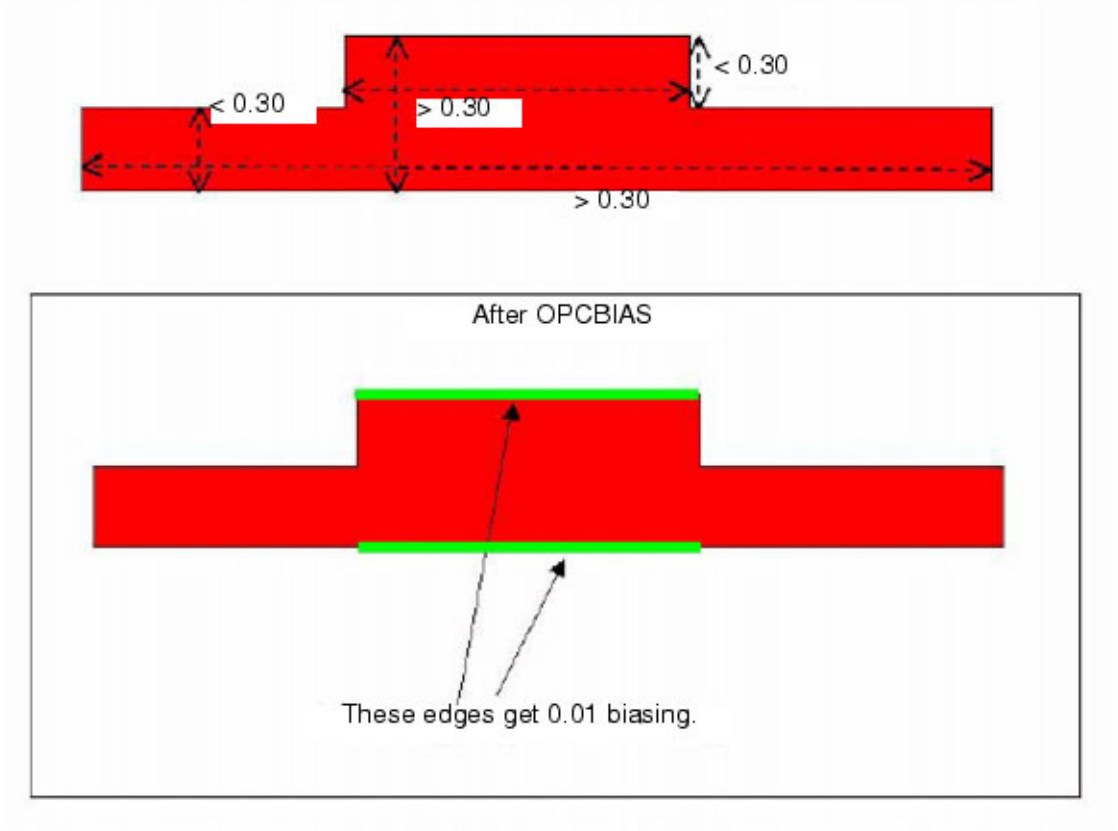
LENGTH1 Biasing

This example shows the basic rule found in the previous section, with the LENGTH1 constraint added.

Figure 4-45. LENGTH1 Biasing Example Code

```
OPCBIAS MET1
   SPACE >= 0.5 WIDTH >= 0.3  MOVE 0.01 LENGTH1 >= 0.3
```

Figure 4-46. LENGTH1 Biasing

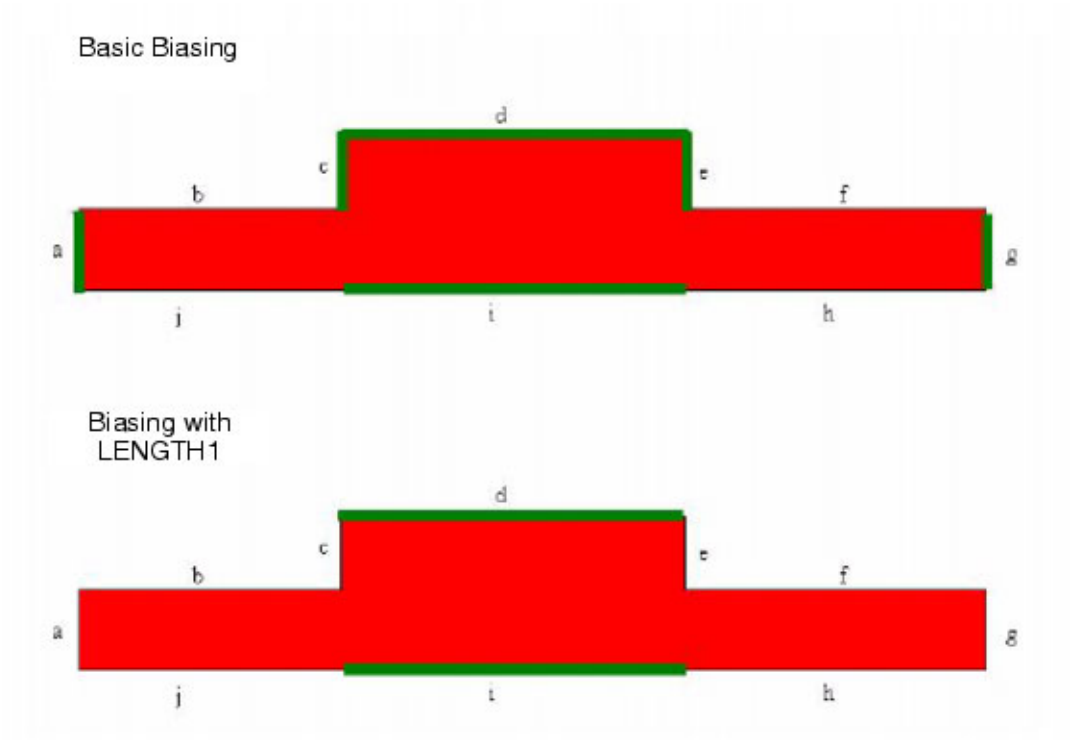


Explanation of Basic Biasing and LENGTH1 Biasing

The letters in the explanation refer to those in Figure 4-47.

- **Basic Biasing —** Edges a, c, d, e, g, and i all meet the WIDTH constraint.
- **LENGTH1 Biasing —** Only edges d and i meet the LENGTH1 constraint.

Figure 4-47. Explanation of Basic Biasing and LENGTH1 Biasing



Chapter 5
OPCLINEEND Command

The OPCLINEEND command performs variable line-end extension, including end extension and hammerhead creation.

OPCLINEEND directs the Calibre nmDRC hierarchical engine to generate line-end extensions based on line-end length and height to ensure extensions do not cause spacing violations. OPCLINEEND is typically used for table-driven OPC. This section introduces you to the OPCLINEEND command and how it functions. Refer to "OPCLINEEND" on page 190 for more detail regarding OPCLINEEND syntax.

OPCLINEEND

SVRF Commands

Biases line-ends of input target shapes.

Usage

output_layer **= OPCLINEEND**
in_layer
[RLAYER *ref_layer*]
[*via_layer* BY *value* [*via_layer2* BY *value2*]]
WIDTH *constraint* [HEIGHT *constraint*] [**SPACE *constraint*** [*metric*]]
END *value_list* SERIF *extension_list depth_list*
[CORNERFILL *fill1* [*fill2*] [ITERATIONS *value*]]
[EUCLIDEAN]
[V2]

Description

The OPCLINEEND command biases line-ends of shapes on the input layer you specify. Line-ends are selected using various constraints and then biased by various keywords provided by the OPCLINEEND command.

Arguments

- ***output_layer*= OPCLINEEND**

  or

  ***output_layer*‘{’ OPCLINEEND ... ‘}’**

  A required output layer name containing line-end data generated by the OPCLINEEND command. The output_layer must precede either an equal sign or a pair of braces surrounding the entire OPCLINEEND command.

- ***in_layer***

  A required name for the input layer containing polygon or edge data. The in_layer must be a subset of the ref_layer, although this dependency is not checked.

- RLAYER *ref_layer*

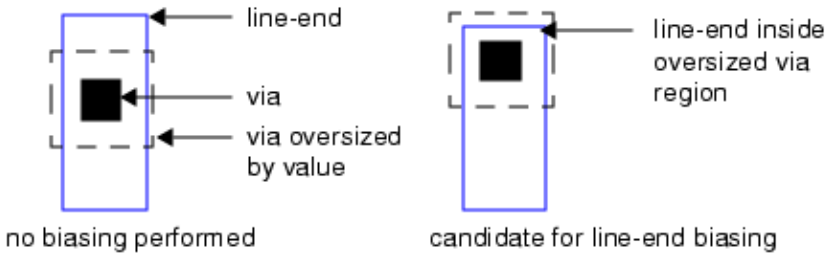
  An optional polygonal layer from which OPCLINEEND performs all spacing checks with respect to. The ref_layer must be a superset of the in_layer, although this dependency is not checked. If you do not specify RLAYER, in_layer defaults as a spacing reference layer.

- *via_layer* BY *value* [*via_layer2* BY *value2*]

  An optional polygon layer name, keyword, and floating-point distance triplet used for calculating which line-ends receive OPC treatment based on placement of vias near line-ends.

Calibre uses the value to calculate an oversize region for the via. Line-ends that fall within the oversize region are candidates for receiving OPC treatment. You can specify a maximum of two via layers. See Figure 5-1 for a graphical description of this technique.

Figure 5-1. Biasing By Via Region



- **WIDTH *constraint*** [**HEIGHT** *constraint*] [**SPACE *constraint*** [*metric*]] **END *value_list* SERIF *extension_list depth_list***

  A required keyword and set of constraints line-ends must meet in order for specified corrections to be applied to them. The WIDTH condition defines the width of the polygon at the line-end (the length of the line-end).

  When you specify multiple rule sets within the same OPCLINEEND command, the WIDTH constraint intervals must all be mutually exclusive. Within a rule set, the default rule and each of the associated correction rules must use the same WIDTH constraint.

  Initial default WIDTH rule usage is:

  WIDTH *constraint* [HEIGHT *constraint*] **END *value_list* SERIF *extension_list depth_list***

  All subsequent WIDTH rule usage must include the SPACE constraint:

  WIDTH *constraint* [HEIGHT *constraint*] {**SPACE *constraint*** [*metric*]} **END *value_list* SERIF *extension_list depth_list***

  WIDTH keyword arguments perform line-end selection and related bias treatment:

  WIDTH *constraint* — The WIDTH constraint is specified in user units as an interval of non-negative real numbers. The constraint must have an upper boundary (< x or <= x must appear in the constraint).

  HEIGHT *constraint* — An optional keyword and constraint specified as a non-negative real number in microns that a line-end must meet in order for the rule to apply to that line-end. The HEIGHT defines the height of the polygon at the line-end, which can also be described in terms of the lengths of the edges that share endpoints with the line-end edge.

A single HEIGHT constraint defines the constraint for both edges adjacent to the line-end. If the HEIGHT constraint is not specified, the default is greater than 0. Figure 5-2 describes how HEIGHT relates to each adjacent edge of a line-end:

Figure 5-2. HEIGHT Constraint

If you specify multiple rule sets within the same OPCLINEEND command, the WIDTH and HEIGHT constraint pairs must be mutually exclusive.

Within a rule set, the default rule and each of the associated correction rules must use the same HEIGHT constraint.

SPACE *constraint* [*metric*] — A required keyword and constraint specified as a range of floating point non-negative numbers used for every subsequent rule. The metric argument is optional and specifies one of three measurement methods:

OPPOSITE SYMMETRIC — Applies special processing to edges that are not parallel. This is the default metric and cannot be specified explicitly.

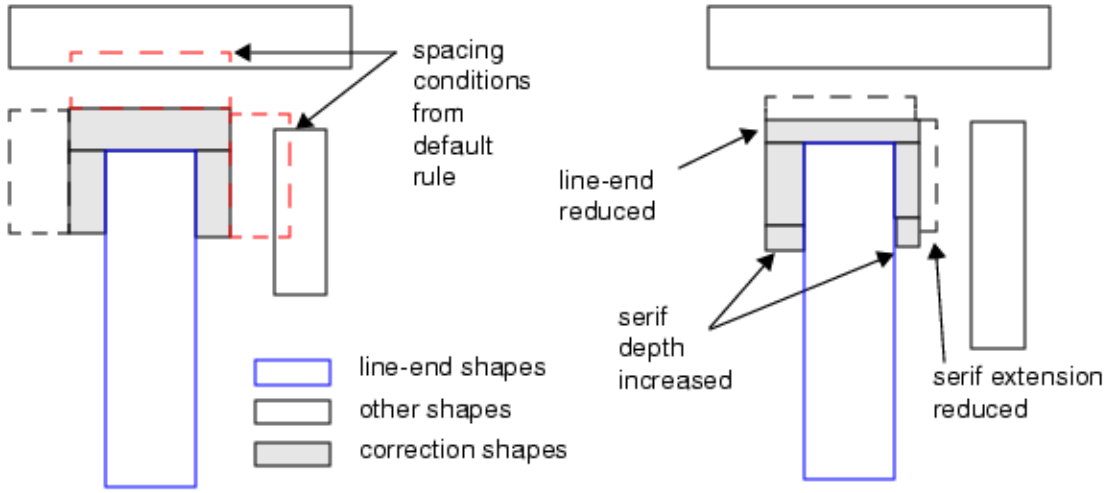
OPPOSITE EXTENDED *range* — Measures an area of orthogonal distances, one away from the line-end using the specified constraint and one perpendicular from the line-end using range. You must specify a positive floating-point number for the range.

EUCLIDEAN — Measures a consistent, radial distance from any point on the line-end using the specified constraint.

For more information regarding metrics, see the Metrics section of the *Calibre Verification User's Manual*.

SPACE is used to define conditions only for subsequent rules and not for the initial, default rule. When a spacing condition is identified, OPCLINEEND replaces the default line-end extension with the associated correction as depicted in Figure 5-3.

Figure 5-3. Applying Corrections with SPACE



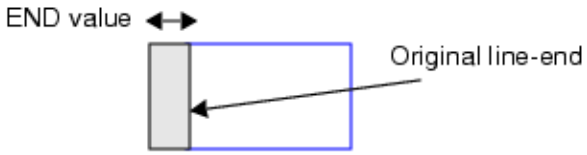
END *val_1 val_2 val_3* — A required keyword and maximum of three values specified as non-negative floating-point numbers in microns used to generate a line-end extension. For default rules, only one END value may be specified for the initial movement of a line-end, which is the maximum movement in the rule set as depicted in Figure 5-4.

val_1 — Used when the opposing edge is not extended. This value is also used as the default for the other list elements when they are not specified.

val_2 — Used if the opposing edge is an extension of the same type, that is, both edges are line-end extensions or both are serifs. If the second list element is greater than half the space between line-ends, the line-ends merge.

val_3 — Used when the opposing edge is an extension of another type, where one is a serif and the other is a line-end extension. If the third value is not specified, the second value is used by default.

Figure 5-4. END value



The number of values you supply for END extensions and SERIF extensions do not need to match. However, the number of SERIF extension values and SERIF depth values must match to avoid ambiguity. Refer to Figure 5-17 on page 203 for an illustration of possible configurations of opposing edges and their extension conditions. The END condition is required for all rules. There is no default for this setting.

SERIF *extension_list depth_list* — A required keyword and argument list applied to a line-end by the extension and depth values you specify. The extension_list defines the

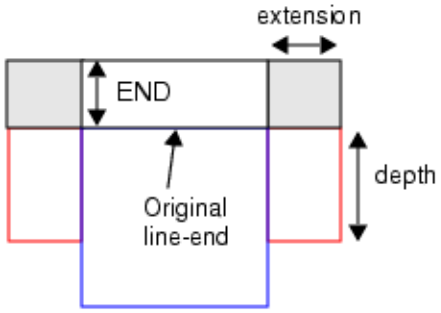
serif width and the depth_list defines the length of the hammerhead in the direction opposite of the line-end extension.

extension_list — A maximum of three non-negative floating-point numbers required for each extension argument list.

depth_list — A maximum of three non-negative floating-point numbers required for each depth argument list.

Figure 5-5 shows the addition of the extension and depth to the segment generated by END.

Figure 5-5. SERIF Extension and Depth



The extension and depth in a default rule define the maximum serif extension and depth allowed. Default rules can contain only one pair of values. If either extension or depth is zero, only the line-end extension is applied, and no serif is created.

Correction rule values are used if spacing violations occur when using default rule values. Correction rules can have up to three values in the extension_list and three values in the depth_list. These values in rule sets are discussed in the section entitled “How Opposing Edges Affect Corrections” on page 202. However, the number of SERIF extension and depth values must match to avoid ambiguity. The number of values you supply for END extensions and SERIF extensions do not need to match.

