

IC Compiler™ Technology File and Routing Rules Reference Manual

Version J-2014.09-SP4, March 2015

SYNOPSYS®

Copyright Notice and Proprietary Information

© 2015 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

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

Third-Party Links

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

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

Copyright Notice for the Command-Line Editing Feature

© 1992, 1993 The Regents of the University of California. All rights reserved. This code is derived from software contributed to Berkeley by Christos Zoulas of Cornell University.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1.Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2.Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3.All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed by the University of California, Berkeley and its contributors.

- 4.Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright Notice for the Line-Editing Library

© 1992 Simmule Turner and Rich Salz. All rights reserved.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

- 1.The authors are not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
- 2.The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
- 3.Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
- 4.This notice may not be removed or altered.

Contents

About This Manual	xiv
Customer Support.	xvi
1. Technology File Syntax	
Creating a Technology File	1-3
Basic Syntax Rules.	1-3
Technology File Contents	1-4
Technology Section.	1-5
Unit Precision and Range	1-6
Defining Units	1-7
Defining Routing Rule Modes	1-12
PrimaryColor Section	1-19
Colors on a Computer Monitor.	1-19
Colors in IC Compiler.	1-19
Color Section	1-20
LineStyle Section	1-22
Tile Section.	1-23
Layer Section	1-23
Layout Attributes	1-28
Display Attributes.	1-30
Parasitic Attributes.	1-33
Resistance	1-34

Capacitance	1-34
Inductance	1-34
Sidewall Capacitance	1-35
Routing Channel Capacitance	1-35
Total Sidewall Routing Channel Capacitance	1-36
Maximum Current Density	1-36
Maximum Intracapacitance Distance Ratio	1-36
Wire Segment Length	1-36
Physical Attributes	1-37
Height From Substrate	1-37
Thickness	1-37
Design Rule Attributes	1-38
LayerDataType Section	1-49
ContactCode Section	1-50
Physical Attributes	1-52
Parasitic Attributes	1-54
Resistance	1-55
Capacitance	1-55
Maximum Current Density	1-55
Design Rule Attributes	1-55
DesignRule Section	1-57
FringeCap Section	1-66
CapModel and CapTable Sections	1-68
ResModel Section	1-69
PRRule Section	1-70
Cell Row Spacing Rules for Double-Back Cell Rows	1-70
Cell Row Spacing Rules for Non-Double-Back Cell Rows	1-71
DensityRule Section	1-72
SlotRule Section	1-73

2. Routing Design Rules

Minimum Area Rules	2-3
Minimum Length Rule	2-3
General Minimum Area Rule	2-4
Two-Stage Special Minimum Area Rule	2-6

Polygon-Edge-Based Multistage Minimum Area Rule	2-7
Dense Wire Minimum Dimensions Rule	2-8
Minimum Enclosed Area Rule	2-9
Minimum Enclosed Width Rule	2-11
Metal Width Rules	2-11
Signal Routing Maximum Metal Width Rule	2-12
Discrete Metal Widths Rule	2-12
No Routing in Nonpreferred Direction	2-13
Minimum Width in Nonpreferred Direction	2-14
Default Width in Nonpreferred Direction	2-14
Minimum Edge Rules	2-15
Minimum Edge Mode	2-16
Maximum Total Number of Short Edges Rule	2-16
Maximum Total Short Edge Length Rule	2-17
Convex-Concave Minimum Edge Length Rule	2-18
Special Notch Rule	2-19
Hook Rule	2-19
H-Shape Rule	2-20
Adjacent Minimum Edge Length Rule	2-21
Short Edge to End-of-Line Rule	2-23
Minimum Edge for Macro Pin Connection Rule	2-24
Minimum Spacing Rules	2-25
Minimum Spacing Rule	2-26
U-Shape Spacing Rule	2-27
Via Corner Spacing Rule	2-29
Via Same Net Minimum Spacing Rule	2-31
Edge-Line Via Spacing Rule	2-32
General Cut Spacing Rule	2-33
Asymmetric Via-to-Metal Spacing Rule	2-41
Fat Metal Spacing Rules	2-42
Fat Metal Spacing Table Rule	2-43
Width Versus Width	2-44
Width Versus Parallel Length	2-46
Fat Metal Orthogonal Spacing Rule	2-47
Fat Metal Parallel Length	2-49

Fat Metal Extension Spacing Rule	2-50
Neighboring Layer Fat Metal Extension Spacing Rule	2-55
Metal Span Spacing Rule	2-57
End-of-Line Spacing Rules	2-60
End-of-Line to End-of-Line Spacing Rule	2-61
One-Neighbor End-to-End Table-Based Spacing Rule	2-62
One-Neighbor End-of-Line Spacing Rule	2-63
Two-Neighbor End-of-Line Spacing Rule	2-68
Three-Neighbor End-of-Line Spacing Rule	2-72
Three-Neighbor End-of-Line Cap Rule	2-74
Two-Sided End-of-Line Spacing Rule	2-76
Two-Sided End and Joint Spacing Rule	2-78
End-of-Line Two-Corner Keepout Rule	2-80
End-of-Line Spacing Rule and Stub Modes	2-82
Stub Mode 1: Single-Edge Spacing at Metal End	2-83
Stub Mode 2: Two-Edge Spacing at Metal End	2-84
Stub Mode 3: Three-Edge Spacing at Metal End	2-86
Stub Mode 4: Two-Edge Spacing With Connected Metal Exclusion	2-87
Enhanced Dense End-of-Line Spacing Rule	2-88
End-of-Line to End-of-Line Spacing Rule (Classic Router)	2-88
L-Shaped End-of-Line Spacing Rule	2-90
End-of-Line Depth Rule	2-91
Bridge Rules	2-92
Bridge Rule 1	2-92
Bridge Rule 2	2-94
Bridge Rule 3	2-96
Fat Metal Contact Rules	2-98
Fat Metal Contact Rule	2-98
One-Dimensional Table Rule	2-98
Two-Dimensional Table Rule	2-99
Area-Based Fat Metal Contact Rule	2-101
Fat Metal Extension Contact Rule	2-103
Area-Based Fat Metal Extension Contact Rule	2-105
Alternative Syntax for Fat Metal Contact and Extension Rules	2-107
Alternative Syntax Example	2-108
Conversion to Standard Syntax	2-109

Fat Poly Contact Rule	2-111
Via Spacing Rules	2-111
Adjacent Via Rule	2-111
Enclosed Via Spacing Rule	2-112
Isolated Via Rule	2-113
Misaligned Via Spacing Rule	2-114
Via Enclosure Rules	2-118
Minimum Enclosure Rule	2-119
End-of-Line Via Enclosure Rules	2-119
Zero-Neighbor End-of-Line Via Enclosure Rule	2-120
T-Shape Metal Extension for End-of-Line Via Enclosure Rule	2-121
Two-Neighbor End-of-Line Via Enclosure Rule	2-122
Three-Neighbor End-of-Line Via Enclosure Rule	2-127
Enclosed Via Metal Minimum Length Rule	2-128
Fat Wire Via Enclosure Rule	2-129
Fat Metal Via Keepout Rules	2-133
Fat Metal Via Keepout Area Rule	2-134
Fat Via Enclosure Rule	2-135
Fat Poly Contact Enclosure Rule	2-136
Concave Metal Corner Via Enclosure Rule	2-137
Convex Metal Corner Via Enclosure Rule	2-138
Via Placement Rules	2-140
Via Stack Rule	2-140
Via Array (Via Farm) Rule	2-142
Via Density Rule	2-144
Vias on Nonpreferred Route Direction Rule	2-144
Metal and Via on Wire Track Rules	2-145
Metal and Via Alignment Using the onGrid Attribute	2-145
Metal and Via Alignment Using the Classic Router	2-145
Wire Shape Rules	2-147
Jog Wire Rules	2-147
Small Jog Rule	2-147
Jog Wire End-of-Line Via Rule	2-148
Jog Wire Via Keepout Region Rule	2-148
Line-End to Jog Distance Rule	2-149
Dog Bone Rule	2-150

Protrusion Length Rule	2-151
Metal Density Rules	2-153
Metal Density Rule	2-153
Metal Density Gradient Rule	2-153
Parallel Length-Based Floating Wire Antenna Rule	2-154

3. Routing Rules for Advanced Geometries

Double-Patterning Spacing Rules	3-2
Double-Patterning Line-End to Line-End Spacing Rule	3-2
Double-Patterning Line-End to Line-Side Spacing Rule	3-5
Double-Patterning Line-Side to Line-Side Spacing Rule	3-6
Double-Patterning Corner-to-Corner Spacing Rule	3-8
Double-Patterning Fat Metal Spacing Rule	3-8
Self-Aligned Via Rules	3-9
Single Self-Aligned Edge Forbidden Rule	3-11
Self-Aligned Via Spacing Rules	3-12
Self-Aligned Via Cluster Rules	3-13
Self-Aligned Via Array Rule	3-17
Self-Aligned Via Diagonal Spacing Rule	3-19
Self-Aligned Via Zigzag Spacing Rule	3-20
Minimum Edge, Width, and Area Rules	3-22
Discrete Metal Dimension Rule	3-23
Two-Convex-Corner Minimum Edge Length Rule	3-24
Line-End Concave Corner Length Rule	3-25
Convex-Concave Minimum Edge Length Rule	3-26
Diagonal Concave Corner Minimum Width Rule	3-27
Minimum Metal Jog Length and Adjacent Edge Length Rules	3-27
Fat Metal Branch Minimum Width and Length Rule	3-29
Width-Based Minimum Area Rule	3-32
Polygon-Edge Multistage Minimum Area Rule Enhancement	3-32
Metal Spacing Rules	3-34
Cut to Metal Orthogonal Spacing Rule	3-35
Line-End to Concave Corner Spacing Rule	3-36
Concave Corner Keepout Rule	3-37
Fat Metal Corner Keepout Rule	3-37

Preferred and Nonpreferred Corner-to-Corner Spacing Rule	3-39
Preferred and Nonpreferred End-of-Line Spacing Rule	3-39
Preferred and Nonpreferred Fat Metal Spacing Rule	3-41
X-Spacing From Shape in Preferred Direction	3-42
Y-Spacing From Shape in Preferred Direction	3-43
X-Spacing From Shape in Nonpreferred Direction	3-43
Y-Spacing From Shape in Nonpreferred Direction	3-44
Fat Metal Spacing Exclusion Range	3-45
Metal Span Orthogonal Spacing Rule	3-47
Line-End Exceptions	3-49
Z-Shape Exceptions	3-50
L-Shape Exceptions	3-51
Spacing Range Exceptions	3-51
Orthogonal Minimum Width Rule	3-52
Forbidden Pitch Rule	3-53
Wide Metal Jog Rule	3-55
Preferred-Direction Forbidden Spacing Rule	3-56
Fat Metal Forbidden Spacing Rule	3-57
Three-Wire Forbidden Space Rule	3-59
Via-Induced Metal Bridge Rule	3-60
Enclosure-Dependent Metal Spacing Rule	3-62
U-Shape Leg Spacing Rule	3-63
Line-End to U-Shape Spacing Rule	3-64
Two-Nighbor End-of-Line Spacing Rule Enhancements	3-65
Via Spacing Rules	3-66
Via to Concave Metal Corner Spacing Rule	3-67
Maximum Number of Unaligned Vias Rule	3-67
Non-Self-Aligned Via Edge to Metal Spacing Rule	3-68
Adjacent Via Spacing Rule	3-69
Constrained Via Spacing Rule	3-71
Line Via Spacing Rule	3-71
Edge Via Spacing Rule	3-74
Matrix Via Spacing Rule	3-75
Via Corner Spacing Rule Enhancement	3-79
Interlayer Cut-to-Metal Spacing Rule	3-80
Non-Self-Aligned Interlayer Via Spacing Rule	3-81
Via Forbidden Spacing Rule	3-82

Separated Via Spacing Rule	3-84
Via Corner Spacing at Zero Projection Rule	3-85
Mixed Edge-to-Edge and Center-to-Center Via Spacing Rule.	3-86
Via Enclosure Rules	3-87
Via to Joint Distance Rule	3-87
Line-End Cut Enclosure Rule	3-91
Convex Metal Via Keepout Rule	3-92
Concave-Convex Edge Via Enclosure Rule.	3-93
Fat Wire Via Enclosure Rule Enhancements.	3-94
Measure Metal Width Orthogonal to Parallel Run	3-95
Overlapped Width Threshold Ranges.	3-96
Minimum Spacing Requirements	3-97
Fat Metal Contact Asymmetric Enclosure Rule Enhancements	3-98
Fat Metal Contact Asymmetric Extra Enclosure Rule.	3-98
Fat Metal Contact Asymmetric Total Enclosure Rule	3-99
Alternative Syntax for Fat Metal Contact Asymmetric Enclosure	3-100
Sparse-Dense Via Enclosure Rule	3-101
Via-to-Jog Metal Enclosure Rule	3-105
Via to Concave Corners Maximum Limit Rule	3-107
Contact Code Rules	3-107
Contact Code Rotation Control	3-108
Fat Metal Contact Rule	3-108
IC Compiler Zroute Error Messages	3-109

Preface

This preface includes the following sections:

- [About This Manual](#)
- [Customer Support](#)

About This Manual

The *IC Compiler Technology File and Routing Rules Reference Manual* provides details about the technology file and how to implement routing design rules. IC Compiler uses this information to perform placement and routing in IC Compiler.

Audience

Users of this manual should be familiar with IC implementation concepts, technology specifications, design rule constraints, and standard-cell-based design.

Related Publications

For additional information about the IC Compiler tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

<https://solvnet.synopsys.com/DocsOnWeb>

Release Notes

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

To see the *IC Compiler Release Notes*,

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

<https://solvnet.synopsys.com/DownloadCenter>

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

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code> .
Courier bold	Indicates user input—text you type verbatim—in examples, such as <code>prompt> write_file top</code>
[]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>low medium high</code>
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

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

Accessing SolvNet

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

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

<https://solvnet.synopsys.com>

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

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

Contacting the Synopsys Technical Support Center

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

- Open a support case to your local support center online by signing in to the SolvNet site at <https://solvnet.synopsys.com>, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
 - E-mail support_center@synopsys.com from within North America.
 - Find other local support center e-mail addresses at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages>
- Telephone your local support center.
 - Call (800) 245-8005 from within North America.
 - Find other local support center telephone numbers at <http://www.synopsys.com/Support/GlobalSupportCenters/Pages>

1

Technology File Syntax

A technology file provides technology-specific information, such as the names and physical and electrical characteristics of each metal layer and the routing design rules. This chapter describes the syntax of the technology file. After you create a technology file, you associate it with a Milkyway design library. IC Compiler uses the Milkyway design library to access the technology information.

This chapter provides instructions for creating and loading a technology file and information about each of the technology file sections. A technology file contains process-specific information such as layer thicknesses and the sheet resistance of the various layers.

The chapter is organized into the following sections:

- [Creating a Technology File](#)
- [Basic Syntax Rules](#)
- [Technology File Contents](#)
- [Technology Section](#)
- [PrimaryColor Section](#)
- [Color Section](#)
- [LineStyle Section](#)
- [Tile Section](#)
- [Layer Section](#)

- [LayerDataType Section](#)
- [ContactCode Section](#)
- [DesignRule Section](#)
- [FringeCap Section](#)
- [CapModel and CapTable Sections](#)
- [ResModel Section](#)
- [PRRule Section](#)
- [DensityRule Section](#)
- [SlotRule Section](#)

Creating a Technology File

You can create a technology file by using the syntax descriptions in this chapter or by editing an existing technology file.

To generate an editable version of the technology file associated with a Milkyway design library, use the `write_mw_lib_files -technology` command (or choose File > Export > Write Library File in the GUI). You specify the name of the generated technology file by using the `-output` option.

```
icc_shell> write_mw_lib_files -technology -output techfile.tf
```

Basic Syntax Rules

These are the basic technology file syntax rules:

- The file extension used for an editable technology file is `.tf`.
- A technology file consists of several sections, each of which uses the following syntax:

```
section_name{  
    attribute_name = attribute_value...  
}
```

- Keywords, such as section and attribute names, are case-sensitive and must be typed with the exact spelling and capitalization given. They are reserved words and cannot be used outside the specified context.
- Parentheses (`)` should be typed as they appear in the syntax.
- You can add comments to the technology file by placing a slash and an asterisk (`/*`) at the beginning of a comment and an asterisk and a slash (`*/`) at the end. All text between `/*` and `*/` is ignored.

Technology File Contents

[Example 1-1](#) shows the contents of the technology file.

Example 1-1 Technology File Contents

```
/* Specifies units and unit values */
Technology {
    units
    operating_conditions
    routing_rule_modes
}

/* Defines the six basic colors used to create display colors */
[PrimaryColor {
    primarycolor_attributes
}]

/* Defines the custom display colors used to display designs in the
library */
Color color_value {
    color_attributes
}...

/* Defines the line styles used to display designs in the library */
LineStyle "name" {
    linestyle_attributes
}...

/* Defines the unit tiles */
Tile "name" {
    tile_attributes
}...

/* Defines the layer-specific characteristics, including display */
/* characteristics and layer-specific routing design rules */
Layer "name" {
    display_attributes
    layout_attributes
    parasitic_attributes
    physical_attributes
}...

/* Defines the via masters used in designs in the library */
ContactCode "name" {
    contactcode_attributes
}...
```

```

/* Defines the interlayer routing design rules that apply to designs in
the library */
DesignRule {
    layer_attributes
    rule_attributes
}...

/* Defines capacitance information for interconnect layers */
FringeCap value {
    fringe_cap_attributes
}...

/* Defines timing information */
CapModel {
    capmodel_attributes
}

/* Defines timing information */
CapTable {
    captable_attributes
}

/* Defines the resistance and temperature coefficient of a layer as a
function of wire
width */
ResModel {
    resmodel_attributes
}

/* Defines cell row spacing rules */
PRRule {
    prrrule_attributes
}...

/* Defines density rules */
DensityRule {
    densityrule_attributes
}...

/* Defines slot rules */
SlotRule {
    slotrule_attributes
}...

```

Technology Section

Use the `Technology` section of a Milkyway technology file to specify global attributes, such as units and routing rule modes. [Example 1-2](#) shows an example of a `Technology` section.

Example 1-2 Technology Section

```

Technology {
    /* define units */
    dielectric = 0.000000e+00
    unitLengthName = "micron"
    lengthPrecision = 1000
    gridResolution = 50
    unitTimeName = "ns"
    timePrecision = 100
    unitCapacitanceName = "pf"
    capacitancePrecision = 10000
    unitResistanceName = "kohm"
    resistancePrecision = 10
    unitInductanceName = "nh"
    inductancePrecision = 1000
    unitPowerName = "mw"
    powerPrecision = 10000
    unitVoltageName = "V"
    voltagePrecision = 1000
    unitCurrentName = "mA"
    currentPrecision = 1000000

    /* define routing rule modes */
    minLengthMode = 0
    minAreaMode = 0
    fatTblMinEnclosedAreaMode = 0
    minEdgeMode = 0
    cornerSpacingMode = 0
    fatTblSpacingMode = 0
    parallelLengthMode = 0
    fatWireExtensionMode = 0
    stubMode = 0
    maxStackLevelMode = 1
}

```

Unit Precision and Range

You specify the unit size and precision of each unit (length, resistance, capacitance, and so on) in the `Technology` section. Unit values are stored as 32-bit integers, so the specified unit size and precision determine the maximum dynamic range of the unit in the design.

The range of integers that can be stored is from -2^{31} to $2^{31} - 1$ (–2,147,483,648 to +2,147,483,647). For example, if you set the unit length specification to micron and the length precision specification to 1,000, the maximum dynamic range is from –2,147,483.648 to +2,147,483.647 microns.

If you want to measure capacitance down to 0.0001 picofarad (pF), you need to set unit capacitance to pF and capacitance precision to 10,000. In that case, the maximum capacitance that can be measured is 214,748 pF.

Defining Units

Use the following attributes to define the units in the `Technology` section:

`dielectric`

The `dielectric` attribute specifies the permittivity of SiO_2 . This attribute is not used by IC Compiler and can be omitted from the technology file.

`unitLengthName`

The `unitLengthName` attribute defines the linear distance unit. The valid values are

- `micron`
- `mil`

`lengthPrecision`

The `lengthPrecision` attribute sets the number of database units per user unit, thereby defining the precision of distance measurements stored in the database.

For example, if you set `unitLengthName` to `micron` and `lengthPrecision` to 1,000, the database unit will be 0.001 micron. Thus, all distance measurements are rounded to the nearest 0.001 micron.

`gridResolution`

The `gridResolution` attribute sets the manufacturing grid resolution in database units. For example, if you set `unitLengthName` to `micron` and `lengthPrecision` to 1,000, the database unit is 0.001 micron. If the minimum grid resolution of the manufacturing process is 0.1 micron, `gridResolution` should be set to 100.

When you place an object with IC Compiler, the object's point of origin falls on the grid.

`unitTimeName`

The `unitTimeName` attribute defines the time unit.

[Table 1-1](#) shows the valid units for `unitTimeName`.

Table 1-1 Valid Units for `unitTimeName`

Unit	Value
<code>fs</code>	1×10^{-15} second
<code>ps</code>	1×10^{-12} second
<code>ns</code>	1×10^{-9} second

Table 1-1 Valid Units for unitTimeName (Continued)

Unit	Value
us	1 x 10 ⁻⁶ second
ms	1 x 10 ⁻³ second
s	second

timePrecision

The `timePrecision` attribute defines the precision of time measurements stored in the database.

For example, if you set `unitTimeName` to `ns` and `timePrecision` to 100, the database unit is 0.01 ns. Thus, all time measurements are rounded to the nearest 0.01 ns.

unitCapacitanceName

The `unitCapacitanceName` attribute defines the capacitance unit.

[Table 1-2](#) shows the valid units for `unitCapacitanceName`.

Table 1-2 Valid Units for unitCapacitanceName

Unit	Value
ff	1 x 10 ⁻¹⁵ farad
pf	1 x 10 ⁻¹² farad
nf	1 x 10 ⁻⁹ farad
uf	1 x 10 ⁻⁶ farad
mf	1 x 10 ⁻³ farad
f	farad

capacitancePrecision

The `capacitancePrecision` attribute defines the precision of capacitance measurements stored in the database.

For example, if you set `unitCapacitanceName` to `pf` and `capacitancePrecision` to 10,000, the database unit is 0.0001 picofarad or 0.1 femtofarad. Thus, all capacitance measurements are rounded to the nearest 0.1 femtofarad.

`unitResistanceName`

The `unitResistanceName` attribute defines the resistance unit.

[Table 1-3](#) shows the valid units for `unitResistanceName`.

Table 1-3 Valid Units for `unitResistanceName`

Unit	Value
mohm	1×10^{-3} ohm
ohm	ohm
kohm	1×10^3 ohm
Mohm	1×10^6 ohm

`resistancePrecision`

The `resistancePrecision` attribute defines the precision of resistance measurements stored in the database.

For example, if you set `unitResistanceName` to `ohm` and `resistancePrecision` to 1,000, the database unit is 0.001 ohm. Thus, all resistance measurements are rounded to the nearest 0.001 ohm.

`unitInductanceName`

The `unitInductanceName` attribute defines the inductance unit.

[Table 1-4](#) shows the valid units for `unitInductanceName`.

Table 1-4 Valid Units for `unitInductanceName`

Unit	Value
fH	1×10^{-15} henry
pH	1×10^{-12} henry
nH	1×10^{-9} henry
uH	1×10^{-6} henry

Table 1-4 Valid Units for unitInductanceName (Continued)

Unit	Value
mH	1 x 10 ⁻³ henry
H	henry

inductancePrecision

The **inductancePrecision** attribute defines the precision of inductance measurements stored in the database.

For example, if you set **unitInductanceName** to nH and **inductancePrecision** to 1,000, the database unit will be 0.001 nanohenry (nH). Thus, all inductance measurements are rounded to the nearest 0.001 nH.

unitPowerName

The **unitPowerName** attribute defines the power unit.

[Table 1-5](#) shows the valid units for **unitPowerName**.

Table 1-5 Valid Units for unitPowerName

Unit	Value
fw	1 x 10 ⁻¹⁵ watt
pw	1 x 10 ⁻¹² watt
nw	1 x 10 ⁻⁹ watt
uw	1 x 10 ⁻⁶ watt
mw	1 x 10 ⁻³ watt
w	watt

powerPrecision

The **powerPrecision** attribute defines the precision of power measurements stored in the database.

For example, if you set **unitPowerName** to mw and **powerPrecision** to 1,000, the database unit is 0.001 milliwatt (mw). Thus, all power measurements are rounded to the nearest 0.001 mw.

`unitVoltageName`

The `unitVoltageName` attribute defines the voltage unit.

[Table 1-6](#) shows the valid units for `unitVoltageName`.

Table 1-6 Valid Units for `unitVoltageName`

Unit	Value
fV	1×10^{-15} volt
pV	1×10^{-12} volt
nV	1×10^{-9} volt
uV	1×10^{-6} volt
mV	1×10^{-3} volt
V	volt

`voltagePrecision`

The `voltagePrecision` attribute defines the precision of voltage measurements stored in the database.

For example, if you set `unitVoltageName` to mV and `voltagePrecision` to 1,000, the database unit is 0.001 millivolt (mV). Thus, all power measurements are rounded to the nearest 0.001 mV.

`unitCurrentName`

The `unitCurrentName` attribute defines the current unit.

Table 1-7 shows the valid units for `unitCurrentName`.

Table 1-7 Valid Units for `unitCurrentName`

Unit	Value
fA	1 x 10 ⁻¹⁵ ampere
pA	1 x 10 ⁻¹² ampere
nA	1 x 10 ⁻⁹ ampere
uA	1 x 10 ⁻⁶ ampere
mA	1 x 10 ⁻³ ampere
A	ampere

`currentPrecision`

The `currentPrecision` attribute defines the precision of current measurements stored in the database.

For example, if you set `unitCurrentName` to milliampere (mA) and `currentPrecision` to 1,000, the database unit is 0.001 mA. Thus, all power measurements are rounded to the nearest 0.001 mA.

Defining Routing Rule Modes

In the `Technology` section, use the following attributes to define the routing rule modes.

`minLengthMode`

Determines whether via cuts are considered in applying the minimum length rule.

Table 1-8 shows the valid values for the `minLengthMode` attribute.

Table 1-8 Valid Values for `minLengthMode`

Mode	Description
0 (the default)	IC Compiler uses the total wire length to check the minimum length rule.
1	IC Compiler uses wire segments that include via cuts to check the minimum length.

For more information, see [“Minimum Length Rule” on page 2-3](#).

`minAreaMode`

Determines whether the shapes of areas are considered in applying the minimum area rule. [Table 1-9](#) shows the valid values for the `minAreaMode` attribute.

Table 1-9 Valid Values for minAreaMode

Mode	Description
0 (the default)	Ignores the <code>specialMinArea</code> value and honors the <code>minArea</code> rule.
1	Honors the <code>specialMinArea</code> value if the polygon is not a rectangle.

For more information, see [“Two-Stage Special Minimum Area Rule” on page 2-6](#).

`fatTblMinEnclosedAreaMode`

Determines how to apply the minimum enclosed area rule. [Table 1-10](#) shows the valid values for the `fatTblMinEnclosedAreaMode` attribute.

Table 1-10 Valid Values for fatTblMinEnclosedAreaMode

Mode	Description
0 (the default)	The fat wire minimum enclosed area mode is triggered when any of the surrounding metal satisfies the width requirement.
1	The fat wire minimum enclosed area mode is triggered only when all of the surrounding metal satisfies the width requirement.

For more information, see [“Minimum Enclosed Area Rule” on page 2-9](#).

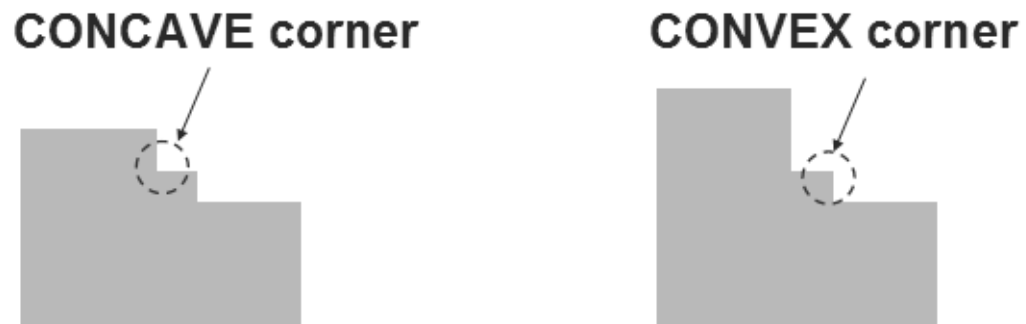
minEdgeMode

Controls the scope of the check for the minimum edge rules. [Table 1-11](#) shows the valid values for the `minEdgeMode` attribute.

Table 1-11 Valid Values for `minEdgeMode`

Mode	Description
0 (the default)	Needs a concave corner to trigger a violation. A concave corner is formed by two adjacent edges when both are less than minimum length. Figure 1-1 shows examples of convex and concave corners.
1	The minimum edge length is violated if the total number of consecutive minimum edges is greater than the value specified by <code>maxNumMinEdge</code> or if the total edge length is greater than <code>maxTotalMinEdgeLength</code> .

Figure 1-1 Concave and Convex Corners



For more information, see [“Minimum Edge Rules” on page 2-15](#).

`cornerSpacingMode`

Specifies whether diagonal or rectilinear distance measurement is used for via spacing. If this attribute is set to 0, diagonal measurement is used; the actual spacing, measured diagonally, must satisfy the via minimum spacing requirement. If it is set to 1, rectilinear measurement is used; at least the x-direction or y-direction must satisfy the via minimum spacing requirement. [Table 1-12](#) shows the valid values for the `cornerSpacingMode` attribute.

Table 1-12 Valid Values for cornerSpacingMode

Mode	Description
0 (the default)	The corner-to-corner via spacing is measured by using the diagonal distance.
1	The corner-to-corner via spacing is measured by using the Manhattan, rectilinear distance.

For more information, see [“Via Corner Spacing Rule” on page 2-29](#).

`fatTblSpacingMode`

Determines the effect of the parallel length (defined with the `fatTblParallelLength` attribute) on the indexing of the `fatTblSpacing` attribute. [Table 1-13](#) shows the valid values for the `fatTblSpacingMode` attribute.

Table 1-13 Valid Values for fatTblSpacingMode

Mode	Description
0 (the default)	Downgrades the <code>fatTblSpacing</code> index according to the value of the <code>fatTblParallelLength</code> attribute.
1	Downgrades the <code>fatTblSpacing</code> index by a maximum of one.

For more information, see [“Fat Metal Spacing Table Rule” on page 2-43](#).

`parallelLengthMode`

The `parallelLengthMode` attribute controls the merging of wire segments based on the parallel length rule. [Table 1-14](#) shows the valid values for the `parallelLengthMode` attribute.

Table 1-14 Valid Values for `parallelLengthMode`

Mode	Description
0 (the default)	Merges wire segments only with fat neighboring shapes.
1	Merges wire segments with all neighboring shapes.

For more information, see [“Fat Metal Parallel Length” on page 2-49](#).

`fatWireExtensionMode`

The `fatWireExtensionMode` attribute controls the application of the fat metal extension spacing rule. [Table 1-15](#) shows the valid values for the `fatWireExtensionMode` attribute.

Table 1-15 Valid Values for `fatWireExtensionMode`

Mode	Description
0 (the default)	The fat spacing check is performed only on extension wires connected to the fat wire and within the extension range from the fat wire edges and corners. The spacing is checked between any two wires. Projection length is not checked.
1	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Spacing between both overlapped and non-overlapped wire pairs is checked. Projection lengths are also checked.
2	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Only the spacing between non-overlapped wire pairs is checked. Projection lengths are not checked.

Table 1-15 Valid Values for fatWireExtensionMode (Continued)

Mode	Description
3	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Only the spacing between overlapped wire pairs is checked. Projection lengths of these wires are also checked.
4	The fat spacing check is performed only on connected wires within the extension range from the fat wire edges, not including fat wire corners. The spacing from these wires to all other wires is checked. Projection lengths are not checked.

For more information, see [“Fat Metal Extension Spacing Rule” on page 2-50](#).

stubMode

Specifies when the stub spacing rule is applied, rather than the minimum spacing rule.

[Table 1-16](#) shows the valid values for the `stubMode` attribute.

Table 1-16 Valid Values for stubMode

Mode	Description
0 (the default)	Stub spacing is used when the distance that two objects run parallel to each other is less than or equal to <code>stubThreshold</code> .
1	Stub spacing is used when an end-of-line metal segment has an edge width that is less than or equal to <code>stubThreshold</code> .
2	Stub spacing is used when a metal segment with a width less than or equal to <code>stubThreshold</code> has neighboring metal shapes along two adjacent edges or any one edge, which is less than <code>stubThreshold</code> from the corner of two adjacent edges.

Table 1-16 Valid Values for stubMode (Continued)

Mode	Description
3	Stub spacing is used when a metal with a width less than or equal to <code>stubThreshold</code> has neighboring metals along three adjacent edges and has neighboring metal shapes along two adjacent edges within <code>stubThreshold</code> .
4	Stub spacing is used when a metal segment has a width that is less than <code>stubThreshold</code> and there is no connecting metal within <code>minWidth</code> , and if it has neighboring metal shapes along two adjacent edges or any one edge that is less than <code>stubThreshold</code> from the corner of two adjacent edges.

For more information, see [“End-of-Line Spacing Rules” on page 2-60](#).

`maxStackLevelMode`

Controls the scope of the via array maximum stack level rule. [Table 1-17](#) shows the valid values for the `maxStackLevelMode` attribute.

Table 1-17 Valid Values for maxStackLevelMode

Mode	Description
0 (the default)	Ignores the maximum stack level rule check when all stacked vias are via arrays.
1	Checks the maximum stack level rule for stacked via arrays.
2	Ignores the maximum stack level rule check when at least one stacked via is a via array.
3	Ignores the maximum stack level rule check when all stacked vias are aligned via arrays (overlapped with at least two cuts).

For more information, see [“Via Stack Rule” on page 2-140](#).

`fixedColor`

Enables or disables cell-level mask swapping during stream-out of physical design data in multiple-patterning technologies. To enable the mask swapping feature, set this attribute to 0 (or omit it from the technology file). To disable the mask swapping feature, set this attribute to 1.

PrimaryColor Section

The `PrimaryColor` section of the technology file defines the six basic colors that IC Compiler uses to create display colors.

To understand how the system combines primary colors to create display colors, you need to understand how a computer monitor creates colors on the screen.

Colors on a Computer Monitor

In any system, the term *primary colors* refers to the colors used to define all other colors. A computer monitor has three primary colors; IC Compiler has six.

All colors displayed on a computer monitor are created by combining red, green, and blue light. Combining colors of light produces results that are very different from the results that come from combining the same colors of paint. Before you decide to change your colors, note the following results that come from combining colors of light:

- green + red = yellow
- green + blue = cyan
- blue + red = magenta

The intensity of any primary color on a computer monitor (red, green, or blue) can be from 0 to 255, with 255 producing the brightest intensity of the color.

If you are working with computer graphics for the first time, you might find that combining colors can produce unexpected results.

Colors in IC Compiler

The `PrimaryColor` section of a technology file defines the six primary colors for IC Compiler: two intensities of red (`lightRed` and `mediumRed`), two intensities of green (`lightGreen` and `mediumGreen`), and two intensities of blue (`lightBlue` and `mediumBlue`). The intensity value is a number between 0 and 255.

IC Compiler uses combinations of these six primary colors to create the display colors you see when you display a cell in the layout window. If you do not include the `PrimaryColor` section in the technology file, IC Compiler uses the default values shown in [Example 1-3](#).