For subsequent correction rules, all extensions must be less than the default rule extension. The depths can be greater than the default rule depth, to keep the serif a constant overall depth in the event the line-end is shortened by the correction rule.

In a subsequent correction rule, a depth or extension value of zero is interpreted as no change from the value specified by the default rule.

- CORNERFILL *fill1* [*fill2*] [ITERATIONS *value*]

  An optional keyword that specifies the maximum notch size and applies corrections to any notched line-ends. CORNERFILL fills notches on line-ends that require correction prior to performing line-end extension. By default, CORNERFILL is disabled.

  fill1 — A required argument specified as a positive floating-point number in user units. This is the maximum dimension in the first direction (arbitrarily selected). If only fill1 is specified, OPCLINEEND fills notches with dimensions up to fill1 x fill1.

  fill2 — An optional argument, dependent on specification of fill1, specified as a positive floating-point number in user units. This is the maximum dimension in the second

direction (the remaining direction). If both fill1 and fill2 values are specified, OPCLINEEND fills notches with dimensions up to fill1 x fill2.

ITERATIONS *value* — An optional keyword and argument pair. The value specified must be a positive integer. You use ITERATIONS to control the number of iterations performed in the event there is a stair-step at the line-end being corrected. The default is 1.

- EUCLIDEAN

  An optional keyword that sets the EUCLIDEAN metric for use by the SPACE keyword. EUCLDEAN can be specified as a keyword in OPCLINEEND or as an argument to the WIDTH keyword.

- V2

  An optional keyword that enables the V2 hierarchical processing engine for OPCLINEEND. V2 enables an improved hierarchical promotion scheme that produces the same results, and, in ambiguous cases, ensures the same results between flat and hierarchical runs. V1 is the default mode of operation.

  In addition, V2 enables a new transcript format for log files.

Examples

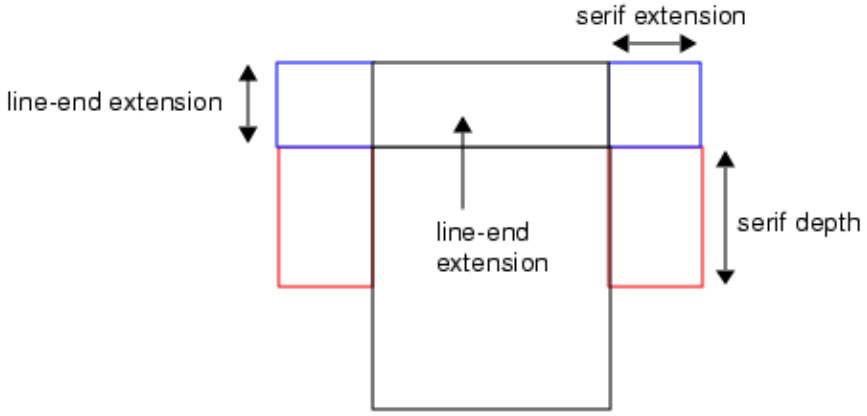
Figure 5-6. OPCLINEEND Example

```
...
LAYER POLY 15
polyCorrection = OPCLINEEND POLY
   WIDTH <0.2 END 0.02 SERIF 0.02 0.1 //default
   WIDTH <0.2 HEIGHT >0.1
      SPACE <=0.2
      END 0.005 SERIF 0 0.115
   WIDTH <0.2 >= 0.1 HEIGHT >0.1
      SPACE <=0.2 OPPOSITE EXTENDED 0.02
      END 0.005 SERIF 0 0.115
   WIDTH <0.2 HEIGHT >0.1
      SPACE >0.2 <=0.25
      END 0.01 SERIF 0.01 0.11

DRC CHECK MAP polyCorrection 115 //Output to layer 115

//Alternative brace-enclosing format
polyCorrection {OPCLINEEND POLY
   WIDTH <0.2 END 0.02 SERIF 0.02 0.1 //default
   ...
}
```

Line-end Corrections

You define line-end corrections in terms of line-end extension, serif extension, and serif depth.

Figure 5-7. Movement Categories for the OPCLINEEND Command



You map the line-end conditions to the line-end corrections by creating rule sets. In general, a rule set contains a default rule and one or more correction rules. The default rule defines the maximum OPC. Correction rules define the actual OPC treatment required for line-ends that would introduce spacing violations if they were corrected using the default rule.

Example 5-8 shows a rule set composed of the default rule plus two correction rules.

Figure 5-8. Default and Correction Rules

```
X {OPCLINEEND POLY
   //default rule
   WIDTH <0.3 HEIGHT >0.2                    END 0.02  SERIF 0.002 0.1
   //correction rule 1
   WIDTH <0.3 HEIGHT >0.2 SPACE <=0.3        END 0.005 SERIF 0     0.115
   //correction rule 2
   WIDTH <0.3 HEIGHT >0.2 SPACE >0.3 <=0.35 END 0.01  SERIF 0.01  0.11
}
```

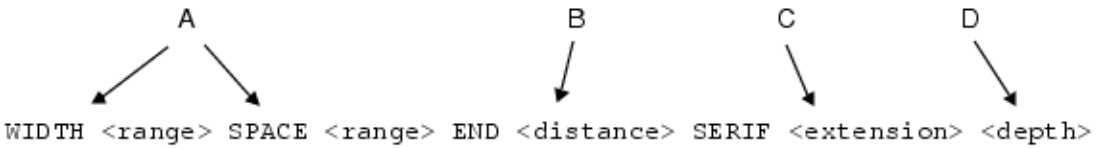
Relation of Process Dimensions and OPCLINEEND Rules

Depending on your process and design constraints, you may use as many as three criterion to describe line-end treatment:

- line-end extension
- serif extension or line widening
- serif height constant

When you develop Calibre OPCLINEEND rules, all the information contained in these tables (and more) is represented by a single set of rules. Figure 5-9 shows how the information in a correction rule relates to the original data in table format:

Figure 5-9. Rule Components



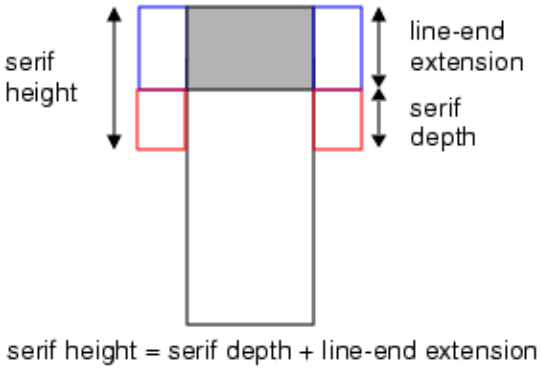
A. WIDTH and SPACE constraints define the row and column in each table

B. END extension distance (line-end movement) from the line extension table

C. SERIF extension (width) from the serif extension table

D. SERIF depth from the serif height table minus the END distance for this rule

Figure 5-10. Serif Height

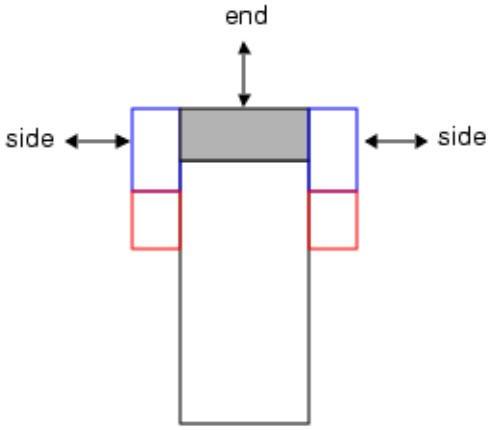


Serif height, which for your process may be defined as a constant value or using a serif height table, does not translate directly into a single value in an OPCLINEEND rule. With Calibre OPCLINEEND you define the serif depth and the line-end extension; these two together define the serif height.

You must also keep in mind that while each table rule represented within the OPCLINEEND command is relative to the same WIDTH and SPACE pair, the pair does not always refer to the same physical spacing measurement. The OPCLINEEND command measures spacing to the ends and sides of each line:

- At the end of the line, which defines the line-end extension and serif depth.
- Spacing to the sides of the line-end, which define the serif extensions on both sides

Figure 5-11. Measuring Spacing



Because of this, a given line-end may have its extension and serif depth defined by one OPCLINEEND command and its serif width defined by one or more different OPCLINEEND commands.

For each line-end, the OPCLINEEND command also checks for OPC spacing violations in these same directions to avoid MRC conflicts.

Avoid Spacing Violations After Treatment

Default rules define corrections required for all line-ends, but Calibre OPCLINEEND avoids post-bias errors by checking for spacing violations after applying the default correction.

The OPCLINEEND command measures the external distance between the generated OPC geometries and other geometries, including the other OPC geometries. If it encounters a spacing violation, as defined by a correction rule, OPCLINEEND removes any initial corrections in the direction of the violation, and replaces them with corrections defined by the END and SERIF arguments for the correction rule that correspond to the spacing violation condition.

Line-ends are always found on the in_layer. You control where OPCLINEEND looks for spacing violations through the layers you supply in the command statement.

- If you supply an in_layer without a reference_layer, the command checks for spacing violations between the line-end and other geometries on the in_layer.
- If you supply an in_layer plus a reference_layer, the command checks for spacing violations in two places: between the line-end on the in_layer and edges on the reference_layer and between the line-end on the in_layer and other corrected line-ends on the in_layer. Note that the reference_layer should include all geometry from the in_layer plus any additional geometry that must be considered for space measurements.

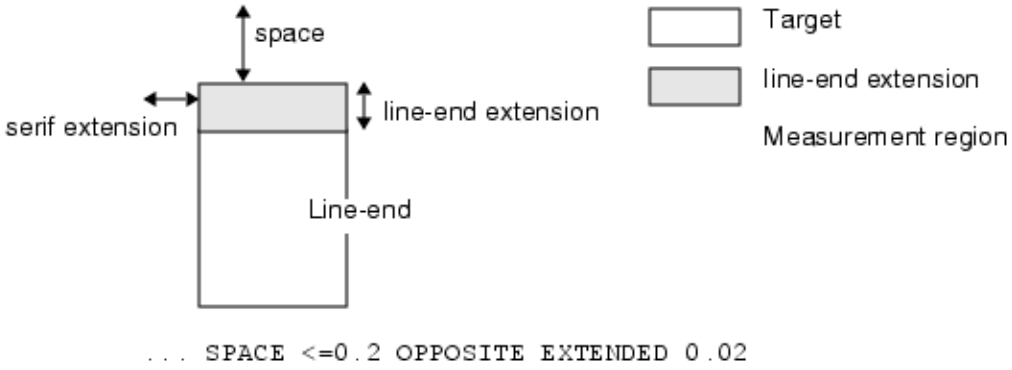
The OPCLINEEND command uses the SPACE violation condition to calculate and correct for two types of spacing violations, line-end OPC spacing violations and serif OPC spacing violations. These violation types and their resolution methods are described here.

Line-end Spacing Violations

Calibre OPCLINEEND identifies line-end spacing violations by generating a measurement region for the line-end edge after it has been extended by the default rule.

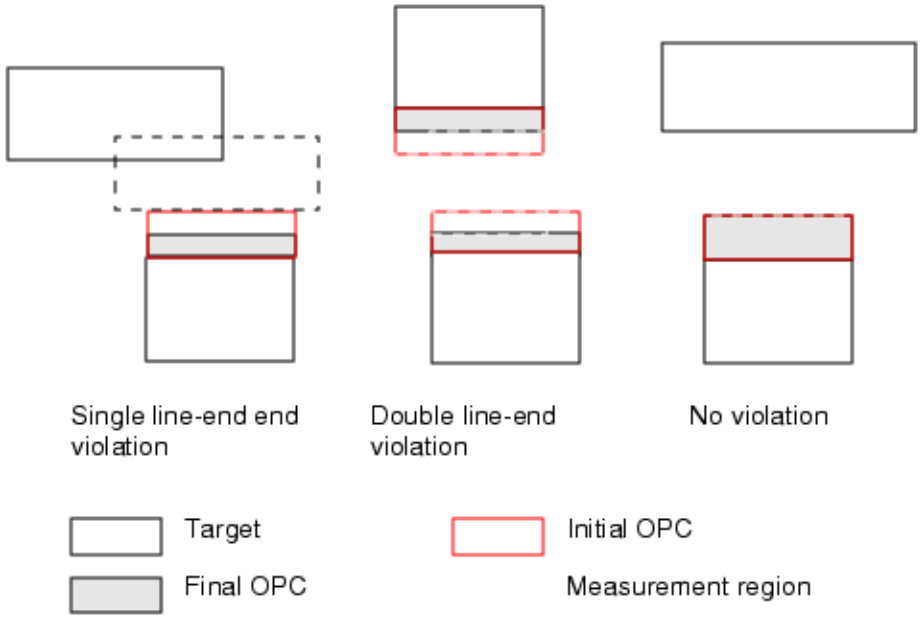
Figure 5-12 shows the OPPOSITE EXTENDED measurement region used for identifying line-end spacing violations as specified by the bit of code shown.

Figure 5-12. The OPPOSITE EXTENDED Measurement Region



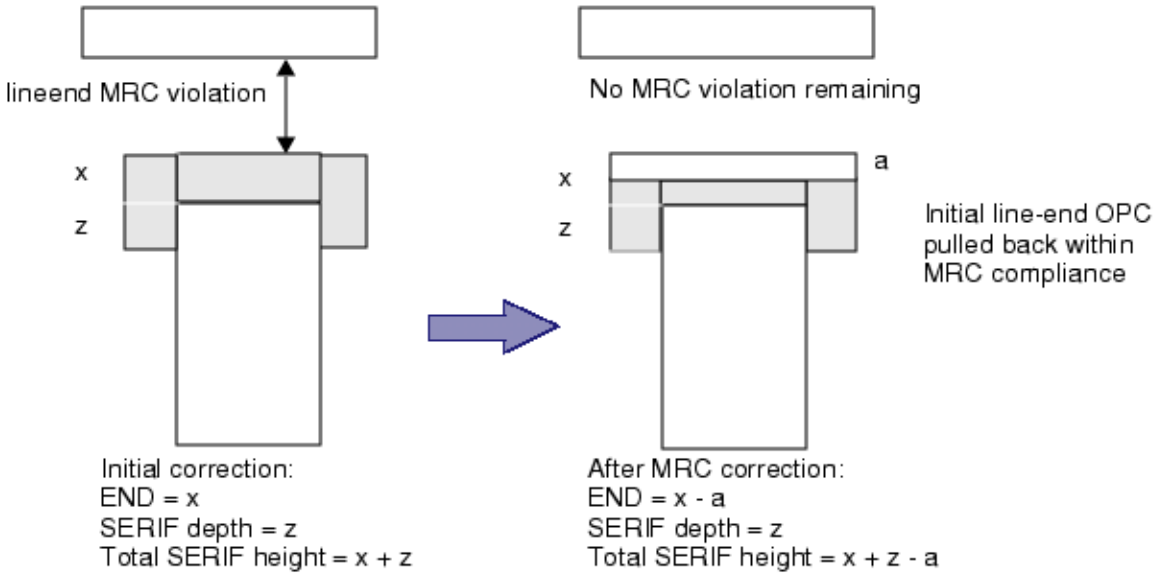
If the system finds geometries inside the measurement region on the same layer or in the ref_layer, a violation is said to exist. Calibre OPCLINEEND corrects that violation by modifying the line-end correction to agree with the line-end extension defined in the correction rule. The new line-end extension values must be smaller than the initial ones.

Figure 5-13. Spacing Violations



If the line-end has serifs, modifying the line-end correction also impacts the serif height. Figure 5-14 shows the results of correcting the space violation without also adjusting the serif depth. Note that the total serif height is shorter after the line-end extension is reduced to avoid the line-end OPC spacing violation.

Figure 5-14. Correcting Line-end Spacing Violations Without Serif Depth Adjustments

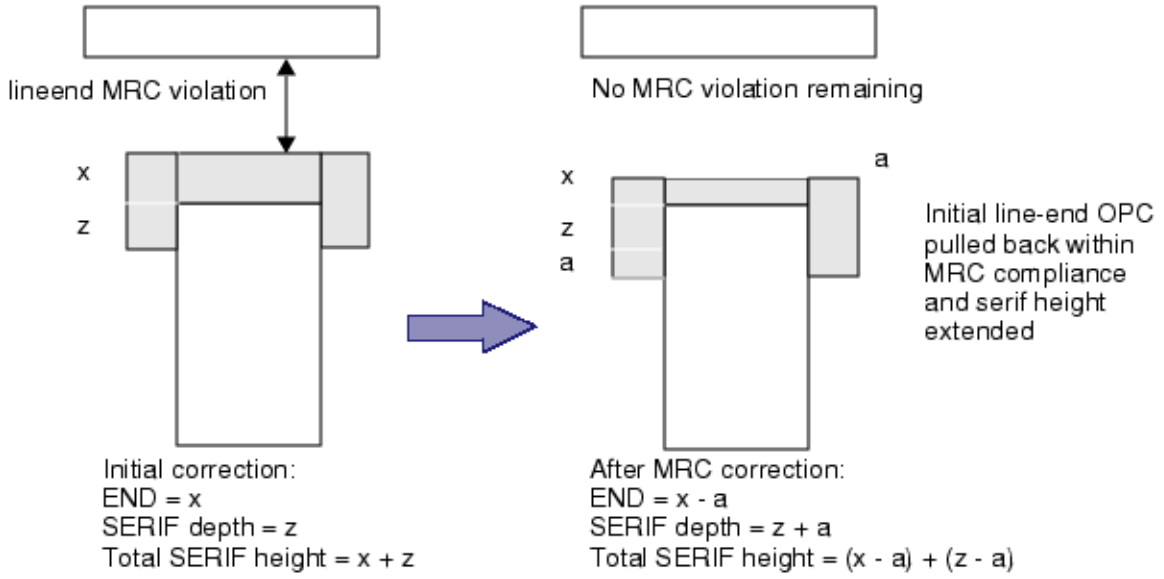


You can control the impact on the serif height by writing a correction rule that includes new values for both the END and the SERIF depth. When a line-end has serifs, OPCLINEEND corrects the violation by modifying both the width of the line-end correction and the depth of the serif to agree with the correction rule line-end extension and serif depth, respectively.

You would use this technique when it is desirable either to adjust the total serif height or to keep the serif height fixed (for example, to ensure that the serif height remains greater than the minimum width allowed by Calibre nmDRC).

You can ensure that the serif height remains the same by increasing the serif depth by the same amount by which the line-end extension is reduced. In order to do this, the system allows for specifying a greater serif depth value after the spacing measurement. Figure 5-15 shows the results of correcting the space violation by adjusting both the line-end extension and the serif depth.

Figure 5-15. Correcting Line-End Spacing Violations With Serif Depth Adjustments



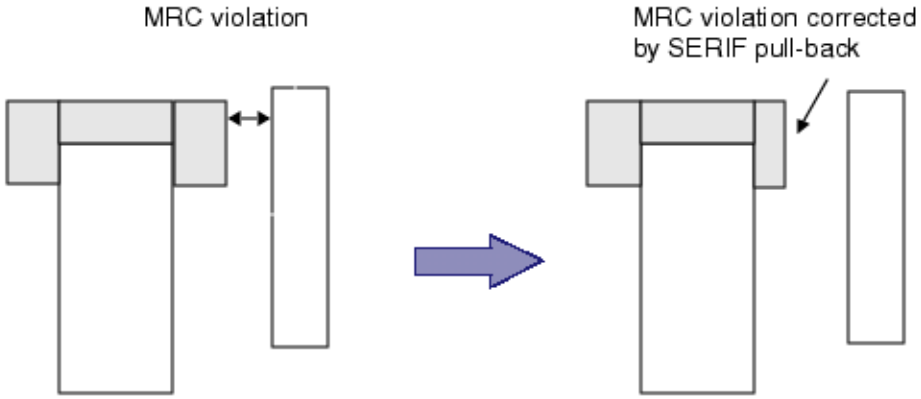
Note

When correcting a line-end spacing violation, OPCLINEEND ignores the serif extension value in the correction rule. It modifies the serif extension only when it discovers a serif violation and corrects the violations using the serif extension values for the rule that applies.

Serif Spacing Violations

Calibre OPCLINEEND identifies serif spacing violations by generating a measurement region for each serif after it has been extended by the default rule. Figure 5-16 shows the OPPOSITE EXTENDED measurement regions used to identify serif spacing violations for a generic hammerhead.

Figure 5-16. Finding Serif Spacing Violations With OPPOSITE EXTENDED

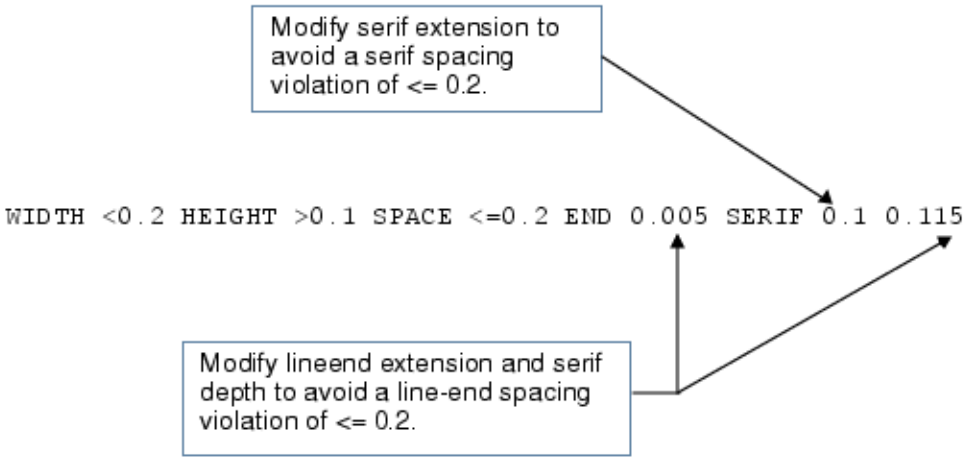


If the system finds geometries inside the measurement region on the same layer or on the ref_layer, a serif violation is said to exist. Calibre OPCLINEEND corrects that violation by modifying the serif width to agree with the corrected serif extension. Notice how the serif extension in Figure 5-16 is adjusted only on the side with the violation. The OPCLINEEND command treats each serif as a separate entity. The new serif extension value must be smaller than the initial one, possibly as small as zero. The line-end extension and serif depth are unaffected.

How Opposing Edges Affect Corrections

Because spacing violations are likely to be different in all three directions, Calibre OPCLINEEND may draw from three different correction rules to arrive at the final line-end treatment; one for each violation.

When you design correction rules, keep in mind that these are multi-purpose rules. They define the corrections for line-end spacing violations and the corrections for serif spacing violations.



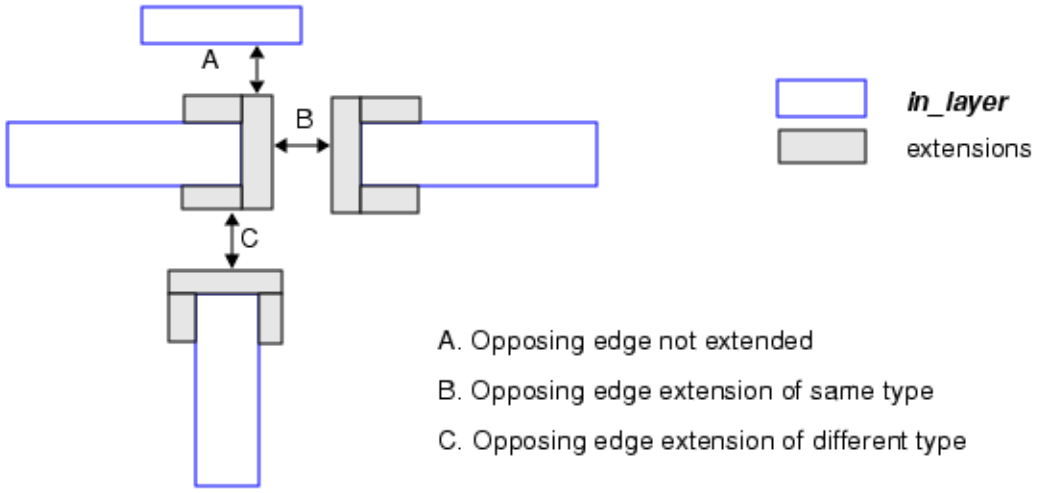
Extension and depth values are dependent on the type of edge directly opposite the fragment being moved. There are three cases:

- The opposing edge is not extended.
- The opposing edge is an extension of the same type.
- The opposing edge is an extension of a different other type.

When a correction rule contains a single value for end, or serif extension, or serif depth, that value applies to all three cases. When it contains two values, the first is used for cases in which the opposing edge is not extended and the second is used for all other situations.

Each is illustrated in Figure 5-17.

Figure 5-17. Extension Conditions



Applying the same correction to case A as to case C results in a wider spacing between the case C edges than the case A edges. This is because both edges pull back in case C and only one edge pulls back in case A.

To avoid this discrepancy, you can define up to three different corrections for each of the three categories of movement (line-end extension, serif extension, and serif depth), one for each case. The values are order dependent:

- **First Value —** Used when the opposing edge is not extended. This value is also used as the default for the other values when they are not specified.
- **Second Value —** Used if the opposing edge is an extension of the same type. (That is, both edges are line-end extensions or both are serifs.)
- **Third Value —** Used if the opposing edge is an extension of the same type, where both edges are line-end extensions or both are serifs. If the third value is not specified, the second value is used by default.

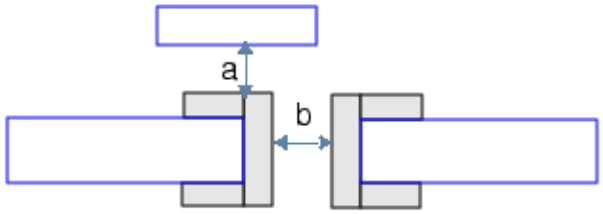
You specify actual extension distances based upon the spacing condition that exists after moving the edge the default amount. Also note that if you specify multiple values for serif extension, you must specify the same number of values for serif depth. In Example 5-18, and as illustrated in Figure 5-19, the four values after the keyword SERIF are respectively interpreted as:

1. serif extension opposite unextended edge
2. serif extension opposite extended edge of any type
3. serif depth opposite unextended edge
4. serif depth opposite extended edge of any type

Figure 5-18. Correction Rules to Anticipate Effects of Opposing Edges (Code)

```
X {OPCLINEEND POLY
   WIDTH <= 0.2                        END 0.15     SERIF 0.15 0.3
   WIDTH <= 0.2 SPACE <= 0.05          END 0 0.05   SERIF 0.00 0.05 0.3 0.3
   WIDTH <= 0.2 SPACE > 0.05 <= 0.10 END 0.0 0.10 SERIF 0.05 0.10 0.3 0.3
   WIDTH <= 0.2 SPACE > 0.10 <= 0.15 END 0.10     SERIF 0.10 0.3
}
```

Figure 5-19. Correction Rules to Anticipate Effects of Opposing Edges



In Figure 5-19, the default extension for both line-ends and serifs is 0.15. The depth is 0.3. The minimum space you must enforce is 0.2 and the input shapes are drawn on a grid of 0.05. In Example 5-18, consider the third rule for the interval > 0.05 <= 0.10:

1. The distance between the serif and the opposing non-extended edge is 0.10. The default movement of 0.15 moved the edge too far to enforce a spacing of 0.2, so you know the correction rule must be less than 0.15. You can calculate that the corrected extension should be 0.05, which is the default extension, 0.15, minus the distance by which the edge overshot the mark, (0.20 -0.10).
2. The distance between the serif and the opposing serif is also 0.10. Since both edges are pulled back by the same amount, you can calculate that the corrected extension should be 0.10, which is the default extension, 0.15, minus half the distance by which the two edges overshot the mark, (0.20 - 0.10)/2.

OPCLINEEND Correction Rules and Summary

In general, a rule set contains a default rule and one or more correction rules. The default rule defines the maximum OPC. Correction rules define the OPC treatment required for line-ends that would introduce spacing violations if they were corrected using the default rule. Example 5-20 shows a rule set comprising the default rule plus two correction rules.

Figure 5-20. OPCLINEEND Rule Structure

```
X {OPCLINEEND POLY
   WIDTH <0.3 HEIGHT >0.2
      END 0.02  SERIF 0.002 0.1  //default rule
   WIDTH <0.3 HEIGHT >0.2 SPACE <=0.3
      END 0.005 SERIF 0 0.115    //correction rule
   WIDTH <0.3 HEIGHT >0.2 SPACE >0.3 <=0.35
      END 0.01  SERIF 0.01 0.11  //correction rule
}
```

All three rules in the example apply to poly line-ends of width less than 0.3 and height greater than 0.2. In the default rule, the line-end extension is 0.02, the serif extension is 0.02, and the serif depth is 0.1.

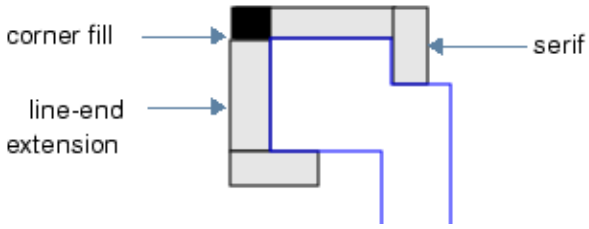
The first correction rule is applied when the default rule introduces a spacing error such that the spacing between adjusted edges is less than or equal to 0.3. For edges that meet the spacing condition, the line-ends are moved 0.005, and no serif would be formed because the serif extension is 0. The serif depth of 0.115 does not matter in this case because of the 0 value for serif extension.

The second correction rule is applied when the default rule introduces a spacing error of the interval > 0.3 <= 0.35. You can see what the line-end and serif values are in the example for this case.

Notice the intervals of the SPACE condition are mutually exclusive. This must be true in any rule set.

In the case where line-end edges are abutting, no serif is generated at the abutting corners. Instead, corner gap filling is performed for the line-end extension geometries at such corners. See Figure 5-21.

Figure 5-21. Corner Filling



Rule Development with OPCLINEEND

The OPCLINEEND command modifies the shape of line-ends.

It provides one rule set for each set of width constraints. Each rule set contains one default rule. Unless your rules contain only one spacing constraint, the rules must also contain one correction rule for each spacing constraint. The basic syntax using all arguments is:

```
WIDTH <range> HEIGHT <value> SPACE <range> END <values> SERIF <values>
```

Rules are written in two sections, default and corrections rules.

Default Rules

Each rule set begins with one and only one default rule. The default rule is always the first rule in the rule set. It defines a condition, based on line-end width and (optionally) height, and the treatment to apply to all line-ends that meet that condition expressed in terms of extension and serif dimensions. The default rule is required. Note that the default rule does not contain a SPACE condition. The default rule defines the maximum OPC. The syntax for a default rule is as follows:

```
WIDTH <range> END <max distance> SERIF <extension> <depth>
```

The END and SERIF extension values in the default rule define the maximum movements defined for that width. The SERIF depth, which is typically not represented in a rules table, is calculated differently depending on whether serif height is constant or varies with space and width conditions:

- If the serif height is constant, the SERIF depth is the constant serif height minus the maximum end movement
- If the serif height varies from cell to cell in your rules table, SERIF depth is the serif height table values minus the maximum end movement.

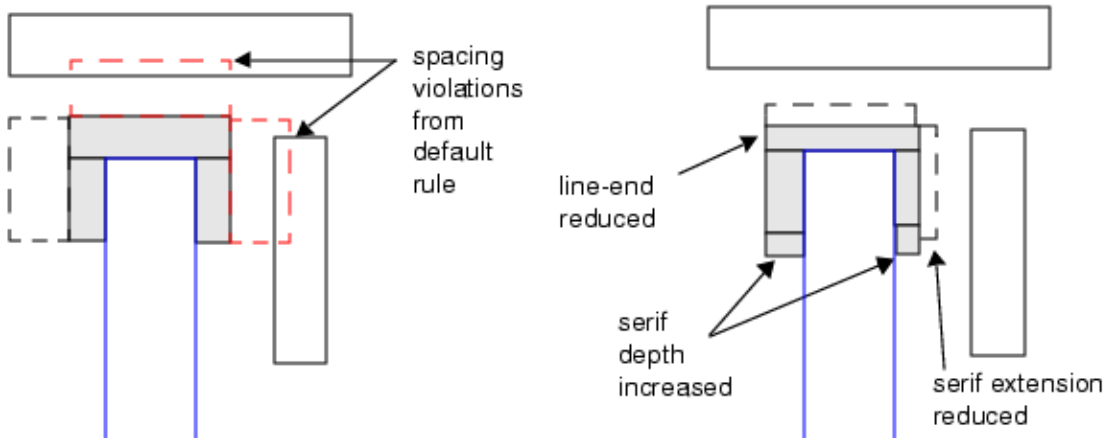
Correction Rules

Correction rules define the OPC treatment required for line-ends that would introduce spacing violations if they were corrected using the default rule. After applying the default rule, if OPCLINEEND encounters a spacing violation, it replaces the default corrections with the corrections defined in the correction rule.