Example 1-3 Defining the Primary Colors

```
PrimaryColor {
    lightRed = 90
    mediumRed = 180
    lightGreen = 80
    mediumGreen = 175
    lightBlue = 100
    mediumBlue = 190
}
```

Color Section

The primary colors defined in the `PrimaryColor` section (or the default primary colors if the technology file does not contain a `PrimaryColor` section) determine the default display colors. There are 64 display colors, which are stored as 6-bit binary numbers. Each bit represents one of the primary colors, as follows:

Medium red	Light red	Medium green	Light green	Medium blue	Light blue
---------------	--------------	-----------------	----------------	----------------	---------------

For example, color number 51, stored as the binary number 110011, consists of the primary colors represented by the bits that are set, including the following:

- Medium red
- Light red
- Medium blue
- Light blue

As described in [Example 1-3](#), the two intensities for red are

```
lightRed 90
mediumRed 180
```

If you add the two intensities together, the result is 270. Because a monitor cannot produce an intensity greater than 255, a display color that combines light red and medium red is red in an intensity of only 255.

In the case of color number 51, the combination of medium red and light red exceeds the maximum intensity of red the monitor can produce. Likewise, the combination of medium blue and light blue exceeds the maximum intensity of blue the monitor can produce. Therefore, creating color number 51 means combining red at an intensity of 255 and blue at an intensity of 255.

You can override the default display color for any color number by defining a custom display color in a `Color` section of the technology file.

The display colors define the colors that IC Compiler uses to display designs in the library. The `Layer` section uses the display colors to specify how layers are displayed.

Note:

When two display colors overlap, the color produced is the result of an OR operation between the two color numbers. For example, if color number 51 (110011) overlaps color number 56 (111000), the result is color number 59 (111011).

You define a custom display color by defining the following attributes in the `Color` section:

`name`

Specifies the name of the color. The name is a user-defined string of up to 31 characters.

`rgbDefined`

Indicates whether the color is red, green, blue (RGB)-defined. Valid values are 1 or 0.

`redIntensity`

Specifies the color's red intensity. Valid values are integers from 0 to 255.

`greenIntensity`

Specifies the color's green intensity. Valid values are integers from 0 to 255.

`blueIntensity`

Specifies the color's blue intensity. Valid values are integers from 0 to 255.

[Example 1-4](#) shows a `Color` section that defines color 62 ("owhite"):

Example 1-4 Defining a Color

```
Color 62 {  
    name = "owhite"  
    rgbDefined = 1  
    redIntensity = 255  
    greenIntensity = 255  
    blueIntensity = 230  
}
```

LineStyle Section

The `LineStyle` section of the technology file defines the line styles IC Compiler uses to display designs in the library. The `Layer` section uses the line styles to determine how layers are displayed.

You define a line style by defining the following attributes in the `LineStyle` section:

`name`

Specifies a name that identifies the line style. The name is a user-defined string of up to 15 characters.

`width`

Specifies the horizontal dimension of the line style pattern in pixels.

`height`

Specifies the vertical dimension of the line style pattern in pixels.

`pattern`

Defines the line style pattern, with 1s representing pixels drawn with the color assigned to the layer and 0s representing pixels with no color (transparent). The pattern representation is enclosed in parentheses.

Note:

Because line styles are predefined, they do not actually need to appear in the technology file.

[Example 1-5](#) shows the definition for a line style named “boundary.”

Example 1-5 Specifying a Line Style

```
LineStyle "boundary" {  
    width = 10  
    height = 1  
    pattern = (1, 0, 0, 1, 1, 1, 1, 1, 0, 0)  
}
```

Tile Section

A `Tile` section defines the unit tiles.

You define a unit tile by defining the following attributes in the `Tile` section:

`name`

Specifies the name of the unit tile. The name is a user-defined string of up to 31 characters.

`width`

Specifies the width of a tile.

`height`

Specifies the height of a tile.

[Example 1-6](#) shows a typical unit tile definition.

Example 1-6 Tile Section of a Technology File

```
Tile "unit" {  
    width = 16  
    height = 168  
}
```

Layer Section

A technology file defines each layer by specifying attributes for that layer. The layer can be a metal layer or a via layer. The attributes fall into several groupings, including

- Layout attributes

Layout attributes associate a physical layer in the layout with the display layer.

- Display attributes

Display attributes specify how objects on the layer are displayed.

- Parasitic attributes

Parasitic attributes define the layer parasitics for a metal layer.

Note:

You define the parasitic attributes for a via layer in a `ContactCode` section.

- Physical attributes

Physical attributes define physical characteristics of the layer and need to be specified if you are using timing-driven layout.

- Design rule attributes

Design rule attributes define the layer-specific design rules associated with objects on the layer.

Example 1-7 shows a selection of attributes that can be used in the `Layer` section.

Example 1-7 *Layer Section*

```
Layer "MET1" {
    /* layout attributes */
    layerNumber = 8
    isDefaultLayer = 0
    maskName = "metall1"
    pitch = 2.2

    /* display attributes */
    color = "blue"
    lineStyle = "solid"
    pattern = "dot"
    blink = 0
    visible = 1
    selectable = 1
    panelNumber = 0

    /* parasitic attributes */
    unitMinResistance = 0
    unitNomResistance = 0
    unitMaxResistance = 0
    unitMinCapacitance = 0
    unitNomCapacitance = 0
    unitMaxCapacitance = 0
    unitMinInductance = 0
    unitNomInductance = 0
    unitMaxInductance = 0
    unitMinSideWallCap = 0
    unitNomSideWallCap = 0
    unitMaxSideWallCap = 0
    unitMinChannelCap = 0
    unitNomChannelCap = 0
    unitMaxChannelCap = 0
    unitMinChannelSideCap = 0
    unitNomChannelSideCap = 0
    unitMaxChannelSideCap = 0
    maxCurrDensity = 0
    maxIntraCapDistRatio = 0
    maxSegLenForRC = 0
}
```



```

/* physical attributes */
unitMinHeightFromSub = 0
unitNomHeightFromSub = 0
unitMaxHeightFromSub = 0
unitMinThickness = 0
unitNomThickness = 0
unitMaxThickness = 0

/* design rule attributes */
maxWidth = 1.0
minWidth = 1.0
nonPreferredWidth = 1.0
defaultWidth = 1.0
xdefaultWidth = 1.0
ydefaultWidth = 1.0
minLength = 1.4
maxLength = 1.4
minArea = 1.0
specialMinArea = 1.0
minAreaEdgeThreshold = 1.0
specialMinAreaTblSize = 0
minAreaEdgeThresholdTbl = (0, 0)
minAreaRectMinLengthTbl = (0, 0)
minAreaRectMinWidthTbl = (0, 0)
minAreaFillMinLengthTbl = (0, 0)
minAreaFillMinWidthTbl = (0, 0)
specialMinAreaTbl = (0, 0)
minEnclosedArea = 1.0
minEnclosedWidth = 1.0
minEdgeLength = 0
maxNumMinEdge = 0
maxTotalMinEdgeLength = 0
minEdgeLength2 = 0
minEdgeLength3 = 0
minEdgeLengthTblSize = 0
minEdgeLengthTbl = (0, 0)
minSpacing = 1.0
cornerMinSpacing = 1.0
sameNetMinSpacing = 0
checkManhattanSpacing = 0
cutTblSize = 0
cutNameTbl = (name1, name2, name3)
cutWidthTbl = (0, 0, 0)
cutHeightTbl = (0, 0, 0)
cutDataTypeTbl = (0, 0, 0)
maxNumAdjacentCut = 0
enclosedCutNumNeighbor = 1.0
enclosedCutNeighborRange = 1.0
enclosedCutMinSpacing = 1.0
enclosedCutToNeighborMinSpacing = 1.0
onWireTrack = 0
endOfLine1NeighborEndToEndThreshold = 0
endOfLine1NeighborEndToEndThreshold2 = 0

```

```

endOfLine1NeighborEndToEndMinLength = 0
endOfLine1NeighborEndToEndParallelWidth = 0
endOfLine1NeighborEndToEndMinSpacing = 0
endOfLine1NeighborThreshold = 0
endOfLine1NeighborWireMinThreshold = 0
endOfLine1NeighborMinLength = 0
endOfLine1NeighborMinSpacing = 0
endOfLine1NeighborCornerKeepoutWidth = 0
endOfLine2NeighborThreshold = 0
endOfLine2NeighborWireMinThreshold = 0
endOfLine2NeighborMinSpacing = 0
endOfLine2NeighborSideMinSpacing = 0
endOfLine2NeighborCornerKeepoutWidth = 0
endOfLine2NeighborSideKeepoutLength = 0
endOfLine3NeighborThreshold = 0
endOfLine3NeighborMinSpacing = 0
endOfLine3NeighborSideMinSpacing = 0
endOfLine3NeighborCornerKeepoutWidth = 0
endOfLine3NeighborSideKeepoutLength = 0
stubThreshold = 0
stubLengthThreshold = 0
stubSpacing = 0
stubToStubSpacing = 0
endOfLineCornerKeepoutWidth = 0
stubMinLength = 0
tJunctionStubWireMaxThreshold = 0
tJunctionOrthoWireMaxThreshold = 0
tJunctionStubKeepoutMinSpacing = 0
tJunctionStubKeepoutMinWidth = 0
uShapeMinSpacing = 0
uShapeDepthThreshold = 0
uShapeMinLength = 0
sameNetWidthThreshold = 0
maxStackLevel = 0
fatWireThreshold = 0
fatFatMinSpacing = 0
fatThinMinSpacing = 0
fatWireExtensionRange = 0
fatTblDimension = 0
fatTblThreshold = (0, 0, 0)
fatTblThreshold2 = (0, 0, 0)
fatTblParallelLength = (0, 0, 0)
fatTblExtensionRange = (0, 0, 0)
fatTblSpacing = (0, 0, 0, 0, 0, 0, 0, 0)
fatTblMinEnclosedArea = (0, 0, 0)
fatTblXDimension = 0
fatTblYDimension = 0
fatTblXThreshold = (0, 0)
fatTblYThreshold = (0, 0)
fatTblXParallelLength = (0, 0)
fatTblYParallelLength = (0, 0)
fatTblXMinSpacing = (0, 0, 0, 0)
fatTblYMinSpacing = (0, 0, 0, 0)

```

```

orthoSpacingExcludeCorner = 0
fatContactThreshold = 0
fatTblFatContactNumber = (0, 0)
fatTblFatContactMinCuts = (0, 0)
fat2DTblFatContactNumber = (0, 0, 0, 0)
fat2DTblFatContactMinCuts = (0, 0, 0, 0)
fatTblDimension2 = 0
fatTblThreshold2 = (0, 0, 0)
fatTblAreaDimension = 0
fatTblAreaThreshold = (0, 0, 0)
fatTblFatContactNumber = (0, 0, 0)
fatTblFatContactMinCuts = (0, 0, 0)
fatTblExtensionDimension = 0
fatTblExtensionThreshold = (0, 0, 0)
fatTblExtensionRangeDimension = 0
fatTblExtensionRange = (0, 0, 0)
fatTblExtensionAreaDimension = 0
fatTblExtensionAreaThreshold = (0, 0, 0)
fatTblExtensionContactNumber = (0, 0, 0)
fatTblExtensionMinCuts = (0, 0, 0)
spanTblDimension = 0
spanTblThreshold = (0, 0, 0, 0)
spanTblParallelLength = (0, 0, 0, 0)
spanTblMinSpacing = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
protrusionTblDim = 0
protrusionFatThresholdTbl = (0, 0, 0)
protrusionLengthLimitTbl = (0, 0, 0)
protrusionMinWidthTbl = (0, 0, 0)
signalRouteMaxWidth = 0
xLegalWidthTblSize = 0
yLegalWidthTblSize = 0
xLegalWidthTbl = (0, 0, 0)
yLegalWidthTbl = (0, 0, 0)
nonPreferredRouteMode = 0
convexConcaveMinEdgeLength = 0
convexMinEdgeLength = 0
hShape1WidthThreshold = 0
hShape2LengthThreshold = 0
hShapeMinWidth = 0
hShapeMinLength = 0
minEdgeLengthTblSize = 0
minEdgeLengthTbl = (0, 0, 0)
minEdgeLengthFacingEdgeExcluded = 0
minSpacingCornerKeepoutWidth = 0
sameSegAlignedUpperWireMaxSpacingThreshold = 0
sameSegAlignedLowerWireMaxSpacingThreshold = 0
sameSegAlignedCutMinSpacing = 0
endOfLine1NeighborEndToEndTblSize = 0
endOfLine1NeighborEndToEndTblThreshold = (0, 0, 0)
endOfLine1NeighborEndToEndTblMinSpacing = (0, 0, 0)
endOfLine1NeighborEndToEndTblParallelWidth = (0, 0, 0)
endOfLine1NeighborThreshold = 0
endOfLine1NeighborMinSpacing = 0

```

```

endOfLine1NeighborCornerKeepoutWidth = 0
endOfLine1NeighborLongEdgeExcluded = 0
endOfLine1NeighborTblSize = 0
endOfLine1NeighborThresholdTbl = (0, 0, 0)
endOfLine1NeighborMinSpacingTbl = (0, 0, 0)
endOfLine1NeighborCornerKeepoutWidthTbl = (0, 0, 0)
endOfLine1NeighborMod1MaxThreshold = 0
endOfLine1NeighborMod1MaxThreshold2 = 0
endOfLine1NeighborMod1MaxLength = 0
endOfLine1NeighborMod1MinSpacing = 0
endOfLine1NeighborMod1ConvexCornerIncluded = 0
endOfLine2NeighborMinLength = L2
endOfLine2NeighborMod1Threshold = 0.070
endOfLine2NeighborMod1WireMinThreshold = 0.072
endOfLine2NeighborMod1MinSpacing = 0.075
endOfLine2NeighborMod1SideKeepoutWidth = 0.150
endOfLine2NeighborMod1CornerKeepoutWidth = 0.030
endOfLine2NeighborMod1SideKeepoutLength = 0.075
endJointOrthoWireMaxThreshold = 0
endWireSideEdgeLengthThreshold = 0
jointWireSpanThreshold = 0
jointToConvexCornerMaxDist = 0
endJointWireThresholdTblSize = 0
endJointWireWidthMaxThresholdTbl = (0, 0, 0)
endJointWireParallelWidthThresholdTbl = (0, 0, 0)
endWireMaxSpacingThreshold = 0
jointWireMaxSpacingThreshold = 0
endEndWireMinSpacing = 0
jointJointWireMinSpacing = 0
endJointWireMinSpacing = 0
jointEndWireMinSpacing = 0

```

Layout Attributes

The layout attributes associate a physical layer in the layout with the display layer.

The layout attributes are listed and described here.

`layerNumber`

Defines the number that identifies the layer.

Valid values are 1–187 (user-defined) and 188–255 (system-defined). Layer numbers 188 through 255 are reserved for IC Compiler to create layers for place and route functionality.

`isDefaultLayer`

Specifies the layer used for routing when there are multiple layers with the same `maskName` value.

Valid values are 0 or 1. Only one layer with the same mask should be designated as the default.

`maskName`

Defines the physical layer associated with the display layer.

The following standard mask names are recognized by IC Compiler:

- `poly`
- `polyCont`
- `metal1, metal2, ... metal15`
- `via1, via2, ... via15`
- `nwell, pwell, deepNwell, deepPwell`
- `passivation` (for bonding pad pins)
- `tsv` (for through-silicon vias)
- `bmetal1, bmetal2, bmetal3` (for backside metal layers)
- `bvia1, bvia2` (for backside vias)

In addition to these standard mask names, you can specify any string of up to 31 characters to be used as a mask name in the Milkyway database. In the Milkyway Environment tool and IC Compiler, these layers with user-defined mask names can be visualized and edited.

`pitch`

Defines the predominant separation distance between the centers of objects on the layer.

The Milkyway Environment tool uses the specified pitch to generate wire tracks in the unit tile.

Note:

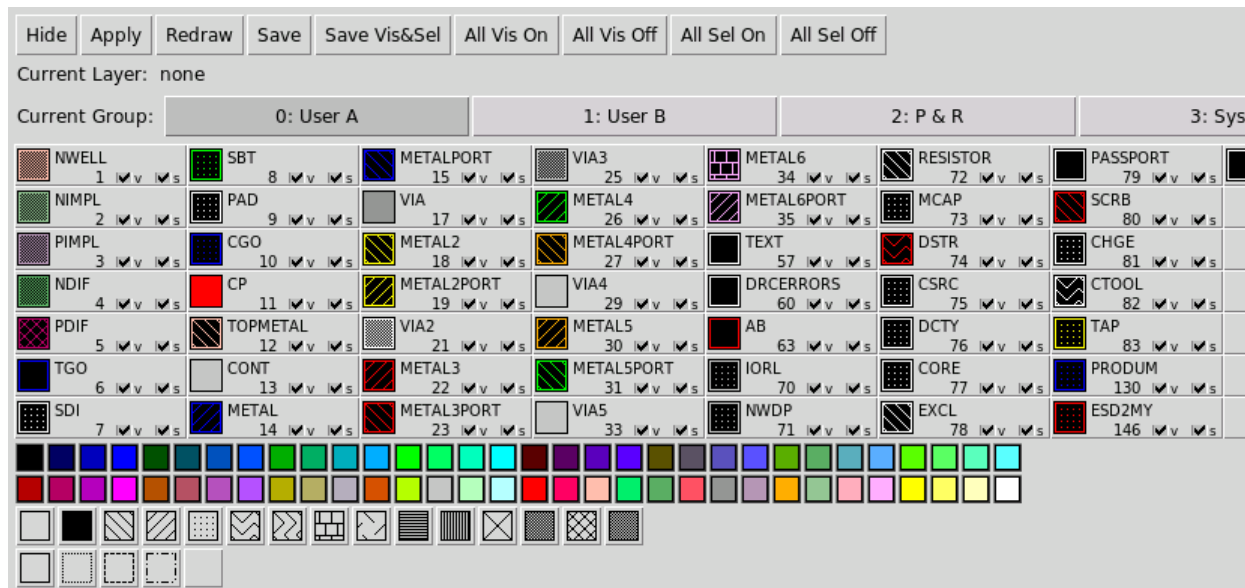
You cannot change the `metal2` pitch after library data preparation because the Milkyway Environment tool uses the `metal2` pitch during blockage, pin, and via (BPV) extraction.

Display Attributes

This section describes the display attributes and their valid values. The display attributes specify how objects on the layer are displayed.

In the Milkyway Environment tool, you can display the available colors, patterns, line styles, and current settings for all the layers defined in the technology file. In a cell layout view window, click the Layer Panel button to open a display similar to the one shown in [Figure 1-2](#).

Figure 1-2 Layer Panel in Milkyway Environment Tool



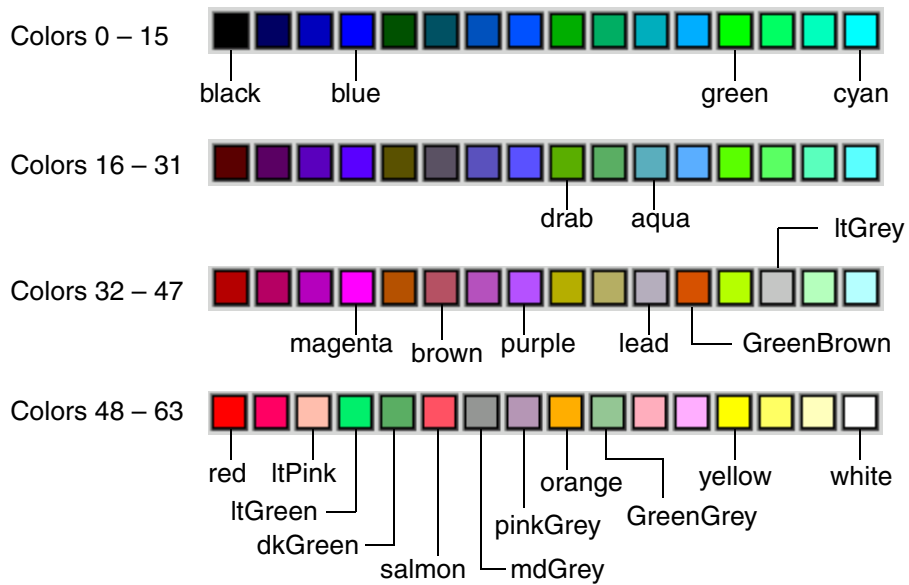
color

Defines the name or number of the color used to display the layer.

Valid values are any integer from 0 to 63 or the name of a color that is defined in the `Color` section.

If you specify a number, the color does not have to be defined in the `Color` section. (See [“Colors in IC Compiler” on page 1-19.](#))

The standard colors and their predefined names are shown in [Figure 1-3](#).

Figure 1-3 Layer Display Colors

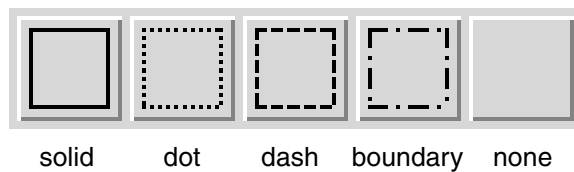
Colors that have a name can be referred to by either the name or number, such as “blue” or “3”. The rest can be referred to by number only.

lineStyle

Sets the defined line style for objects on the layer.

Valid values are the name of a style defined in the `LineStyle` section. (See “[LineStyle Section](#)” on page 1-22.)

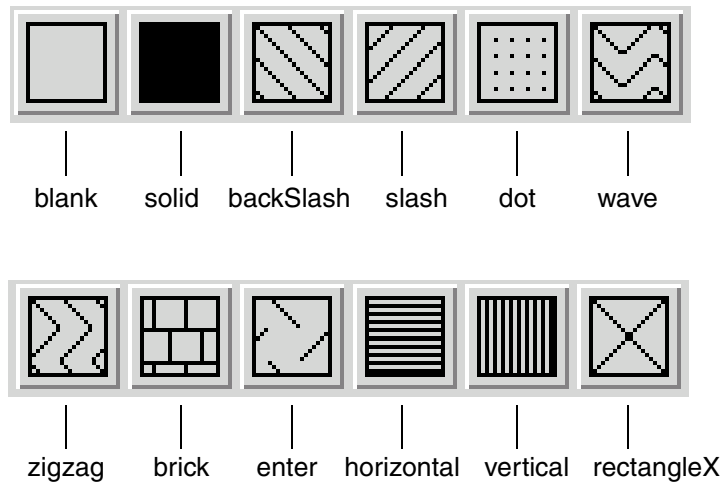
The standard line styles and their predefined names are shown in [Figure 1-4](#).

Figure 1-4 Layer Display Line Styles

pattern

Sets the pattern displayed inside the layer geometries.

The standard patterns and their predefined names are shown in [Figure 1-4](#).

Figure 1-5 Layer Display Patterns

`blink`

Sets the layer to blink or not to blink.

Valid values are 1 (on) or 0 (off).

`visible`

Sets the layer visibility.

Valid values are 1 (on) or 0 (off).

`selectable`

Sets the layer selectability.

Valid values are 1 (on) or 0 (off).

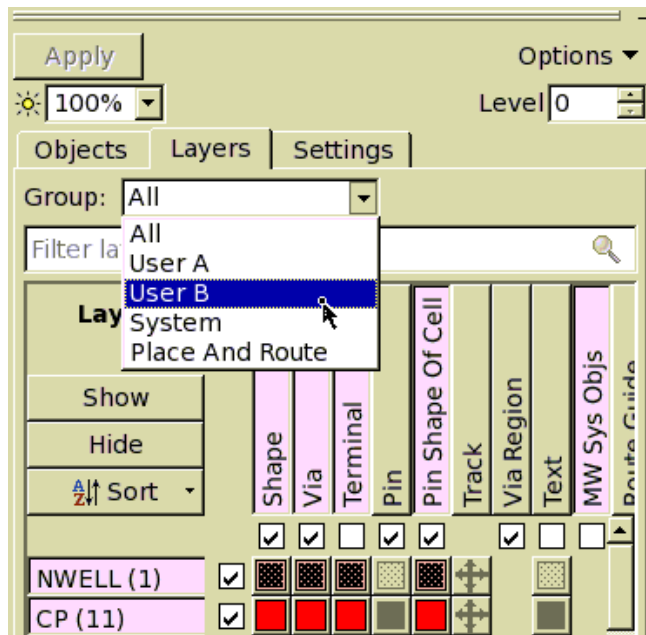
`panelNumber`

Specifies the group to which the layer belongs. Valid values are integers from 0 to 3, representing the following layer groups:

- 0 – User A, the default group for user-defined layers
- 1 – User B, an alternative user-defined layer group
- 2 – Place and Route, a group used for blockages, channels, and similar objects
- 3 – System, the system-defined layer group

In the View Settings panel of the IC Compiler GUI, the Layers tab shows information about the layers defined in the technology file. The Group setting lets you restrict the list of layers displayed in the panel to a specified group, as shown in [Figure 1-6](#).

Figure 1-6 Layer Group Display Options in View Settings Panel



Parasitic Attributes

The parasitic attributes define the metal layer parasitics. In general, IC Compiler gets the parasitic information from the TLUPlus files rather than from the technology file; however, extraction does use the maximum intracapacitance distance ratio and the wire segment length attributes.

This section describes the following parasitic attributes:

- [Resistance](#)
- [Capacitance](#)
- [Inductance](#)
- [Sidewall Capacitance](#)
- [Routing Channel Capacitance](#)
- [Total Sidewall Routing Channel Capacitance](#)
- [Maximum Current Density](#)

- [Maximum Intracapacitance Distance Ratio](#)
- [Wire Segment Length](#)

Note:

For via layers, you specify the parasitic attributes in the `ContactCode` section. (See [“ContactCode Section” on page 1-50.](#))

Resistance

Resistance is defined as the resistance per square (length divided by width) of a poly or metal layer. The resistance attributes are

`unitMinResistance`

A floating-point number representing the minimum resistance.

`unitNomResistance`

A floating-point number representing the nominal resistance.

`unitMaxResistance`

A floating-point number representing the maximum resistance.

Capacitance

Capacitance is defined per square user unit of a poly or metal layer in a cell instance or over a macro. The capacitance attributes are

`unitMinCapacitance`

A floating-point number representing the minimum capacitance.

`unitNomCapacitance`

A floating-point number representing the nominal capacitance.

`unitMaxCapacitance`

A floating-point number representing the maximum capacitance.

Inductance

Inductance is defined per unit length of a poly or metal layer.

Note:

IC Compiler does not use the inductance values.

The inductance attributes are

`unitMinInductance`

A floating-point number representing the minimum inductance.

`unitNomInductance`

A floating-point number representing the nominal inductance.

`unitMaxInductance`

A floating-point number representing the maximum inductance.

Sidewall Capacitance

Sidewall capacitance is defined as the total sidewall capacitance per unit length for both sides of a poly or metal layer in a cell instance or over a macro.

The sidewall capacitance attributes are

`unitMinSideWallCap`

A floating-point number representing the minimum capacitance.

`unitNomSideWallCap`

A floating-point number representing the nominal capacitance.

`unitMaxSideWallCap`

A floating-point number representing the maximum capacitance.

If you specify a zero (0), IC Compiler calculates sidewall capacitance based on the height from the substrate and the thickness of the layer, as specified in the physical attributes. (See [“Physical Attributes” on page 1-37.](#))

Routing Channel Capacitance

Routing channel capacitance is defined per square user unit of a poly or metal layer in a routing channel.

The routing channel capacitance attributes are

`unitMinChannelCap`

A floating-point number representing the minimum capacitance.

`unitNomChannelCap`

A floating-point number representing the nominal capacitance.

`unitMaxChannelCap`

A floating-point number representing the maximum capacitance.

Total Sidewall Routing Channel Capacitance

Total sidewall routing channel capacitance is defined as the total sidewall capacitance per unit length for both sides of a poly or metal layer in a routing channel.

The sidewall routing channel capacitance attributes are

`unitMinChannelSideCap`

A floating-point number representing the minimum capacitance.

`unitNomChannelSideCap`

A floating-point number representing the nominal capacitance.

`unitMaxChannelSideCap`

A floating-point number representing the maximum capacitance.

Maximum Current Density

Use the `maxCurrDensity` attribute to define the maximum current density in amps per centimeter of thickness for the layer. For each layer, the actual current density is the `maxCurrDensity` value divided by the route width in centimeters.

Maximum Intracapacitance Distance Ratio

Use the `maxIntraCapDistRatio` attribute to specify the distance ratio required to control the extraction range. This attribute is a unitless floating-point number.

For example, where three times the `minSpacing` is required to control the extraction, specify a value of 3 for the `maxIntraCapDistRatio` attribute.

Wire Segment Length

Use the `maxSegLenForRC` attribute to define the maximum length in user units of a wire segment on the layer.

During extraction, IC Compiler splits wire segments longer than the specified length into segments of the specified length or less to compute a more accurate model. If you specify zero (0), the wire segments on the layer are not split.

Physical Attributes

The physical attributes define physical characteristics of the layer. This information is needed to determine the timing characteristics of the design. This section describes the following physical attributes:

- [Height From Substrate](#)
- [Thickness](#)

Height From Substrate

The height from the substrate is the distance in user units between the layer and the substrate.

The height attributes are

`unitMinHeightFromSub`

A floating-point number representing the minimum distance.

`unitNomHeightFromSub`

A floating-point number representing the nominal distance.

`unitMaxHeightFromSub`

A floating-point number representing the maximum distance.

These values are used for calculating the sidewall capacitance when you do not specify sidewall capacitance. (See [“Parasitic Attributes” on page 1-33.](#))

The height from substrate values are stored internally as double-precision numbers and therefore are not limited by the database per-user unit limitations described in [“Unit Precision and Range” on page 1-6.](#)

Thickness

Thickness is the vertical thickness in user units of a poly or metal layer.

The thickness attributes are

`unitMinThickness`

A floating-point number representing the minimum thickness.

`unitNomThickness`

A floating-point number representing the nominal thickness.

`unitMaxThickness`

A floating-point number representing the maximum thickness.

These values are stored internally as double-precision numbers and therefore are not limited by the database unit limitations described in [“Unit Precision and Range” on page 1-6](#).

Design Rule Attributes

The design rule attributes in a `Layer` section define layer-specific design rules; they apply only to the associated layer. All measurements in this section are in the user units specified in the `Technology` section of the technology file. (See [“Technology Section” on page 1-5](#).) For more information about these design rules, see [Chapter 2, “Routing Design Rules.”](#)

Note:

Interlayer design rules are defined in the `DesignRule` section. For more information, see [“DesignRule Section” on page 1-57](#).

These are design rule attributes used in the `Layer` section:

`maxWidth`

Specifies the value used to identify wide metals for slotting and allows the `signoff_drc` command to check the result of issuing the `slot_wire` command.

Note:

The signal router does not recognize the `maxWidth` rule.

`minWidth`

Defines the minimum width of any dimension of an object on the layer.

`defaultWidth`

Defines the default width of any dimension of an object on the layer.

The default width is used with all operations except the design rule checker (DRC), which uses the minimum width.

`minLength`

Defines the minimum wire length allowed on the layer.

`maxLength`

Defines the maximum length of an object (rectangle or polygon) on the layer. This rule applies only to the classic router, not Zroute. If this attribute is set to 0, the rule is unspecified for the layer, and the router does not check it.

`minArea`

The minimum area for any object on the layer.

`specialMinArea`

The minimum area for a special shape in the two-stage minimum area rule.

`minAreaEdgeThreshold`

The edge length threshold used to define special shapes for the two-stage minimum area rule.

`specialMinAreaTblSize`

The table size for the multistage minimum area rules.

`minAreaEdgeThresholdTbl`

The edge thresholds for the polygon-edge-based multistage minimum area rule.

`minAreaRectMinLengthTbl`

The rectangular area minimum-length thresholds for the rectangular multistage minimum area rule.

`minAreaRectMinWidthTbl`

The rectangular area minimum-width thresholds for the rectangular multistage minimum area rule.

`minAreaFillMinLengthTbl`

The fill area minimum-length thresholds for the polygon-edge-based multistage minimum area rule.

`minAreaFillMinWidthTbl`

The fill area minimum-width thresholds for the polygon-edge-based multistage minimum area rule.

`specialMinAreaTbl`

The area thresholds for the multistage minimum area rules.

`minEnclosedArea`

The minimum area enclosed by ring-shaped wires or vias.

`minEnclosedWidth`

The minimum width of any dimension of an enclosed area on the layer.

`minEdgeLength`

The threshold for short edges.

`maxNumMinEdge`

The maximum number of consecutive short edges.

`maxTotalMinEdgeLength`

The maximum total length of consecutive edges shorter than `minEdgeLength`.

`minEdgeLength2`

The length threshold for applying the special notch rule. Use with the `minEdgeLength3` attribute.

`minEdgeLength3`

The minimum width of the notch for the special notch rule. Use with `minEdgeLength2`.

`minEdgeLengthTblSize`

The table size of the short edge end-of-line rule.

`minEdgeLengthTbl`

The width threshold and minimum edge length for the short edge end-of-line rule.

`minSpacing`

The minimum spacing between the edges of objects on the layer.

`cornerMinSpacing`

The minimum corner-to-corner spacing between two vias. This value is smaller than `minSpacing`.

`sameNetMinSpacing`

The minimum spacing for two shapes belonging to the same net. This value is smaller than `minSpacing`.

`checkManhattanSpacing`

Specifies whether diagonal or rectilinear distance measurement is used for DRC violation analysis. If this attribute is set to 0, diagonal measurement is used; the actual spacing, measured diagonally, must satisfy the minimum spacing requirement. If it is set to 1, Manhattan (rectilinear) measurement is used; at least the x-direction or y-direction must satisfy the minimum spacing requirement.

`cutTblSize`

The size of the cut table for the general cut spacing rule.

`cutNameTbl`

A list of names assigned to different cut sizes for the general cut spacing rule.

`cutWidthTbl`

The list of cut widths corresponding to the list of names defined by `cutNameTbl`.

`cutHeightTbl`

The list of cut heights corresponding to the list of names defined by `cutNameTbl`.

`cutDataTypeTbl`

The list of data types corresponding to the list of names defined by `cutNameTbl`.

`maxNumAdjacentCut`

The maximum number of adjacent vias.

`adjacentCutRange`

The distance for defining adjacent vias.

`enclosedCutNumNeighbor`

The minimum number of neighboring vias allowed for defining an enclosed via.

`enclosedCutNeighborRange`

The distance range of neighboring vias for defining an enclosed via.

`enclosedCutMinSpacing`

The minimum spacing between two enclosed vias.

`enclosedCutToNeighborMinSpacing`

The minimum spacing between an enclosed via and its neighboring vias.

`onWireTrack`

The routes and vias are placed on wire tracks.

`endOfLine1NeighborThreshold`

The line width threshold that determines the application of the end-of-line to end-of-line and one-neighbor end-of-line spacing rules.

`endOfLine1NeighborEndToEndMinSpacing`

The minimum spacing between the two line-ends in the end-of-line to end-of-line spacing rule.

`endOfLine1NeighborEndToEndParallelWidth`

The parallel run between the two line-ends in the end-of-line to end-of-line spacing rule.

`endOfLine1NeighborMinSpacing`

The minimum spacing between the line-end and the neighboring metal in the one-neighbor end-of-line spacing rule.

`endOfLine1NeighborThreshold`

The line-width threshold for the one-neighbor end-of-line spacing rule.

`endOfLine1NeighborWireMinThreshold`

The opposite-metal width threshold for the one-neighbor end-of-line spacing rule.

`endOfLine1NeighborMinLength`

The narrow-end length threshold for the one-neighbor end-of-line spacing rule.