Correction rules adhere to the same general syntax as default rules, except that they must contain the additional SPACE violation condition and they support lists of values for END and SERIF conditions. The END extension, SERIF extension, and SERIF depth can each contain up to three values. These account for up to three different spacing violations a default rule can introduce. These are discussed in detail in the section entitled “How Opposing Edges Affect Corrections” on page 202.

Figure 5-22 illustrates this process. Notice that the movement values in the correction rule are specified with respect to the original line-end.

Figure 5-22. Applying Correction Rules



The following syntax is for a simple correction rule:

```
WIDTH <range> SPACE <range> END <distance_list>
   SERIF <extension_list> <depth_list>
```

The SPACE constraint in a correction rule represents both types of spacing violations:

- Line-end OPC violations
- Serif OPC violations

Because OPCLINEEND checks for violations after applying the default rule, you must adjust the spacing values in the tables before you can use them as correction rules. For instructions on adjusting the spacing values, refer to the tables in the section entitled "Make your Conversion Step by Step" on page 209.

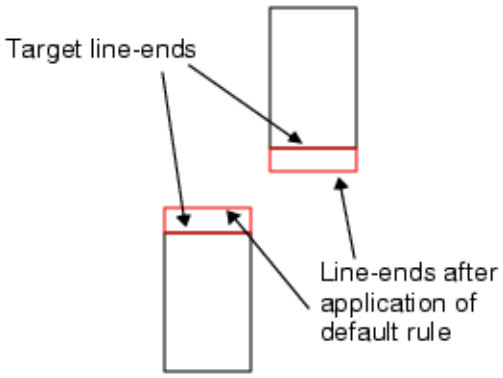
You create one correction rule for every spacing value found in any of the rules tables represented by the rule set.

Avoiding Problems

Use the OPPOSITE EXTENDED metric with SPACE to avoid kissing corners and other problems caused by diagonally opposite geometries. Diagonally opposite geometries can cause problems with serifs as well as simple extensions, and so the OPPOSITE EXTENDED metric is equally relevant to all parts of Calibre OPCLINEEND. If you use OPPOSITE EXTENDED for one correction rule for a given width, you must use the same metric and value for all correction rules for that width.

Figure 5-23 demonstrates this problem:

Figure 5-23. Avoidance of Kissing Corners



Use the optional HEIGHT constraint to direct OPCLINEEND to ignore stubby line-ends which are defined as shorter than the specified height.

Table Conversion to OPCLINEEND Rules

This section describes some planning thoughts and the sequence of steps to convert a table of values into valid OPCLINEEND syntax.

Plan your Conversion

In most cases you can perform all the corrections indicated by all relevant rules tables with a single OPCLINEEND command. In order to successfully combine rules tables, the tables must meet one of the following conditions:

- All tables use the same width and spacing intervals.
- The tables use different intervals for width and/or spacing, but the intervals are mutually exclusive.
- The tables use different intervals for width and/or spacing, and the intervals for one table map cleanly into the intervals for the other table.

There are two assumptions behind the default rule/correction rules structure:

- You can do a better job anticipating spacing violations (either due to less than the minimum spacing allowed after a RET operation or to nearby geometries impinging on corner space) after the move than before the move.
- For every line width, there is a maximum correction.

Based on these assumptions, you apply that maximum correction if OPCLINEEND discovers a spacing violation you replace the maximum correction with the greatest correction that does not result in a spacing violation, as defined in the correction rule for that spacing.

To write the rules you must know three things:

1. The number of line-ends corrected for each space that is examined.
2. The maximum correction for each line width.
3. The minimum spacing to enforce for line-ends and serifs.

The number of line-ends corrected for each space is really a question about the assumptions under which the table was created. If the assumption is that line-ends are opposite (for line extensions) and adjacent (for serif extensions) to other line-ends, both must be corrected when OPCLINEEND encounters a violation. If the assumption is that line-ends are opposite and adjacent to other types of lines, only the line-end is corrected.

You can check this assumption by examining the rules tables.

- If the spacing interval[1]equals the correction interval[2], the number of edges to be corrected is 1.
- If the spacing interval is twice the correction interval, the number of edges to be corrected is 2.

The maximum correction for each line width is the maximum table value for that width.

Knowing the number of line-ends corrected for each space, N, makes it easy to calculate the minimum spacing to enforce for line-ends and serifs. In the rules table, for each spacing/width combination, calculate the enforced spacing as:

```
Spacing - (N * correction)
```

- In most tables, the value is the same for every space or width pair in the table. This value is the minimum spacing.
- If the values are not the same for every space or width pair in the table, check to see if they are the same for each width. Use the smallest value as the minimum spacing value.

Make your Conversion Step by Step

1. Convert table values into microns.

1. The spacing interval is the step size between each successive spacing interval.
2. The correction interval is the stepsize between corrections. Typically there is a consistent step size between corrections, such as 0.05 micron or 0.1 micron.

Table 5-1. Line-End Extension Table in User Units

Spacing \ Line Width	0.1	0.2
0	0	0
0.8	0.05	0.05
0.9	0.10	0.10
1.0	0.15	0.15
1.1	0.20	0.20
1.2	0.25	0.25
1.3	0.30	0.30

Table 5-2. Serif Extension Table in User Units

Spacing \ Line Width	0	0.1	0.15	0.20
0	0	0	0	0
0.8	0	0	0	0
0.9	0	0	0	0
0.10	0	0.05	0.05	0.05
1.1	0	0.10	0.10	0.10
1.2	0	0.15	0.15	0.10
1.3	0	0.20	0.20	0.10
1.4	0	0.25	0.20	0.10
1.5	0	0.30	0.20	0.10

2. Analyze your process and tables and calculate values for:

   a. N = The number of edges to be corrected for each space that is examined. In this example, N = 2.

   b. C = The maximum correction(s) for each line width. In this example, the values are:

   Line extension — C^{all} = 0.30 Serif Extension — $C^{0.1}$ = 0.30, $C^{0.15}$= 0.20, $C^{0.2}$= 0.10

c. S_L = The minimum spacing for line-ends. In this example S_L = 0.7.

d. S_S = The minimum spacing for serifs. In this example S_S = 0.9.

3. For each space/width pair, calculate the spacing constraint to be measured after the default rule is applied by subtracting ($N*C^{width}$) from the original spacing. Then rewrite the tables using this new spacing. In the following tables, the diagonal line through a table cell enables you to compare the line-end extension (upper half) with the spacing for that cell (lower half).

Table 5-3. Spacing Adjustment for Line-End Extensions

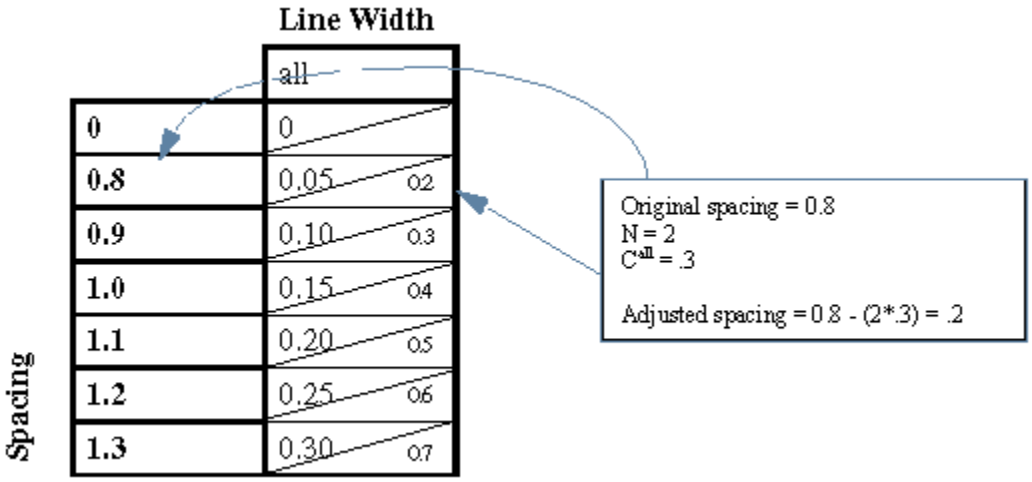


Table 5-4. Line-End Extensions Based on Adjusted Spacing

Line Width

Spacing	all
0	0
0.2	0.05
0.3	0.10
0.4	0.15
0.5	0.20
0.6	0.25
0.7	0.30

Table 5-5. Spacing Adjustment for Serifs

Spacing \ Line Width	0.1	0.15	0.20
0.8	0	0	0
0.9	0 / 0.3	0 / 0.5	0 / 0.7
0.10	0.05 / 0.4	0.05 / 0.6	0.05 / 0.8
1.1	0.10 / 0.5	0.10 / 0.7	0.10 / 0.9
1.2	0.15 / 0.6	0.15 / 0.8	0.10 / 1.0
1.3	0.20 / 0.7	0.20 / 0.9	0.10 / 1.1
1.4	0.25 / 0.8	0.20 / 1.0	0.10 / 1.2
1.5	0.30 / 0.9	0.20 / 1.1	0.10 / 1.3

Table 5-6. Serifs Based on Adjusted Spacing

Spacing \ Line Width	0.1	0.15	0.20
0.3	0	0	0
0.4	0.05	0	0
0.5	0.10	0	0
0.6	0.15	0.05	0
0.7	0.20	0.10	0
0.8	0.25	0.15	0.05
0.9	0.30	0.20	0.10
1.0	0.30	0.20	0.10
1.1	0.30	0.20	0.10
1.2	0.30	0.20	0.10
1.3	0.30	0.20	0.10

4. Expand each correction cell into four cells, as shown in Table 5-8 on page 214 and Table 5-9 on page 215. In step 6, you convert the spacing and width categories into

ranges of values. You can fill in the cells with three different movement distances, one for each of the three conditions as follows:

a. The opposing edge is not extended.

b. The opposing edge is an extension of the same type.

c. The opposing edge is an extension of a different type.

5. As mentioned previously, rules tables are typically created with an assumption about how many edges are moved for a given space. In step 2 you calculated this as N (for this example N=2). In this step you calculate the movement for all cases (not just the case assumed) and use this assumption to check your rules against the original table.

   Tables 5-4 and 5-6 show the spacing after the default rule has been applied, plus the recommended movement from the original table. If the spacing is less than the minimum spacing to enforce, you must write a correction rule for that space. You do this by calculating the distance by which the default rule overshot the desired correction (V), then figuring out the correct movement for the edges. See Table 5-7 for equation notation.

   a. When the opposing edge is not extended:

      ```
      V1 = SM - S
      C1 = Cd-V
      ```

      (If N = 1, this should be the same as the value from the original table.)

   b. When the opposing edge is an extension of the same type:

      ```
      V2 = SM - S
      C2 = Cd-V/2
      ```

      (If N = 2, this should be the same as the value from the original table.)

   c. When the opposing edge is an extension of another type:

      ```
      SLS = the greater of SL and SS
      V3 = (SLS- S)
      ```

      You choose how to distribute the movement between the serif and the line-end. For example, to subtract V/2 from the default correction for both tables[3]:

      ```
      C3 = Cd-V/2
      ```

3. Base this decision on knowledge of your process. As a rule of thumb, the D correction for the line-end plus the D correction for the serif extension of the same space should add up to V.

Table 5-7. Notation Used in Equations

Known Values	N = Number of edges assumed to be corrected for each space. Cd = Default correction for the table. SL = Minimum spacing for line-end extensions. SS = Minimum spacing for serif extensions S = Spacing after default rule was applied.
Values to Calculate	V= Violation — distance by which correction overshot desired results. SM = Minimum spacing for the table. If for a line-end extensions table, this is SL. For a serif extensions table this is SS. SLS = Minimum spacing for line-end opposite serif extensions. C = Correction.

Table 5-8. Modified Line-End Extensions Table

Spacing \ Line Width	all		
0.1	0		
	0	0	0
0.2	0.05		
	0	0.05	0
0.3	0.10		
	0	0.10	0
0.4	0.15		
	0	0.15	0.05
0.5	0.20		
	0.10	0.20	0.1
0.6	0.25		
	0.20	0.25	0.15
0.7	0.30		
	0.30	0.30	0.20

$S_L = 0.7$
$N = 2$
$C_d = 0.3$
$S_{LS} = 0.9$
$V_1 = V_2 = 0.7 - 0.3 = 0.4$
$V_3 = 0.9 - 0.3 = 0.6$
$C_1 = 0.3 - 0.4 => 0$
$C_2 = 0.3 - 0.2 = 0.1$
$C_3 = 0.3 - 0.3 = 0$

Table 5-9. Modified Serif Extensions Table

Spacing \ Line Width	0.1			0.15			0.2		
0.3	0			0			0		
	0	0	0	0	0	0	0	0	0
0.4	0.05			0			0		
	0	0.05	0.05	0	0	0	0	0	0
0.5	0.10			0			0		
	0	0.10	0.10	0	0	0	0	0	0
0.6	0.15			0.05			0		
	0	.015	0.15	0	0.05	0.05	0	0	0
0.7	0.20			0.10			0		
	0.10	0.20	0.20	0	0.10	0.10	0	0	0
0.8	0.25			0.15			0.05		
	0.20	0.25	0.25	0.10	0.15	0.15	0	0.05	0.05
0.9	0.30			0.20			0.10		
	0.30	0.30	0.30	.020	0.20	0.20	0.10	0.10	0.10

Where:
Width = 0.15
Space = 0.7
$S_S = 0.9$
$N = 2$
$C_d = 0.2$
$V = 0.2$
$C_1 = 0$
$C_2 = 0.1$
$C_3 = 0.1$

6. Table 5-10 shows the conversion of the spacing and width categories into ranges of values:

Table 5-10. Final Line-End Extensions Table

Spacing \ Line Width	all		
<=0.1	0		
	0	0	0
>0.1<=0.2	0.05		
	0	0.05	0
>0.2<=0.3	0.10		
	0	0.10	0
>0.3<=0.4	0.15		
	0	0.15	0.05
>0.4<=0.5	0.20		
	0.10	0.20	0.1
>0.5<=0.6	0.25		
	0.20	0.25	0.15
>0.6	0.30		
	0.30	0.30	0.20

Table 5-11. Final Serif Extensions Table

Spacing \ Line Width	>0.1<=0.15			>0.15<=0.2			>0.2		
<=0.3	0			0			0		
	0	0	0	0	0	0	0	0	0
>0.3<=0.4	0.05			0			0		
	0	0.05	0.05	0	0	0	0	0	0
>0.4<=0.5	0.10			0			0		
	0	0.10	0.10	0	0	0	0	0	0
>0.5<=0.6	0.15			0.05			0		
	0	.015	0.15	0	0.05	0.05	0	0	0
>0.6<=0.7	0.20			0.10			0		
	0.10	0.20	0.20	0	0.10	0.10	0	0	0
>0.7<=0.8	0.25			0.15			0.05		
	0.20	0.25	0.25	0.10	0.15	0.15	0	0.05	0.05
>0.8	0.30			0.20			0.10		
	0.30	0.30	0.30	0.20	0.20	0.20	0.10	0.10	0.10

7. Select a value for SERIF height. If your process calls for a single serif height, use that value. If your process uses a serif height table to define the serif height, select the value that corresponds to the greatest spacing. For this example, a constant value of 0.5 is assumed.
8. Use the OPPOSITE EXTENDED metric with SPACE to catch corner spacing violations. Calculate the extension to use for this metric as the maximum vertical and horizontal clearance required between diagonally opposite corners. (For more information on this concept, refer to “Avoiding Problems” on page 207.) For this example, 0.3 is used.
9. Create a rule set for the smallest width in either table.
    a. Create the default rule with:
        - END = maximum movement for that width in Table 5-8 on page 214.
        - SERIF extension = maximum for that width in Table 5-9 on page 215.
        - SERIF depth = serif height from step 7 minus END.

```
WIDTH >0.1<=0.15 END 0.3 SERIF 0.3 0.2
```

b. Create one correction rule for each spacing constraint in each table.

- WIDTH = WIDTH from the default rule and SPACE = SPACE from the table.
- OPPOSITE EXTENDED = as defined in step 8.
- END = movements for that WIDTH and SPACE in the Modified Line-End Extensions table. See Table 5-10 on page 216.
- SERIF extensions = movement for that WIDTH and SPACE in the Modified Serif Extension. See Table 5-11 on page 217.
- SERIF depth = serif height minus END. There must be the same number of depth values to match the number of serif extension values. In this example all are the same. (If serif height is not constant, find the value in the serif height table that corresponds to the WIDTH and SPACE values for the rule.) The rules for WIDTH > 0.1 <= 0.15 are shown in Example 5-24.

Figure 5-24. LINEENDBIAS Example

```
//default rule
lineendBias {OPCLINEEND POLY
    WIDTH > 0.1 <= 0.15 END 0.3 SERIF 0.3 0.2

// remainder are correction rules
    WIDTH > 0.1 <= 0.15 SPACE <= 0.1 OPPOSITE EXTENDED 0.3
        END 0.0 0.0 0.0 SERIF 0.0 0.0 0.0 0.50 0.50 0.50
    WIDTH > 0.1 <= 0.15 SPACE > 0.1 <= 0.2 OPPOSITE EXTENDED 0.3
        END 0.0 0.05 0.0 SERIF 0.0 0.0 0.0 0.50 0.45 0.45
    WIDTH > 0.1 <= 0.15 SPACE > 0.2 <= 0.3 OPPOSITE EXTENDED 0.3
        END 0.0 0.10 0.0 SERIF 0.0 0.0 0.0 0.50 0.40 0.40
    WIDTH > 0.1 <= 0.15 SPACE > 0.3 <= 0.4 OPPOSITE EXTENDED 0.3
        END 0.0 0.15 0.05 SERIF 0.0 0.5 0.5 0.50 0.35 0.35
    WIDTH > 0.1 <= 0.15 SPACE > 0.4 <= 0.5 OPPOSITE EXTENDED 0.3
        END 0.1 0.2 0.1 SERIF 0.0 0.10 0.10 0.40 0.30 0.30
    WIDTH > 0.1 <= 0.15 SPACE > 0.5 <= 0.6 OPPOSITE EXTENDED 0.3
        END 0.2 0.25 0.15 SERIF 0.0 0.15 0.15 0.30 0.25 0.25
    WIDTH > 0.1 <= 0.15 SPACE > 0.6 <= 0.7 OPPOSITE EXTENDED 0.3
        END 0.3 0.30 0.20 SERIF 0.1 0.20 0.20 0.20 0.20 0.20
    WIDTH > 0.1 <= 0.15 SPACE > 0.7 <= 0.8 OPPOSITE EXTENDED 0.3
        END 0.3 0.30 0.25 SERIF 0.2 0.25 0.25 0.20 0.20 0.15
    WIDTH > 0.1 <= 0.15 SPACE > 0.8 OPPOSITE EXTENDED 0.3
        END 0.3 0.30 0.30 SERIF 0.3 0.3 0.3 0.20 0.20 0.10}
```

10. Repeat the process for each additional WIDTH.

Irregularly Shaped Line-ends

When the data you pass to OPCLINEEND contains notched or stair-step shaped line-ends, the default behavior ignores those line-ends.

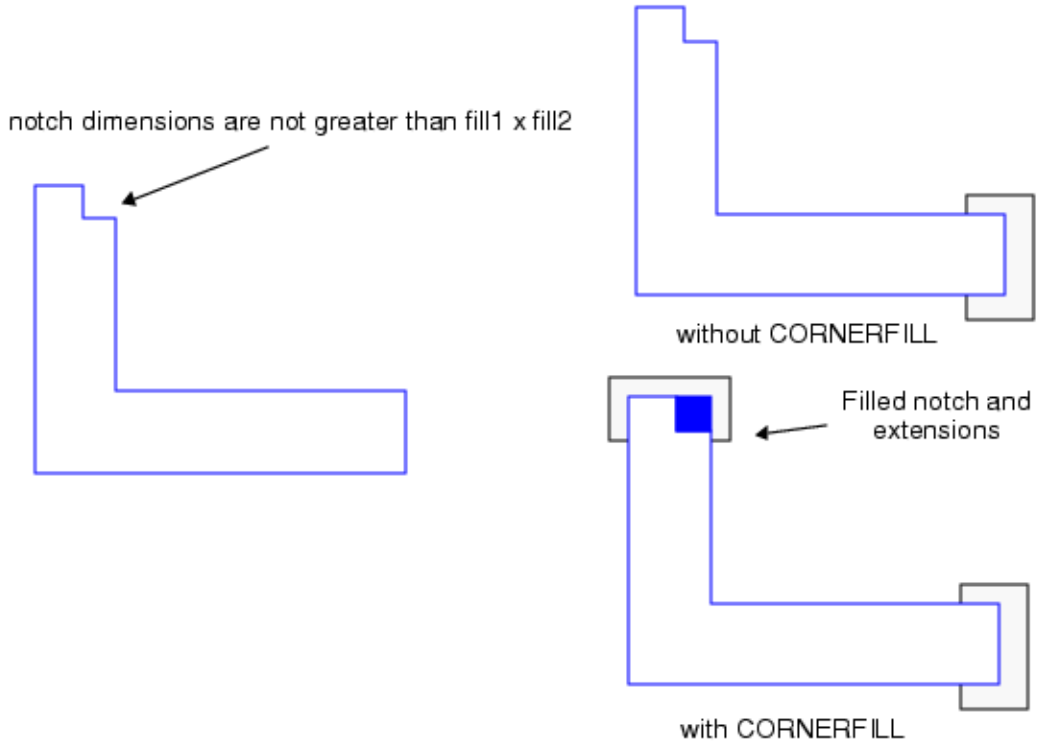
You can override this default behavior and force OPCLINEEND to apply extensions to the irregular line-ends by supplying the OPCLINEEND keyword.

When you specify CORNERFILL, you must define the maximum size of notch to fill.

CORNERFILL occurs as part of the line-end correction process. If a notched line-end satisfies the conditions specified for requiring correction, and you supply CORNERFILL, OPCLINEEND behaves as follows:

1. If notch dimensions are greater than fill1 x fill2, then stop.
2. If notch dimensions are not greater than fill1 x fill2, then fill the notch.
3. Apply extensions to the resulting line-end.

Figure 5-25. Correcting Notched Line-ends using CORNERFILL



To direct OPCLINEEND to fill and treat line-ends containing stair-step configurations, supply the ITERATIONS keyword with CORNERFILL. This forces OPCLINEEND to perform the specified number of iterations of notch filling before applying the line-end extension.

OPCLINEEND Example 1

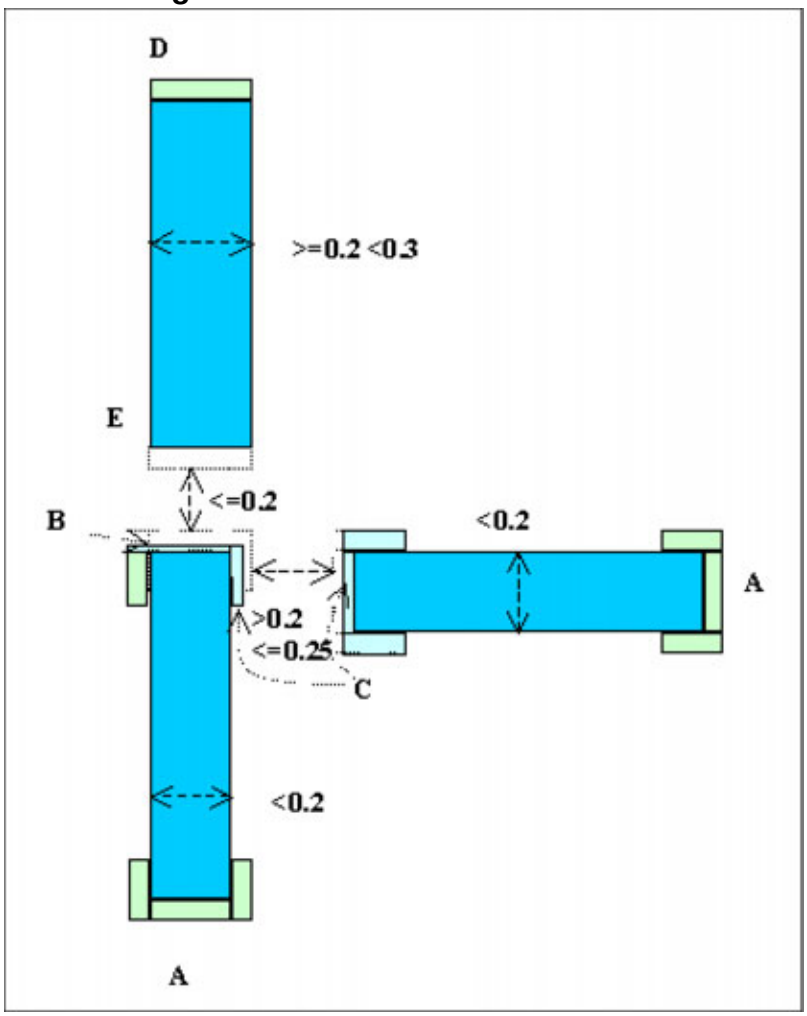
This example shows two rule sets. The first rule set applies to all line-ends that meet the WIDTH < 0.2 and HEIGHT > 0.1 conditions. The second rule set applies to all line-ends that

meet the WIDTH >= 0.2 < 0.3 and HEIGHT > 0.1 conditions. Both rule sets check and adjust for spacing violations.

Figure 5-26. Examples of the OPCLINEEND Command

```
X {OPCLINEEND POLY
   WIDTH <0.2 HEIGHT >0.1                      END 0.02  SERIF 0.02 0.1  //A
   WIDTH <0.2 HEIGHT >0.1 SPACE <=0.2          END 0.005 SERIF 0 0.115   //B
   WIDTH <0.2 HEIGHT >0.1 SPACE >0.2 <=0.25 END 0.01  SERIF 0.01 0.11 //C
   WIDTH >=0.2 <0.3 HEIGHT >0.1                END 0.02  SERIF 0 0       //D
   WIDTH >=0.2 <0.3 HEIGHT >0.1 SPACE <=0.2 END 0     SERIF 0 0       //E
}
```

Figure 5-27. OPCLINEEND Results



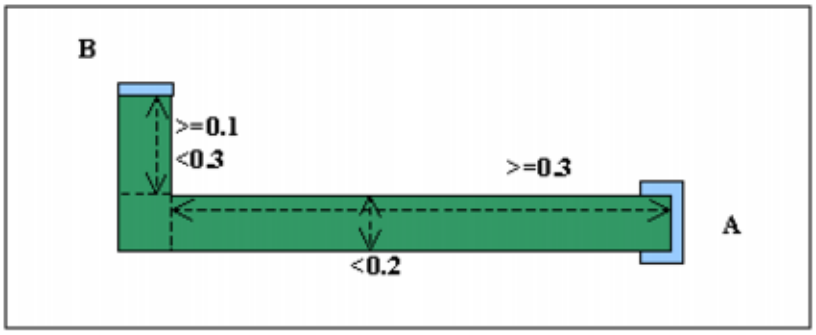
OPCLINEEND Example 2

This example shows the command used with two rule sets, neither of which adjusts for spacing violations.

Figure 5-28. Examples of the OPCLINEEND Command

```
X {OPCLINEEND MET1
   WIDTH <0.2 HEIGHT >=0.3              END 0.02  SERIF 0.02 0.1     //A
   WIDTH <0.2 HEIGHT >=0.1 <0.3         END 0.02  SERIF 0    0      //B
}
```

Figure 5-29. OPCLINEEND Results



Chapter 6
Calibre OPCBIAS Best Practices

Best practices improve Calibre OPCBIAS results.

Recommended Settings for Best OPCBIAS Results

These are recommended settings to improve results produced by the OPCBIAS command.

Table 6-1. Recommended Settings for OPCBIAS Results

Option	Default Setting	Recommended Setting	Reason for Recommendation/Notes
V2 (Processing Mode)	V1	V2	V2 processing mode provides higher scalability on Calibre® MTflex™ runs.
MINBIASLENGTH value	none	Set value to approximately 1.5x the minimum feature size	Setting value too short produces a lot of jogs, and setting it too long leads to excessive smoothing and incorrect sizing.
CLOSEREDGE	V2	V2	CLOSEREDGE V2 is an improved algorithm for treating short edges (<MINBIASLENGTH) generated by edge classification. We recommend that you use CLOSEREDGE V2 instead of CLOSERSYMMETRIC because the smoothing in CLOSEREDGE V2 is much closer to the results obtained from flat processing.
OPTION IMPLICIT_METRIC	NO	YES	This option enables the OPPOSITE EXTENDED metric for spacing rules that are created implicitly if the neighboring interval also uses the OPPOSITE EXTENDED metric.

Table 6-1. Recommended Settings for OPCBIAS Results (cont.)

Option	Default Setting	Recommended Setting	Reason for Recommendation/Notes
OPTION CAREFUL_MEASURE	NO	YES	If using large values with OPPOSITE EXTENDED, use OPTION CAREFUL_MEASURE YES. This option reduces the difference between hierarchical and flat runs, especially if large values of OPPOSITE EXTENDED are used. This option can increase runtime.
OPPOSITE EXTENDED value	none	Set value to approximately 1.1x the minimum feature size	This recommended value is large enough to detect other features but small enough not to create unnecessary interactions.

Appendix A
Calibre nmDRC Hierarchical Engine

The Calibre nmDRC hierarchical engine is the foundation for all Calibre RET solutions, and runs all Calibre programs hierarchically.

Calibre nmDRC Hierarchical Engine Overview and Workflow

Calibre nmDRC is a powerful layout design verification and correction tool that manipulates layer data and performs design and manufacturability rule checks quickly and efficiently. It serves as the starting point for all Calibre RET batch processes, coordinating data exchange between layout files and Calibre tools.

You begin with the Calibre nmDRC hierarchical engine, no matter which Calibre RET solution you plan to use.

Requirements

Before invoking the Calibre hierarchical engine, you must set your environment variables. Calibre tools require that the CALIBRE_HOME environment variable be set.

When you invoke the Calibre nmDRC hierarchical engine, you supply two forms of input:

- One or more layout databases — GDS databases containing the design layer or layers on which you perform OPC.
- One or more rule files — ASCII files composed of Standard Verification Rule Format (SVRF) statements. These statements control various functions:
  - Design environment
  - Loading data from the GDS database(s)
  - Layer preprocessing (sizing, creating and isolating new layers, and other tasks)

- Design and manufacturability checks
- Invoking batch RET processes
- Rule-based OPC
- Writing output data to a results database

This section introduces some basic concepts to take full advantage of Calibre functionality. For more information on SVRF rule files, refer to the section entitled SVRF Rule Files of this manual, and to the *Standard Verification Rule Format (SVRF) Manual*. For more information on the Calibre nmDRC hierarchical engine, refer to the *Calibre Verification User's Manual*.

Modes of Operation

The Calibre hierarchical engine can be invoked in batch mode by using the calibre command in the command line of a Linux®[1]shell window as described in "How to Invoke the Calibre Hierarchical Engine" on page 226.

Related Topics

Calibre Administrator's Guide [Calibre Administrator's Guide]

Standard Verification Rule Format (SVRF) Manual

Calibre Verification User's Manual

How to Invoke the Calibre Hierarchical Engine

Calibre is invoked on the command line from any Linux® terminal window for many Calibre RET solutions.

To invoke the Calibre hierarchical engine, enter the calibre command in the command line of a Linux shell window. The usage syntax is as follows:

```
$CALIBRE_HOME/calibre -drc [-hier]
   [-turbo [n]]
   [-turbo_litho [n]]
   [-turbo_all]

   rules
```

Arguments include:

- **-drc —** A required switch specifying DRC checking.
- **-hier —** An optional switch specifying hierarchical DRC checking (nmDRC-H).
- **-pto —** An optional switch enabling FullScale for Calibre nmBIAS.

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

- **-turbo *n* —** An optional argument that selects multi-threaded parallel processing for DRC. The turbo value specifies the number of CPUs to use for non-RET processes. If you do not specify the turbo value, the application uses the maximum number of CPUs on the machine. For more information on multi-threaded processing, refer to the *Calibre Verification User's Manual*.
- **-turbo_all —** An optional argument used in conjunction with -turbo or -turbo_litho. When the number of licenses specified in -turbo and -turbo_litho are not available, the Calibre hierarchical engine normally runs with fewer threads than requested. This command line option causes the Calibre hierarchical engine to exit if it cannot get the number of licenses specified.
- **-turbo_litho *n* —** An optional argument that selects multi-threaded parallel processing for batch operations invoked by any of the RET operations. The turbo_litho value specifies the number of CPUs to use for these batch processes. If you do not specify the turbo_litho value, the application uses the maximum number of CPUs on the machine.

  Note

  The number specified for -turbo_litho does not need to agree with the number specified for -turbo. For more information on multi-threaded processing, refer to the *Calibre Verification User's Manual*.