`endOfLine1NeighborMinSpacing`

The threshold value for the one-neighbor end-of-line spacing rule.

`endOfLine1NeighborCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the one-neighbor end-of-line spacing rule.

`endOfLine2NeighborThreshold`

The line width threshold that determines the application of the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborWireMinThreshold`

The nearby line width threshold that determines the application of the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborMinSpacing`

The minimum spacing between the line-end and the neighboring metal in the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborSideMinSpacing`

The minimum spacing between the side of the line-end and the neighboring metal in the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the two-neighbor end-of-line spacing rule.

`endOfLine2NeighborSideKeepoutLength`

The length of the keepout area along the side of the line-end metal in the two-neighbor end-of-line spacing rule.

`endOfLine3NeighborThreshold`

The line width threshold that determines the application of the three-neighbor end-of-line spacing rule.

`endOfLine3NeighborMinSpacing`

The minimum spacing between the line-end and the neighboring metal in the three-neighbor end-of-line spacing rule.

`endOfLine3NeighborSideMinSpacing`

The minimum spacing between the sides of the line-end and the two side-neighboring metals in the three-neighbor end-of-line spacing rule.

`endOfLine3NeighborCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the three-neighbor end-of-line spacing rule.

`endOfLine3NeighborSideKeepoutLength`

The length of the keepout areas along the sides of the line-end metal in the three-neighbor end-of-line spacing rule.

`stubThreshold`

The maximum parallel length for applying the `stubSpacing` rule instead of the `minSpacing` rule.

`stubLengthThreshold`

The length threshold of wire edges that share the same corners with the end-of-line edge. Setting the `stubSpacing` attribute is required when the end-of-line edge's width is less than or equal to `stubThreshold` and either adjacent edge's length is less than or equal to `stubLengthThreshold`.

`stubSpacing`

The minimum spacing for two objects that run parallel for less than the distance specified by the `stubThreshold` attribute. This value is smaller than `minSpacing`.

`stubToStubSpacing`

The spacing between two end-of-line wire segments.

Note:

The `stubToStubSpacing` rule is not supported by Zroute.

`endOfLineCornerKeepoutWidth`

The distance from the corner of the end-of-line metal segment to the neighboring metal segment, defining a keepout region.

`stubMinLength`

The minimum distance from the end-of-line edge to the opposite internal edge.

`tJunctionStubWireMaxThreshold`

The end-of-line wire width threshold for applying the two-sided end-of-line spacing rule.

`tJunctionOrthoWireMaxThreshold`

The nearby-wire width threshold for applying the two-sided end-of-line spacing rule.

`tJunctionStubKeepoutMinSpacing`

The width of the keepout area, perpendicular to the wire end, for the two-sided end-of-line spacing rule.

`tJunctionStubKeepoutMinWidth`

The length of the keepout area, from wire end to wire end, for the two-sided end-of-line spacing rule.

`uShapeMinSpacing`

The minimum spacing between two edges when at least one edge has a U-shaped notch.

`uShapeDepthThreshold`

The minimum depth threshold of the notch for applying the U-shape minimum spacing rule.

`uShapeMinLength`

The maximum length threshold of the notch for applying the U-shape minimum spacing rule.

`sameNetWidthThreshold`

The width threshold for the thin wire width for the dog-bone rule.

`maxStackLevel`

The maximum number of vias that can stack at the same point.

`fatWireThreshold`

The threshold for using the `fatFatMinSpacing` rule instead of the `minSpacing` rule.

`fatFatMinSpacing`

The minimum distance required between wires on a layer when the widths of both wires are greater than or equal to `fatWireThreshold`.

`fatThinMinSpacing`

The minimum distance required between wires on a layer when the width of one of the wires is greater than or equal to `fatWireThreshold`.

`fatWireExtensionRange`

The extension range required for using the fat wire spacing rules (`fatFatMinSpacing` or `fatThinMinSpacing`) instead of the `minSpacing`, even when the wire in the extension portion is no longer fat. Any metal extending out from the fat wire must be treated as fat within the specified extension range.

`fatTblDimension`

The size of the fat table.

`fatTblThreshold`, `fatTblThreshold2`

The thresholds used to define the different fat degrees of a metal segment. You must specify n values, where n is the table size.

`fatTblParallelLength`

The parallel length thresholds used to adjust the corresponding fat degree of a metal when the metal segment is adjacent to another metal segment. You must specify n values, where n is the table size.

`fatTblParallelLengthDimension`

The number of thresholds M used to determine which rules apply, based on the length that two metal segments are parallel to each other.

`fatTblExtensionRange`

The extension ranges required for a metal segment of the corresponding fat degree.

`fatTblSpacing`

The spacing required for the corresponding fat condition when two metal segments are adjacent. You must specify an n by n table, where n is the table size.

`fatTblMinEnclosedArea`

The minimum enclosed area for the fat wires. You must specify n values, where n is the table size.

`fatTblXDimension`

The size of the fat metal orthogonal X spacing rule table.

`fatTblYDimension`

The size of the fat metal orthogonal Y spacing rule table.

`fatTblXThreshold`

The thresholds used to define the different fat degrees of a metal segment in the x-direction. You must specify n values, where n is the X table size.

`fatTblYThreshold`

The thresholds used to define the different fat degrees of a metal segment in the y-direction. You must specify n values, where n is the Y table size.

`fatTblXParallelLength`

The parallel length thresholds used to adjust the corresponding fat degree of a metal when the metal segment is adjacent to another metal segment, for the X spacing rule. You must specify n values, where n is the X table size.

`fatTblYParallelLength`

The parallel length thresholds used to adjust the corresponding fat degree of a metal when the metal segment is adjacent to another metal segment, for the Y spacing rule. You must specify n values, where n is the Y table size.

`fatTblXMinSpacing`

The minimum X spacing required for the corresponding fat condition when two metal segments are adjacent. You must specify an n by n table, where n is the table size.

`fatTblYMinSpacing`

The minimum Y spacing required for the corresponding fat condition when two metal segments are adjacent. You must specify an n by n table, where n is the table size.

`orthoSpacingExcludeCorner`

Specifies whether to extend the spacing check to the corner area for the given layer for the fat metal orthogonal spacing rule.

`fatContactThreshold`

The threshold for using a fat wire contact instead of the default contact.

Any wire on the specified layer whose width is equal to or greater than this threshold requires a fat wire contact. For more information, see the description of the `isFatContact` attribute in [“ContactCode Section” on page 1-50](#).

`fatTblFatContactNumber`

The contact code numbers for the fat wires in a one-dimensional table rule. You must specify n values, where n is the table size.

`fatTblFatContactMinCuts`

The minimum number of vias in a one-dimensional table rule. You must specify n values, where n is the table size.

`fat2DTblFatContactNumber`

The contact code numbers for the fat wires in a two-dimensional table rule. You must specify an n by n table, where n is the table size.

`fat2DTblFatContactMinCuts`

The minimum number of vias in a two-dimensional table rule. You must specify an n by n table, where n is the table size.

`fatTblDimension2`

The number of upper-metal ranges considered in the area-based fat metal contact rule.

`fatTblThreshold2`

The upper-metal width thresholds considered in the area-based fat metal contact rule.

`fatTblAreaDimension`

The number of area ranges considered in the area-based fat metal contact rule.

`fatTblAreaThreshold`

The area thresholds considered in the area-based fat metal contact rule.

`fatTblFatContactNumber`

The allowed contact numbers for vias used in the area-based fat metal contact rule.

`fatTblFatContactMinCuts`

The minimum number of cuts that can be used in the area-based fat metal contact rule.

`fatTblExtensionDimension`

The number of width ranges considered in the area-based fat metal extension contact rule.

`fatTblExtensionThreshold`

The width thresholds considered in the area-based fat metal extension contact rule.

`fatTblExtensionRangeDimension`

The number of extension ranges considered in the area-based fat metal extension contact rule.

`fatTblExtensionRange`

The extension range thresholds considered in the area-based fat metal extension contact rule.

`fatTblExtensionAreaDimension`

The number of metal area ranges considered in the area-based fat metal extension contact rule.

`fatTblExtensionAreaThreshold`

The area thresholds considered in the area-based fat metal extension contact rule.

`fatTblExtensionContactNumber`

The contact numbers of the allowed contacts for each combination of width, extension, and metal area in the area-based fat metal extension contact rule.

`fatTblExtensionMinCuts`

The number of allowed via cuts for each combination of width, extension, and metal area in the area-based fat metal extension contact rule.

`spanTblDimension`

The table size for the metal span spacing rule.

`spanTblThreshold`

A table of metal span values for the metal span spacing rule.

`spanTblParallelLength`

A table of parallel length values for the metal span spacing rule.

`spanTblMinSpacing`

A table of minimum spacing values corresponding to the combinations of span thresholds and parallel lengths for the metal span spacing rule.

`protrusionTblDim`

The size of the protrusion table.

`protrusionFatThresholdTbl`

The threshold value (minimum widths) for the fat wire. You must specify n values, where n is the table size.

`protrusionLengthLimitTbl`

The length threshold (maximum lengths) for the connected thin wire. You must specify n values, where n is the table size.

`protrusionMinWidthTbl`

The minimum width value for the connected thin wire. You must specify n values, where n is the table size.

LayerDataType Section

The `LayerDataType` section defines a layer data object in the Milkyway database and provides information about a particular layer. The selectability, visibility, and display features of the layer data object can be different from those of the associated layer. The layer data object can be included with, or excluded from, the associated mask layer.

A layer data object designation consists of the layer number, a colon, and a data type number. For example, if layer 20 represents the metal1 layer, then layer data type 20:1 could represent metal1 pin text, 20:2 could represent metal1 DEF obstruction (OBS) geometries, and so on. Different data type numbers can also be used to set different pattern styles for different usage of the same layer.

Data type numbers can range from 0 to 255. The data type number 0 represents the layer itself, so layer data type 20:0 is exactly the same as layer 20. Each routing mask layer, such as metal1 or via1, must be defined as a plain layer, having 0 as the data type number.

The data objects (geometry or text) defined by the `LayerDataType` statement can be included with, or excluded from, the corresponding layer, depending on your requirements. Set the `nonMask` attribute to 1 to indicate a nonmask layer that is to be omitted from mask generation for the associated layer. Set this attribute to 0 to include the layer data type object as part of the mask layer. The default for this attribute is 0.

[Example 1-8](#) shows a typical `LayerDataType` section.

Example 1-8 LayerDataType Section

```
LayerDataType "MET1ZONE" {
    layerNumber = 4
    dataTypeNumber = 3
    nonMask = 1
    visible = 1
    selectable = 1
    blink = 0
    color = "red"
    lineStyle = "solid"
    pattern = "dot"
}
```

In IC Compiler, you can find out the layer data type objects associated with a specific layer and the object characteristics by using commands similar to the following:

```
icc_shell> get_layer_attribute -layer metall data_types
...
icc_shell> get_layer_attribute -layer metall data_type_details
...
```

ContactCode Section

A technology file defines each via master (contact code) by specifying attributes for that via master. A via master can be a default via master (default contact) or a fat via master (fat contact). The attributes fall into several groupings, including

- [Physical Attributes](#)

Physical attributes define the physical attributes of the via master.

- [Parasitic Attributes](#)

Parasitic attributes define the layer parasitics for a via layer.

- [Design Rule Attributes](#)

Design rule attributes define the layer-specific design rules associated with the via layer.

The `ContactCode` section defines the via masters (contact codes) used in designs in the library.

[Example 1-9](#) shows a typical `ContactCode` section for a default via.

Example 1-9 ContactCode Section (DefaultContact)

```
ContactCode "PCON" {
    /* physical attributes */
    contactCodeNumber = 1
    contactSourceType = 0
    cutLayer = "CONT"
    lowerLayer = "M1"
    upperLayer = "M2"
    isDefaultContact = 1
    cutWidth = 0.8
    cutHeight = 0.8
    minNumRows = 1
    minNumCols = 1
}
```

```

    /* parasitic attributes */
    unitMinResistance = 0.00025
    unitNomResistance = 0.00025
    unitMaxResistance = 0.00025
    temperatureCoeff = 2.5e-06
    unitMinCapacitance = 0.0004
    unitNomCapacitance = 0.0004
    unitMaxCapacitance = 0.0004

    /* design rule attributes */
    upperLayerEncWidth = 0.6
    upperLayerEncHeight = 0.6
    lowerLayerEncWidth = 0.6
    lowerLayerEncHeight = 0.6
    minCutSpacing = 1.2
    maxNumRowsNonTurning = 1
}

```

Example 1-10 shows a typical `ContactCode` section for a fat via.

Example 1-10 *ContactCode Section (FatContact)*

```

ContactCode "PCON" {
    /* physical attributes */
    contactCodeNumber = 1
    contactSourceType = 0
    cutLayer = "CONT"
    lowerLayer = "M1"
    upperLayer = "M2"
    isFatContact = 1
    cutWidth = 0.8
    cutHeight = 0.8

    /* parasitic attributes */
    unitMinResistance = 0.00025
    unitNomResistance = 0.00025
    unitMaxResistance = 0.00025
    unitMinCapacitance = 0.0004
    unitNomCapacitance = 0.0004
    unitMaxCapacitance = 0.0004

    /* design rule attributes */
    upperLayerEncWidth = 0.6
    upperLayerEncHeight = 0.6
    lowerLayerEncWidth = 0.6
    lowerLayerEncHeight = 0.6
    minCutSpacing = 1.2
    maxNumRowsNonTurning = 1
}

```

For vias that are to be used during redundant via insertion for yield improvement, you can specify multiple-cut, odd-shaped vias, called rectangle-based vias. In that case, you specify the physical attributes as a list of rectangles for the lower metal layer, for the cut layer, and for the upper metal layer. The Zroute router checks such vias for design rule violations. However, it adds such vias to the design only during redundant via insertion, not ordinary routing. For that reason, you cannot define such a via as the default via (`isDefaultContact=1`).

For example, the following `ContactCode` section defines a three-cut, H-shaped via to be used for redundant via insertion:

```
ContactCode "via3H" {
  contactCodeNumber = 53
  cutLayer          = "VIA3"
  lowerLayer        = "M3"
  upperLayer        = "M4"
  contactSourceType = 2 ; Multiple-cut fixed via
  lowerLayerRectTbl = ("0.0, 0.0, 0.2, 0.6",
                      "0.4, 0.0, 0.6, 0.6",
                      "0.0, 0.2, 0.6, 0.4")
  upperLayerRectTbl = ("0.0, 0.2, 0.6, 0.4")
  cutLayerRectTbl   = ("0.1, 0.3, 0.2, 0.4",
                      "0.3, 0.3, 0.4, 0.4",
                      "0.5, 0.3, 0.6, 0.4")
}
```

Physical Attributes

The physical attributes define the physical characteristics of the via master.

The physical attributes are

`name`

The name of the via master.

`contactCodeNumber`

Identifies the via master. This must be an integer between 1 and 65535. Each `ContactCode` section must have a unique `contactCodeNumber`.

`contactSourceType`

Identifies the contact type. Valid values and their meanings are

- 0 (single-cut fixed via – this is the default contact type)
- 1 (generated via for regular nets)
- 2 (multiple-cut fixed via)

- 3 (single-cut via for nondefault routing rule)
- 4 (multiple-cut via for nondefault routing rule)
- 5 (generated via for special nets, such as power nets)

`cutLayer`

Specifies the name of the display layer used for the via. The value must be the name of one of the layers specified in a `Layer` section.

`lowerLayer`

Specifies the name of the display layer used for one of the conduction layers. The value must be the name of one of the layers specified in a `Layer` section.

`upperLayer`

Specifies the name of the display layer used for the other conduction layer. The value must be the name of one of the layers specified in a `Layer` section.

`isDefaultContact`

Specifies whether the contact is a default contact. Valid values are 0 or 1. A given `cutLayer` can have multiple default contacts defined. In that case, the router uses the lowest-cost default `ContactCode`. If this attribute statement is omitted, the default is 0, meaning not a default contact.

`isFatContact`

Specifies whether the contact is a fat contact. Valid values are 0 or 1.

A fat contact is used when the wire is wider than the `fatContactThreshold` value specified in the associated `Layer` section.

`cutHeight`

Specifies the vertical dimension of the cut.

`cutWidth`

Specifies the horizontal dimension of the cut.

`minNumRows`

Specifies the number of rows in a minimum-size via array.

`minNumCols`

Specifies the number of columns in a minimum-size via array.

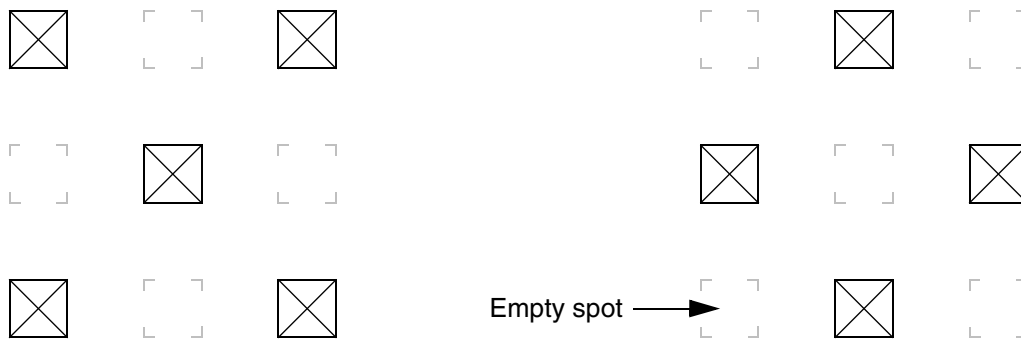
isStaggeredPattern

Specifies whether a multiple-cut via array has a full pattern (0) or staggered pattern (1). A staggered pattern has via cuts in alternating positions in the array, like the black squares of a chess board, as shown in [Figure 1-7](#).

isStaggeredEmptyAtLowerLeft

For a staggered via array, specifies whether the pattern starts with a cut (0) or an empty spot (1) at the lower-left corner of the array. [Figure 1-7](#) shows two 3-by-3 staggered via arrays, one with a via cut in the lower-left corner and the other with an empty spot in the lower-left corner.

Figure 1-7 Staggered Via Arrays



`isStaggeredEmptyAtLowerLeft = 0`

`isStaggeredEmptyAtLowerLeft = 1`

Parasitic Attributes

The parasitic attributes define the layer parasitics. In general, IC Compiler gets the parasitic information from the TLUPlus files, rather than the technology file.

This section describes the following parasitic attributes:

- [Resistance](#)
- [Capacitance](#)
- [Maximum Current Density](#)

Resistance

Resistance is defined as the resistance per contact. The resistance attributes are

`unitMinResistance`

A floating-point number representing the minimum resistance.

`unitNomResistance`

A floating-point number representing the nominal resistance.

`unitMaxResistance`

A floating-point number representing the maximum resistance.

Capacitance

Capacitance is defined per contact in a cell instance or over a macro. The capacitance attributes are

`unitMinCapacitance`

A floating-point number representing the minimum capacitance.

`unitNomCapacitance`

A floating-point number representing the nominal capacitance.

`unitMaxCapacitance`

A floating-point number representing the maximum capacitance.

Maximum Current Density

Use the `maxCurrDensity` attribute to define the maximum current density in amps per square centimeter of contact or via area.

Design Rule Attributes

The design rule attributes in a `ContactCode` section define design rules specific to a via layer. All measurements in this section are in the user units specified in the `Technology` section of the technology file. See [“Technology Section” on page 1-5](#).

Note:

Additional via design rules are defined in the `Layer` and `DesignRule` sections. For more information, see [“Design Rule Attributes” on page 1-38](#) and [“DesignRule Section” on page 1-57](#).

The design rule attributes for the `ContactCode` section are

`upperLayerEncHeight`

The distance at which the first of two layers encloses the via in the vertical dimension.

`upperLayerEncWidth`

The distance at which the first of two layers encloses the via in the horizontal dimension.

`lowerLayerEncHeight`

The distance at which the second of two layers encloses the via in the vertical dimension.

`lowerLayerEncWidth`

The distance at which the second of two layers encloses the via in the horizontal dimension.

`minCutSpacing`

The minimum separation distance between the edges of the via.

`viaFarmSpacing`

The minimum separation distance between two via farms.

`maxNumRows`

When `viaFarmSpacing` is defined in the same `ContactCode` section, the `maxNumRows` value represents the exact number of rows in each via farm in a via farm array; otherwise, the `maxNumRows` value represents the maximum number of rows in a via array.

`excludedForSignalRoute`

When `excludedForSignalRoute` is present and is set to 1, it prevents the router from using the via for signal routing. Existing instances of the via, such as those used in standard cells or macros, are allowed to remain and are still checked for design rule violations, but the router does not use the via in new signal routes.

`maxNumRowsNonTurning`

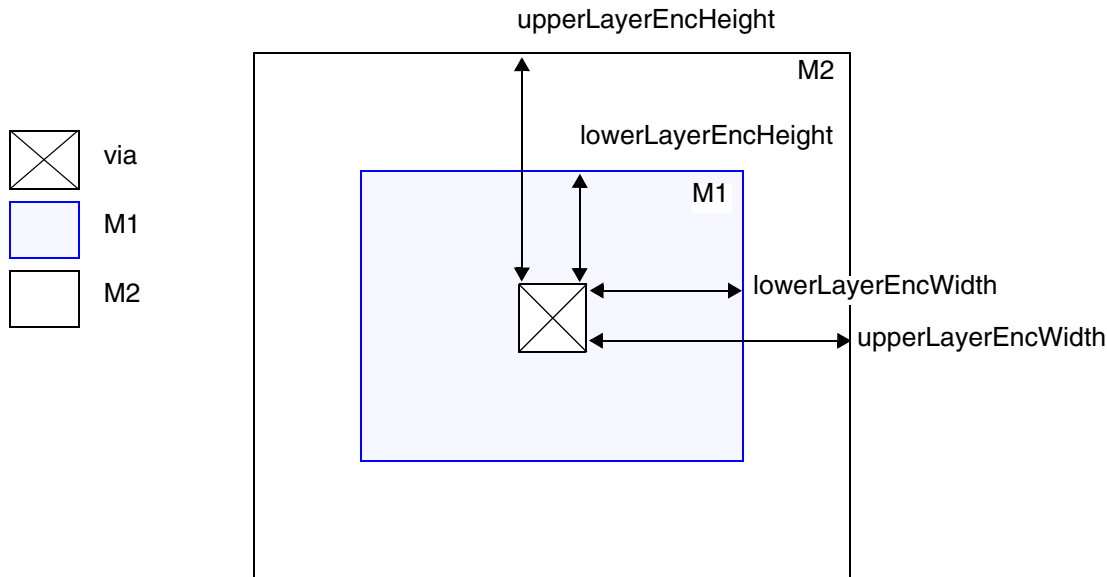
Restricts a via size by placing an upper limit on the number of rows of cuts allowed in the routing direction.

Note:

When the prerouter needs to switch layers without changing the routing direction, it uses a square for a drop via. The width and height of the square is the fixed width of the wires.

Figure 1-8 shows the interpretation of the `upperLayerEncHeight`, `lowerLayerEncHeight`, `upperLayerEncWidth`, and `lowerLayerEncWidth` attributes when the lower layer is M1 and the upper layer is M2.

Figure 1-8 Via Enclosure Attributes



DesignRule Section

The `DesignRule` section of the technology file defines the interlayer design rules that apply to designs in the library. All measurements in this section are in the user units specified in the `Technology` section of the technology file. (See [“Technology Section” on page 1-5](#).) For more information about these design rules, see [Chapter 2, “Routing Design Rules.”](#)

Note:

Layer-specific design rules are defined in a `Layer` section. For more information, see [“Design Rule Attributes” on page 1-38](#).

Example 1-11 shows attributes that can be used in the `DesignRule` section.

Example 1-11 DesignRule Section

```
DesignRule {
    layer1 = "name"
    layer2 = "name"
    minSpacing = 1.0
    diffNetMinSpacing = 1.0
    cornerMinSpacing = 0
    minEnclosure = 0
    cut1TblSize = 0
}
```

```

cut2TblSize = 0
cut1NameTbl = (name1, name2)
cut2NameTbl = (name1, name2)
orthoSpacingExcludeCornerTbl = (0, 0, 0, 0)
sameNetXMinSpacingTbl = (0, 0, 0, 0)
sameNetYMinSpacingTbl = (0, 0, 0, 0)
sameNetCornerMinSpacingTbl = (0, 0, 0, 0)
sameNetCenterMinSpacingTbl = (0, 0, 0, 0)
diffNetXMinSpacingTbl = (0, 0, 0, 0)
diffNetYMinSpacingTbl = (0, 0, 0, 0)
diffNetCornerMinSpacingTbl = (0, 0, 0, 0)
diffNetCenterMinSpacingTbl = (0, 0, 0, 0)
sameSegXMinSpacingTbl = (0, 0, 0, 0)
sameSegYMinSpacingTbl = (0, 0, 0, 0)
sameSegCornerMinSpacingTbl = (0, 0, 0, 0)
sameSegCenterMinSpacingTbl = (0, 0, 0, 0)
diffSegXMinSpacingTbl = (0, 0, 0, 0)
diffSegYMinSpacingTbl = (0, 0, 0, 0)
diffSegCornerMinSpacingTbl = (0, 0, 0, 0)
diffSegCenterMinSpacingTbl = (0, 0, 0, 0)
diffNetKeepoutMinWidth = 0
diffNetKeepoutMinLength = 0
diffNetKeepoutMinRadius = 0
endOfLineEnclosure = 0
endOfLineViaJogLength = 0
endOfLineViaJogWidth = 0
endOfLineViaEncWidth = 0
stackable = 0
endOfLineEncTblSize = 0
endOfLineEncTbl = (0, 0, 0, 0, 0, 0, 0, 0, 0)
endOfLineEncSideThreshold = (0, 0, 0, 0, 0, 0, 0, 0, 0)
endOfLineEncSpacingTbl = (0, 0, 0, 0, 0, 0, 0, 0, 0)
endOfLineTShapeEncTblSize = 0
endOfLineTShapeCornerMinSpacing = (0, 0, 0, 0, 0, 0, 0, 0, 0)
endOfLineTShapeEncSideThreshold = (0, 0, 0, 0, 0, 0, 0, 0, 0)
endOfLineTShapeEncTbl = (0, 0, 0, 0, 0, 0, 0, 0, 0)
endOfLineTShapeEncSpacingTbl = (0, 0, 0, 0, 0, 0, 0, 0, 0)
endOfLineEnc2NeighborTblSize = 0
endOfLineEnc2NeighborThreshold = 0
endOfLineEnc2NeighborCornerKeepoutWidth = 0
endOfLineEnc2NeighborSideKeepoutLength = 0
endOfLineEnc2NeighborSideMinSpacing = 0
endOfLineEnc2NeighborMinEnclosure = 0
endOfLineEnc2NeighborTbl = (0, 0, 0)
endOfLineEnc2NeighborSpacingTbl = (0, 0, 0)
endOfLineEnc2NeighborViaArrayExcludedTbl = (0, 0, 0)
endOfLineEnc2NeighborWireMinThreshold = 0
fatWireViaEncTblSize = 0
fatWireViaEncWidthThresholdTbl = (0, 0, 0, 0)
fatWireViaEncParallelLengthThresholdTbl = (0, 0, 0, 0)
fatWireViaEncMaxSpacingThresholdTbl = (0, 0, 0, 0)
fatWireViaEnclosureTbl = (0, 0, 0, 0)
fatWireViaArrayExcludedTbl = (0, 0, 0, 0)

```

```

    concaveMetalToCutMinDist = 0
    jogWireViaKeepoutTblSize = 0
    jogWireViaKeepoutEncThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
    jogWireViaKeepoutMinSize = (0, 0, 0, 0, 0, 0, 0, 0)
    fatWireViaKeepoutTblSize = 0
    fatWireViaKeepoutWidthThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
    fatWireViaKeepoutParallelLengthThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
    fatWireViaKeepoutMaxSpacingThreshold = (0, 0, 0, 0, 0, 0, 0, 0)
    fatWireViaKeepoutMinSize = (0, 0, 0, 0, 0, 0, 0, 0)
    fatWireViaKeepoutEnclosure = (0, 0, 0, 0, 0, 0, 0, 0)
}

```

These are the attribute definitions:

`layer1`

One of the layers to which the rule applies. The value must be the name of one of the layers specified in a `Layer` section.

`layer2`

The other layer to which the rule applies. The value must be the name of one of the layers specified in a `Layer` section.

`minSpacing`

The minimum spacing between the specified layers.

`diffNetMinSpacing`

The minimum spacing between cuts in different nets for the specified layers.

`cornerMinSpacing`

The minimum corner-to-corner spacing between two vias on different via layers. This value is smaller than the `minSpacing` value.

`minEnclosure`

The minimum distance at which a layer must enclose another layer when the two layers overlap.

`cut1TblSize`

The size of the cut 1 name table used in the general cut spacing rule.

`cut2TblSize`

The size of the cut 2 name table used in the general cut spacing rule.

`cut1NameTbl`

A list of cut names previously defined by the `cutNameTbl` attribute, used in the general cut spacing rule, first cut layer.

`cut2NameTbl`

A list of cut names previously defined by the `cutNameTbl` attribute, used in the general cut spacing rule, second cut layer.

`orthoSpacingExcludeCornerTbl`

Specifies whether the general cut spacing rule is extended beyond the corners of the cut.

`sameNetXMinSpacingTbl`

The minimum X spacing between cuts in the same net for the general cut spacing rule.

`sameNetYMinSpacingTbl`

The minimum Y spacing between cuts in the same net for the general cut spacing rule.

`sameNetCornerMinSpacingTbl`

The minimum corner-to-corner spacing between cuts in the same net for the general cut spacing rule.

`sameNetCenterMinSpacingTbl`

The minimum center-to-center spacing between cuts in the same net for the general cut spacing rule.

`diffNetXMinSpacingTbl`

The minimum X spacing between cuts in different nets for the general cut spacing rule.

`diffNetYMinSpacingTbl`

The minimum Y spacing between cuts in different nets for the general cut spacing rule.

`diffNetCornerMinSpacingTbl`

The minimum corner-to-corner spacing between cuts in different nets for the general cut spacing rule.

`diffNetCenterMinSpacingTbl`

The minimum center-to-center spacing between cuts in different nets for the general cut spacing rule.

`sameSegXMinSpacingTbl`

The minimum X spacing between cuts in the same wire segment for the general cut spacing rule.

`sameSegYMinSpacingTbl`

The minimum Y spacing between cuts in the same wire segment for the general cut spacing rule.

`sameSegCornerMinSpacingTbl`

The minimum corner-to-corner spacing between cuts in the same wire segment for the general cut spacing rule.

`sameSegCenterMinSpacingTbl`

The minimum center-to-center spacing between cuts in the same wire segment for the general cut spacing rule.

`diffSegXMinSpacingTbl`

The minimum X spacing between cuts in different wire segments for the general cut spacing rule.

`diffSegYMinSpacingTbl`

The minimum Y spacing between cuts in different wire segments for the general cut spacing rule.

`diffSegCornerMinSpacingTbl`

The minimum corner-to-corner spacing between cuts in different wire segments for the general cut spacing rule.

`diffSegCenterMinSpacingTbl`

The minimum center-to-center spacing between cuts in different wire segments for the general cut spacing rule.

`diffNetKeepoutMinWidth`

The minimum shorter spacing between a via and an unconnected metal shape for the asymmetric via-to-metal spacing rule.

`diffNetKeepoutMinLength`

The minimum longer spacing between a via and an unconnected metal shape for the asymmetric via-to-metal spacing rule.

`diffNetKeepoutMinRadius`

The radius of curvature for the asymmetric via-to-metal spacing rule.

`endOfLineEnclosure`

An enclosure size to specify the end-of-line rule for routing wire segments.

`endOfLineViaJogLength`

The length from the jog wire edge to the end-of-line edge, which is also the minimum spacing between the end-of-line edge and a single via.

`endOfLineViaJogWidth`

The maximum width of the end-of-line edge.

`endOfLineViaEncWidth`

The maximum width of the single-via enclosure.

`stackable`

Specifies whether via cuts on layer1 and layer2 can be stacked: 0 for not stackable or 1 for stackable. The default is 0, not stackable.

To enforce the not-stackable rule, whether by default or by explicitly setting the `stackable` attribute to 0, you must also specify a nonzero minimum spacing between vias in layer1 and layer2. You can define this rule by either of the following methods:

- Specify a nonzero `minSpacing` value in the same `DesignRule` section
- Specify a general cut spacing rule between the two layers using these attributes:

```
sameNetXMinSpacingTbl
sameNetYMinSpacingTbl
sameNetCornerMinSpacingTbl
```

Any nonzero spacing value triggers enforcement of the not-stackable rule, including a very small value. This value can be the minimum grid resolution of the technology, for example, 0.001. If no minimum spacing rule exists, the router can overlap via cuts in the two layers by any amount, irrespective of the `stackable` attribute setting.

When the `stackable` attribute is set to 1, the router has the flexibility to either stack the two vias with their centers exactly aligned or enforce any minimum spacing rule that has been defined for the layers. By default, the router can violate a minimum spacing rule only by exactly aligning the via centers. You can specify an allowable amount of center-to-center misalignment by setting the `stackViaCenterSpacingThreshold` attribute.

`endOfLineEncTblSize`

The size of the end-of-line enclosure table.

`endOfLineEncTbl`

The minimum metal enclosure required at the end-of-line edge. You must specify *n* values, where *n* is the table size.

`endOfLineEncSideThreshold`

The metal enclosure thresholds at the side edges of an end-of-line wire that encloses a via near the end-of-line edge. You must specify *n* values, where *n* is the table size.

`endOfLineEncSpacingTbl`

The minimum end-of-line spacing for wire ends with a via dropped to a lower via layer when the width of the wire is less than `stubThreshold`. You must specify *n* values, where *n* is the table size.

`endOfLineTShapeEncTblSize`

The size of the T-shape end-of-line enclosure table.

`endOfLineTShapeCornerMinSpacing`

The maximum spacing threshold between the corner of the T-shape wire and the corner of the enclosed via for applying the T-shape wire via enclosure rule. If the spacing is greater than this value, the rule is waived.

`endOfLineTShapeEncSideThreshold`

The width threshold between the edge of the via and the edge of the enclosing T-shape wire end. You must specify *n* values, where *n* is the table size.

`endOfLineTShapeEncTbl`

The minimum extension between a single via's edge and the edges of the enclosing T-shape wire. You must specify *n* values, where *n* is the table size.

`endOfLineTShapeEncSpacingTbl`

The minimum end-of-line spacing for T-shape wire ends with a via dropped to a lower via layer based on the wire width ranges specified in the `endOfLineTShapeEncSideThreshold` attribute. You must specify *n* values, where *n* is the table size.

`fatWireViaEncTblSize`

The table size for the fat via enclosure rule.

`fatWireViaEncWidthThresholdTbl`

The width thresholds for the fat via enclosure rule.

`fatWireViaEncParallelLengthThresholdTbl`

The parallel length thresholds for the fat via enclosure rule.

`fatWireViaEncMaxSpacingThresholdTbl`

The maximum spacing thresholds for the fat via enclosure rule.

`fatWireViaEnclosureTbl`

The minimum enclosure values for the fat via enclosure rule.

`fatWireViaArrayExcludedTbl`

The via array size exclusion thresholds for the fat via enclosure rule.

`concaveMetalToCutMinDist`

The minimum spacing value for the concave metal corner via enclosure rule.

`jogWireViaKeepoutTblSize`

The size of the jog wire via keepout table.