- **rules —** A required user-supplied argument specifying the name of the SVRF rule file. The SVRF file includes statements and operations to perform DRC, OPC, or verification and correction tasks. You can reference multiple rule files using INCLUDE inside the primary rule file, which you must specify when invoking the tool. For more information on rule files, refer to section B of this manual, and to the *Standard Verification Rule Format (SVRF) Manual*.

Calibre Results

Each Calibre run saves results and status information.

Results Database

This database stores the results of the Calibre run. The Calibre nmDRC hierarchical engine writes results to the database file specified with the DRC RESULTS DATABASE statement (detailed in the *Standard Verification Rule Format (SVRF) Manual*) in the SVRF rule file. If this is a GDS database, which is typical of OPC runs, the Calibre nmDRC hierarchical engine writes all the data to layer 0. Using the DRC CHECK MAP statement, you can assign different types of data to different layers and you can direct Calibre to write the results to different database files. For more information, refer to the sections entitled "DRC Statements" on page 246 and "Output Statements" on page 250 in this manual.

Calibre Transcript

The Calibre nmDRC hierarchical engine automatically generates a text transcript at run-time and sends the transcript to stdout, which is typically the shell window from which you invoked the application. During the Calibre run, the transcript displays event logs, warning messages, and summary information. This transcript documents the tasks the application performs: compiling the rule file, loading in the design data, and operating on that data. When you invoke any of the model-based batch tools, it also records the full setup file referenced and the optical and resist models used for simulation. It is good practice to save the transcript by redirecting the output to a file, which you can do when you invoke the Calibre nmDRC hierarchical engine. Methods you can use include:

- pipe and tee syntax:

  ```
  calibre -drc -hier -turbo_litho rulefile.drc | tee rulefile.log
  ```

- redirection:

  csh or tcsh:

  ```
  calibre -drc -hier rulefile.drc >& log
  ```

  ksh or bsh:

  ```
  calibre -drc -hier rulefile.drc > log 2>&1
  ```

For a more complete description of the Calibre transcript, refer to the "Calibre nmDRC Results" chapter of the *Calibre Verification User's Manual.*

DRC Results Summary

When you specify a pathname using the DRC SUMMARY REPORT statement detailed in the *Standard Verification Rule Format (SVRF) Manual*, the Calibre nmDRC hierarchical engine generates a results summary text file and saves it to this file. The file contains heading information, describing the particulars of the run, runtime warnings list, statistics for the layers before and after processing, and a runtime summary.

Related Topics

Rule File Contents

Calibre Data

Several types of Calibre data are output for user-examination.

Raw data for the Calibre nmDRC hierarchical engine consists of polygons located on layers. This section discusses the data that the Calibre nmDRC hierarchical engine recognizes and creates, and the impact of layer types on the operations it performs.

Layer Types

The Calibre nmDRC hierarchical engine distinguishes between four types of layers:

- **Original layers (also called drawn or design layers) —** Layers that represent original layout data. In the GDS file, you reference original layers by number. In a rule file, you assign a layer name to each original layer that you intend to use, then reference it by its name.
- **Derived polygon layers —** Layers that contain polygons generated as the result of layer operations, such as Boolean functions, area functions, and polygon-directed dimension check operations.
- **Derived edge layers —** Layers that contain edges or sub-edges of polygons generated as the result of layer operations, such as edge operations and edge-directed dimension check operations.
- **Derived error layers —** Layers that represent clusters of one, two, three, or four edge segments. Each cluster is the result of an error-directed dimensional check operation. You cannot use error layers for any purpose other than reporting errors that exist between the edges within each cluster. For more information on error layers, refer to the Calibre Verification User's Manual.

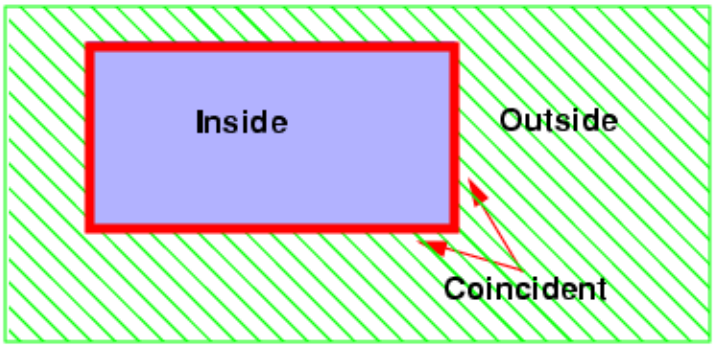
Edge and Polygon Data

Each Calibre SVRF operation operates on polygon data, edge data, or both. As you classify and manipulate data with SVRF, you must pay close attention to the type of data each operation returns: edge layers or polygon layers.

Polygon data consists of whole polygons. In most cases, the Calibre nmDRC hierarchical engine automatically merges the polygon data as part of any polygon operation. That is, if any polygons on a single layer overlap or share an edge, the Calibre nmDRC hierarchical engine merges them into one polygon. Merged data normally provides more accurate depiction of the true mask than unmerged data.

Each polygon divides space into three categories: inside, outside, and coincident. In Figure A-1, imagine that the space extends to the edge of the mask, in all directions. The red rectangle represents the polygon itself and the portion of the space that is coincident with it. The blue area shows the portion of space that is inside, the green striped area shows the space that is outside.

Figure A-1. How Polygon Data Divides Space

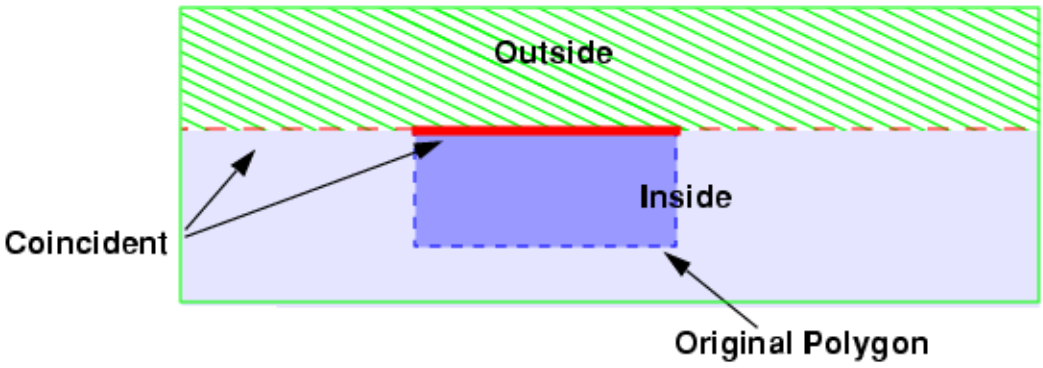


Edge data consists of individual polygon edges, or portions of edges. These edges contain both geometric data and polygon reference data. Geometric data refers to the x and y coordinates of the edge. Reference data refers to a record of the polygon to which the edge belongs and its orientation (which direction is inside, which is outside).

Edge data also divides space into three categories: inside, outside, and coincident; however, the meanings are slightly different. All points on the side of the line that was inside of the original polygon are on the inside of the edge. All points on the side of the line that was outside of the original polygon are on the outside of the edge. All points collinear with the line are said to be coincident with the edge. In Figure A-2, the small rectangle drawn with a thin dotted line is included to show how the orientation of the edge relates to the original polygon. Notice that the inside of the polygon is a small subset of the inside of the edge.

Figure A-2 shows the edge drawn as a thick red line, with a dashed red line drawn through the edge and extending beyond it in both directions. The red lines represent the portion of space that is coincident with the edge. The light blue area shows the portion of space that is inside, the green striped area shows the space that is outside.

Figure A-2. How Edge Data Divides Space



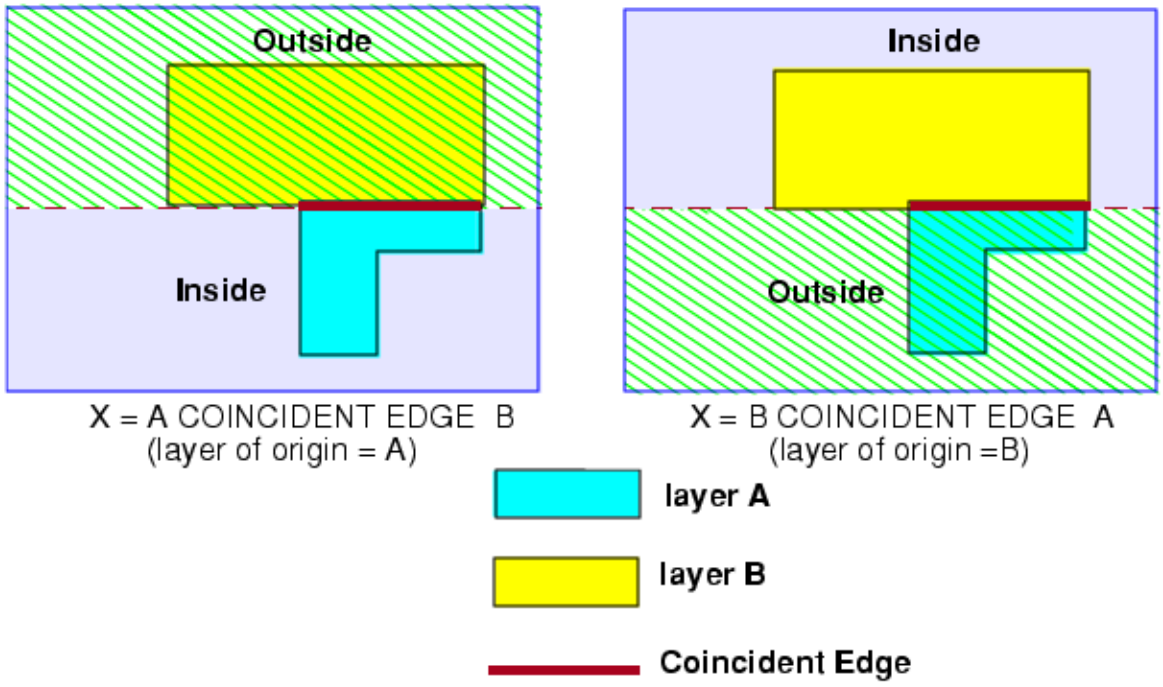
Layer of Origin

Because polygon reference data associates each edge with a specific polygon, you must pay close attention to the layer of origin, that is, where the data comes from. For example, it might appear that the following layer definitions produce the same data in the layer X:

```
X = A COINCIDENT EDGE B       // Edges on layer A that are coincident
                              // with edges on layer B.
X = B COINCIDENT EDGE A       // Edges on layer B that are coincident
                              // with edges on layer A.
```

However, the polygon reference data stored with the results are different, depending on the layer of origin. While both operations appear to generate the same geometric data, one contains edges selected off layer A, and the other contains edges selected off layer B. In the following figure, inside and outside are different depending on the layer of origin.

Figure A-3. Reference Data Dependence on Layer of Origin



Dimensional checks, which measure distances relative to selected edges, provide a good example of how important the layer of origin is. For extended information see “Dimension Checks” on page 235. In the following example, the INTERNAL operation in check A, has different effects than the INTERNAL operation in check B:

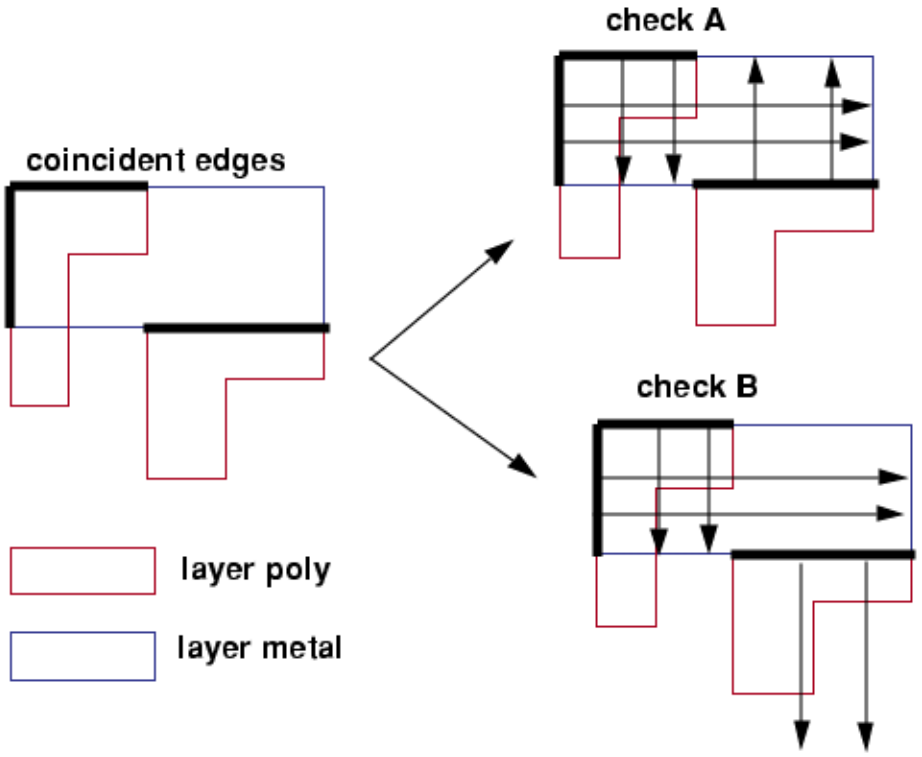
```
// check A:
X = metal COINCIDENT edge poly //metal edges coincident with poly edges
INTERNAL X metal < 3 //internally facing edges closer than 3 microns.
```

```
// check B:
X = poly COINCIDENT edge metal //poly edges coincident with metal edges
INTERNAL X metal < 3 //internally facing edges closer than 3 microns.
```

In check A, the output contains only metal edges coincident with poly, while in check B, the output contains poly edges coincident with metal.

Figure A-4 shows how the COINCIDENT operation results with the INTERNAL check, which looks toward the interior of shapes from edges in layer X.

Figure A-4. The Effect of Reference Data on Dimension Checks



Derived Layers and Rule File Statements

The result of any SVRF operation is a stored collection of data.

The Calibre nmDRC hierarchical engine allows you to store collections of data in the following forms:

- As derived error layers, which are named layers that contain error data. Because error data differs from standard geometric data, you cannot reference error layers in subsequent operations, and they are therefore not relevant to this discussion of rule based RET.

- As derived edge or derived polygon layers, which are named layers containing geometric data that you can reference in subsequent operations. These layers contain intermediate data used as input for other layer operations. The data are composed of either edges or polygons, but not both. The Calibre nmDRC hierarchical engine does not write derived layers to the results database unless explicitly instructed to do so.
- As data that are written directly to the results database. You direct the Calibre nmDRC hierarchical engine to write data to the results database, without first creating a derived layer, by using "Rule File Statements" on page 233.

Note

The only types of data you can pass to the LITHO operation are derived polygon layers and original design layers.

Derived Layers

You create a derived layer using a layer definition statement in an SVRF rule file:

```
layer_name = layer_operation
```

where layer_name assigns a name to the derived layer and layer_operation generates the contents of the layer. See "Layer Operations" on page 247 for more information.

Derived layers can be either global or local.

- Any derived layers you create outside a rule file statement are global. You use global derived layers to store data that you may need to reference multiple times within a single SVRF rule file.
- Any derived layers you create inside Rule File Statements are local. You use local derived layers as intermediate steps when isolating or creating the data the rule file statement writes to the database. You can reference only local derived layers from within that rule file statement.

The names of global derived layers must be unique. The names of local derived layers must be unique within the rule file statement in which they are created.

Rule File Statements

Rule file statements are named output statements that direct the Calibre nmDRC hierarchical engine to write the results of layer operations to the results database. The syntax for a rule file statement is as follows:

```
name {layer_operation | layer_definition_statement
   ...
layer_operation | layer_definition_statement}
```

Note

For rule file statements, braces are required ({ }) .

You define the data resulting from a rule file statement using a series of layer operations and/or layer definition statements within the braces portion of a rule file statement. When used, layer definition statements create local derived layers, used as intermediate data. The layer operations generate output that is written directly to the results database.

The following rule file statement shows how you might combine layer operations and layer definition statements to return the desired data.

```
metal_end_cap {
   x = CONVEX EDGE m1 == 2          // Derive layer x from original layer.
   y = LENGTH x == 3                // Derive layer y from layer x.
   z = EXPAND EDGE y OUTSIDE BY 1   // Derive layer z from layer y.
   m1 OR z                          // Layer operation -- merge layers
                                    //   and store results in database.
}
```

When a rule file statement contains multiple layer operations, the Calibre nmDRC hierarchical engine merges the data resulting from the layer operations into the final results. For example, the following two rule file statements output the same results:

```
metal_end_cap1 {
   INTERNAL metal < 3            // Generates short edges
   long_met = metal LENGTH > 5   // Generates long edges
   INTERNAL metal long_met < 4   // Generates edges near long edges
}

metal_end_cap2 {
   M1 = INTERNAL [metal] < 3           // Generates short edges
   long_met = metal LENGTH > 5         // Generates long edges
   M2 = INTERNAL metal [long_met] < 4  // Generates edges near long edges
   M1 OR M2                            // Layer operation - derives edges
                                       //   that combine short edges and
                                       //   those near long edges
}
```

Rule filenames must be unique within a single rule file. If a rule file includes other rule files using the INCLUDE statement, then rule filenames in all included files must also be unique.