`endOfLineEnc2NeighborTblSize`

The table size for the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborThreshold`

The line-width threshold for applying the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborCornerKeepoutWidth`

The extension of the keepout area beyond the line-end corners in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborSideKeepoutLength`

The length of the keepout area along the side of the line-end metal in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborSideMinSpacing`

The minimum spacing between the side of the line-end and the neighboring metal in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborMinEnclosure`

The minimum overlap of a line-end over the via in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborTbl`

A table of minimum line-end enclosure values in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborSpacingTbl`

A table of line-end to neighboring-metal spacing values in the two-neighbor end-of-line via enclosure rule.

`endOfLineEnc2NeighborViaArrayExcludedTbl`

A table of values, either 0 or 1, used to include or exclude the application of the rule to via arrays.

`endOfLineEnc2NeighborWireMinThreshold`

The minimum edge length for the two-neighbor end-of-line via enclosure rule to be applied.

`jogWireViaKeepoutEncThreshold`

The maximum width thresholds for applying the jog wire via keepout rule. You must specify n values, where n is the table size.

`jogWireViaKeepoutMinSize`

The minimum spacing between the edge of the via and the corner edges of a jog wire.

`fatWireViaKeepoutTblSize`

The size of the fat wire via keepout table.

`fatWireViaKeepoutWidthThreshold`

The minimum width thresholds of the enclosing metal segment. You must specify n values, where n is the table size.

`fatWireViaKeepoutParallelLengthThreshold`

The minimum parallel length thresholds between the enclosing metal segment and the neighboring metal segment.

`fatWireViaKeepoutMaxSpacingThreshold`

The maximum spacing thresholds between the enclosing metal segment and the neighboring metal segment.

`fatWireViaKeepoutMinSize`

The minimum required length of the keepout area in each direction from the corner. You must specify n values, where n is the table size.

`fatWireViaKeepoutEnclosure`

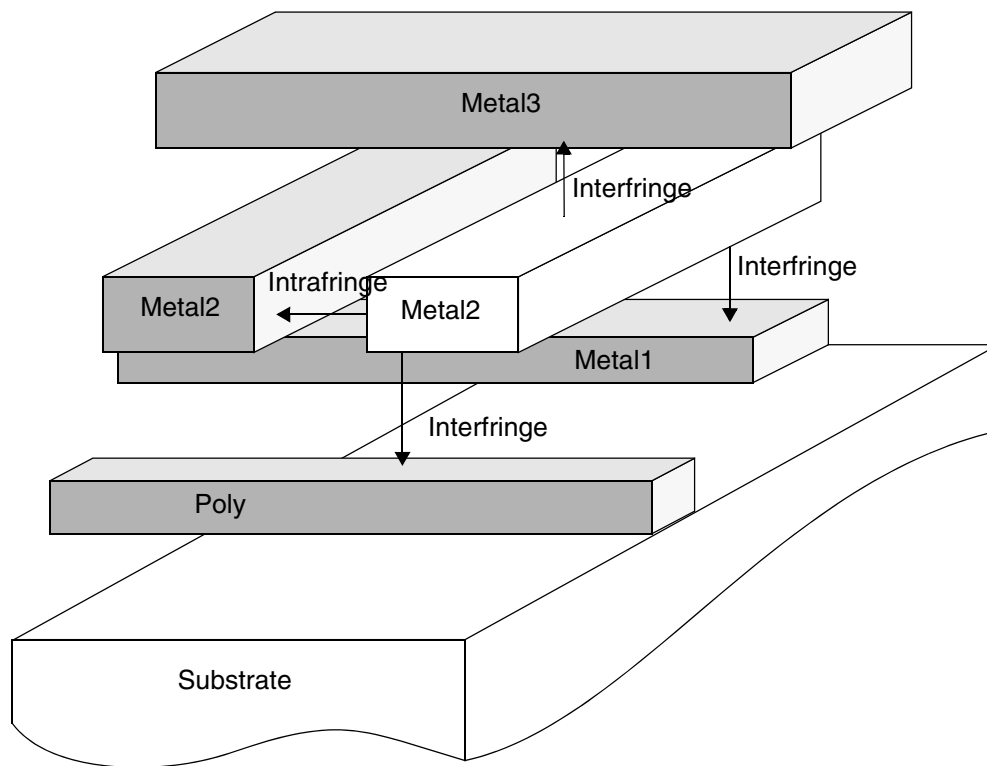
The minimum required width of the metal enclosure. You must specify n values, where n is the table size.

FringeCap Section

A `FringeCap` section specifies capacitance information for interconnect layers when they are overlapping or parallel to each other. This information includes the following:

- Capacitance per unit area when objects on different layers overlap (interfringe in [Figure 1-9](#))
- Capacitance per unit length when objects on the same layer are separated by the minimum spacing specified in the `Layer` section (intrafringe in [Figure 1-9](#))

Figure 1-9 Interfringe and Intrafringe Examples



Default values for coupling capacitance between adjoining parallel objects on the same layer or adjoining objects on different layers are defined in the respective `Layer` sections of the technology file. Other coupling capacitance values are entered in the `FringeCap` section. This information is used for calculating coupling capacitance for timing-driven layout.

Note:

In general, IC Compiler gets the parasitic information from the TLUPlus files, rather than the technology file.

The attributes in a `FringeCap` section are

`number`

The coupling capacitance number, which must be an integer less than the number of metal layers.

`layer1`

One of the two metal layers on which different metal layers cross each other.

`layer2`

The other metal layer on which different metal layers cross each other.

`minFringeCap`

The minimum capacitance for the layer.

`nomFringeCap`

The nominal capacitance for the layer.

`maxFringeCap`

The maximum capacitance for the layer.

[Example 1-12](#) shows a typical interfringe `FringeCap` section.

Example 1-12 *FringeCap Section (Interfringe)*

```
FringeCap 4 {
    number = 4
    layer1 = "M2"
    layer2 = "M3"
    minFringeCap = 0.00022
    nomFringeCap = 0.00022
    maxFringeCap = 0.00022
}
```

[Example 1-13](#) shows a typical intrafringe `FringeCap` section.

Example 1-13 *FringeCap Section (Intrafringe)*

```
FringeCap 4 {
    number = 4
    layer1 = "M2"
    layer2 = "M2"
    minFringeCap = 0.00017
    nomFringeCap = 0.00017
    maxFringeCap = 0.00017
}
```

CapModel and CapTable Sections

The `CapModel` and `CapTable` sections of a technology file let you specify the material type (conductor or dielectric), thickness, and dielectric value for each layer, as well as the wire width and spacing for each layer.

Note:

In general, IC Compiler gets the parasitic information from the TLUPlus files, rather than the technology file.

You can create these two sections manually with a text editor, but you normally create them by using the Milkyway Environment `cmCreateCapModel` command. For more information about this command, see the *Library Data Preparation for IC Compiler User Guide*.

[Example 1-14](#) is an example of a `CapModel` section.

Example 1-14 CapModel Section

```
CapModel "polyConfig1" {
    refLayer = "poly"
    groundPlaneBelow = ""
    groundPlaneAbove = "metal1"
    bottomCapType = "Table"
    bottomCapDataMin = "poly_C_BOTTOM_GP"
    bottomCapDataNom = "poly_C_BOTTOM_GP"
    bottomCapDataMax = "poly_C_BOTTOM_GP"
    topCapType = "Table"
    topCapDataMin = "poly_C_TOP_GP"
    topCapDataNom = "poly_C_TOP_GP"
    topCapDataMax = "poly_C_TOP_GP"
    lateralCapType = "Table"
    lateralCapDataMin = "poly_C_LATERAL"
    lateralCapDataNom = "poly_C_LATERAL"
    lateralCapDataMax = "poly_C_LATERAL"
}
```

[Example 1-15](#) is an example of a `CapTable` section.

Example 1-15 CapTable Section

```
CapTable "metal3_C_BOTTOM_GP_21" {
    wireWidthSize = 3
    wireSpacingSize = 5
    wireWidth = (0.6, 1.2, 1.8)
    wireSpacing = (0.5, 1, 1.5, 2, 2.5)
    capValue = (
        2.4798e-05, 3.05934e-05, 3.5548e-05, 3.99643e-05,
        4.39498e-05, 3.1404e-05, 3.74482e-05, 4.2877e-05,
        4.77828e-05, 5.22164e-05, 3.87834e-05, 4.50642e-05,
        5.07664e-05, 5.59294e-05, 6.05912e-05
    )
}
```

[Example 1-16](#) is an example of a table lookup capacitance model.

Example 1-16 Table Lookup Capacitance Model

```
Technology {
    name = ""
    dielectric = 0
}

CapTable "poly_C_TOP_GP" {
    wireWidthSize = 2
    wireSpacingSize = 3
    wireWidth = (0.4, 0.8)
    wireSpacing = (0.5, 1, 1.5)
    capValue = (4.68407e-05, 5.76469e-05,
                6.10734e-05, 6.97226e-05,
                8.06076e-05, 8.40257e-05)
}
...
```

ResModel Section

The `ResModel` section in the technology file allows the resistance and temperature coefficient of the layer to be expressed as a function of wire width.

Note:

In general, IC Compiler gets the parasitic information from the TLUPlus files, rather than the technology file.

When the `ResModel` section for a given layer is present, it overrides the values for the following attributes in the corresponding `Layer` section: `unitMinResistance`, `unitNomResistance`, and `unitMaxResistance`.

[Example 1-17](#) shows a `ResModel` section.

Example 1-17 ResModel Section

```
ResModel "metallResModel" {
    layerNumber = 1
    size = 3
    wireWidth = (0.48, 0.6, 0.72)
    tempCoeff = (0.01, 0.01, 0.01)
    minRes = (0.0003, 0.0002, 0.0001)
    nomRes = (0.0003, 0.0002, 0.0001)
    maxRes = (0.0003, 0.0002, 0.0001)
}
```

PRRule Section

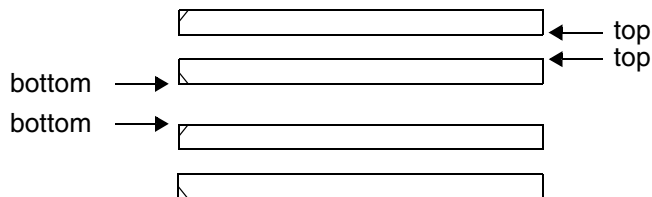
The `PRRule` section of a technology file defines cell row spacing. The cell row spacing rules differ, depending on whether you are using double-back cell rows.

Cell Row Spacing Rules for Double-Back Cell Rows

When using double-back cell rows in your floorplan, as shown in [Figure 1-10](#), you specify the following row spacing rules:

- Between top edge and top edge
- Between bottom edge and bottom edge

Figure 1-10 Row Spacing Rule for Double-Back Cells



The attributes used to define these rules are

`rowSpacingTopTop`

The spacing between the top edges when you are using double-back cell rows in your floorplan.

`rowSpacingBotBot`

The spacing between the bottom edges when you are using double-back cell rows in your floorplan.

`abutableTopTop`

Whether or not the two top edges can be abutted when you are using double-back cell rows in your floorplan. Valid values are 1 or 0.

`abutableBotBot`

Whether or not the two bottom edges can be abutted when you are using double-back cell rows in your floorplan. Valid values are 1 or 0.

[Example 1-18](#) shows a typical `PRRule` section for double-back cell rows.

Example 1-18 *PRRule Section for Double-Back Cell Rows*

```

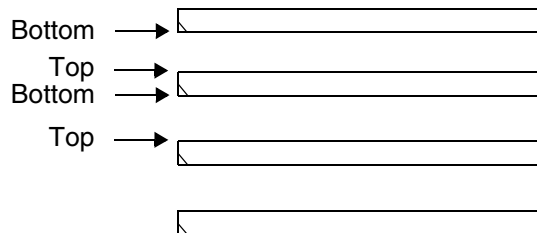
PRRule {
    rowSpacingTopTop = 2.8
    rowSpacingBotBot = 1.4
    abutableTopTop = 1
    abutableBotBot = 1
}

```

Cell Row Spacing Rules for Non-Double-Back Cell Rows

When not using double-back cell rows in your floorplan, you specify a row spacing rule between the top edge and the bottom edge, as shown in [Figure 1-11](#).

Figure 1-11 *Row Spacing Rule for Non-Double-Back Cells*



The attributes used to define these rules are

`rowSpacingTopBot`

The spacing between the top edge and the bottom edge when you are not using double-back cell rows in your floorplan.

`abutableTopBot`

Whether or not the top edge and the bottom edge can be abutted when you are not using double-back cell rows in your floorplan. Valid values are 1 or 0.

[Example 1-19](#) shows a `PRRule` section for non-double-back cell rows.

Example 1-19 *PRRule Section for Non-Double-Back Cell Rows*

```

PRRule {
    rowSpacingTopBot = 2.8
    abutableTopBot = 0
}

```

DensityRule Section

A `DensityRule` section defines the metal density rules used by the `signoff_metal_fill`, `report_metal_density`, and `insert_metal_filler` commands. The technology file should contain a `DensityRule` section for each metal layer.

Use the following attributes to define the metal density rules:

`layer`

The metal layer.

`windowSize`

The size of the window for the density check. By default, the window step size is half of the defined `windowSize`.

The check setup is assumed to be half of the window size.

`minDensity`

The minimum percentage of metal allowed in the window.

`maxDensity`

The maximum percentage of metal allowed in the window.

`maxGradientDensity`

The maximum percentage difference between the fill density of adjacent windows.

For more information about the metal density rules, see [“Metal Density Rules” on page 2-153](#).

[Example 1-20](#) shows an example of a `DensityRule` section.

Example 1-20 DensityRule Section

```
DensityRule {  
    layer = "M1"  
    windowSize = 200  
    minDensity = 20  
    maxDensity = 80  
}
```

SlotRule Section

A `SlotRule` section defines the slotting rules used by the `slot_wire` command for a specific metal layer. The technology file should contain a `SlotRule` section for each metal layer.

Use the following attributes to define the slotting rules:

`layerNumber`

The layer number.

`lengthThreshold`

The length threshold of metal wires requiring slotting.

`widthThreshold`

The width threshold of metal wires requiring slotting.

`minSlotWidth`

The minimum width of the slot.

`maxSlotWidth`

The maximum width of the slot.

`minSlotLength`

The minimum length of the slot.

`maxSlotLength`

The maximum length of the slot.

`minSideSpace`

The minimum space between adjacent slots in a direction perpendicular to the wire (current flow) direction.

`maxSideSpace`

The maximum space between adjacent slots in a direction perpendicular to the wire (current flow) direction.

`minSideClearance`

The minimum space from the side edge of a wire to its outermost slot.

`maxSideClearance`

The maximum space from the side edge of a wire to its outermost slot.

`minEndSpace`

The minimum space between adjacent slots in the wire (current flow) direction.

`maxEndSpace`

The maximum space between adjacent slots in the wire (current flow) direction.

`minEndClearance`

The minimum space from the end edge of a wire to its outermost slot.

`maxEndClearance`

The maximum space from the end edge of a wire to its outermost slot.

`maxMetalDensity`

The maximum metal density allowed for a slotted wire.

[Example 1-21](#) shows an example of a `SlotRule` section.

Example 1-21 *SlotRule Section*

```
SlotRule "" {  
    layerNumber = 1  
    lengthThreshold = 0  
    widthThreshold = 0  
    minSlotWidth = 0  
    maxSlotWidth = 0  
    minSlotLength = 0  
    maxSlotLength = 0  
    minSideSpace = 0  
    maxSideSpace = 0  
    minSideClearance = 0  
    maxSideClearance = 0  
    minEndSpace = 0  
    maxEndSpace = 0  
    minEndClearance = 0  
    maxEndClearance = 0  
    maxMetalDensity = 0  
}
```

2

Routing Design Rules

IC Compiler supports routing design rules for advanced technologies extending to 90 nm, 65 nm, 45 nm, and 32 nm. All the design rules are supported by both the detail route operation and the search-and-repair operation, unless otherwise noted.

For Zroute, all of the routing design rules must be defined in the technology file.

For the classic router, most of the routing design rules are defined in the technology file. Some of the rules are defined with variables or detail route options or have additional controls that are defined with variables or detail route options that you enter in the command window during an IC Compiler session.

This chapter contains the following sections:

- [Minimum Area Rules](#)
- [Minimum Enclosed Area Rule](#)
- [Metal Width Rules](#)
- [Minimum Edge Rules](#)
- [Minimum Spacing Rules](#)
- [Fat Metal Spacing Rules](#)
- [Metal Span Spacing Rule](#)
- [End-of-Line Spacing Rules](#)

- [Bridge Rules](#)
- [Fat Metal Contact Rules](#)
- [Via Spacing Rules](#)
- [Via Enclosure Rules](#)
- [Via Placement Rules](#)
- [Metal and Via on Wire Track Rules](#)
- [Wire Shape Rules](#)
- [Metal Density Rules](#)
- [Parallel Length-Based Floating Wire Antenna Rule](#)

Minimum Area Rules

The minimum area rules are described in the following sections:

- [Minimum Length Rule](#)
- [General Minimum Area Rule](#)
- [Two-Stage Special Minimum Area Rule](#)
- [Polygon-Edge-Based Multistage Minimum Area Rule](#)
- [Dense Wire Minimum Dimensions Rule](#)

Minimum Length Rule

The minimum length rule specifies the minimum length of a polygon. For a single rectangle, the longest dimension must meet this rule. For a polygon composed of two or more rectangles, the length of the bounding box that encloses the entire geometry must meet this rule.

To define the minimum length rule, use the `minLength` attribute in a metal `Layer` section of the technology file. For example,

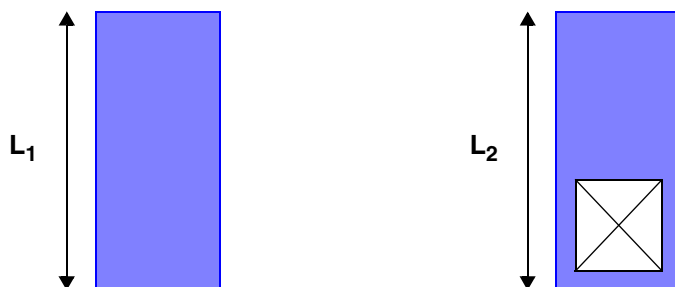
```
Layer "M1" {  
    minLength = 0.35  
}
```

A minimum length violation can be fixed by adding metal stubs or by rerouting the wire to meet the `minLength` value.

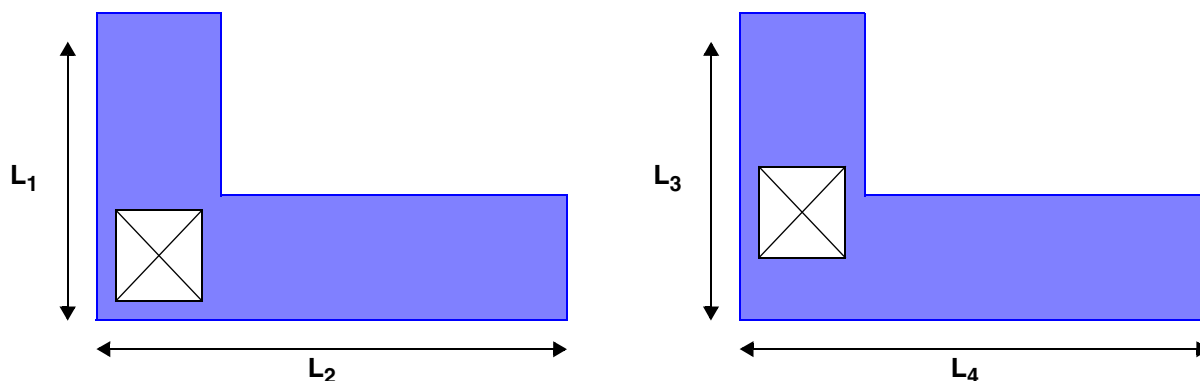
By default, the `minLengthMode` attribute is set to 0, and the minimum length rule is enforced for all shapes, irrespective of vias. You can change this behavior by setting the `minLengthMode` attribute to 1, which causes the check to be performed only on segments that fully enclose vias:

```
Technology {  
    minLengthMode = 1  
}
```

In [Figure 2-1](#), by default, both L_1 and L_2 must be at least the minimum length, L_{\min} , which is specified by the `minLength` attribute. However, if the `minLengthMode` attribute is set to 1, L_1 is not checked because it does not enclose a via.

Figure 2-1 Minimum Length Rule for Enclosed Vias

With the `minLengthMode` attribute set to 1, if a via is fully enclosed by more than one wire segment in a given layer, the minimum length check is performed only on the longest segment. For example, in [Figure 2-2](#), L_2 is longer than L_1 , so the minimum length check is performed only on L_2 and not on L_1 . However, L_3 fully encloses the via and L_4 does not, so the minimum length check is performed on L_3 and not on L_4 .

Figure 2-2 Minimum Length Rule for Multiple Wire Segments Enclosing a Via

General Minimum Area Rule

The `minArea` attribute defines the minimum area for all geometries. Specify this attribute in a `metal Layer` section of the technology file. For example,

```
Layer "M1" {
    minArea = 0.2
}
```

This general rule works with both Zroute and the classic router.

The rectangle-based multistage minimum area rule defines a set of minimum areas for polygons. The minimum area depends on whether the polygon is a rectangle, and if so, its dimensions. A rectangle that can completely cover another rectangle of specified dimensions has a smaller minimum area requirement. The length and width thresholds and the corresponding minimum areas are specified in a table. This rule is supported by Zroute only, not the classic router.

For a three-stage minimum area rule, you define a table size of two and specify two length thresholds, L1 and L2, and two width thresholds, W1 and W1. There are three minimum area values: A0, A1, and A2. A0 is the largest value, A1 is the medium value, and A2 is the smallest value. The rule defines the minimum area in three stages:

- Stage One: Any metal geometry must have an area of at least A0, the largest of the three area values, unless the stage two or stage three conditions apply.
- Stage Two: Any rectangle that can completely cover another rectangle measuring L1 by W1 must have an area of at least A1, the medium area value, unless the stage three condition applies.
- Stage Three: Any rectangle that can completely cover another rectangle measuring L2 by W2 must have an area of at least A2, the smallest area value.

This is the syntax for the rule:

```
Layer "MetalX" {
  minArea                = A0
  specialMinAreaTblSize  = 2
  minAreaRectMinLengthTbl = (L1, L2)
  minAreaRectMinWidthTbl  = (W1, W2)
  specialMinAreaTbl       = (A1, A2)
}
```

For example,

```
Layer "MetalX" {
  minArea                = 0.026
  specialMinAreaTblSize  = 2
  minAreaRectMinLengthTbl = (0, 0.024)
  minAreaRectMinWidthTbl  = (0, 0.076)
  specialMinAreaTbl       = (0.020, 0.018)
}
```

In this example, all geometries must have an area of at least 0.026 unless one of the other conditions applies. Any rectangle must have an area of at least 0.020, unless the last condition applies. (The area L1 by W1 is zero by zero, so all rectangles can cover it.) A rectangle that can completely cover another rectangle measuring 0.024 by 0.076 must have an area of at least 0.018.

Two-Stage Special Minimum Area Rule

The two-stage minimum area rule allows you to define a smaller minimum area for shapes in general and a larger minimum area for special shapes. Applying different minimum-area rules for different shapes can help conserve routing resources. Without this capability, you would need to specify a larger minimum area to cover all kinds of shapes.

Use the `minArea` attribute to define a general minimum area for all geometries. Use the `specialMinArea` attribute to define a different, larger minimum area for special shapes. A special shape is a polygon having the following characteristics:

- All edge lengths are less than a specified threshold, `minAreaEdgeThreshold`
- It cannot cover a rectangle measuring `minAreaEdgeThreshold` by `minWidth`

Both of these conditions must be met for the `specialMinArea` rule to apply. If a shape has any edge longer than `minAreaEdgeThreshold` or can completely cover a rectangle measuring `minAreaEdgeThreshold` by `minWidth`, then the `specialMinArea` rule does not apply; the lower minimum area `minArea` rule is used instead. The `specialMinArea` setting must be greater than the `minArea` setting.

You enable the two-stage minimum area rule by setting the `minAreaMode` attribute to 1 in the `Technology` section of the technology file. By default, it is set to 0 and the router checks only the `minArea` rule. With the attribute set to 1, the router honors the `specialMinArea` value for special shapes and the `minArea` value for ordinary shapes.

In the following example, the two-stage minimum area rule is enabled. For layer M1, the general minimum area is 0.02, but for special shapes, the minimum area is 0.05. A special shape is a polygon composed entirely of segments less than 0.20 in length and cannot cover a rectangle measuring 0.20 by 0.07.

```
Technology {
    minAreaMode = 1
}

Layer "M1" {
    minWidth           = 0.07
    minArea            = 0.02
    minAreaEdgeThreshold = 0.20
    specialMinArea     = 0.05
    ...
}
```

This rule is supported by both Zroute and the classic router. It is not necessary to specify `minAreaMode = 1` for Zroute, but it is required for the classic router.

Polygon-Edge-Based Multistage Minimum Area Rule

The polygon-edge-based multistage minimum area rule defines a set of minimum areas that depend on the lengths of the segments of the polygon. A polygon made exclusively with shorter segments requires a larger minimum area. The segment length thresholds and the corresponding minimum areas are specified in a table.

This rule is supported by Zroute only, not the classic router.

For a three-stage minimum area rule, you define a table size of two and specify two length thresholds, L1 and L2. This creates three ranges of polygon edge lengths: short (less than L1), medium (L1 to less than L2), and long (greater than or equal to L2). There are three minimum area values: A0, A1, and A2. A0 is the smallest value, A1 is the largest value, and A2 is an intermediate value. The rule defines the minimum area in three stages:

- Stage One: Any metal geometry must have an area of at least A0, the smallest of the three area values.
- Stage Two: Any polygon, including a rectangle, consisting completely of short edges less than L1, must have an area of at least A1, the largest of the three area values.

If all edges of a polygon are less than the edge length threshold L1, but a rectangle with dimensions L1 by W1 can be completely filled into the polygon, the stage-two special minimum area requirement A1 is waived.

- Stage Three: Any polygon, including a rectangle, consisting completely of short and medium edges or just medium edges, must have an area of at least A2, the intermediate area value.

If all edges of a polygon are less than the edge length threshold L2, but a rectangle with dimensions L2 by W2 can be completely filled into the polygon, the stage-three special minimum area requirement A2 is waived.

This is the syntax for the three-stage minimum area rule:

```
Layer "MetalX" {
    minArea                = A0
    specialMinAreaTblSize  = 2
    minAreaEdgeThresholdTbl = (L1, L2)
    minAreaFillMinLengthTbl = (L1, L2)
    minAreaFillMinWidthTbl  = (W1, W2)
    specialMinAreaTbl      = (A1, A2)
}
```

For example,

```
Layer "metalX" {
    minArea                = 0.0120
    specialMinAreaTblSize  = 2
    minAreaEdgeThresholdTbl = (0.15, 0.21)
    minAreaFillMinLengthTbl = (0.15, 0.21)
    minAreaFillMinWidthTbl  = (0.06, 0.06)
    specialMinAreaTbl      = (0.0450, 0.0160)
}
```

In this example, all geometries must have an area of at least 0.0120. A polygon made entirely of short segments less than 0.15 must have an area of at least 0.0450, unless that polygon can completely cover a rectangle measuring 0.15 by 0.06. A polygon made entirely of short and medium segments, or just medium segments, must have an area of at least 0.0160, unless that polygon can completely cover a rectangle measuring 0.15 by 0.06.

Setting the table size to 1 is an alternative method to specify a two-stage special minimum area rule. For example,

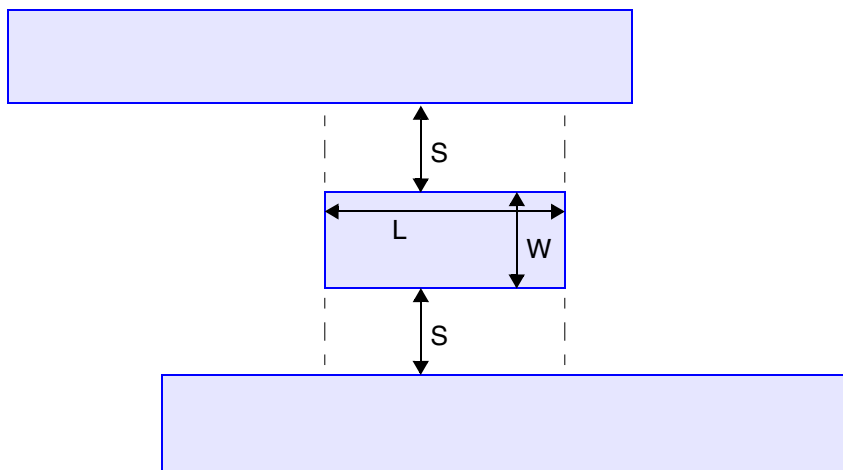
```
Layer "MetalX" {
    minArea                = 0.02
    specialMinAreaTblSize  = 1
    minAreaEdgeThresholdTbl = (0.20)
    minAreaFillMinLengthTbl = (0.20)
    minAreaFillMinWidthTbl  = (0.07)
    specialMinAreaTbl      = (0.05)
}
```

If you do not want to waive the rule, specify an invalid value such as -1 for the length and width table attributes. For example,

```
Layer "metalX" {
    minArea                = 0.02
    specialMinAreaTblSize  = 1
    minAreaEdgeThresholdTbl = (0.20)
    minAreaFillMinLengthTbl = (-1)
    minAreaFillMinWidthTbl  = (-1)
    specialMinAreaTbl      = (0.05)
}
```

Dense Wire Minimum Dimensions Rule

The dense wire minimum dimensions rule specifies the minimum length and width of a metal rectangle between closely space wires, like the rectangle shown in [Figure 2-3](#). When the space between the rectangle and either nearby wire is less than the value S, then the rectangle must have a length of at least L or a width of at least W.

Figure 2-3 Dense Wire Minimum Dimensions Rule

This is the general syntax of the rule:

```
Layer "Mx" {
    denseWireMinWidth      = W
    denseWireMinLength     = L
    denseWireMinSpacing    = S
}
```

For example,

```
Layer "M2" {
    denseWireMinWidth      = 0.22
    denseWireMinLength     = 0.78
    denseWireMinSpacing    = 0.23
}
```

Minimum Enclosed Area Rule

The minimum enclosed area rule defines the minimum area enclosed by ring-shaped wires or vias. You define this rule in a `Layer` section of the technology file by specifying the `minEnclosedArea` attribute for the associated layer. Define this attribute in a metal `Layer` section of the technology file. For example,

```
Layer "M1" {
    minEnclosedArea        = 0.2
}
```

IC Compiler honors the minimum enclosed area rule by avoiding the creation of fat wires. Use the `fatTblThreshold` attribute to specify the thresholds that the router uses to determine whether a metal wire is fat. Use the `fatTblMinEnclosedArea` attribute to specify

the minimum enclosed area for thin metal; the router uses the first nonzero value in the list. These attributes are defined in a `Layer` section of the technology file.

In the following example, the router avoids creating any metal wires with widths greater than or equal to 0.155.

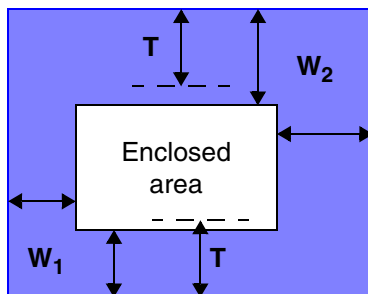
```
Layer "M1" {
    fatTblDimension      = 3
    fatTblThreshold      = (0, 0.155, 1.605)
    fatTblMinEnclosedArea = (0.3, 1.0, 1.0)
}
```

You can also set the `fatTblMinEnclosedAreaMode` attribute in the `Technology` section of the technology file. This attribute determines the mode in which the router checks for and avoids a violation:

- When `fatTblMinEnclosedAreaMode = 0`, which is the default, the fat metal minimum enclosed area mode is triggered when any of the surrounding metal is narrower than the threshold width.
- When `fatTblMinEnclosedAreaMode = 1`, the fat metal minimum enclosed area mode is triggered only when all of the surrounding metal is less than the threshold width.

In [Figure 2-4](#), the width of the metal surrounding the enclosed area is W_1 on two sides and W_2 on the other two sides. If W_1 is less than the fat threshold T and W_2 is more than the fat threshold T , the `fatTblMinEnclosedAreaMode` setting determines whether the minimum enclosed area rule is applied. If the mode is set to 0 (the default), the rule is applied because there is at least one surrounding width below the threshold. If the mode is set to 1, the rule is not applied because not all of the surrounding widths are below the threshold.

Figure 2-4 Minimum Enclosed Area Rule Modes



This rule can be useful when standard cells have ring-shaped pins. The cells themselves satisfy the rule, possibly using a minimum-sized enclosed area. When a router connects to these pins, it must make the connection in such a manner that no fat wire is created on any edges or on all edges of the enclosed area, depending on the technology requirements, so that there is no requirement to have a larger enclosed area.

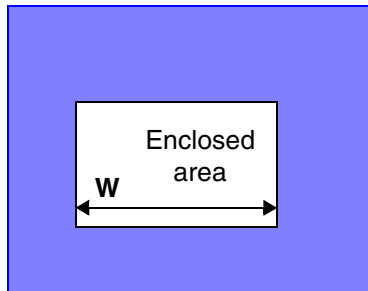
Minimum Enclosed Width Rule

The `minEnclosedWidth` attribute specifies the minimum width of an enclosed area for the given routing layer. Define this attribute in a metal `Layer` section of the technology file. For example,

```
Layer "M1" {  
    minEnclosedWidth = 0.7  
}
```

The width of an enclosed area checked by this rule is the larger of the two dimensions of the area, as shown in [Figure 2-5](#).

Figure 2-5 Minimum Enclosed Width



This rule can be useful when standard cells have ring-shaped or U-shaped pins. The cells themselves satisfy the rule. When a router connects to these pins, it must make the connection in such a manner that no enclosed area is created, or if an enclosed area is created or changed, the larger dimension of the enclosed area is at least as large as the specified minimum width.

Metal Width Rules

The following rules restrict the allowed widths of metal routes:

- [Signal Routing Maximum Metal Width Rule](#)
- [Discrete Metal Widths Rule](#)
- [No Routing in Nonpreferred Direction](#)
- [Minimum Width in Nonpreferred Direction](#)
- [Default Width in Nonpreferred Direction](#)

Signal Routing Maximum Metal Width Rule

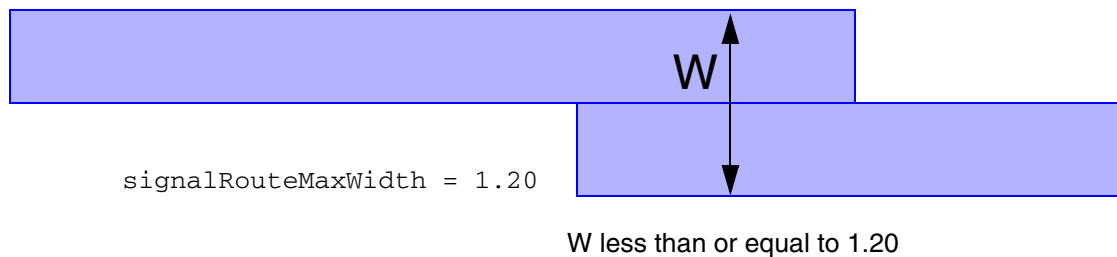
To specify a maximum allowed width for any signal route in a layer, set the `signalRouteMaxWidth` attribute to the maximum value in the `Layer` section.

For example, to set the maximum width for signal routing to 1.20 microns in layer M1, use:

```
Layer    "M1" {
    signalRouteMaxWidth = 1.20
}
```

This rule applies to signal routing only, not power supply and ground routing. See [Figure 2-6](#).

Figure 2-6 Routing Maximum Metal Width Rule



If the tool has nondefault routing rules with a width defined, make sure that the specified width does not violate the `signalRouteMaxWidth` rule. Otherwise, it will trigger an RT-068 error and cause the nondefault routing rule to be ignored.