Controlled Data Destinations

When the Calibre nmDRC hierarchical engine writes rule file output directly to a results database that is in GDS format, it merges all the data onto layer 0. In most cases, it is more useful to store each type of data on a different layer. Using the DRC Check Map statement, you can map the results of a rule file statement to an additional layer, and if necessary, specify a data type. This additional layer can be in the same database or in a different database.

The basic syntax for using the DRC Check Map statement this way is as follows:

```
DRC CHECK MAP rule file_name layer [ datatype ] [ filename ]
```

The DRC CHECK MAP statement provides additional functionality not described in this manual. For more information, refer to the DRC CHECK MAPentry in the *Standard Verification Rule Format (SVRF) Manual* or to the "Rule Check Statements" section in the *Calibre Verification User's Manual*.

Derived Layers Storage With Rule File Statements

Calibre derived layers are intermediate data that do not exist outside the Calibre nmDRC hierarchical engine. To save global derived layer data for use with other applications you must write the data to the results database using a rule file statement and the COPY operation. The syntax is as follows:

```
X = layerA AND layerB
X {COPY X} // rule_check_name {COPY derived_layer_name}
```

Note

Because derived layers are data while rule file statements are output statements, you can create rule file statements with the same name as a derived layer.

Related Topics

Rule File Contents

Dimension Checks

The dimension check is a commonly used SVRF operation.

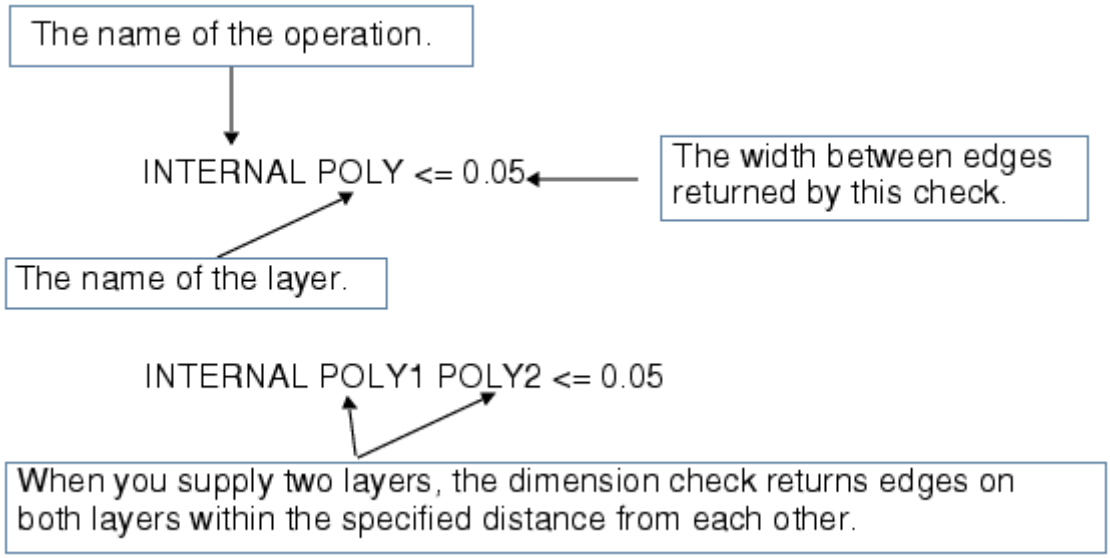
These operations measure distances between opposing edges. Because dimension checks allow you to classify edges based on key properties, such as width and spacing, they serve as the foundation for all RET related processing.

The Calibre nmDRC hierarchical engine provides three dimension checks:

- INTERNAL — Defines a minimum separation between the interior sides of two edges.
- EXTERNAL — Defines a minimum separation between the exterior sides of two edges.
- ENCLOSURE — Defines a minimum separation between the interior side of an edge on one layer and the exterior side of an edge on another layer. This results in returning edges on one layer that are enclosed by polygons on the other layer.

Figure A-5 shows two simple dimension checks. Using additional arguments, you can fine-tune the dimension check to return only the data you are interested in. For more information on dimension check arguments, refer to the *Standard Verification Rule Format (SVRF) Manual.*

Figure A-5. Dimension Checks



The Calibre nmDRC hierarchical engine performs dimension checks as a three-step process:

1. Identify appropriate edge pairs.
2. Construct measurement regions.
3. Return those portions of an edge that intersect the measurement region for the opposite edge.

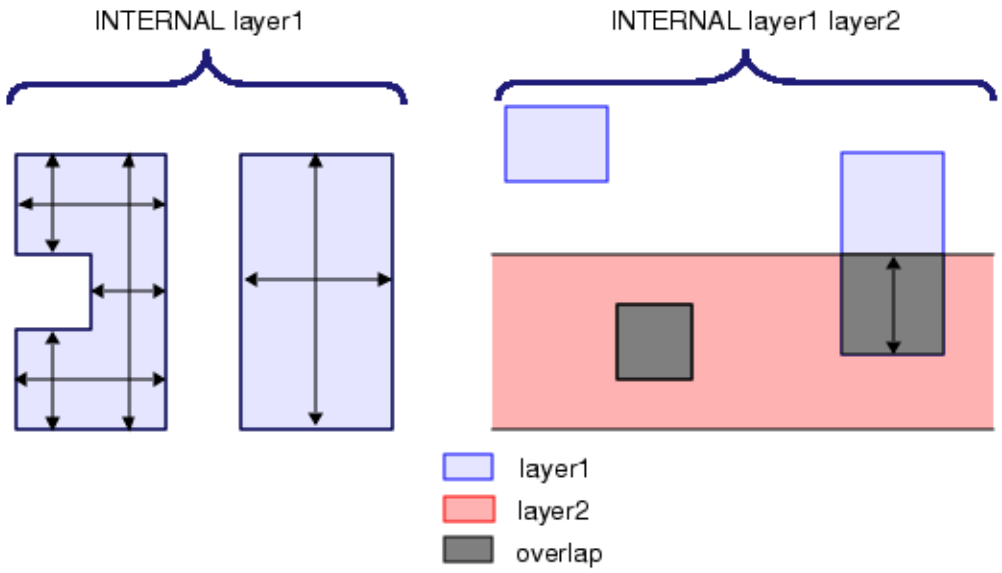
Each step is described in the following sections.

Edge Pair Identification

The Calibre nmDRC hierarchical engine begins the measurement process by evaluating all the edges on the layer (or pair of layers) to find appropriate edge pairs; all pairs of edges that meet the criteria for the operation. The first relates to the orientation of the edges. Each dimension check measures from a specific side of the first edge to a specific side of the second edge:

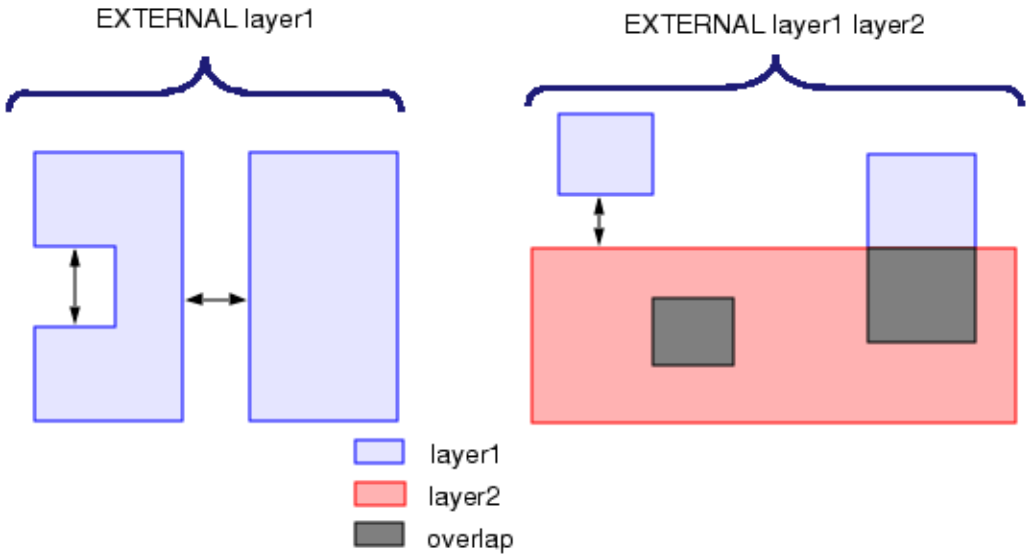
- For the Internal dimension check, the application measures from the inside of one edge to the inside of another edge.

Figure A-6. Edge Pairs Measured by the Internal Operation



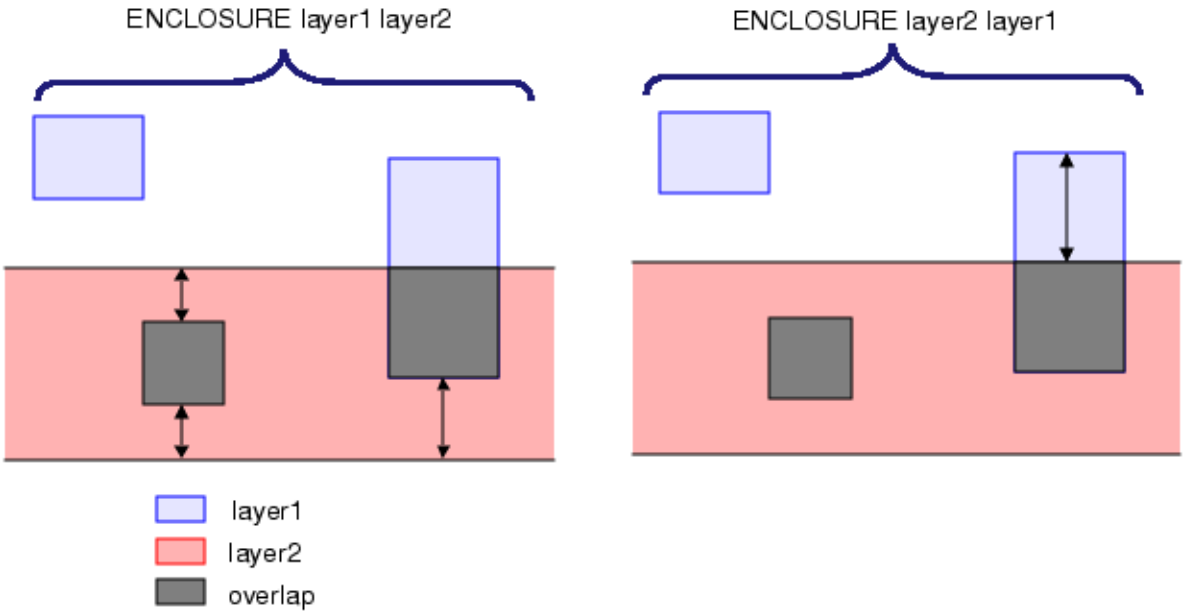
- For the EXTERNAL dimension check, the application measures from the outside of one edge to the outside of another edge.

Figure A-7. Edge Pairs Measured by the External Operation



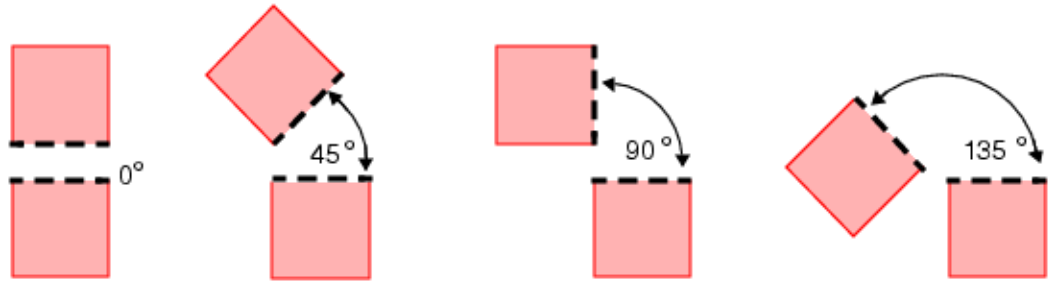
- For the ENCLOSURE dimension check, the application measures between the outside side of an edge on the first input layer to the inside of an edge B on the second input layer.

Figure A-8. Edge Pairings Measured by the Enclosure Operation



Another requirement for appropriate edge pairs relates to the angle between the specified sides of the edges. Figures A-6 through A-8 show edge pairings that measure the distance between parallel edges (angle = 0). By default, the application considers a pair of edges to be appropriate if the angle between the two edges >=0 and <90. You can use additional arguments with dimension checks to override this default behavior; however, the angle must always be less than 180 degrees.

Figure A-9. Measuring Appropriate Angles for Edge Pairings



For the EXTERNAL dimension check, the appropriate angle is between the outsides of the dashed edges.

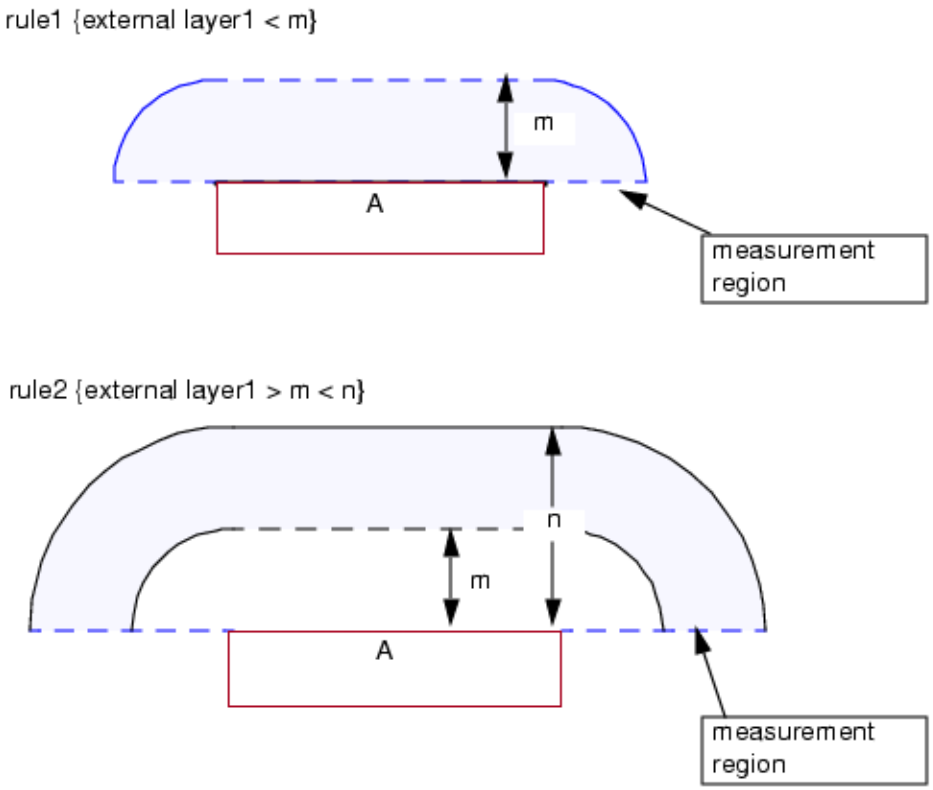
Measurement Region Creation

Once the Calibre nmDRC hierarchical engine has identified pairs of edges that meet the criteria, it creates measurement regions for each of the edges in a pair. The application uses measurement regions to measure the separation between edges. Each measurement region

extends in the direction of the dimension check, either outside or inside. It contains all points in that direction such that the distance from the edge satisfies the constraint for the operation.

Figure A-10 shows two regions outside of edge A. The first region assumes the constraint x < m, and the second region assumes the constraint m < x <= n. By definition, the measurement region does not contain the points on the line containing edge A.