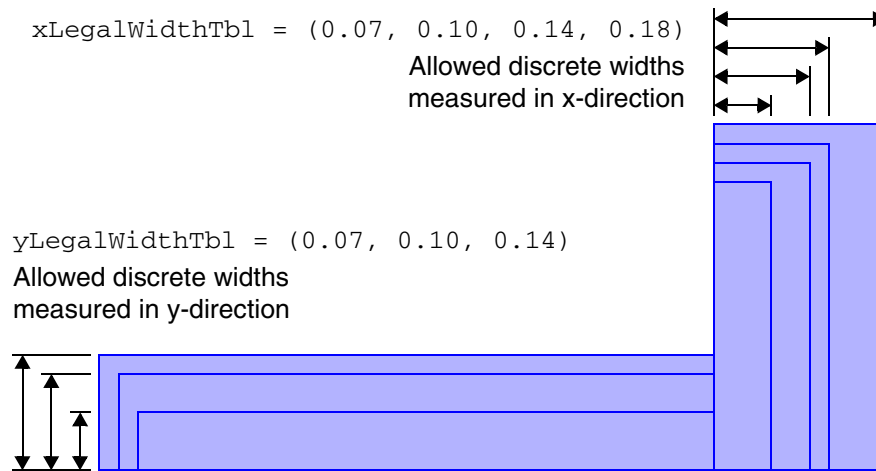
Discrete Metal Widths Rule

The discrete metal widths rule specifies a limited set of allowed widths for any shape in a given layer. Only the discrete widths specified in the list are allowed. No width is allowed to be between two successive values in the list or outside the range of values in the list (by default). The list can be different for the x-direction and y-direction.

In the rule, you specify the number of allowed widths and the list of widths, measured in each direction. The following example shows the syntax.

```
Layer "Metal1" {
    xLegalWidthTblSize = 4
    yLegalWidthTblSize = 3
    xLegalWidthTbl     = (0.07, 0.10, 0.14, 0.18)
    yLegalWidthTbl     = (0.07, 0.10, 0.14)
}
```

The resulting allowed discrete widths are shown in [Figure 2-7](#).

Figure 2-7 Discrete Metal Widths Rule

In each checking direction, one of the legal widths must match the default width defined by `xDefaultWidth` or `yDefaultWidth`.

To allow any width beyond the last value in each table, use the following additional attribute:

```
legalWidthBeyondLastValue = 1
```

By adding this line to the previous example, all widths greater than or equal to 0.18 measured in the x-direction, and greater than or equal to 0.14 in the y-direction, are allowed.

A square shape in a metal layer is treated as a wire running in the preferred direction. For example, a square measuring 0.18 by 0.18 is checked with the `xLegalWidthTbl` list if the preferred direction is vertical, or checked with the `yLegalWidthTbl` list if the preferred direction is horizontal. To have square shapes treated as running the nonpreferred direction instead, use the following attribute setting:

```
legalWidthCheckNonPrefForSquare = 1
```

No Routing in Nonpreferred Direction

To prohibit all routing in the nonpreferred direction for a layer, set the `nonPreferredRouteMode` attribute to 1 in the `Layer` section. For example,

```
Layer "M2" {
    nonPreferredRouteMode = 1
}
```

Then the router creates routes only in the preferred direction.

Minimum Width in Nonpreferred Direction

The minimum width for routes in a metal layer is defined by the `minWidth` attribute in the `Layer` section of the technology file. For example,

```
Layer "M2" {  
    minWidth = 0.07  
}
```

By default, this minimum width requirement applies to all metal shapes in both the preferred and nonpreferred directions.

To specify a different minimum width requirement for routes running in the nonpreferred direction, set the `nonPreferredWidth` attribute to the desired value, which must be greater than or equal to the `minWidth` setting. For example,

```
Layer "M2" {  
    minWidth = 0.07  
    nonPreferredWidth = 0.09  
}
```

In this example, the `minWidth` setting of 0.07 applies to routes running in the preferred direction and the `nonPreferredWidth` setting of 0.09 applies to routes running in the nonpreferred direction.

Default Width in Nonpreferred Direction

The default width for signal routes in a metal layer is defined by the `defaultWidth` attribute in the `Layer` section of the technology file. For example,

```
Layer "M2" {  
    defaultWidth = 0.10  
}
```

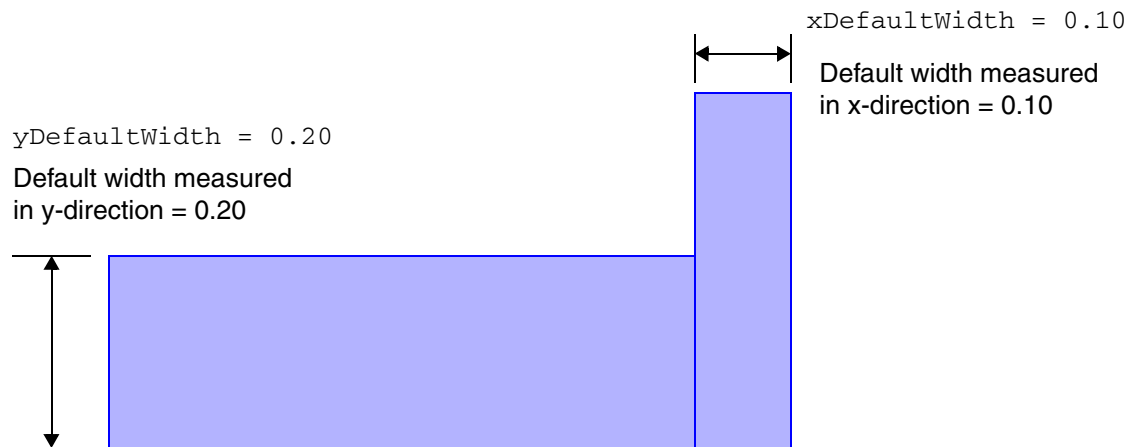
By default, this default width applies to signal routing in both the preferred and nonpreferred directions.

To specify different default widths measured in the x-direction and y-direction, define `xDefaultWidth` for vertical wires and `yDefaultWidth` for horizontal wires in addition to `defaultWidth`. For example,

```
Layer "M2" {  
    defaultWidth = 0.10  
    xDefaultWidth = 0.10  
    yDefaultWidth = 0.20  
}
```

The router uses these widths by default, as shown in [Figure 2-8](#).

Figure 2-8 Different Default Metal Widths in the X-Direction and Y-Direction



Specifying `defaultWidth` is mandatory for each layer, whereas using `xDefaultWidth` and `yDefaultWidth` is optional. If used, `xDefaultWidth` and `yDefaultWidth` override the value specified by `defaultWidth` as measured in the specified directions.

Note that `defaultWidth`, `xDefaultWidth`, and `yDefaultWidth` specify the default width chosen by the router, not the minimum allowed width, which is controlled by the `minWidth` attribute. The `xDefaultWidth` and `yDefaultWidth` settings must be greater than or equal to the `defaultWidth` setting, and the `defaultWidth` setting must be greater than or equal to the `minWidth` setting. Specifying `minWidth` is mandatory for each layer.

Minimum Edge Rules

The minimum edge rules are described in the following sections:

- [Minimum Edge Mode](#)
- [Maximum Total Number of Short Edges Rule](#)
- [Maximum Total Short Edge Length Rule](#)
- [Convex-Concave Minimum Edge Length Rule](#)
- [Special Notch Rule](#)
- [Hook Rule](#)
- [H-Shape Rule](#)
- [Adjacent Minimum Edge Length Rule](#)

- [Short Edge to End-of-Line Rule](#)
- [Minimum Edge for Macro Pin Connection Rule](#)

Minimum Edge Mode

You control the scope of the check for the minimum edge rules by setting the `minEdgeMode` attribute in the `Technology` section of the technology file. For example,

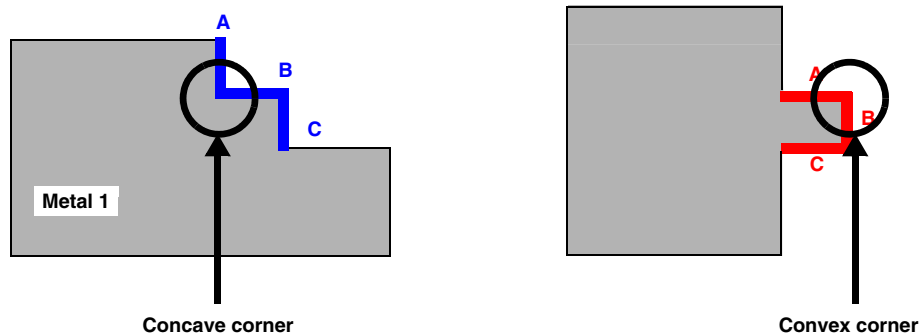
```
Technology {
    minEdgeMode = 0
}
```

When the mode is set to 0 (the default), a small concave corner must be present to trigger a violation. A small concave corner is formed by two adjacent edges that are both less than the minimum edge length. When the mode is set to 1, a small concave corner need not be present to trigger a violation; a convex corner alone can trigger a violation.

Maximum Total Number of Short Edges Rule

The maximum total number of short edges rule specifies the maximum number of consecutive short edges. The scope of the rule is controlled by the `minEdgeMode` attribute in the `Technology` section of the technology file. When `minEdgeMode` is 0, violations are triggered only when the short edges include a concave corner. When `minEdgeMode` is 1, a concave corner is not necessary to trigger a violation; a convex corner alone can trigger a violation. For example, in [Figure 2-9](#), assume edges A, B, and C are all short edges and the maximum number of consecutive short edges is 2. This rule is violated in the example on the left, regardless of the `minEdgeMode` setting, because it contains a concave corner, but the rule is violated in example on the right only when `minEdgeMode` is 1.

Figure 2-9 Minimum Edge Examples



To define this rule, set the following attributes in a metal `Layer` section of the technology file:

- `minEdgeLength`

This attribute specifies the maximum length that defines a short edge.

- `maxNumMinEdge`

This attribute specifies the number of consecutive edges to check. This rule is violated only when the number of consecutive short edges is greater than `maxNumEdge`.

For example, to define a short edge as an edge less than 0.07 microns and the maximum number of consecutive short edges as 2, enter

```
Layer "M1" {
    minEdgeLength = 0.07
    maxNumMinEdge = 2
}
```

Maximum Total Short Edge Length Rule

The maximum total short edge length rule specifies the maximum total length of consecutive short edges. The scope of the rule is controlled by the `minEdgeMode` attribute in the `Technology` section of the technology file. When `minEdgeMode` is 0, violations are triggered only when the short edges include a concave corner. When `minEdgeMode` is 1, a concave corner is not necessary to trigger a violation; a convex corner alone can trigger a violation. For example, in [Figure 2-9 on page 2-16](#), assume edges A, B, and C are all short edges and the total length of these edges exceeds the maximum total short edge length. This rule is violated in the left-hand figure regardless of the `minEdgeMode` setting because it contains a concave corner, but the rule is violated in the right-hand figure only when `minEdgeMode` is 1.

To define this rule, set the following attributes in a metal `Layer` section of the technology file:

- `minEdgeLength`

This attribute specifies the maximum length that defines a short edge.

- `maxTotalMinEdgeLength`

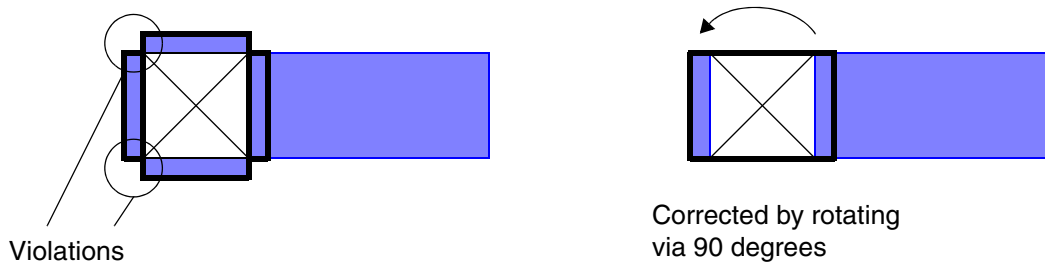
This attribute specifies the maximum total length of the consecutive short edges.

For example, to define a short edge as an edge less than 0.07 microns and the maximum total length of consecutive short edges as 0.11 microns, enter

```
Layer "M1" {
    minEdgeLength = 0.07
    maxTotalMinEdgeLength = 0.11
}
```

Figure 2-10 shows an example of an application of the rule. On the left, two metal enclosures of stacked vias are perpendicular to each other, causing the short edge length errors circled in the diagram. This can be corrected by rotating one of the vias by 90 degrees, as shown on the right, causing the vias to coincide perfectly.

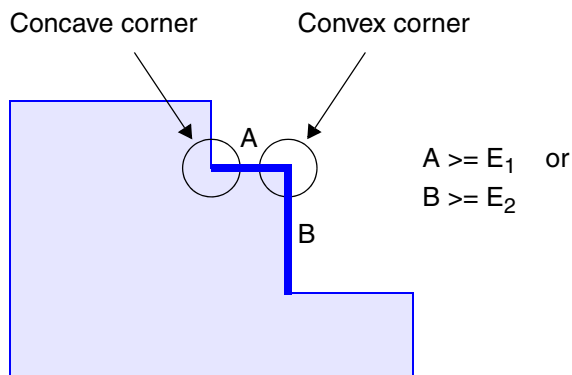
Figure 2-10 Total Short Edge Length Rule



Convex-Concave Minimum Edge Length Rule

The convex-concave minimum edge length rule specifies that a metal edge connecting a concave corner and a convex corner must have a length of at least E_1 , or if less than E_1 , the adjacent edge that forms the convex corner must have a length of at least E_2 . See [Figure 2-11](#).

Figure 2-11 Concave-Convex Minimum Length Rule



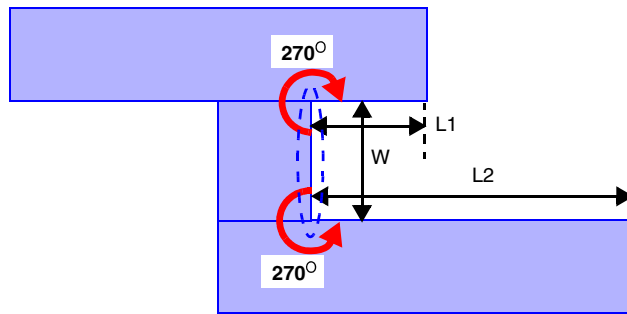
This is the syntax of the rule:

```
Layer "MetalX" {
    convexConcaveMinEdgeLength = E1
    convexMinEdgeLength        = E2
}
```

Special Notch Rule

The special notch rule specifies the minimum width (W in [Figure 2-12](#)) of a notch when at least one of the adjacent edges (L1 and L2 in [Figure 2-12](#)) is less than a specified length.

Figure 2-12 Special Notch Rule Example



To define this rule, set the following attributes in a metal `Layer` section of the technology file:

- `minEdgeLength2`

This attribute specifies the length threshold for applying this rule. This rule applies if at least one of the edges adjacent to the notch is less than `minEdgeLength2`.

- `minEdgeLength3`

This attribute specifies the minimum width of the notch, W.

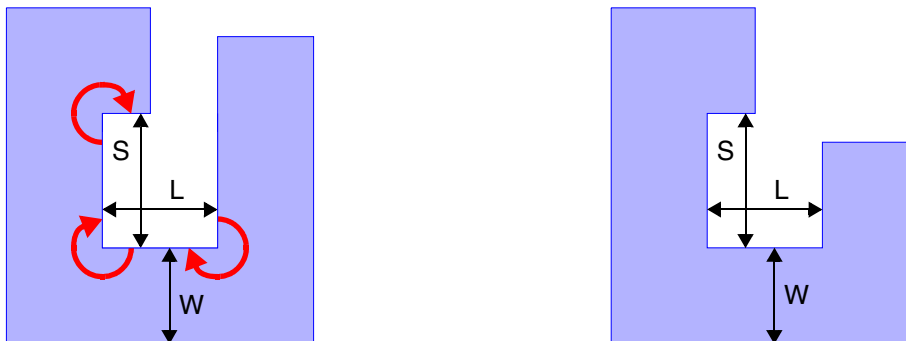
For example, to specify that the minimum notch width is 0.26 microns when at least one of the adjacent edges is less than 0.4 microns, enter the following:

```
Layer "M1" {
    minEdgeLength2 = 0.4
    minEdgeLength3 = 0.26
}
```

Special notch rule violations can be fixed by rerouting wires, rotating vias, or adding stubs.

Hook Rule

The hook rule specifies the minimum spacing between facing edges in a hook shape, where the metal edge has three consecutive 270-turns within a short space, as shown in [Figure 2-13](#). When the inside length of the hook base is less than L and the width of the hook base is less than or equal to W, then the inside height of the hook must be at least S. The rule applies to both of the cases shown in the figure.

Figure 2-13 Hook Rule Examples

This is the general syntax of the rule:

```

Layer "M1" {
    minEdgeLength2           = S
    minEdgeLength3           = L
    minEdgeLengthCheckConcaveCorner = 1
    minEdgeLength2MaxWireWidth = W
}

```

For example,

```

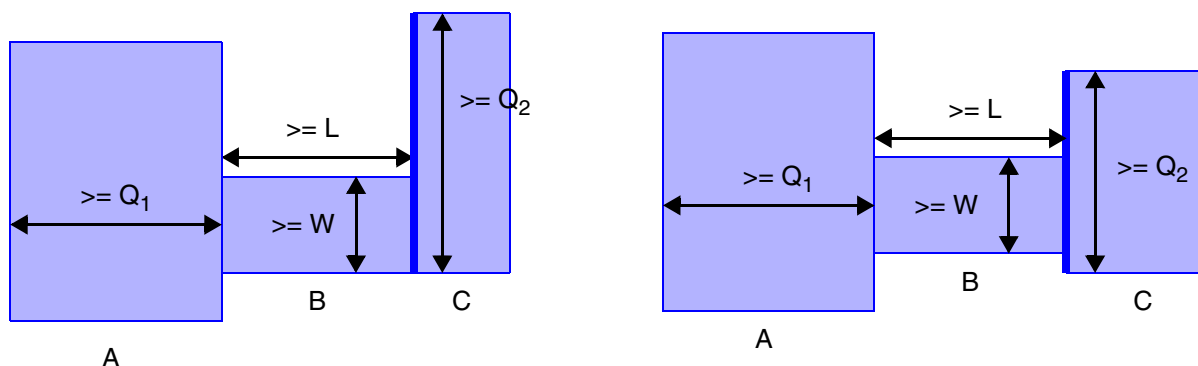
Layer "M1" {
    minEdgeLength2           = 0.170
    minEdgeLength3           = 0.077
    minEdgeLengthCheckConcaveCorner = 1
    minEdgeLength2MaxWireWidth = 0.061
}

```

H-Shape Rule

The H-shape rule requires a minimum length or minimum width requirement to be satisfied for a metal shape that joins two other metal shapes. In the two examples shown in [Figure 2-14](#), shape B joins shapes A and C, forming a shape resembling the letter “H.” If the width of shape A is at least Q_1 and the length of shape C is at least Q_2 , then either the width of shape B must be at least W or the length of shape B must be at least L.

Figure 2-14 H-Shape Rule



This is the syntax of the rule:

```

Layer "MetalX" {
    hShape1WidthThreshold      = Q1
    hShape2LengthThreshold    = Q2
    hShapeMinWidth            = W
    hShapeMinLength           = L
}

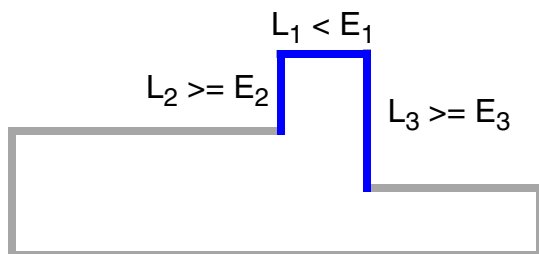
```

The length threshold attribute Q_2 must be greater than the minimum width attribute W . Otherwise, the shape is not considered an H-shape and the rule is not checked.

Adjacent Minimum Edge Length Rule

The adjacent minimum edge length rule specifies the minimum lengths of edges adjacent to a short edge of a metal polygon. If one edge has a length less than E_1 , the two adjacent edges must have lengths of at least E_2 and E_3 , as shown in [Figure 2-15](#).

Figure 2-15 Adjacent Minimum Edge Length Rule Example



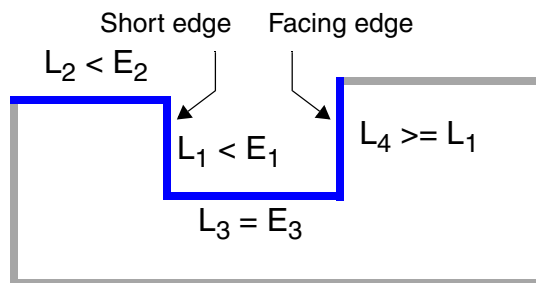
Here is an example of the rule:

```
Layer "M1" {
  minEdgeLengthTblSize = 3
  minEdgeLengthTbl = (0.26, 0.30, 0.40)
}
```

The three numbers in the `minEdgeLengthTbl` table are E1, E2, and E3, respectively. This rule says that for layer M1, if there is an edge less than 0.26 microns, the one adjacent edge must be at least 0.30 microns and the other must be at least 0.40 microns.

In some technologies, the adjacent minimum edge length rule is waived if another edge of the polygon faces the short edge, and that other edge is at least as long as the short edge, as shown in [Figure 2-16](#).

Figure 2-16 Short Edge Waived With Facing Edge

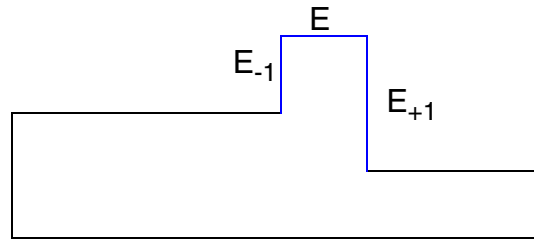


To allow the rule to be waived under these conditions, set the `minEdgeLengthFacingEdgeExcluded` attribute to 1, as in the following example:

```
Layer "M1" {
  minEdgeLengthTblSize = 3
  minEdgeLengthTbl = (0.26, 0.30, 0.40)
  minEdgeLengthFacingEdgeExcluded = 1
}
```

If `minEdgeLengthFacingEdgeExcluded` is set to 0 or omitted, then the rule is not waived for the facing edge condition.

The following alternative rule is supported by the classic router, but not Zroute. This three-adjacent-edge minimum length rule specifies the minimum lengths of the edges E_{-1} and E_{+1} connected to a short edge E , as shown in [Figure 2-17](#).

Figure 2-17 Three Adjacent Edges Considered

To define this rule, set the following detail route options:

- `minEdgeLengthMode`
This option must be set to 0 for this rule.
- `MxMinEdgeLength4`
This option specifies the maximum length for edge E to be considered a short edge.
- `MxMinEdgeLength5`
This option specifies the minimum length of edge E_{-1} . This value must be less than or equal to the value specified for `MxMinEdgeLength6`.
- `MxMinEdgeLength6`
This option specifies the minimum length of edge E_{+1} .

where x is the metal layer to which the rule applies. Valid values for x are 1 through 15. The settings of these detail route options are saved with the cell.

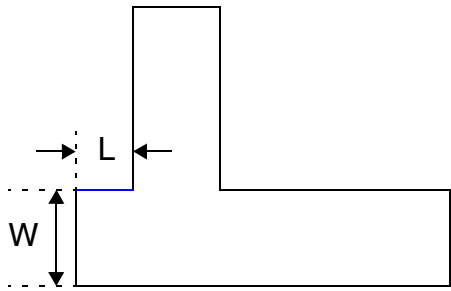
For example, to define the rule for the metal 1 layer such that when E is less than 0.4 microns, E_{-1} must be at least 0.26 microns and E_{+1} must be at least 0.3 microns, enter the following commands:

Note: This form is supported by classic router only, not Zroute

```
icc_shell> set_droute_options -name minEdgeLengthMode -value 0
icc_shell> set_droute_options -name M1MinEdgeLength4 -value 0.4
icc_shell> set_droute_options -name M1MinEdgeLength5 -value 0.26
icc_shell> set_droute_options -name M1MinEdgeLength6 -value 0.3
```

Short Edge to End-of-Line Rule

The short edge to end-of-line rule specifies the minimum length of an edge adjacent to a line-end edge that is less than a specified width. In [Figure 2-18](#), when the width of the line-end is less than W, the short edge L must be at least a specified value.

Figure 2-18 Short Edge to End-of-Line Rule

For Zroute, you define this rule in a `Layer` section by using the `minEdgeLengthTblSize` and `minEdgeLengthTbl` attributes. For example,

```
Layer "M1" {
  minEdgeLengthTblSize = 2
  minEdgeLengthTbl = (0.26, 0.4)
}
```

In this example, for a line-end width `W` of less than 0.4, the short edge `L` must be at least 0.26.

For the classic router, you define this rule by using the `minEdgeLengthMode`, `MnMinEdgeLength4`, and `MnMinEdgeLength5` detail route options.

When the `minEdgeLengthMode` detail route option is set to 1,

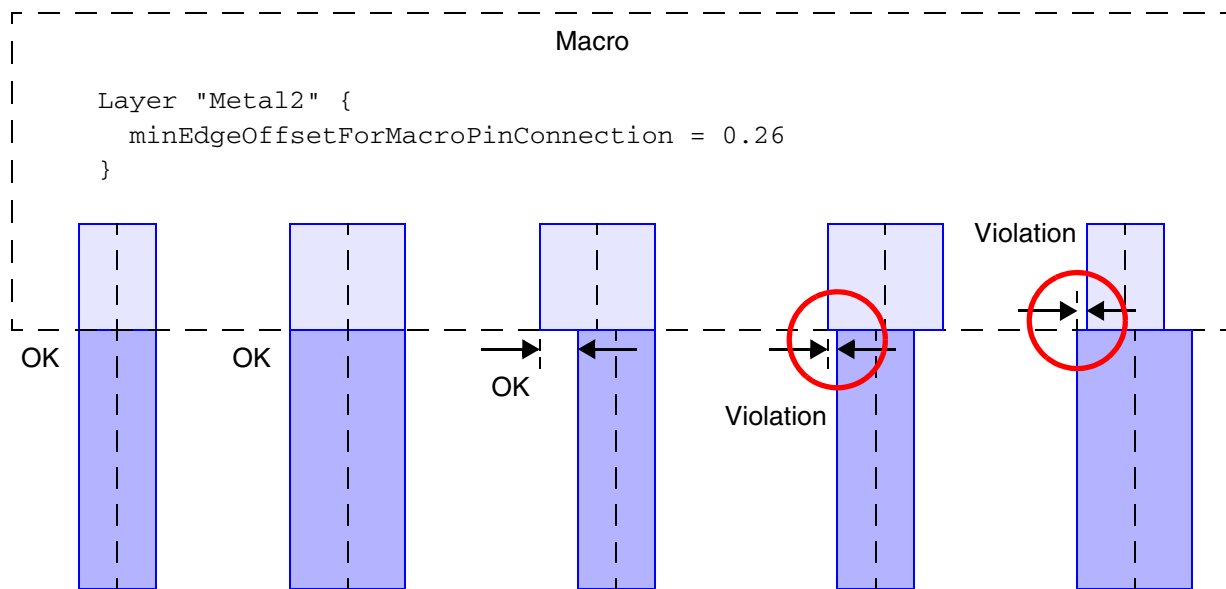
- `MnMinEdgeLength4` defines the short edge length
- `MnMinEdgeLength5` defines the end-of-line width threshold that triggers the rule

For example, to set the threshold `W` to 0.4 and the minimum short edge to 0.26 for layer `M4`, use the following commands:

```
icc_shell> set_droute_options -name minEdgeLengthMode -value 1
icc_shell> set_droute_options -name M4MinEdgeLength4 -value 0.26
icc_shell> set_droute_options -name M4MinEdgeLength5 -value 0.4
```

Minimum Edge for Macro Pin Connection Rule

The minimum edge for macro pin connection rule enforces exactly aligned connections between metal wires and macro pins where they connect. If the macro pin and wire have different widths, the short edges resulting at the connection point must have at least a specified length. See [Figure 2-19](#).

Figure 2-19 Minimum Edge for Macro Pin Connection Rule

To invoke the rule, set the `minEdgeOffsetForMacroPinConnection` attribute to the desired minimum edge length at the metal-to-pin connection for the metal layer. For example,

```
Layer "Metal2" {
  minEdgeOffsetForMacroPinConnection = 0.26
}
```

Minimum Spacing Rules

The minimum spacing rules are described in the following sections:

- [Minimum Spacing Rule](#)
- [U-Shape Spacing Rule](#)
- [Via Corner Spacing Rule](#)
- [Via Same Net Minimum Spacing Rule](#)
- [Edge-Line Via Spacing Rule](#)
- [General Cut Spacing Rule](#)
- [Asymmetric Via-to-Metal Spacing Rule](#)

Minimum Spacing Rule

Use the `minSpacing` attribute in a `Layer` section of the technology file to specify the minimum spacing allowed for that layer between the edges of two objects on different nets. This is the default minimum spacing requirement for that layer.

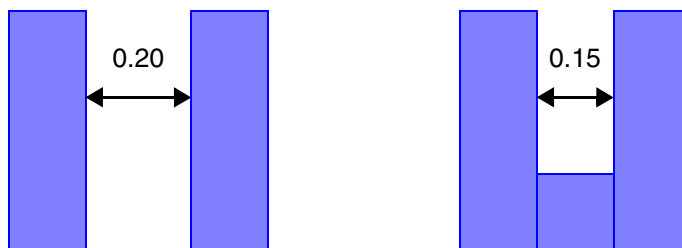
You can specify a smaller minimum spacing rule for objects that are on the same net. Use the `sameNetMinSpacing` attribute in a `Layer` section of the technology file to specify the minimum spacing allowed between the edges of two objects on the same net.

For example,

```
Layer "M1" {
  minSpacing          = 0.20
  sameNetMinSpacing   = 0.15
}
```

In [Figure 2-20](#), the unconnected wires must have a spacing of at least 0.20, whereas the connected wires must have a spacing of at least 0.15.

Figure 2-20 Unconnected and Connected Minimum Spacing Rule

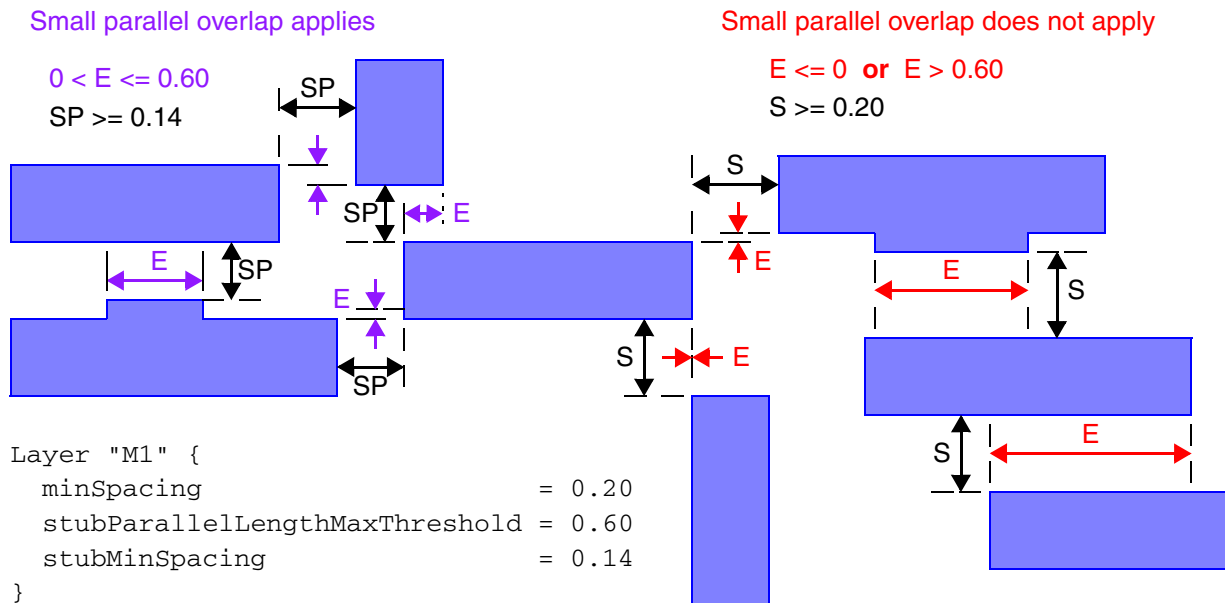


If the minimum spacing is allowed to be less when the parallel overlap between the metal shapes is no more than a specific threshold, you can specify the smaller minimum spacing and parallel overlap threshold using the following syntax:

```
Layer "M1" {
  minSpacing          = 0.20
  stubParallelLengthMaxThreshold = 0.60
  stubMinSpacing       = 0.14
}
```

In this example, if the parallel overlap between the two metal shapes is greater than zero but less than or equal to 0.60, then the minimum spacing is 0.14. This rule applies to metal shapes belonging to the same net or different nets. If the parallel overlap is zero or less, or more than 0.60, the `minSpacing` value (0.20) applies instead. [Figure 2-21](#) shows some examples where the parallel overlap rule does and does not apply.

Figure 2-21 Less Restrictive Minimum Spacing for Small Parallel Overlap

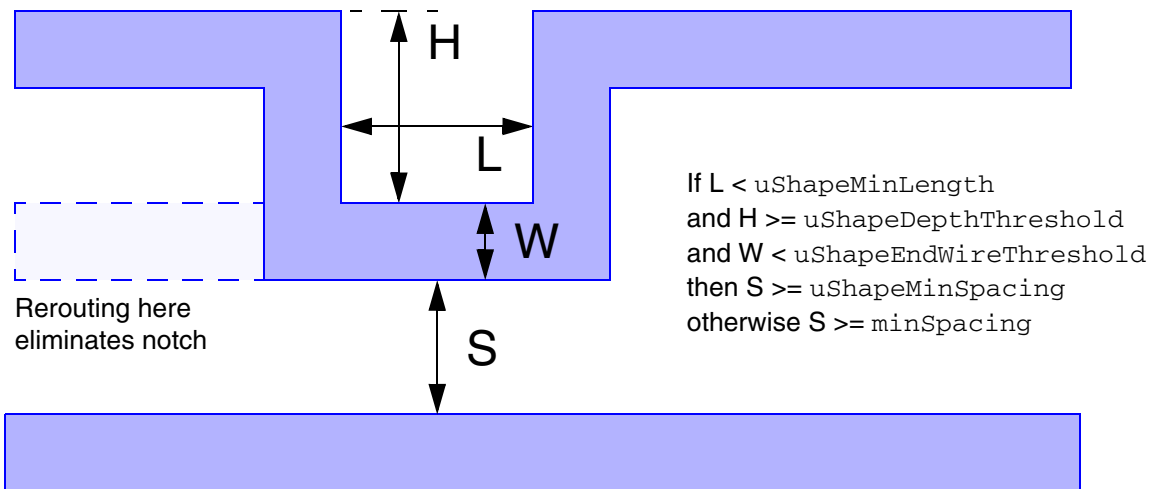


U-Shape Spacing Rule

To specify the minimum spacing between a short U-shaped notch and a neighboring wire, use the `uShapeMinSpacing`, `uShapeDepthThreshold`, `uShapeEndWireThreshold`, and `uShapeMinLength` attributes. Define these attributes in a metal `Layer` section of the technology file.

In [Figure 2-22](#), the rule defines the minimum spacing (S), as specified with `uShapeMinSpacing`, between two edges when the height (H) is greater than or equal to `uShapeDepthThreshold`, the length (L) is less than `uShapeMinLength`, and the width of the nearby U-shaped wire base (W) is less than `uShapeEndWireThreshold`. Otherwise, the smaller default minimum spacing specified with `minSpacing` applies.

Figure 2-22 U-Shape Spacing Rule



For example,

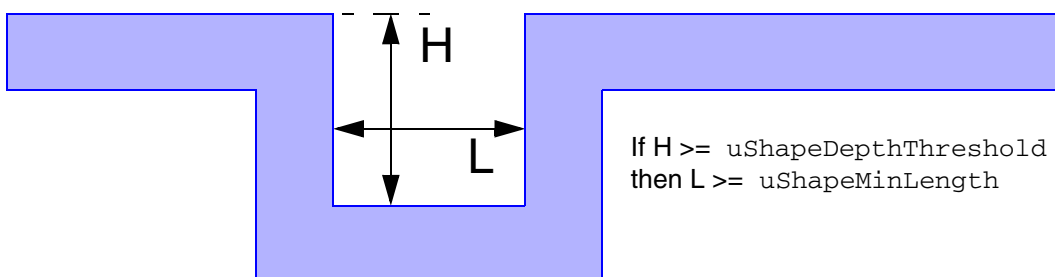
```

Layer "M1" {
    uShapeMinSpacing      = 0.13
    uShapeDepthThreshold  = 0.05
    uShapeEndWireThreshold = 0.04
    uShapeMinLength       = 0.16
}

```

An alternative form of the U-shape spacing rule specifies only the minimum depth and minimum length of the U shape, without considering the distance to any nearby metal, as shown in Figure 2-23.

Figure 2-23 Alternative U-Shape Spacing Rule



For example,

```

Layer "M1" {
    uShapeDepthThreshold  = 0.05
    uShapeMinLength       = 0.16
}

```

Via Corner Spacing Rule

Use the `cornerMinSpacing` attribute to specify the minimum corner-to-corner spacing allowed between two vias, which must be smaller than the minimum spacing value specified with the `minSpacing` attribute.

For corner spacing on the same via layer, define this attribute in a via `Layer` section of the technology file. For example,

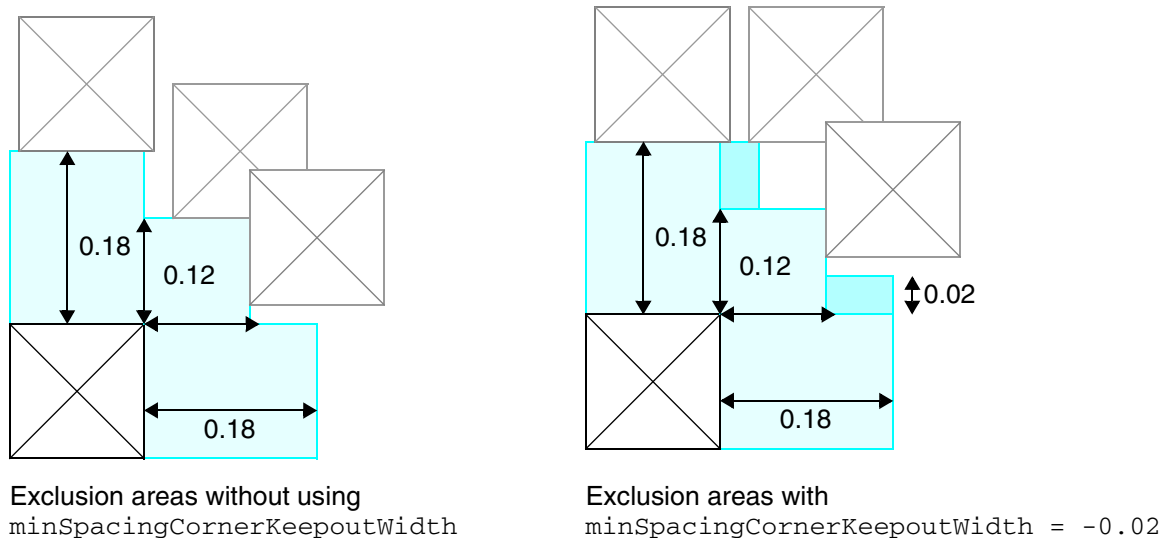
```
Layer "VIA1" {
  minSpacing      = 0.18
  cornerMinSpacing = 0.12
}
```

If the `cornerMinSpacing` value applies only when the parallel spacing between the two vias is less than a given threshold, use `minSpacingCornerKeepoutWidth` to specify that threshold. If the parallel spacing is more than the threshold, then the `minSpacing` value applies instead of `cornerMinSpacing`. For example,

```
Layer "VIA1" {
  minSpacing              = 0.18
  cornerMinSpacing        = 0.12
  minSpacingCornerKeepoutWidth = -0.02
}
```

This rule creates the exclusion area shown in [Figure 2-24](#).

Figure 2-24 Via Corner Spacing Rule

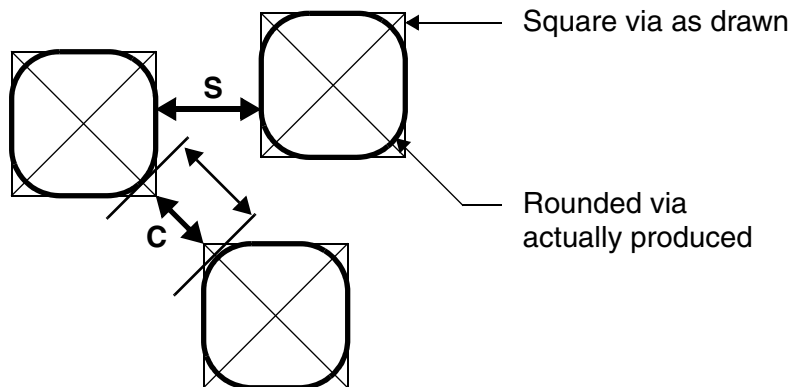


For corner spacing between different via layers, define this attribute in the `DesignRule` section of the technology file. For example,

```
DesignRule {
  layer1          = "VIA1"
  layer2          = "VIA2"
  minSpacing      = 0.20
  cornerMinSpacing = 0.12
}
```

The reason for allowing closer spacing between corners is that the edges of real physical vias are typically rounded within the boundaries of the drawn vias, as depicted in [Figure 2-25](#). Therefore, the minimum spacing requirement C between drawn corners might be less than the minimum spacing requirement S between side-by-side edges. By allowing a closer spacing C between drawn corners, routing resources can be conserved.

Figure 2-25 Drawn Vias Versus Actual Physical Vias

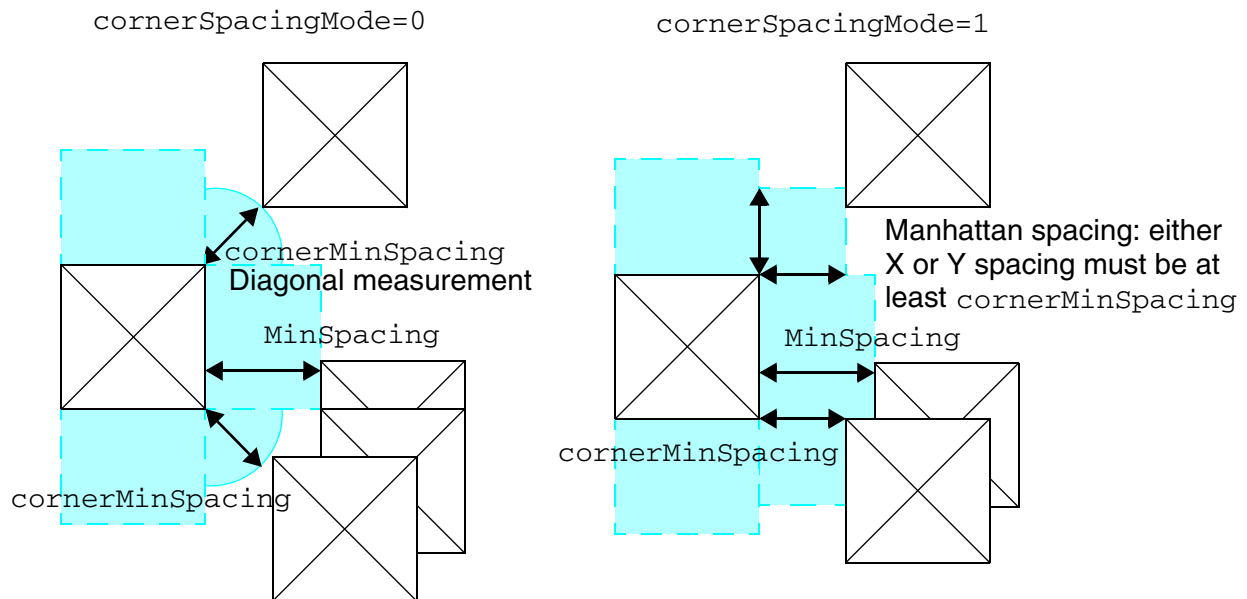


You can control the measurement method, either diagonal or Manhattan, for corner-to-corner spacing between vias by setting the `cornerSpacingMode` attribute in the `Technology` section of the technology file. By default (`cornerSpacingMode = 0`), IC Compiler measures the diagonal distance. To use Manhattan distance instead, set the `cornerSpacingMode` attribute to 1. For example,

```
Technology {
  cornerSpacingMode = 1
}
```

[Figure 2-26](#) demonstrates the difference between diagonal and Manhattan distance measurements, as determined by the `cornerSpacingMode` setting. For the case where the outside edges of two vias line up exactly, the `minSpace` value is used if `cornerSpacingMode` is set to 0, or the `cornerMinSpacing` value is used if `cornerSpacingMode` is set to 1.

Figure 2-26 Diagonal and Manhattan Distance Measurements



Via Same Net Minimum Spacing Rule

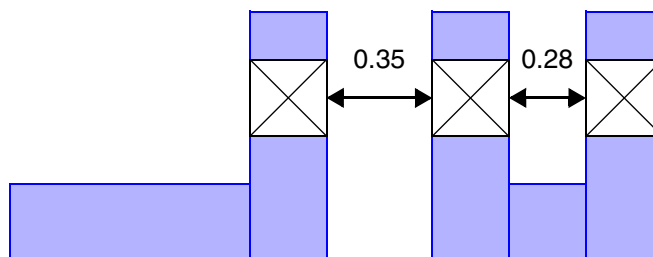
Use the `sameNetMinSpacing` attribute to override the `minSpacing` default and set a smaller value for the minimum spacing between two vias belonging to the same net.

Define this attribute in a `via Layer` section of the technology file. For example,

```
Layer "VIA5" {
  sameNetMinSpacing = 0.28
  minSpacing        = 0.35
}
```

In [Figure 2-27](#), the minimum spacing between the two vias belonging to different nets is 0.35, whereas the minimum spacing between the two vias belonging to the same net is 0.28.

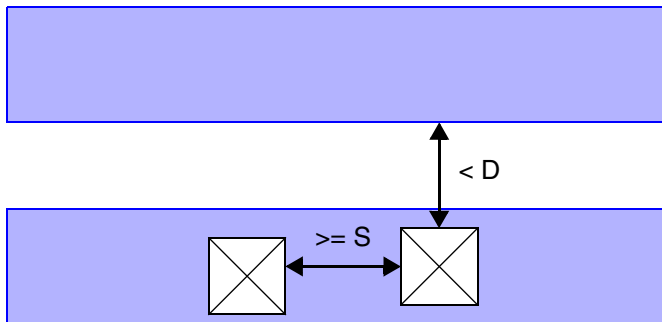
Figure 2-27 Unconnected and Connected Via Minimum Spacing Rule



Edge-Line Via Spacing Rule

The edge-line via spacing rule specifies the minimum spacing between neighboring vias if the vias are in the same metal segment, and there is a neighboring metal segment in the same metal layer within a distance D between a via edge and the neighboring metal. See [Figure 2-28](#).

Figure 2-28 Edge-Line Via Minimum Spacing Rule

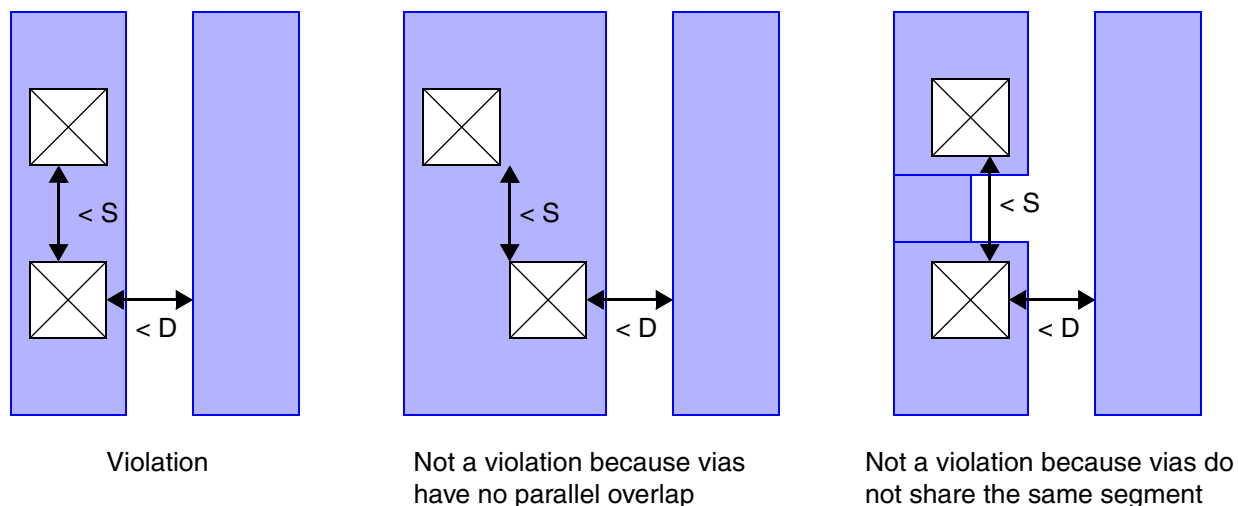


If the distance between the via and the neighboring metal is less than D , and the two vias have a parallel overlap that is greater than zero, then the spacing between the two vias must be at least S . This is the syntax of the rule:

```
Layer "VIAX" {
    sameSegAlignedUpperWireMaxSpacingThreshold = D
    sameSegAlignedLowerWireMaxSpacingThreshold = D
    sameSegAlignedCutMinSpacing                = S
}
```

The `sameSegAlignedUpperWireMaxSpacingThreshold` attribute is the threshold D for the via's upper layer. The `sameSegAlignedLowerWireMaxSpacingThreshold` attribute is the threshold D for the via's lower layer. If only one of these two attributes is included in the rule, only that layer (upper or lower) is checked.

The examples shown in [Figure 2-29](#) demonstrate the conditions under which the rule is applied.

Figure 2-29 Edge-Line Via Minimum Spacing Rule Examples

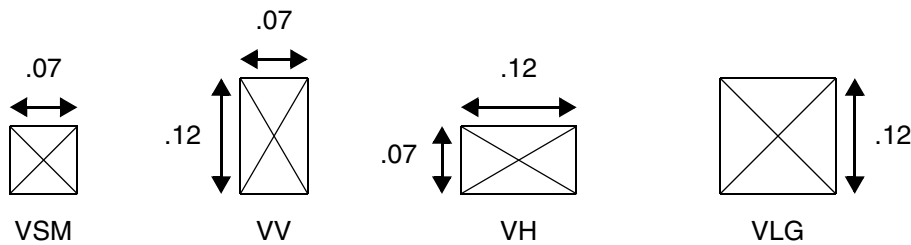
General Cut Spacing Rule

You can specify the minimum spacing between cuts (vias and contacts) based on the cut dimensions and the net and wire-segment relationships of the cuts. To use this rule, you specify the dimensions of the cuts and assign a name to each cut size. Then you create a table of minimum spacing values that apply to each possible combination of named cut sizes. You can specify separate minimum spacing values for cuts in the same net, in different nets, in the same metal segment, and in different metal segments.

Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see the previous sections, [“Via Corner Spacing Rule” on page 2-29](#) and [“Via Same Net Minimum Spacing Rule” on page 2-31](#).

Usage of this rule is best explained by example. Suppose that you use the four cut sizes shown in [Figure 2-30](#) in layer Via1.

Figure 2-30 Specifying Cut Sizes for General Cut Spacing Rule

Assign the names to the four via sizes for layer Via1 using the following syntax:

```
Layer "Via1" {
  cutTblSize = 4
  cutNameTbl = (VSM, VV, VH, VLG)
  cutWidthTbl = (.07, .07, .12, .12)
  cutHeightTbl = (.07, .12, .07, .12)
}
```

The defined cut names can be used in the `Layer`, `DesignRule`, and `ViaRule` sections. The cut names must be defined in the file before they are used in later sections.

Suppose that you use the same cut sizes in layer Via2. You similarly assign the same set of names to the same four via sizes in layer Via2 as follows:

```
Layer "Via2" {
  cutTblSize = 4
  cutNameTbl = (VSM, VV, VH, VLG)
  cutWidthTbl = (.07, .07, .12, .12)
  cutHeightTbl = (.07, .12, .07, .12)
}
```

Now you need to specify the spacing between just the square vias in layers Via1 and Via2, with different spacing values for vias in the same net versus vias in different nets. To specify these spacing values, use the following syntax:

```
DesignRule {
  layer1 = "Via1"
  layer2 = "Via2"
  cut1TblSize = 2
  cut2TblSize = 2
  cut1NameTbl = (VSM, VLG)
  cut2NameTbl = (VSM, VLG)
  orthoSpacingExcludeCornerTbl = (0, 1,
                                  1, 0)
  sameNetXMinSpacingTbl = (0.10, 0.10,
                           0.10, 0.12)
  diffNetXMinSpacingTbl = (0.11, 0.11,
                           0.11, 0.13)
}
```

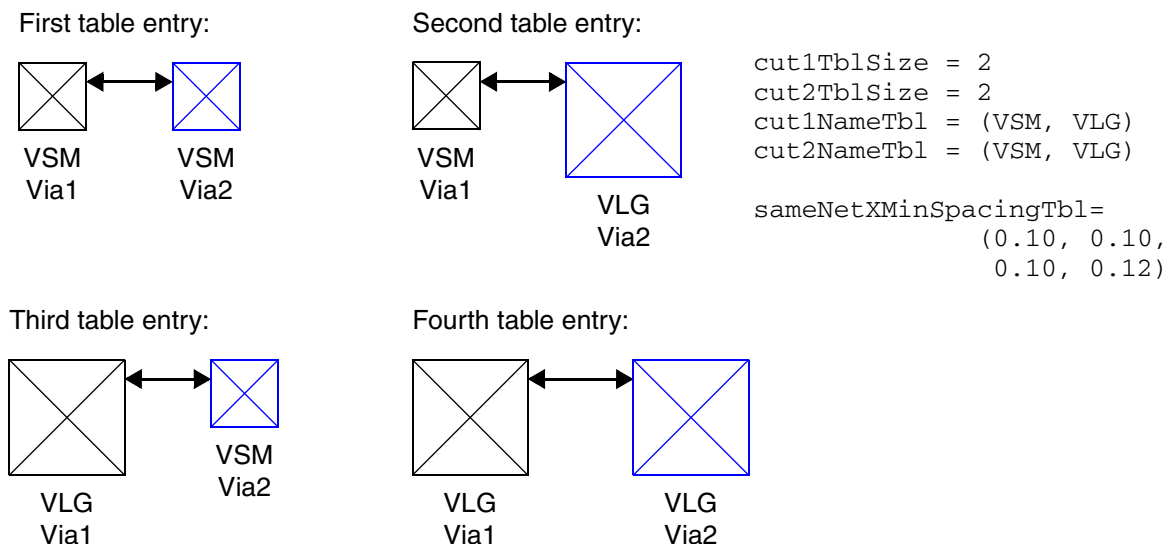
The `layer1` and `layer2` attributes specify the cut layers involved in the rule. The two layers can be different layers or the same layer. The rule specifies the minimum distance between cuts on the two specified layers or between cuts in the same layer. The `cut1TblSize` and `cut2TblSize` attributes specify the number of named cuts in each layer for which the current rule applies. The `cut1NameTbl` and `cut2NameTbl` attributes list the named cuts for which the current rule applies. The total number of entries in each table is the product of the `cut1TblSize` and `cut2TblSize` attributes, which is equal to the total number of combinations between the named cuts in the two lists.

In this example, the rule specifies the minimum spacing between cuts in layer Via1 to cuts in layer Via2. Each table has four entries, each corresponding to a combination of named cuts taken in order from the `cut1NameTbl` and `cut2NameTbl` tables:

- VSM in layer Via1 to VSM in layer Via2
- VSM in layer Via1 to VLG in layer Via2
- VLG in layer Via1 to VSM in layer Via2
- VLG in layer Via1 to VLG in layer Via2

The `orthoSpacingExcludeCornerTbl` attribute is a table of Boolean values that specifies whether the rule is extended beyond the corners of the cut. The `sameNetXMinSpacingTbl` and `diffNetXMinSpacingTbl` attributes are tables that specify the minimum spacing values between the named cuts for same-net and different-net cuts, respectively. For an example, see [Figure 2-31](#).

Figure 2-31 Cut-to-Cut Spacing Specified in a 2-by-2 Table



The following table-based attributes can be used to specify the edge-to-edge X spacing, edge-to-edge Y spacing, corner-to-corner spacing, or center-to-center spacing of two cuts in the same net or in different nets, respectively:

```
sameNetXMinSpacingTbl
sameNetYMinSpacingTbl
sameNetCornerMinSpacingTbl
sameNetCenterMinSpacingTbl
```

```
diffNetXMinSpacingTbl
diffNetYMinSpacingTbl
diffNetCornerMinSpacingTbl
diffNetCenterMinSpacingTbl
```

Similarly, the following table-based attributes can be used to specify the edge-to-edge X spacing, edge-to-edge Y spacing, corner-to-corner spacing, or center-to-center spacing of two cuts in the same wire segment or in different wire segments, respectively:

```
sameSegXMinSpacingTbl
sameSegYMinSpacingTbl
sameSegCornerMinSpacingTbl
sameSegCenterMinSpacingTbl
```

```
diffSegXMinSpacingTbl
diffSegYMinSpacingTbl
diffSegCornerMinSpacingTbl
diffSegCenterMinSpacingTbl
```

If you do not want to specify a minimum spacing value for a particular combination of named cuts, enter `-1.0` in the corresponding position in the minimum spacing table. Entering a negative value prevents the rule from checking that particular combination.

You can define different rules in separate `DesignRule` sections as long as the sections have different cut name tables. For example, you can define cut rules `sameNetXMinSpacing` and `diffNetXMinSpacing` for all cuts in one table:

```
DesignRule {
  layer1 = "VIA1"
  layer2 = "VIA2"
  cut1TblSize = 4
  cut2TblSize = 4
  cut1NameTbl = (VSM, VV, VH, VLG)
  cut2NameTbl = (VSM, VV, VH, VLG)
  sameNetXMinSpacingTbl = (0.10,0.10,0.12,0.10,
                           0.10,0.10,0.15,0.10,
                           0.12,0.15,0.15,0.12,
                           0.10,0.10,0.12,0.12)
  diffNetXMinSpacingTbl = (0.10,0.10,0.12,0.10,
                           0.10,0.10,0.15,0.10,
                           0.12,0.15,0.15,0.12,
                           0.10,0.10,0.12,0.12)
}
```

At the same time, you can define a different cut rule based on wire segment relationships using `diffSegCenterMinSpacing` in another `DesignRule` section. The table size can be different, as in the following example:

```
DesignRule {
  layer1 = "VIA1"
  layer2 = "VIA2"
  cut1TblSize = 2
  cut2TblSize = 2
  cut1NameTbl = (VSM, VLG)
  cut2NameTbl = (VSM, VLG)
  diffSegCenterMinSpacing = (0.09, 0.09,
                             0.09, 0.11)
}
```

You can define cut rules for cuts on the same layer, as in the following example:

```
DesignRule {
  layer1 = "VIA2"
  layer2 = "VIA2"
  cut1TblSize = 2
  cut2TblSize = 2
  cut1NameTbl = (VSM, VLG)
  cut2NameTbl = (VSM, VLG)
  diffSegCenterMinSpacing = (0.10, 0.10,
                             0.10, 0.12)
}
```

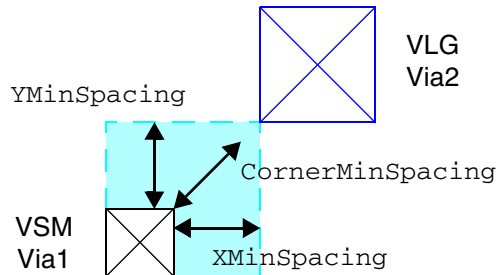
The cut rule minimum spacing values should meet the following requirements when the two cuts are on the same layer:

```
sameNetXMinSpacingTbl[i] >= xMinSpacing >= minSpacing
sameNetYMinSpacingTbl[i] >= yMinSpacing >= minSpacing
diffNetXMinSpacingTbl[i] >= xMinSpacing >= minSpacing
diffNetYMinSpacingTbl[i] >= yMinSpacing >= minSpacing
sameSegXMinSpacingTbl[i] >= max(minCutSpacing, xMinSpacing)
sameSegYMinSpacingTbl[i] >= max(minCutSpacing, yMinSpacing)
diffSegXMinSpacingTbl[i] >= xMinSpacing >= minSpacing
diffSegYMinSpacingTbl[i] >= yMinSpacing >= minSpacing
```

where $i = 0, 1, \dots, \text{table_size} - 1$

`table_size = (cut1TblSize) x (cut2TblSize)`

At a corner area between two cuts, if both `xMinSpacing/yMinSpacing` and `CornerMinSpacing` are defined, by default, the router checks both types of rules, as shown in [Figure 2-32](#).

Figure 2-32 XMinSpacing and YMinSpacing Enforced at Corners

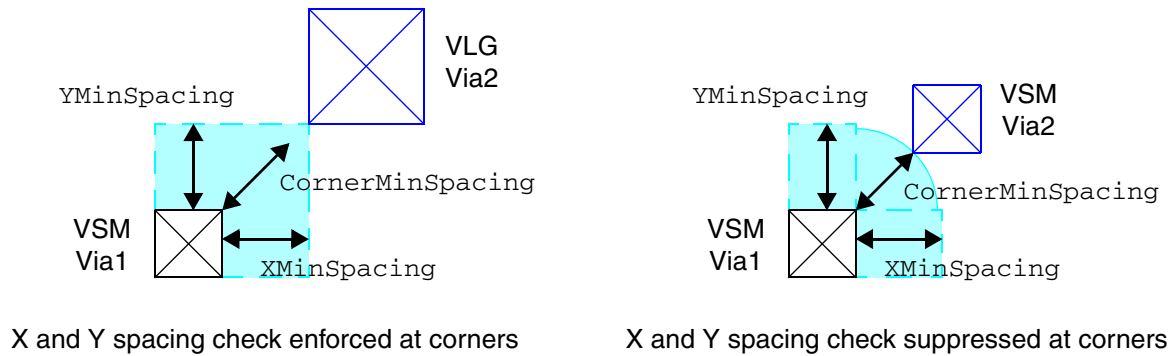
To suppress checking of `XMinSpacing`/`YMinSpacing` at corners and allow only `CornerMinSpacing` to be checked, set the `orthoSpacingExcludeCornerTbl` attribute to 1 for the corresponding check.

In the following example, the router checks both `XMinSpacing`/`YMinSpacing` and `CornerMinSpacing` between cuts VSM and VLG in corner areas for cuts belonging to the same net, whereas it checks only `CornerMinSpacing` between two VSM cuts or between two VLG cuts belonging to the same net. This rule checking is shown in [Figure 2-33](#).

```
DesignRule {
  layer1 = "VIA1"
  layer2 = "VIA2"
  cut1TblSize = 2
  cut2TblSize = 2
  cut1NameTbl = (VSM, VLG)
  cut2NameTbl = (VSM, VLG)
  orthoSpacingExcludeCornerTbl = (1, 0,
                                0, 1)

  sameNetXMinSpacingTbl= (0.10, 0.10
                          0.10, 0.12)
  sameNetYMinSpacingTbl= (0.10, 0.10
                          0.10, 0.12)
  sameNetCornerMinSpacingTbl= (0.14, 0.14)
                              (0.14, 0.17)

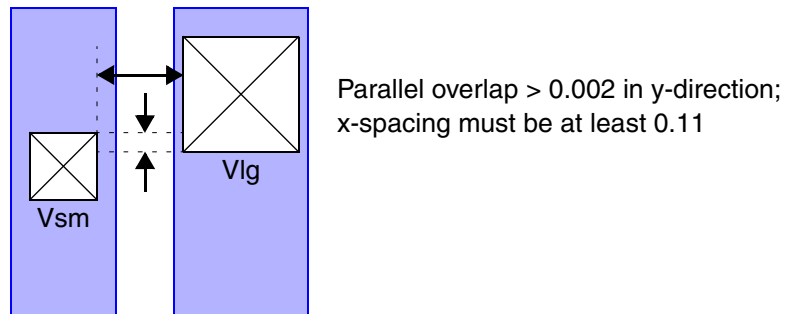
  ...
}
```


Figure 2-33 *XMinSpacing and YMinSpacing Suppressed at Corners*

If the cut edge-to-edge or diagonal spacing depends on the amount of parallel overlap between the two cuts, use syntax similar to the following example:

```
DesignRule {
    layer1                      = "VIA1"
    layer2                      = "VIA2"
    cut1TblSize                 = 2
    cut1NameTbl                 = (Vsm, Vlg)
    cut2TblSize                 = 2
    cut2NameTbl                 = (Vsm, Vlg)
    minSpacingYParallelLengthThresholdTbl = (0.005, 0.002,
                                           0.002, 0)
    diffNetXMinSpacingTbl       = (0.11, 0.11, # for X-spacing
                                   0.11, 0.13)
    minSpacingXParallelLengthThresholdTbl = (0.005, 0.002,
                                           0.002, 0)
    diffNetYMinSpacingTbl       = (0.11, 0.11, # for Y-spacing
                                   0.11, 0.13)
}
```

The `minSpacingYParallelLengthThresholdTbl` attribute specifies the parallel overlap threshold, measured in the y-direction between the edges of two cuts belonging to different nets. If the threshold is positive and the parallel overlap at least this threshold, then the x-spacing rule applies between the edges of the cuts. The x-spacing rule is a table of values corresponding to each combination of cuts, using the cuts listed by the `cut1` and `cut2` name attributes. See the parallel overlap and edge-to-edge spacing measurements in [Figure 2-34](#).

Figure 2-34 Parallel Overlap Test in Y-Direction for X-Spacing Rule With a Positive Threshold

Similarly, the `minSpacingXParallelLengthThresholdTbl` attribute specifies the parallel overlap threshold, measured in the x-direction between the cut edges. If the threshold is positive and the parallel overlap is at least this threshold, then the edge-to-edge y-spacing rule applies between the edges of the cuts.

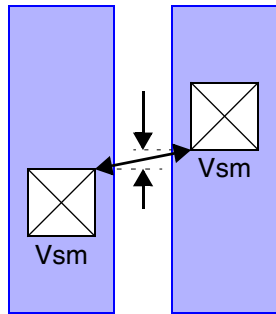
If the parallel length threshold is a negative number, the minimum spacing rule applies if the parallel overlap is greater than the specified negative threshold but less than zero, or in other words, when the two cuts are close to overlapping but do not quite overlap.

To apply the minimum spacing rule as a diagonal measure, use the `minSpacingDiagonalCheckTbl` attribute as shown in the following example:

```
DesignRule {
    layer1                = "VIA1"
    layer2                = "VIA2"
    cut1TblSize           = 2
    cut1NameTbl           = (Vsm, Vlg)
    cut2TblSize           = 2
    cut2NameTbl           = (Vsm, Vlg)
    minSpacingYParallelLengthThresholdTbl = (-0.007, 0,
                                           0, 0)
    diffNetXMinSpacingTbl = (0.12, 0.14,
                           0.14, 0.15)
    minSpacingXParallelLengthThresholdTbl = (-0.007, 0,
                                           0, 0)
    diffNetYMinSpacingTbl = (0.12, 0.14,
                           0.14, 0.15)
    minSpacingDiagonalCheckTbl = (1, 0,
                                 0, 1)
}
```

The `minSpacingDiagonalCheckTbl` attribute, when set to 1, causes the minimum spacing to be measured diagonally. See the parallel overlap and diagonal spacing measurements in [Figure 2-35](#).

Figure 2-35 Parallel Overlap Test in Y-Direction with a Negative Threshold



Parallel overlap > -0.007 and < 0 in y-direction;
diagonal spacing must be at least 0.12

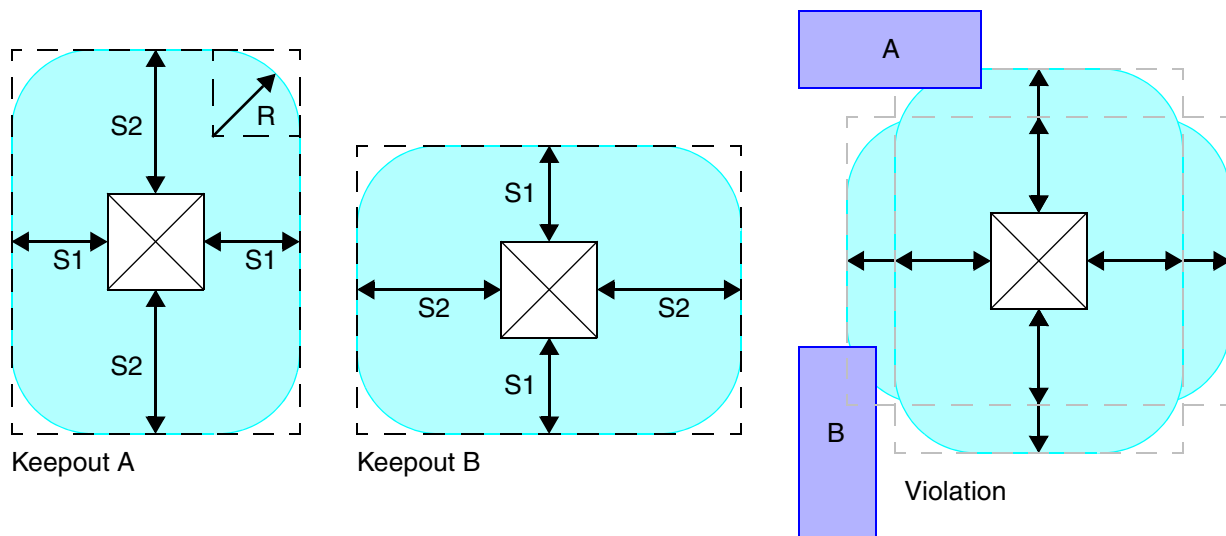
Asymmetric Via-to-Metal Spacing Rule

The asymmetric via-to-metal spacing rule specifies the minimum distance from a cut to an unconnected metal using different spacing values in the x-direction and y-direction and using corner-rounding of the exclusion area. This is the syntax for the rule:

```
DesignRule {
  layer1 = "ViaX"
  layer2 = "MetalX"
  diffNetKeepoutMinWidth = S1
  diffNetKeepoutMinLength = S2
  diffNetKeepoutMinRadius = R
}
```

The rule defines two keepout regions around each via, labeled Keepout A and Keepout B in [Figure 2-36](#). A violation occurs when unrelated metal, not connected to the via, overlaps both Keepout A and Keepout B. A single piece or two different pieces of unrelated metal can cause a violation. In this example, metal A overlaps Keepout A and metal B overlaps Keepout B, triggering a violation.