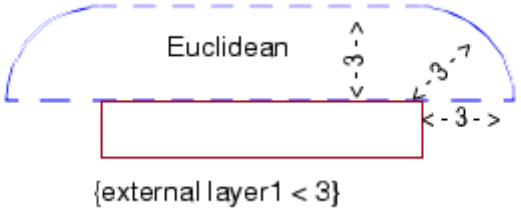
Figure A-10. Measurement Regions for Edge A



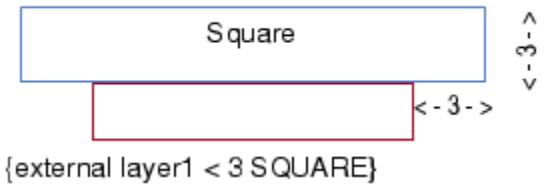
Metrics

You control the shape of the measurement regions by specifying the metric, or measurement style. The Calibre nmDRC hierarchical engine supports five metrics:

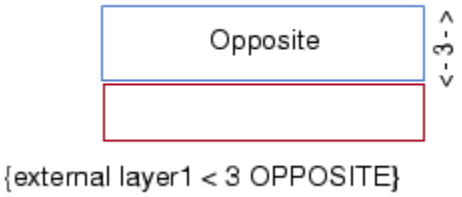
- **Euclidean —** The Euclidean metric forms a region with rounded boundaries that extend past the corners of the selected edges. This is the default metric.



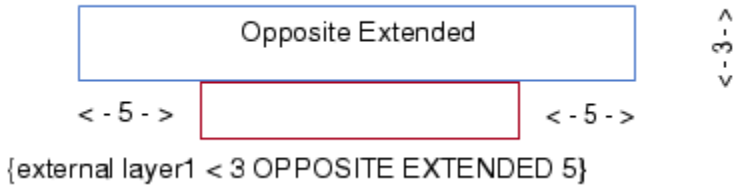
- **Square —** The square metric forms a region with right-angle boundaries that extend past the corners of the selected edges.



- **Opposite —** The opposite metric forms a region with right-angle boundaries that do not extend past the corners of the selected edges.



- **Opposite Extended —** The opposite extended metric forms a region with right-angle boundaries that can extend past the corners of the selected edges by a value you specify.



- **Symmetric —** The opposite symmetric metric uses the opposite metric with post-processing for use with non-parallel edges. For more information on this metric, refer to the *Calibre Verification User's Manual*.

Intersection Edges

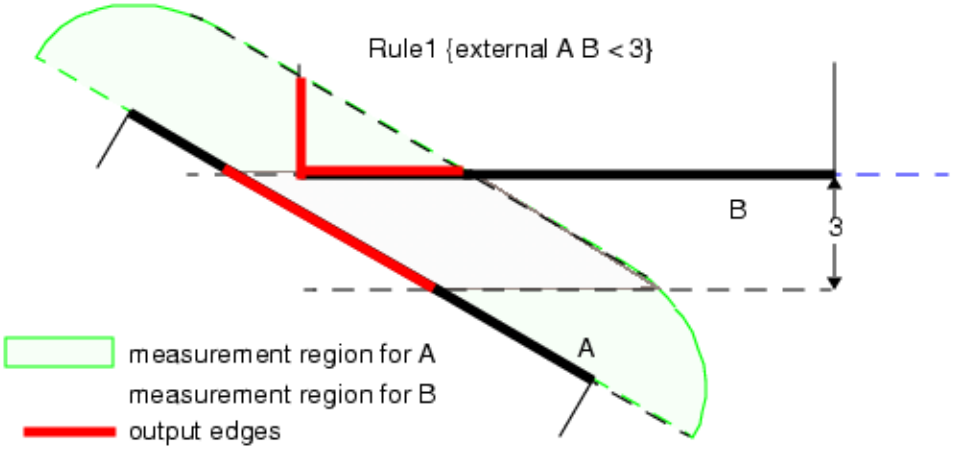
The final step in performing the dimension check returns those portions of the edges that intersect the measurement region for the opposing edge. Because the measurement region for edge A does not contain the line containing edge A, dimension checks do not return any points of intersection, if they exist.

Examples

Assume the measurement is from the outside of edge A to the outside of edge B and that the operation instructs the application to return all edges closer than 3 microns. Using the default metric, which is Euclidean (refer to "Metrics" on page 239), the operation begins processing by constructing two regions. One region consists of all points in the half-plane on the outside of edge A that are within 3 microns of edge A; a similar region consists of all such points around edge B.

The output is an edge pair consisting of the portion of edge A that intersects the measurement region for edge B and the portion of edge B that intersects the measurement region for edge A.

Figure A-11. Creating Measurement Regions



Appendix B
SVRF Rule Files

An SVRF file is an ASCII text file comprising SVRF statements. It contains everything needed to read layout data, process layer data, and apply RET solutions.

You can create a new SVRF rule file from scratch using a text editor, or by copying an existing rule file and modifying it.

This section describes a basic SVRF rule file. It is intended to provide enough information to enable you to read and create a simple rule file. For more information, refer to the *Standard Verification Rule Format (SVRF) Manual*.

All elements in a rule file are either operations or statements. The basic distinction between these is that operations manipulate layout data and statements prepare the environment in which the operations work.

Rule File Requirements

The SVRF file has structure and syntax requirements.

Order

SVRF rule files are compiled before processing. Except for nested variable declarations and conditional statements, you can write statements and operations in any order. For clarity, the examples in this manual organize the operations according to function and the expected order of processing. This is also good practice, because it enhances readability and maintainability.

Within individual SVRF operations, keyword order is only important when changing the order results in ambiguity. For example, the following four operations are all equivalent:

- INTERNAL metal < 3
- INTERNAL < 3 metal
- metal INTERNAL < 3
- < 3 metal INTERNAL

Case Sensitivity

Keywords and names are not case-sensitive.

Because of their relationship to the host platform's directory system, structure names, and pathnames are case-sensitive. To avoid ambiguity, you should enclose pathnames in single or double quotes.

Reserved Words

All SVRF keywords are reserved words. Do not use them as names of variables or layers. For a complete list of reserved words, refer to "Keyword - Name Conflicts" in the *Standard Verification Rule Format (SVRF) Manual.*

Abbreviations

When an SVRF keyword can be abbreviated, its entry in the *Standard Verification Rule Format (SVRF) Manual* shows the keyword with the portion that can be used as the abbreviation displayed in uppercase letters.

Note

Not all SVRF keywords can be abbreviated.

Comments

SVRF supports three types of comments:

- **In-line comments** begin with slashes (//), and can begin anywhere on the input line, and end at the end of the line. In-line comments are ignored by the Calibre nmDRC hierarchical engine.
- **Block comments** begin with /* and end with */. They may begin anywhere on a line and span multiple lines. The Calibre nmDRC hierarchical engine ignores block comments.
- **Calibre nmDRC rulecheck comments** begin with @, and can begin anywhere on the input line, and end at the end of the line, as long as they are inside of a rulecheck statement. Calibre nmDRC rulecheck comments are included in rulecheck output, so you can view them in the Calibre® RVE™ results viewer.

For more information, refer to the *Calibre RVE User's Manual*.

Rule File Contents

The contents of an SVRF file are divided into statements and operations.

Specification Statements

Specification statements in a rule file identify the layout data to be processed, provide some basic information about how it is to be processed, and specify where to store the output. The sample code that follows shows some specification statements taken from a sample SVRF rule file. The discussion that follows describes some of the statements you can use.

```
LAYOUT SYSTEM GDSII
LAYOUT PATH "gdsout/demo_ab.gds"
LAYOUT PRIMARY "demo"
PRECISION 1000
RESOLUTION 1
FLAG SKEW YES
FLAG ACUTE YES
FLAG OFFGRID YES
DRC MAXIMUM RESULTS ALL
DRC RESULTS DATABASE "gdsout/ab_sense.gds" GDSII PSEUDO
DRC SUMMARY REPORT "reports/demo_ab.rep"
DRC MAXIMUM VERTEX 199
```

Note

When using Calibre Interactive, many of the required specification statements are created in a control file that defines the inputs, outputs and other information required for processing the data. The control file contains an INCLUDE statement pointing to the main rule file. Refer to the *Calibre Interactive User's Manual* for more information on control files.

Layout Statements

Layout statements define the database(s) on which the Calibre nmDRC hierarchical engine operates. If you specify multiple database paths, the databases are merged before processing. All SVRF operations in the rule file operate on the same data. Table B-1 lists the mandatory layout statements found in the *Standard Verification Rule Format* manual.

Table B-1. Layout Statements

Statement	Description
LAYOUT PATH	Specifies the pathname for the layout design database. It is required for most situations. When multiple LAYOUT PATH statements are supplied, the first database must contain the top cell specified by LAYOUT PRIMARY.
LAYOUT PRIMARY	Identifies the top level or cell on which to operate. It is required.
LAYOUT SYSTEM	Identifies the database format of the layout design database. It is required.

In addition to these required statements, to obtain the best performance from the Calibre Hierarchical Engine, it is recommended that you supply an optional statement (LAYOUT BASE LAYER or LAYOUT TOP LAYER, found in the *Standard Verification Rule Format* manual) to identify the layout base layers. For more usage information, refer to the *Calibre Post-Tapeout Flow User's Manual*.

Unit Statements

Unit statements provide information about the database unit precision and resolution. Table B-2 lists the more commonly used unit statements:

Table B-2. Unit Statements

Statement	Description
UNIT LENGTH	Specifies the user unit for length. When a number is supplied, it specifies the user unit of length in terms of meters. When a factor is supplied, it specifies the user unit in terms of another unit, such as mil, mm, cm, or inch. The default is 1E-6, or 1 micron.
PRECISION	Defines the ratio of database units to user units. The default is 1000. Given the default unit length of 1 micron, and the default precision of 1000, the default database unit is a nanometer.
RESOLUTION	Defines the layout grid step size. When only one value is supplied, the grid step size is the same in both the x and y direction. The default value is 1.

Flag Statements

Flag statements let you control how the Calibre nmDRC hierarchical engine responds with respect to certain types of errors. Table B-3 lists the more commonly used flag statements found in the *Standard Verification Rule Format* manual.

Table B-3. Flag Statements

Statement	Description
FLAG SKEW	Issues a warning when it encounters a skew edge.
FLAG ACUTE	Issues a warning when an acute angle is encountered.
FLAG OFFGRID	Issues a warning when an off-grid vertex is encountered.

DRC Statements

DRC statements, found in the *Standard Verification Rule Format* manual, let you specify how the Calibre nmDRC hierarchical engine treats the results of the tool run. Table B-4 lists the more commonly used nmDRC statements. For more information on results, refer to "Calibre Results."

Table B-4. DRC Statements

Statement	Description
DRC MAXIMUM RESULTS	Specifies the number of results to report. The default value is 1000. For OPC this statement must set DRC MAXIMUM RESULTS to ALL to ensure that the application saves all corrected geometries.

Table B-4. DRC Statements (cont.)

Statement	Description
DRC RESULTS DATABASE	Specifies the full pathname to the primary results database, and its format. The optional keyword PSEUDO specifies that the results database should include the additional levels of hierarchy that the application creates.
DRC SUMMARY REPORT	Specifies the full pathname for the summary report file. Additional optional arguments let you control how this file is saved.
LAYOUT ERROR ON INPUT	Specifies whether errors or warnings encountered while reading in a GDS database result in fatal errors.

Layer Specification Statements

Layer specification statements, found in the *Standard Verification Rule Format* manual, are statements that make data in the design database available to the Calibre nmDRC hierarchical engine. During any run, the application ignores all data except the layers specified using these statements.

Table B-5. The Layer Specification Statement

Statement	Description
LAYER	Declares a layer in the design database by number and assigns it a name by which it is referenced. The layer name is a user-defined alphanumeric string. It cannot be an SVRF reserved word, unless it is in quotes. Calibre names must be unique within the SVRF file. You can assign the original layer as many names as needed. However you can only assign the Calibre name to one original layer.
LAYER MAP	Generates a new “original” layer by mapping data of the specified type(s) on the specified GDS layer(s) to a new GDS layer.

Layer Operations

Layer operations and layer creation statements isolate specific layer data and modify that data to generate new data. This section describes some of the more commonly used layer operations. This sequence of layer statements and operations that follows illustrates how you can use layer operations to find and modify data.

```
// Layer Specifications
LAYER ACTIVE 2
LAYER POLY 4
// Layer Operations in Layer Creation Statements
LE = CONVEX EDGE POLY WITH LENGTH == 0.18
   ANGLE1 == 90 LENGTH1 > 0.18 ANGLE2 == 90
LENGTH2 > 0.18
LE_ISLAND = EXPAND EDGE LE OUTSIDE BY 0.01
INSIDE_CORNi = EXT [POLY] <= 0.13 ABUT == 90 INTERSECTING ONLY
INSIDE_CORN = (LENGTH POLY >= 0.39) COINCIDENT EDGE INSIDE_CORNi
INSIDE_CORN_ISL = EXPAND EDGE INSIDE_CORN INSIDE BY 0.01
```

Operations let you manipulate layer data. Some operate on a single layer, some on a pair of layers. When deciding which operation to use, think in terms of what each operates on and what it does with that data.

Each operation operates on either polygons or edges. The resulting data consists of only one of these types of data. Because of this, you must pass the correct type of data to the operation.

Note

To the Calibre nmDRC hierarchical engine, edge data are more than just isolated line segments. Each edge maintains a record of the polygon from which it was derived, as well as which side is interior and which is exterior.

Each operation can do one of two things. If it is a Layer Selector, it can select existing data from specified input layer(s). If it is a Layer Constructor, it can create new polygon or edge data by modifying existing data.

Layer Selectors

Layer selectors are operations that select existing data that meet the specified criteria. They may select data from one layer or from all specified layers. Some select polygons, others select edges. Most operations are layer selectors. Table B-6 describes a few of the more commonly used layer selector operations. For a complete list of layer operations, refer to the *Standard Verification Rule Format (SVRF) Manual*.

Table B-6. Commonly Used Layer Selector Operations

Operation	Description
COPY	Copies (selects) all data on the specified layer.
INSIDE	Selects all polygons on layer1 that lie inside polygons on layer2.
OUTSIDE	Selects all polygons on layer1 that lie outside polygons on layer2.
LENGTH	Selects all edges on the specified layer that satisfy the length constraint.
INTERNAL[1]	Defines a required separation between the interior sides of two edges.

Table B-6. Commonly Used Layer Selector Operations (cont.)

Operation	Description
EXTERNAL[1]	Defines a required separation between the exterior sides of two edges.
ENCLOSURE[1]	Defines a required separation between the interior side of an edge on one layer and the exterior side of an edge on another layer.
CONVEX EDGE	Selects edges based on the specified constraints, which can include the value of the angle at one or both ends of the edge, the length of the edge, or the length of abutting edges.
COINCIDENT EDGE	Selects all layer1 edges or edge segments that are coincident with layer2 edges.

1. INTERNAL, EXTERNAL, and ENCLOSURE are dimension checks that you can use as layer selectors or layer creators.

Layer Constructors

Layer constructors are operations that create new data by modifying existing data. Some construct polygon data from existing polygons. Others create polygon data from existing edges.

Table B-7 describes a few of the more commonly used layer constructor operations. For a complete list of operations, refer to the *Standard Verification Rule Format (SVRF) Manual*.

Table B-7. Commonly Used Layer Constructor Operations

Operation	Description
AND	Generates new polygon data from all polygon regions that are common to one or more polygons on layer1 and one or more polygons on layer2.
OR	Generates new polygon data by combining all polygons on layer1 and all polygons on layer2, merging overlapping polygons into larger polygons.
EXPAND EDGE	Expands edges into rectangles. The expansion parameters define how the application expands the edge.
SIZE	Expands or shrinks polygons according to the sizing parameters. The sizing parameters define the factor by which the polygons are sized. They can also define bounding areas and specify how the application treats the resulting polygons (such as, merge polygons, output overlaps only, or truncate).

Layer Generation Statements

Layer generation statements store the results from a layer operation as derived layers, which you can reference in other layer operations. For a complete discussion, refer to “Derived Layers and Rule File Statements” on page 232.

Table B-8. The Layer Definition Statement

Statement	Description
Layer Definition Statement *layer_name = layer_operation*	Generates a **derived layer** with the specified name. The contents of this layer are the results of the operation.

Output Statements

You write data to the results database using the rule file statement found in the *Standard Verification Rule Format* manual. By default, the Calibre nmDRC hierarchical engine writes all data resulting from a rule file statement to layer 0 of the primary results database as defined by the DRC RESULTS DATABASE statement. The DRC CHECK MAP statement enables rule file results to be saved to a specific layer in a database file.

Table B-9. Output Statement

Statement	Description
Rulecheck Statement *name* {*layer_operation* | *layer_definition* … [*layer_operation* | *layer_definition*] }	Outputs the results of the specified layer operations to the results database.
DRC CHECK MAP	Outputs the results of the specified rule file statement on the specified layer in the results database. Also outputs the results of the rule file statement to a database other than the primary results database. Requires the specification of the pathname and format of the database.

Index

— S —

— T —

— U —

— W —

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.