Figure 2-36 Asymmetric Via-to-Metal Spacing Rule



Each keepout region extends away from the edges of the via by the length and width attributes, taken separately in the x-direction and y-direction. The shorter extension S1 applies to the x-direction for Keepout A and applies to the y-direction for Keepout B. Each rectangular keepout region is reduced in the corners by the curvature of radius R. Note that the corner-rounding radius R is specified separately from S1 and S2, so the center point of the quarter-circle used for rounding the corner is not necessarily aligned to any via edge.

Fat Metal Spacing Rules

The fat metal spacing rules are described in the following sections:

- [Fat Metal Spacing Table Rule](#)
- [Fat Metal Orthogonal Spacing Rule](#)
- [Fat Metal Parallel Length](#)
- [Fat Metal Extension Spacing Rule](#)
- [Neighboring Layer Fat Metal Extension Spacing Rule](#)

Fat Metal Spacing Table Rule

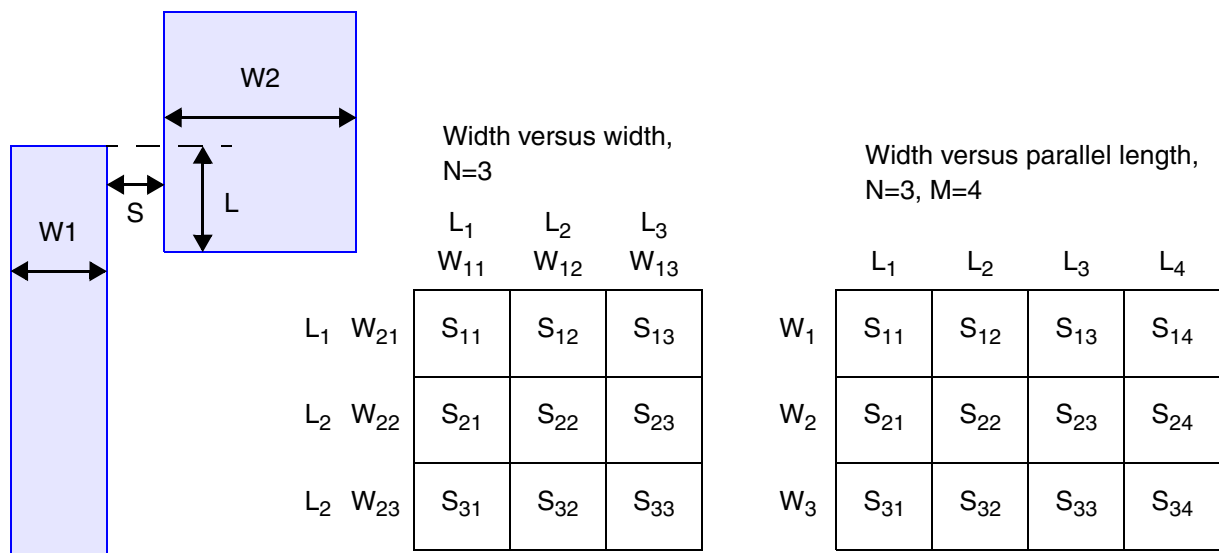
The fat metal spacing table rule defines the minimum spacing requirement between two parallel metal segments, based on the width of the metal segments and the parallel length between them. This rule applies only to fat metal, not minimum-width metal.

There are two forms of this rule:

- In the width-versus-width form, a table of N by N values specifies the minimum spacing between two segments for each combination of two widths. An additional list of N parallel length thresholds causes a downgrade of the spacing requirement if the parallel overlap length of the two segments is less than the threshold. One parallel length threshold is associated with each width. You specify N widths and N parallel length thresholds.
- In the width-versus-parallel-length form, a table of N by M values specifies the minimum spacing between two segments for each combination of width and parallel overlap length. In this case, the rule considers only the width of the wider of the two segments. You specify N widths and M parallel length thresholds.

Figure 2-37 shows how the table of spacing values is organized in the two forms of the rule. In the first table, using the width-versus-width form of the rule, $N=3$. The nine spacing values correspond to each combination of widths for the two metal segments. In the second table, using the width-versus-parallel-length form of the rule, $N=3$ and $M=4$. The 12 table entries correspond to each combination of width and parallel overlap length.

Figure 2-37 Width Versus Width and Width Versus Parallel Length Forms



Width Versus Width

This is the syntax of the width-versus-width form of the fat metal spacing table rule:

```
Layer "MetalX" {
    fatTblDimension      = N
    fatTblThreshold      = (W1,  W2,  ..., WN)
    fatTblParallelLength = (L1,  L2,  ..., LN)
    fatTblSpacing        = (S11, S12, ..., S1N,
                          S11, S12, ..., S1N,
                          ...  ...  ..., ...
                          SN1, SN2, ..., SNN, )
}
```

The `fatTblDimension` attribute specifies the size N of the two-dimensional fat table.

The `fatTblThreshold` attribute is a list of N values specifying the width thresholds for the rule.

The `fatTblParallelLength` attribute is a list of N values specifying the parallel length thresholds for the rule. If the parallel overlap length of the two segments is less than or equal to the threshold corresponding width, the minimum spacing requirement is downgraded to a lower spacing value in the spacing table.

The `fatTblSpacing` attribute is a list of N by N values specifying the minimum spacing requirement for each combination of width range.

For example, assume that the fat metal spacing rule requires the following:

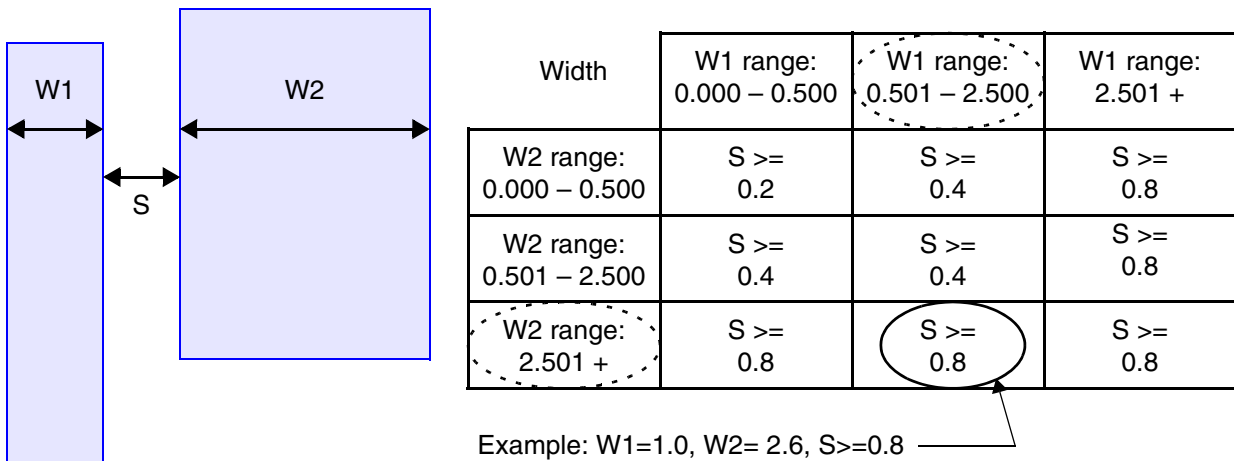
- A minimum spacing of 0.2 between any two parallel metal segments.
- A minimum spacing of 0.4 between two parallel metal segments when either one is greater than 0.5 and the parallel length is greater than 1.5.
- A minimum spacing of 0.8 between two parallel metal segments when either one is greater than 2.5 and the parallel length is greater than 2.5.

To specify this in the technology file, enter

```
Layer "M1" {
    fatTblDimension      = 3
    fatTblThreshold      = (0.0, 0.501, 2.501)
    fatTblParallelLength = (0.0, 1.501, 2.501)
    fatTblSpacing        = (0.2, 0.4, 0.8,
                          0.4, 0.4, 0.8,
                          0.8, 0.8, 0.8)
}
```

The values in the $N \times N$ table represent the minimum spacing values for different combinations of two fat metal widths. The width ranges are established by the `fatTblThreshold` values. In the foregoing example, the table entries represent the minimum spacing between metal wires as indicated in [Figure 2-38](#).

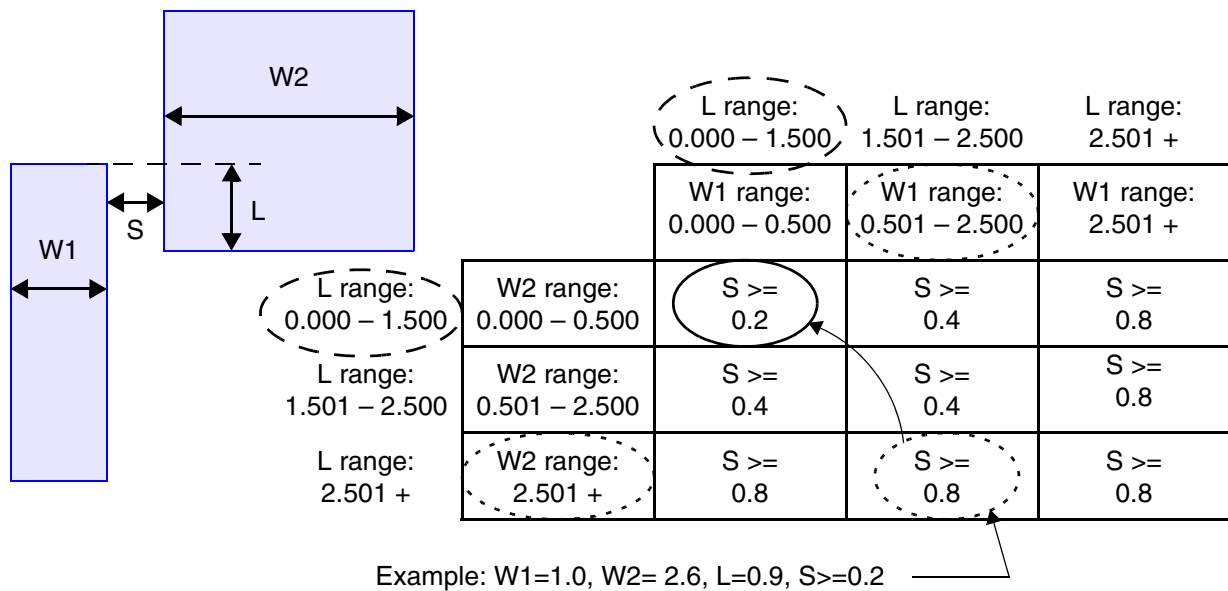
Figure 2-38 Fat Metal Minimum Spacing Table, Width Versus Width



The width values should be specified in the distance measurement precision of the technology. For example, if the `Technology` section specifies the `unitLengthName` as `micron` and `lengthPrecision` as `1000`, then the thresholds should be written with a precision of 0.001 micron to specify contiguous ranges, as in the foregoing example.

If the parallel overlap length of the two metal wires is small enough, the minimum spacing requirement is reduced or downgraded as specified by `fatTblParallelLength`. See [Figure 2-39](#) for an example. Because the parallel overlap length L is small, the minimum spacing requirement is downgraded to the designated lesser entry in the table.

Figure 2-39 Parallel Overlap Length Downgrade of Minimum Spacing



You can optionally restrict the amount of the downgrade to no more than one row and one column in the table by setting `fatTblParallelLength` to 1. In that case, in [Figure 2-39](#), the downgrade would be to a minimum spacing of 0.4 rather than 0.2. Otherwise, there is no restriction on the amount of downgrade that can result from having a smaller parallel overlap.

Width Versus Parallel Length

This is the general syntax of the width-versus-parallel-length form of the fat metal spacing table rule:

```

Layer "MetalX" {
    fatTblDimension           = N
    fatTblThreshold           = (W1,  W2,  ..., WN)
    fatTblParallelLengthDimension = M
    fatTblParallelLength      = (L1,  L2,  ..., LM)
    fatTblSpacing             = (S11,  S12,  ..., S1M,
                                S11,  S12,  ..., S1M,
                                ...,  ...,  ...,  ...,
                                SN1,  SN2,  ..., SNM)
}

```

The `fatTblDimension` attribute specifies the number of width thresholds N in the two-dimensional fat table.

The `fatTblThreshold` attribute is the list of N values specifying the width thresholds for the rule.

The `fatTblParallelLengthDimension` attribute specifies the number of parallel length thresholds M in the two-dimensional fat table.

The `fatTblParallelLength` attribute is the list of N values specifying the parallel length thresholds for the rule.

The `fatTblSpacing` attribute is a list of N by M values specifying the minimum spacing requirement for each combination of segment width and parallel overlap. The rule considers only the width of the wider of the two segments.

For example, consider the following rule:

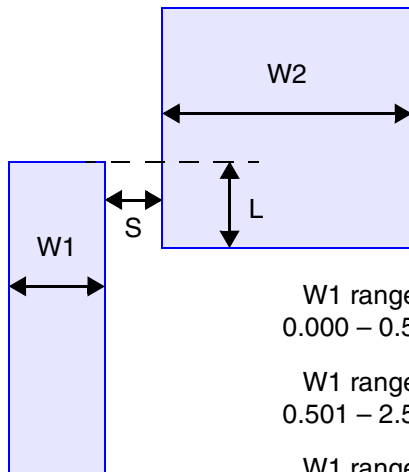
```

Layer "M1" {
    fatTblDimension           = 3
    fatTblThreshold           = (0.0, 0.501, 2.501)
    fatTblParallelLengthDimension = 4
    fatTblParallelLength      = (0.001, 0.081, 1.501, 2.501)
    fatTblSpacing             = (0.2,   0.3,   0.5,   0.8,
                                0.4,   0.5,   0.7,   0.8,
                                0.8,   0.8,   0.8,   0.8)
}

```


This example specifies the minimum spacing requirement for each combination of wider-metal width and parallel overlap length, as indicated in [Figure 2-40](#).

Figure 2-40 Fat Metal Minimum Spacing Table, Width Versus Parallel Length



	L range: 0.001 – 0.080	L range: 0.081 – 1.500	L range: 1.501 – 2.500	L range: 2.501 +
W1 range: 0.000 – 0.500	S ≥ 0.2	S ≥ 0.3	S ≥ 0.5	S ≥ 0.8
W1 range: 0.501 – 2.500	S ≥ 0.4	S ≥ 0.5	S ≥ 0.7	S ≥ 0.8
W1 range: 2.501 +	S ≥ 0.8	S ≥ 0.8	S ≥ 0.8	S ≥ 0.8

In the `fatTblParallelLength` list, a value of 0.0 causes the minimum spacing requirement to apply, irrespective of the parallel overlap length, including negative overlaps.

If the spacing rule needs to be applied for a parallel overlap of exactly 0.0 but not negative overlaps, you can combine the rule with the `orthoSpacingExcludeCorner` attribute, as shown in the following example.

```
fatTblDimension           = 5
fatTblThreshold           = (0.000, 0.050, 0.140, 0.201, 0.221)
fatTblParallelLengthDimension = 4
fatTblParallelLength      = (0.000, 0.000, 0.000, 0.500)
orthoSpacingExcludeCorner = 1
fatTblSpacing             = ( 4-by-5 table )
```

Fat Metal Orthogonal Spacing Rule

The fat metal orthogonal spacing rule extends the fat metal spacing rule described in the previous section to handle the case of different spacing values in the x-direction and y-direction, and to specify the application of the rule at metal corners. The following example demonstrates the usage of this rule.

```
Layer "M1" {
  fatTblXDimension = 3
  fatTblYDimension = 3
  fatTblXThreshold = (0.0, 0.501, 2.501)
```

```

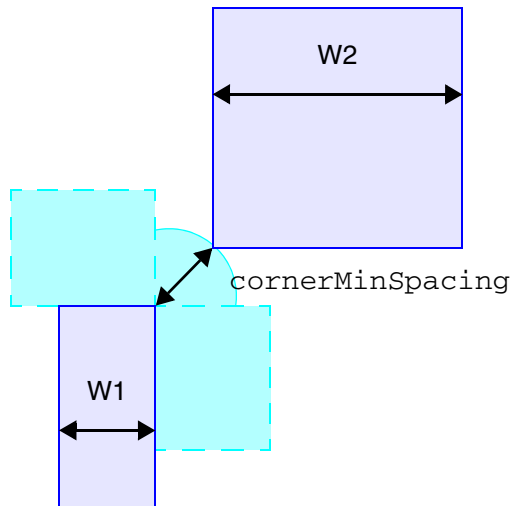
fatTblYThreshold = (0.0, 1.001, 5.001)
fatTblXParallelLength = (0.0, 1.501, 2.501)
fatTblYParallelLength = (0.0, 1.501, 2.501)
fatTblXMinSpacing = (0.2, 0.4, 0.8,
                    0.4, 0.4, 0.8,
                    0.8, 0.8, 0.8)
fatTblYMinSpacing = (0.3, 0.6, 1.2,
                    0.6, 0.6, 1.2,
                    1.2, 1.2, 1.2)
orthoSpacingExcludeCorner = 1
}

```

The spacing attributes are specified separately for the x-direction and y-direction, using different values and possibly different table sizes.

The `orthoSpacingExcludeCorner` attribute specifies whether to extend the spacing check to the corner area for the given layer. The default setting is 0, which enables extension of the spacing check to the corner areas, whereas 1 disables extension of the spacing check to the corner areas. The `cornerMinSpacing` attribute determines the minimum spacing requirement, as illustrated in [Figure 2-41](#). A single, not table-based, corner spacing value applies to the layer.

Figure 2-41 Fat Metal Corner Spacing Check



You can use either the simpler fat table syntax described in the previous section or the orthogonal X-Y syntax just described, but not both within the same technology file.

Note:

If you use the fat metal orthogonal spacing rule and the fat metal extension spacing rule ([“Fat Metal Extension Spacing Rule” on page 2-50](#)) in the same technology file, replace the fat metal extension spacing attribute `fatTblExtensionRange` with the two attributes

`fatTblXExtensionRange` and `fatTblYExtensionRange`, to ensure correct operation of both rules.

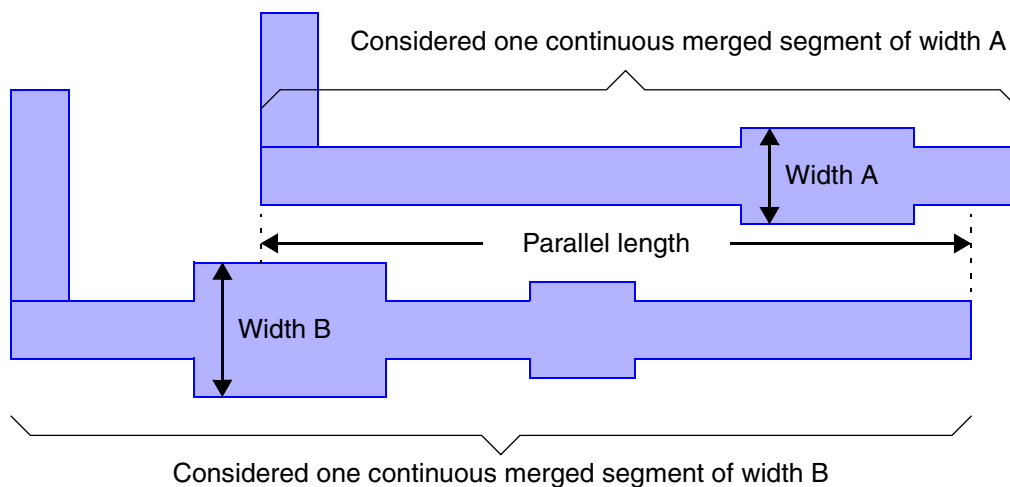
Note:

If you use the fat metal orthogonal spacing rule and the fat metal enclosed minimum area rule (“[Minimum Enclosed Area Rule](#)” on page 2-9) in the same technology file, replace the enclosed minimum area attributes `fatTblThreshold` and `fatTblDimension` with the two attributes `fatTblMinEnclosedAreaDimension` and `fatTblMinEnclosedWidthThreshold`, to ensure correct operation of both rules.

Fat Metal Parallel Length

If two or more fat metal segments of different widths are connected in a straight line, the router considers the segments as a single long segment having a constant width for calculating the parallel lengths of nearby wires. The width of the whole “merged” fat wire segment is considered the same as the widest part. In other words, the spacing between parallel fat wires is based on the widest part of each long segment, as illustrated by the example in [Figure 2-42](#). The minimum spacing between a merged wire and adjacent parallel wires depends on the width and length of the merged wire.

Figure 2-42 Parallel Length Determination



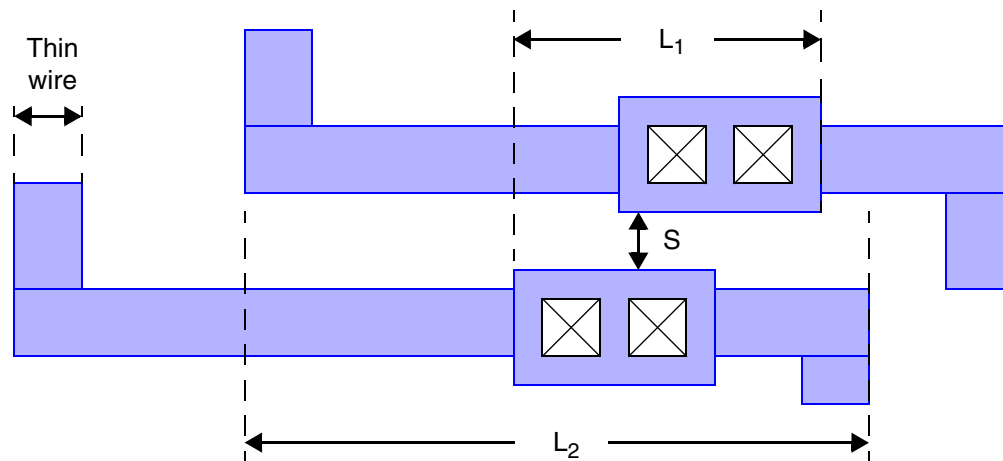
You can control the merging behavior by setting the `parallelLengthMode` attribute in the `Technology` section of the technology file. By default (`parallelLengthMode = 0`), IC Compiler merges the wire segments with fat neighboring shapes only. To merge the wire

segments with all neighboring shapes, set the `parallelLengthMode` attribute to 1. For example,

```
Technology {
    parallelLengthMode = 1
}
```

Figure 2-43 demonstrates the effects of this setting. Each of the two parallel thin wires has a fat via metal enclosure. When the attribute `ParallelLengthMode` is 0 (the default), the parallel length L_1 is based on the extent of the fat enclosures only. When `ParallelLengthMode` is set to 1, the longer parallel length L_2 is based on the full extent of the merged wire.

Figure 2-43 Parallel Length Rule Example

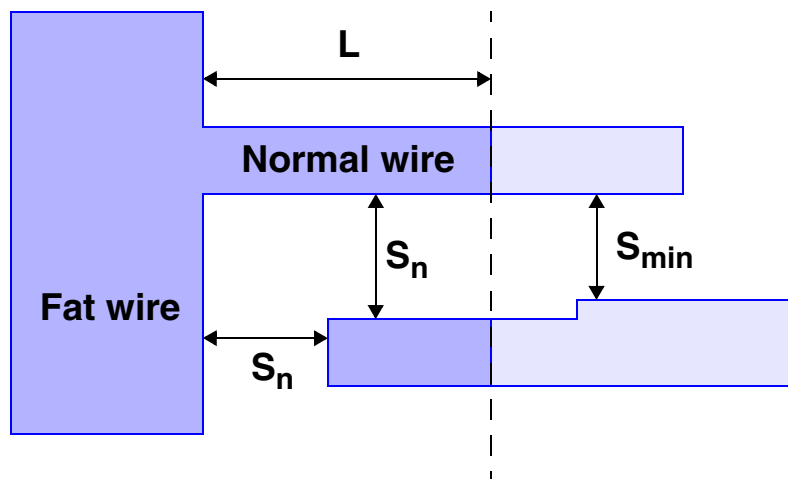


The minimum spacing between these wires depends on the parallel overlap length between them, as defined by the `FatTblParallelLength` and `fatTblSpacing` attributes described in the previous section, “Fat Metal Spacing Table Rule” on page 2-43. In Figure 2-43, the parallel length is either L_1 or L_2 , depending on the `ParallelLengthMode` setting. Setting `ParallelLengthMode` to 1 results in the use of the longer parallel length L_2 , which in turn can result in a larger minimum spacing requirement between the adjacent wires.

Fat Metal Extension Spacing Rule

The fat metal extension spacing rule determines when the fat metal spacing rules apply to normal wires that extend from a fat wire. Figure 2-44 shows that the portion of the extension wire within the extension threshold, L , uses the fat metal spacing rules, S_n , while the portion of the extension wire beyond the extension threshold uses normal wire spacing rules, S_{min} . You specify the extension thresholds by using the `fatTblExtensionRange` attribute.

Figure 2-44 Fat Metal Wire Having Extension Wire With Normal Width



For example,

```
Layer "M1" {
  fatTblDimension = 3
  fatTblThreshold = (0.0, 0.501, 2.501)
  fatTblParallelLength = (0.0, 1.501, 2.501)
  fatTblExtensionRange = (0.0, 0.0, 0.8)
  fatTblSpacing = (0.2, 0.4, 0.8,
                  0.4, 0.4, 0.8,
                  0.8, 0.8, 0.8)
```

You can control how the tool treats this rule by setting the `fatWireExtensionMode` attribute in the `Technology` section, as shown in [Table 2-1](#).

Table 2-1 `fatWireExtensionMode` Attribute Values

Mode value	Description
0 (the default)	The fat spacing check is performed only on extension wires connected to the fat wire and within the extension range from the fat wire edges and corners. The spacing is checked between any two wires. Projection length is not checked. See Figure 2-45 on page 2-53 .
1	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Spacing between both overlapped and non-overlapped wire pairs is checked. Projection lengths are also checked. See Figure 2-46 on page 2-53 .

Table 2-1 *fatWireExtensionMode Attribute Values (Continued)*

Mode value	Description
2	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Only the spacing between non-overlapped wire pairs is checked. Projection lengths are not checked. See Figure 2-47 on page 2-54 .
3	The fat spacing check is performed on both connected and unconnected wires within the extension range from the fat wire edges, not including fat wire corners. Only the spacing between overlapped wire pairs is checked. Projection lengths of these wires are also checked. See Figure 2-48 on page 2-54 .
4	The fat spacing check is performed only on connected wires within the extension range from the fat wire edges, not including fat wire corners. The spacing from these wires to all other wires is checked. Projection lengths are not checked. See Figure 2-49 on page 2-55 .

In the following figures, the fat metal spacing applies to the shaded wire segments, based on the value of the `fatWireExtensionMode` attribute. These are the spacing values checked by the rule (which depend on the `fatWireExtensionMode` attribute setting):

- S_n : Fat spacing of metal n layer; the minimum spacing between a normal-width metal line and another metal edge when located within a specified distance (the “extension range”) of a fat wire edge.
- S_o : Fat spacing between overlapped metals; the minimum spacing between overlapping normal-width metal lines within a the extension range of a fat wire edge.
- S_{min} : The default minimum spacing between a normal-width metal lines, which is used outside of the extension range.
- L_n : Projection length, which is the cumulative length of the projections of multiple adjacent wire segments onto the fat wire edge.
- L_o : Overlapped projection length, which is the cumulative length of the overlapping adjacent wire segments parallel the fat wire edge.

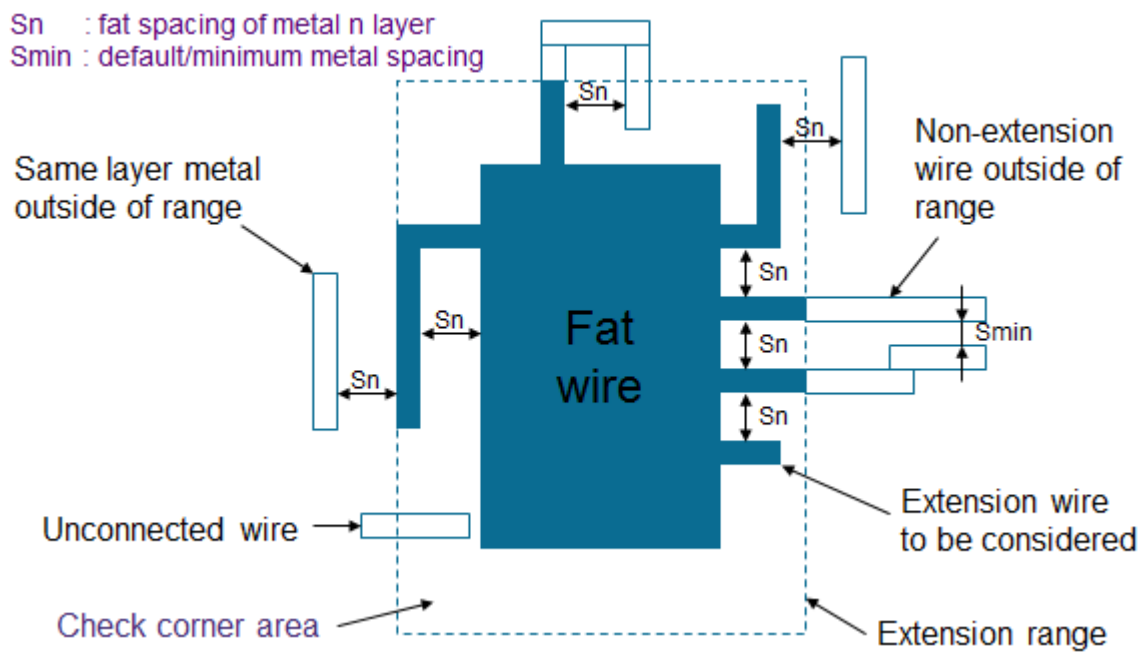
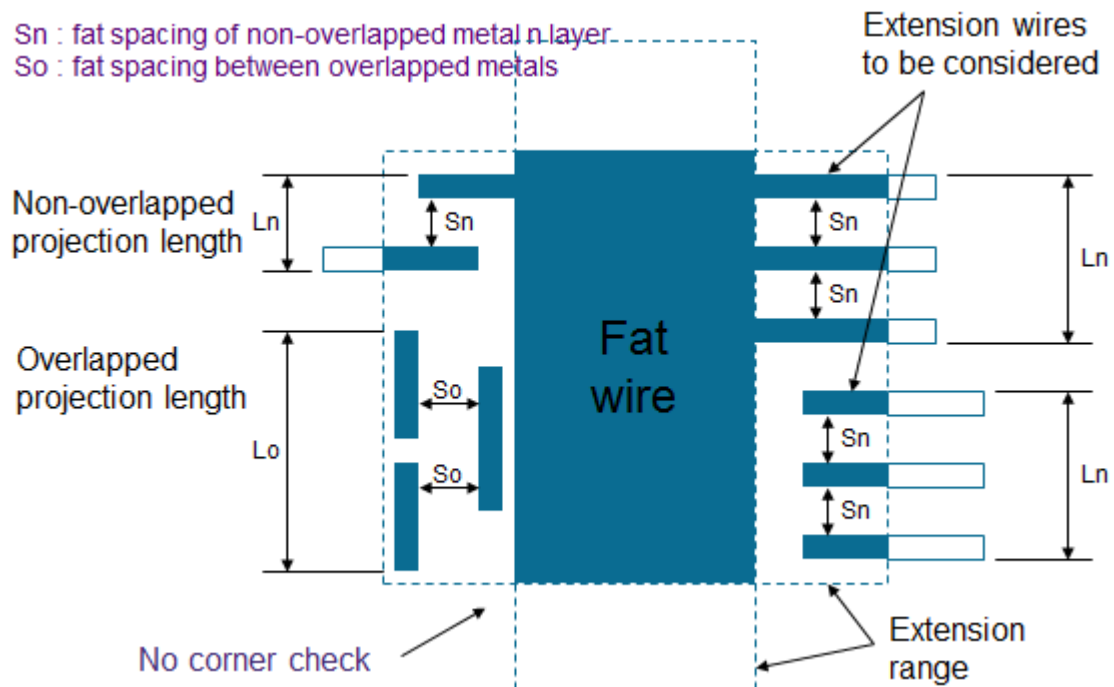
Figure 2-45 *fatWireExtensionMode = 0*Figure 2-46 *fatWireExtensionMode = 1*

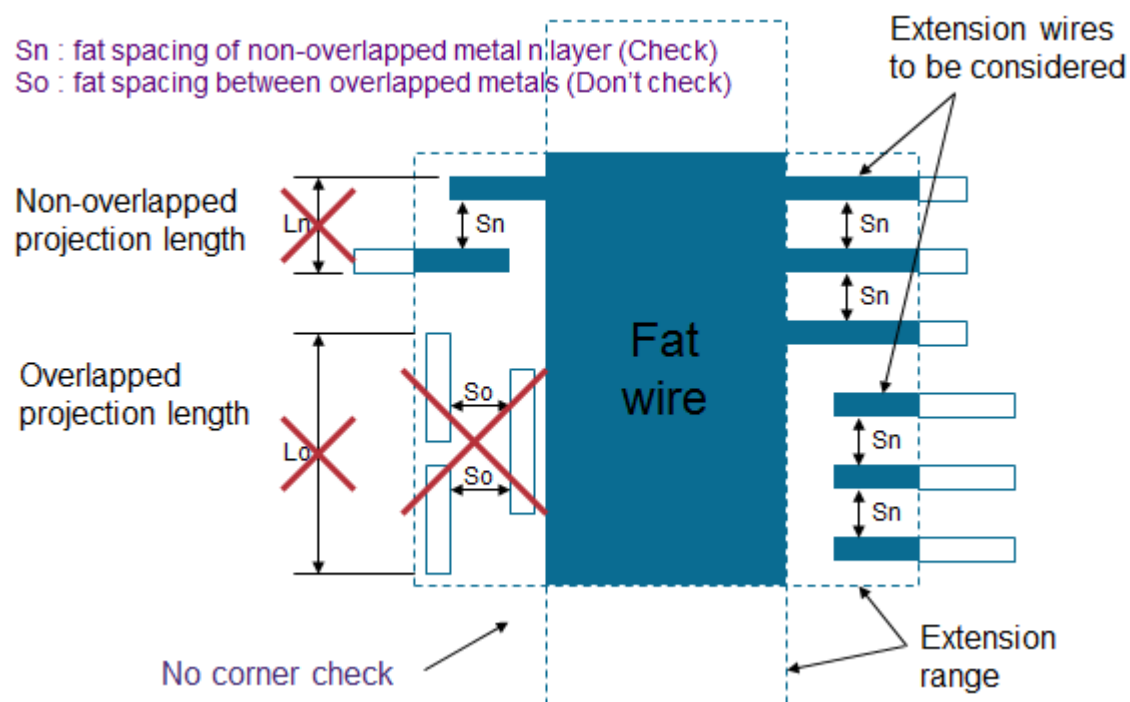
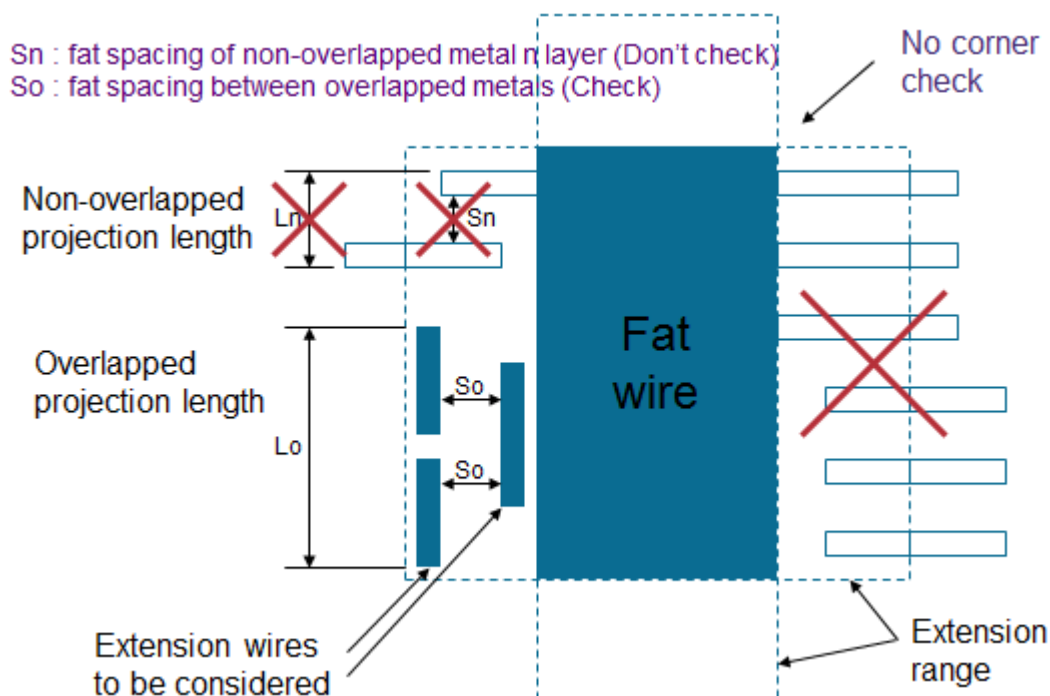
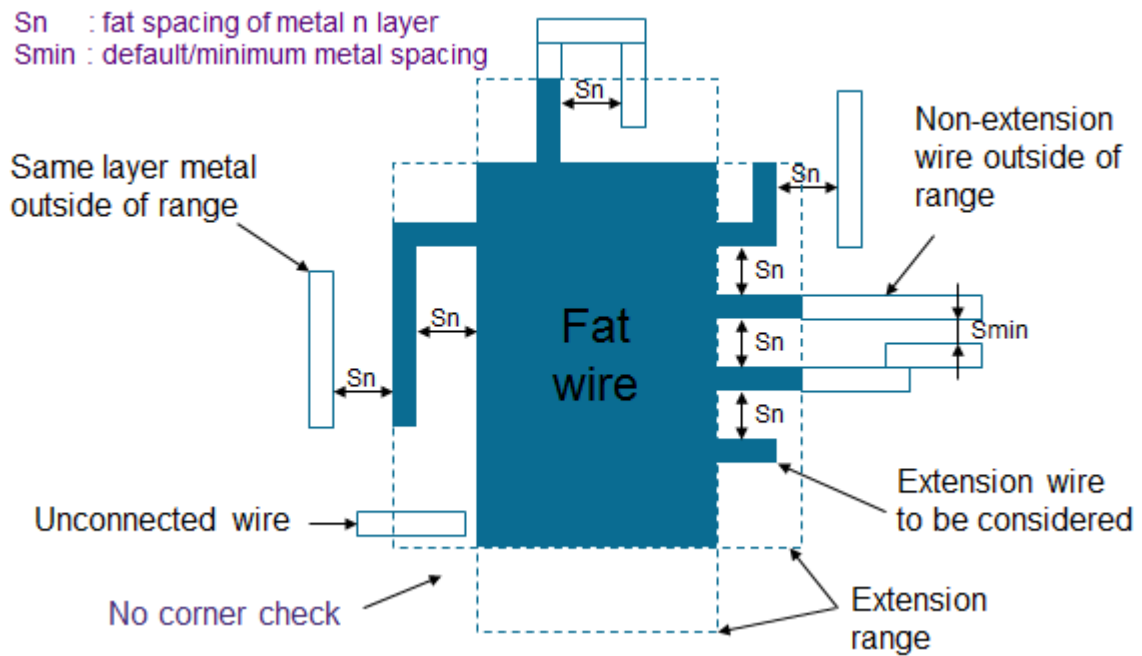
Figure 2-47 *fatWireExtensionMode = 2*Figure 2-48 *fatWireExtensionMode = 3*

Figure 2-49 *fatWireExtensionMode = 4*

Neighboring Layer Fat Metal Extension Spacing Rule

You can specify a fat wire threshold and extension range. A wire on another layer that is within the defined extension range of the fat wire must meet the recommended spacing.

Note:

This rule is supported only by the classic router, not Zroute.

Use the following detail route options to specify the neighboring layer fat extension spacing rule:

```

neighboringLayerFatThreshold
neighboringLayerFatExtensionRange
neighboringLayerM1RecommendedSpacing
neighboringLayerM2RecommendedSpacing
...
neighboringLayerM12RecommendedSpacing
  
```

You set these options with the `set_droute_options` command. These option settings are stored with the cell.

The syntax is

```
set_droute_options -name neighboringLayerFatThreshold -value 0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer fat threshold (applied to all layers)

set_droute_options -name neighboringLayerFatExtensionRange -value 0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer fat extension range (applied to all
# layers)

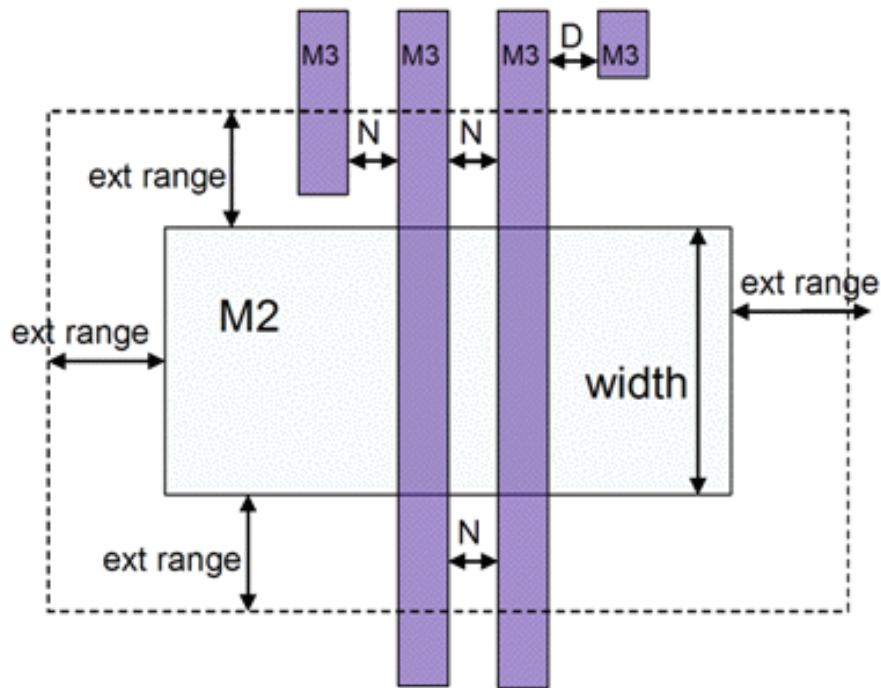
set_droute_options -name neighboringLayerM1RecommendedSpacing -value
0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer recommended spacing for M1

set_droute_options -name neighboringLayerM2RecommendedSpacing -value
0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer recommended spacing for M2

...

set_droute_options -name neighboringLayerM12RecommendedSpacing -value
0.000
# range [0.000,100.000], default=0.000, stored in cell;
# N: neighboring layer recommended spacing for M12
```

For example, as shown in [Figure 2-50](#), if M2 width is greater than or equal to `neighboringLayerFatThreshold` and M3 is within `neighboringLayerFatExtensionRange`, the recommended spacing (N) applies; otherwise, the minimum spacing (D) applies.

Figure 2-50 Neighboring Layer Fat Metal Extension Spacing Rule

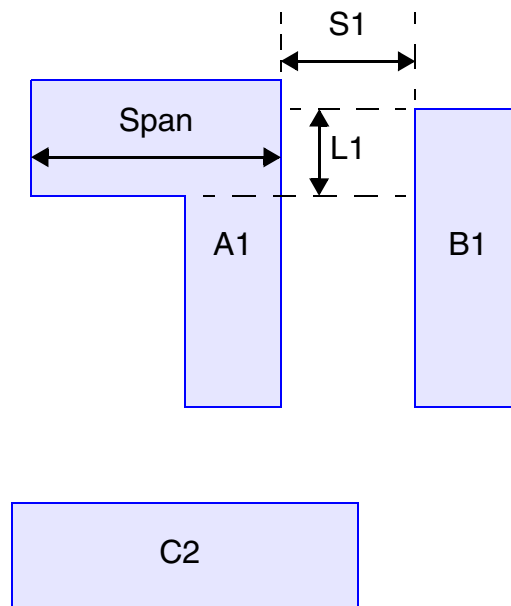
Metal Span Spacing Rule

The metal span spacing rule is similar to the fat metal spacing rule described in the section [“Fat Metal Spacing Table Rule” on page 2-43](#). Like the fat metal spacing rule, the metal span spacing rule defines the minimum space between two parallel metal segments, subject to the parallel length between them. However, the minimum spacing value is based on the span rather than the width of the metal segments.

The span of a metal segment, unlike its width, is always measured perpendicular to the edges of the two metal segments whose spacing is under consideration. The span can be either the smaller or larger dimension of a rectangular metal segment, depending on its shape and its location with respect to the nearby metal.

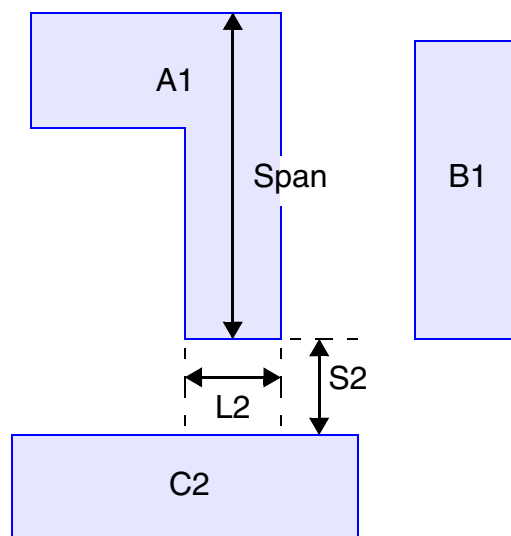
In [Figure 2-51](#), when the metal span spacing rule considers the horizontal spacing $S1$ between metal A1 and metal B1, it considers the horizontal span across A1, perpendicular to the edges separating A1 and B1. It also considers the parallel length $L1$ where the wider top portion of metal A1 overlaps the vertical run of B1.

Figure 2-51 Metal Span Attributes for a Horizontal Spacing Check



In [Figure 2-52](#), when the metal span spacing rule considers the vertical spacing S2 between metal A1 and metal C1, it considers the vertical span across A1 perpendicular to the edges separating A1 and C2. It also considers the parallel length L2 where the width of metal A1 overlaps the horizontal run of C2.

Figure 2-52 Metal Span Attributes for a Vertical Spacing Check



The metal span spacing rule is table-based, allowing you to specify different spacing values for different ranges of span values. The following example demonstrates usage of the rule:

```
Layer "M1" {
  spanTblDimension      = 4
  spanTblThreshold      = (0.000,0.080,0.160,0.220)
  spanTblParallelLength = (0.000,0.110,0.110,0.110)
  spanTblMinSpacing     = (0.060,0.068,0.070,0.062)
```

The `spanTblDimension` attribute specifies the size of the table.

The `spanTblThreshold` attribute specifies the span thresholds. If the measured span is greater than or equal to the span threshold, the corresponding span index value is used to apply the spacing check. If the measured span is less than the threshold, that particular span index does not apply.

The `spanTblParallelLength` specifies the parallel length (L) requirement for the rule corresponding to each entry in the `spanTblThreshold` attribute. The parallel length must be at least the specified value for the corresponding span index to apply.

The `spanTblMinSpacing` table specifies the minimum spacing requirement for each range of span. It can be either a 1-dimensional or 2-dimensional table. A 1-dimensional table, such as the foregoing example, performs a spacing check by considering the span of one metal shape while ignoring the span of the nearby metal shape.

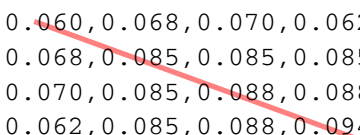
A 2-dimensional table performs a spacing check by considering the span of one metal shape while also consider the span of the nearby metal shape. For example, consider the following 2-dimensional minimum spacing table:

```
Layer "M1" {
  spanTblDimension      = 4
  spanTblThreshold      = (0.000,0.080,0.160,0.220)
  spanTblParallelLength = (0.000,0.110,0.110,0.110)
  spanTblMinSpacing     = (0.060,0.068,0.070,0.062,
                          0.068,0.085,0.085,0.085,
                          0.070,0.085,0.088,0.088,
                          0.062,0.085,0.088,0.094)
```

If one metal shape has a span of 0.080 and a nearby metal shape has a span of 0.160, and if the parallel length between them is at least 0.110, then the minimum spacing between the shapes must be at least 0.085.

The contents of a 2-dimensional table must be symmetrical with respect to a diagonal through the square matrix:

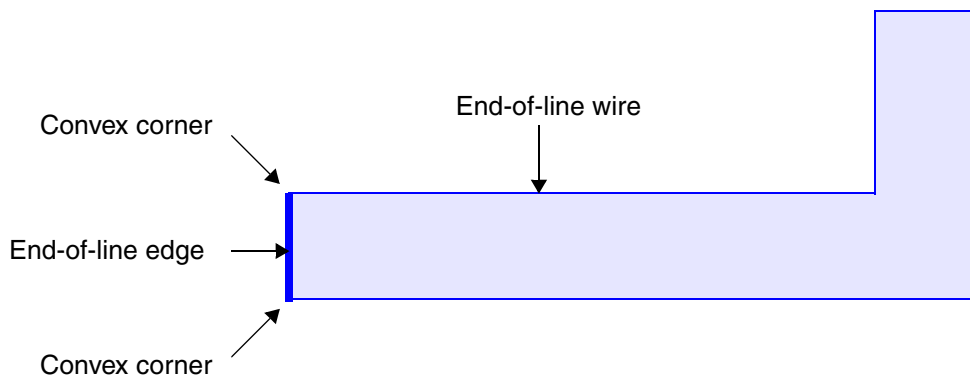
```
spanTblMinSpacing      = (0.060,0.068,0.070,0.062,
                          0.068,0.085,0.085,0.085,
                          0.070,0.085,0.088,0.088,
                          0.062,0.085,0.088,0.094)
```



End-of-Line Spacing Rules

The end-of-line spacing rules define the minimum spacing between an end-of-line edge and an adjacent wire. An end-of-line edge is an edge that connects two convex corners, having a length no more than a width threshold Q . A wire that has at least one end-of-line edge is an end-of-line wire. See [Figure 2-53](#).

Figure 2-53 End-of-Line Edge and End-of-Line Wire



The end-of-line spacing rules are described in the following sections:

- [End-of-Line to End-of-Line Spacing Rule](#)
- [One-Neighbor End-to-End Table-Based Spacing Rule](#)
- [One-Neighbor End-of-Line Spacing Rule](#)
- [Two-Neighbor End-of-Line Spacing Rule](#)
- [Three-Neighbor End-of-Line Spacing Rule](#)
- [Three-Neighbor End-of-Line Cap Rule](#)
- [Two-Sided End-of-Line Spacing Rule](#)
- [Two-Sided End and Joint Spacing Rule](#)
- [End-of-Line Two-Corner Keepout Rule](#)
- [End-of-Line Spacing Rule and Stub Modes](#)
- [Enhanced Dense End-of-Line Spacing Rule](#)
- [End-of-Line to End-of-Line Spacing Rule \(Classic Router\)](#)

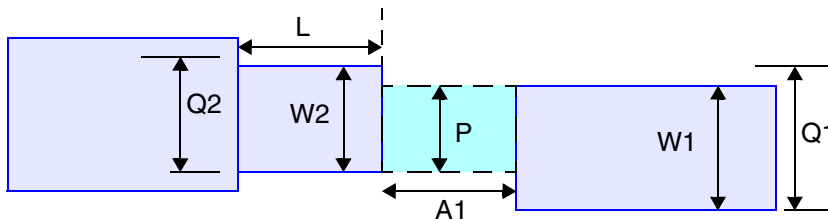
- [L-Shaped End-of-Line Spacing Rule](#)
- [End-of-Line Depth Rule](#)

End-of-Line to End-of-Line Spacing Rule

The end-of-line to end-of-line spacing rule specifies the minimum distance between the ends of two narrow metal lines approaching each other from opposite directions.

In [Figure 2-54](#), if the metal width $W1$ is less than or equal to the width threshold $Q1$, the metal width $W2$ is less than or equal to the width threshold $Q2$, and the parallel width overlap of the two metal lines is at least the parallel width threshold P , then the distance between the two line ends must be at least the minimum spacing value $A2$. If one or both metal lines widen at some distance away from the line-ends, the length of the minimum-width line-end must be at least L for this rule to apply.

Figure 2-54 End-of-Line to End-of-Line Spacing Rule



```
endOfLine1NeighborEndToEndThreshold      = Q1
endOfLine1NeighborEndToEndThreshold2     = Q2
endOfLine1NeighborEndToEndMinLength      = L
endOfLine1NeighborEndToEndParallelWidth  = P
endOfLine1NeighborEndToEndMinSpacing     = A1
```

Here is an example of the syntax used for this rule:

```
Layer "M2" {
  endOfLine1NeighborEndToEndThreshold      = 0.288
  endOfLine1NeighborEndToEndThreshold2     = 0.270
  endOfLine1NeighborEndToEndMinLength      = 0.300
  endOfLine1NeighborEndToEndParallelWidth  = 0.144
  endOfLine1NeighborEndToEndMinSpacing     = 0.104
}
```

In this example, for layer M2, when two lines, one having a width of no more than 0.288 and another with a width no more than 0.270, approach from opposite directions, the two line ends must have a spacing of at least 0.104. For wires that vary in width, the lengths of the minimum-width line-ends must be at least 0.300 for this rule to apply.

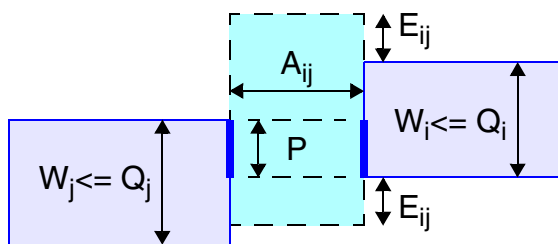
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see [“End-of-Line to End-of-Line Spacing Rule \(Classic Router\)”](#) on page 2-88.

One-Neighbor End-to-End Table-Based Spacing Rule

The one-neighbor end-to-end table-based spacing rule specifies a minimum distance between two facing end-of-line edges based on a table of exclusion-area extension values and width values. The rule specifies the table dimension N , a one-dimensional table of N width threshold values, an N -by- N table of parallel width extension values, and an N -by- N table of minimum-spacing values. See [Figure 2-55](#).

Figure 2-55 One-Neighbor End-of-Line Spacing Rule, Two Widths



```
endOfLine1NeighborEndToEndTblSize = N
endOfLine1NeighborEndToEndTblThreshold = (Q1, ... QN)
endOfLine1NeighborEndToEndTblParallelWidth = ( NxN table Eij values)
endOfLine1NeighborEndToEndTblMinSpacing = ( NxN table Aij values)
```

The minimum spacing A_{ij} applies between two end-of-line wires whose edge widths are W_i and W_j when W_i is no more than Q_i and W_j is no more than Q_j , and if the parallel run between the two metals P is at least the extension range E_{ij} . When the value of E_{ij} is specified as a negative number, the rule effectively creates an exclusion region extending the distance A_{ij} between the two line-ends and extending outward by the absolute value of E_{ij} .

Here is an example of a rule based on a 2-by-2 table.

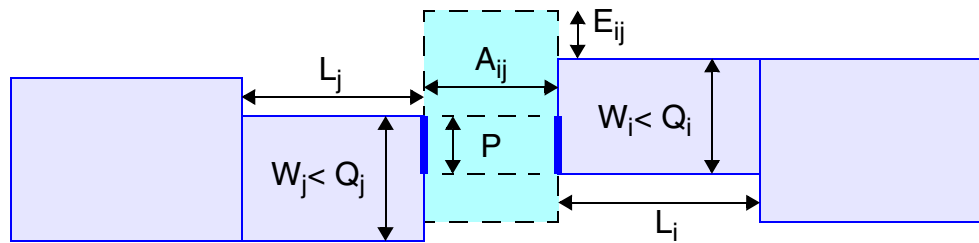
```
Layer "M1" {
  endOfLine1NeighborEndToEndTblSize           = 2
  endOfLine1NeighborEndToEndTblThreshold       = (0.066,0.114)
  endOfLine1NeighborEndToEndTblMinSpacing      = (0.070,0.066,
                                                  0.066,0.062)
  endOfLine1NeighborEndToEndTblParallelWidth  = (-0.058,-0.056,
                                                  -0.056,-0.055)
}
```

In this example, if both metal widths are between the thresholds 0.066 and 0.114, then the minimum spacing between the two line-ends must be at least 0.062 apart as long as the

parallel run between the two line-end edges is no more than -0.055 . This effectively creates an exclusion area that separates the two line-ends by 0.062 and extends outward by 0.055 from the corners of the line-end.

If the rule applies only when the lengths of the two line-ends are at least the length L as shown in Figure 2-56, use `endOfLine1NeighborEndToEndTblMinLength` to specify the length thresholds.

Figure 2-56 One-Nighbor End-of-Line Spacing Rule, Two Widths



```
endOfLine1NeighborEndToEndTblSize = N
endOfLine1NeighborEndToEndTblThreshold = (Q1, ... QN)
endOfLine1NeighborEndToEndTblParallelWidth = ( NxN table Eij values)
endOfLine1NeighborEndToEndTblMinSpacing = ( NxN table Aij values)
```

For example,

```
Layer "M1" {
  endOfLine1NeighborEndToEndTblSize           = 2
  endOfLine1NeighborEndToEndTblThreshold       = (0.066, 0.114)
  endOfLine1NeighborEndToEndTblMinLength      = (0.057, 0.060)
  endOfLine1NeighborEndToEndTblMinSpacing      = (0.070, 0.066,
                                                    0.066, 0.062)
  endOfLine1NeighborEndToEndTblParallelWidth  = (-0.058, -0.056,
                                                    -0.056, -0.055)
}
```

One-Nighbor End-of-Line Spacing Rule

The one-neighbor end-of-line spacing rule specifies the minimum distance between the end of a narrow metal line and a wider metal line or other nearby metal.

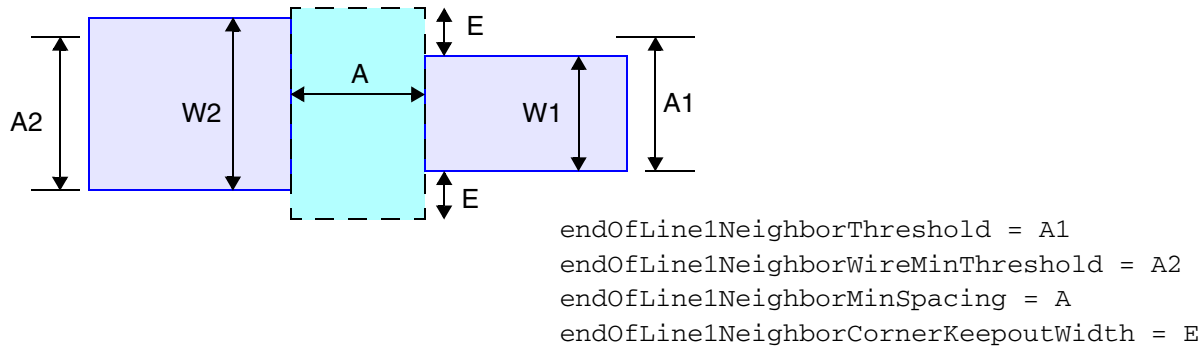
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see [“Stub Mode 1: Single-Edge Spacing at Metal End” on page 2-83](#).

In Figure 2-57, if the metal width $W1$ is less than or equal to the width threshold $A1$ and the metal width $W2$ is greater than or equal to $A2$, then the distance between the line end and

the other metal must be at least the minimum spacing value A. The two metals must be outside of the dashed rectangle extending outward from the line-end corners by the keepout width E.

Figure 2-57 One-Neighbor End-of-Line Spacing Rule, Two Widths



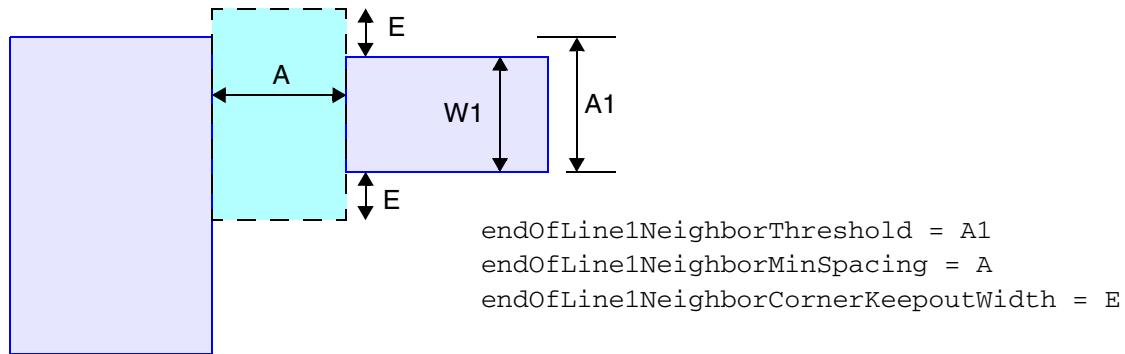
Here is an example of the syntax used for this rule:

```
Layer "M2" {
  endOfLine1NeighborThreshold = 0.288
  endOfLine1NeighborWireMinThreshold = 0.290
  endOfLine1NeighborMinSpacing = 0.140
  endOfLine1NeighborCornerKeepoutWidth = 0.020
}
```

In this example, for layer M2, when a metal line having a width less than or equal to 0.288 approaches another metal having a width of at least 0.290, the spacing between them must be at least 0.140, including a keepout region extending 0.020 away from the line-end corners.

If the neighbor's width is not defined, the rule is triggered by a single width threshold A1. In [Figure 2-58](#), if the metal width W1 is less than or equal to the width threshold A1, then the distance between the line end and the other metal must be at least the minimum spacing value A. The two metals must be outside of the dashed rectangle extending outward from the line-end corners by the keepout width E.

Figure 2-58 One-Neighbor End-of-Line Spacing Rule, One Width



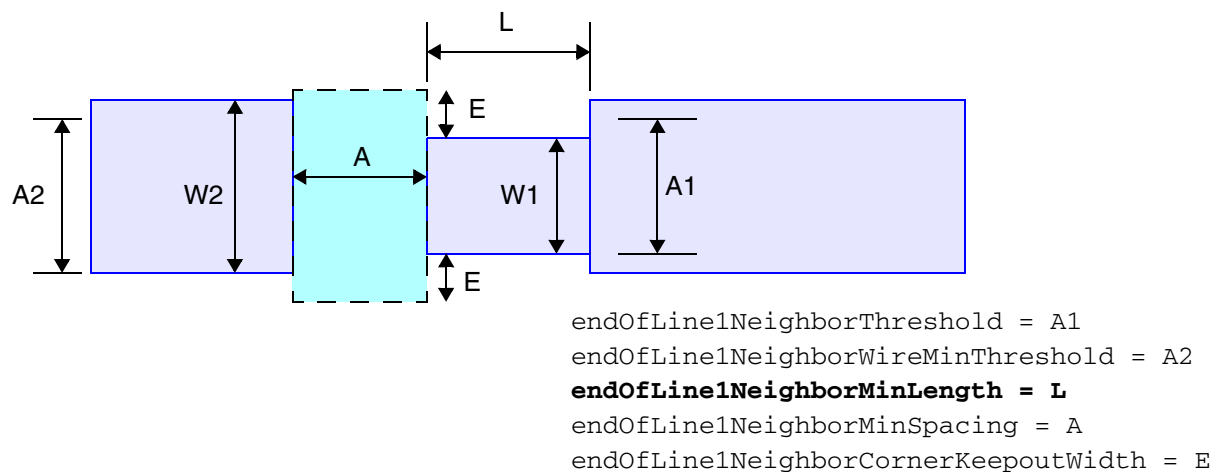
Here is an example of the syntax used for this rule:

```
Layer "M2" {
  endOfLine1NeighborThreshold = 0.288
  endOfLine1NeighborMinSpacing = 0.140
  endOfLine1NeighborCornerKeepoutWidth = 0.020
}
```

In this example, for layer M2, when a metal line having a width of less than 0.288 approaches another metal, the spacing between them must be at least 0.140, including a keepout region extending 0.020 away from the line-end corners.

If the narrow metal line widens and you want the rule to apply only if the narrowest portion is at least a specified length, specify the minimum length threshold using the syntax shown in [Figure 2-59](#).

Figure 2-59 One-Neighbor End-of-Line Spacing Rule, Variable Width



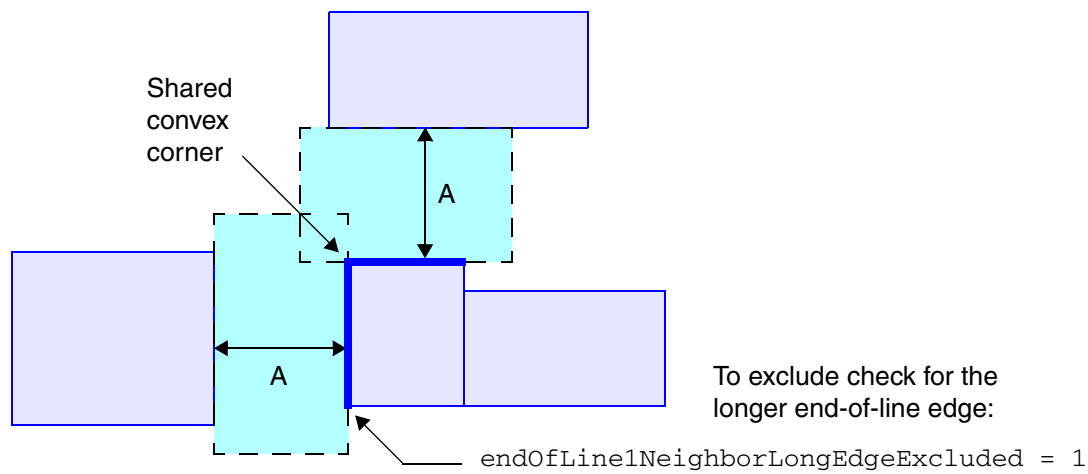
Here is an example of the syntax used for this rule:

```
Layer "M2" {
  endOfLine1NeighborThreshold = 0.288
  endOfLine1NeighborWireMinThreshold = 0.290
  endOfLine1NeighborMinLength = 0.300
  endOfLine1NeighborMinSpacing = 0.140
  endOfLine1NeighborCornerKeepoutWidth = 0.020
}
```

In this example, for layer M2, when a metal line having a width of less than or equal to 0.288 approaches another metal having a width of at least 0.290, and the length of the narrow line end is at least 0.300, then the spacing between them must be at least 0.140, including a keepout region extending 0.020 away from the line-end corners.

When two end-of-line edges share a common convex corner as shown in [Figure 2-60](#), by default, the one-neighbor end-of-line spacing rule applies to both edges.

Figure 2-60 Two Adjacent End-of-Line Edges

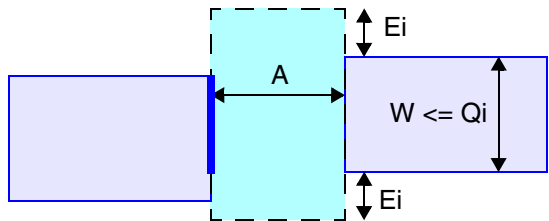


To apply the rule only to the shorter of the two adjacent edges, define an additional attribute, `endOfLine1NeighborLongEdgeExcluded`, and set it to 1:

```
Layer "M2" {
  endOfLine1NeighborThreshold = 0.288
  endOfLine1NeighborMinSpacing = 0.140
  endOfLine1NeighborCornerKeepoutWidth = 0.020
  endOfLine1NeighborLongEdgeExcluded = 1
}
```

A similar, table-based rule specifies a minimum spacing A_i , where $i=1, 2, 3, \dots, N$, between the end-of-line metal whose edge width W is no more than Q_i to a nearby metal edge and the parallel length between the two metal edges is within an extension range E_i . See [Figure 2-61](#).

Figure 2-61 One-Neighbor End-of-Line Spacing Rule, Table-Based



```

endOfLine1NeighborTblSize = N
endOfLine1NeighborWireThresholdTbl = (Q1, Q2, ... QN)
endOfLine1NeighborMinSpacingTbl = (A1, A2, ... AN)
endOfLine1NeighborCornerKeepoutWidthTbl = (E1, ... EN)

```

For example,

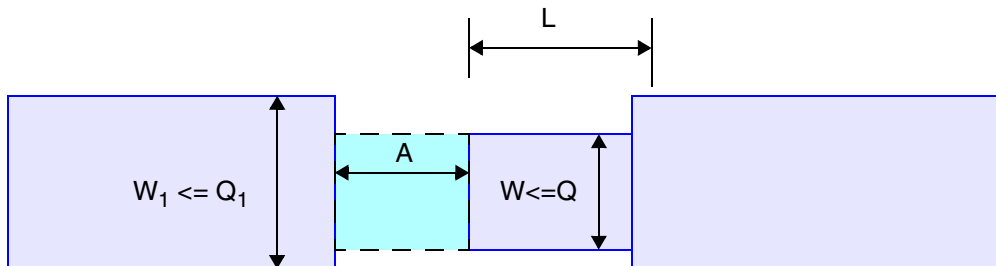
```

Layer "M2" {
  endOfLine1NeighborTblSize = 2
  endOfLine1NeighborThresholdTbl = (0.286, 0.310)
  endOfLine1NeighborMinSpacingTbl = (0.138, 0.160)
  endOfLine1NeighborCornerKeepoutWidthTbl = (0.018, 0.024)
}

```

If a minimum spacing A is required between the end-of-line of a metal whose edge width W is no more than Q and whose length is no more than L , and the nearby metal has an edge width W_1 that is no more than Q_1 , you can use the modified rule shown in [Figure 2-62](#).

Figure 2-62 Modified One-Neighbor End-of-Line Spacing Rule, Variable Width



```

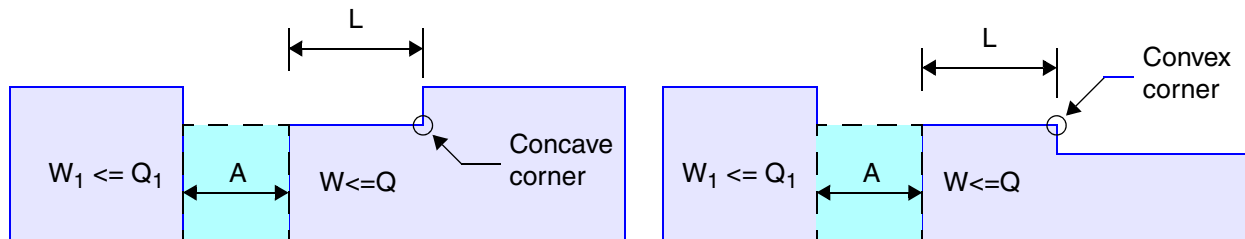
endOfLine1NeighborMod1MaxThreshold = Q
endOfLine1NeighborMod1MaxThreshold2 = Q1
endOfLine1NeighborMod1MaxLength = L
endOfLine1NeighborMod1MinSpacing = A

```

Specifying `endOfLine1NeighborMod1MaxThreshold2` is optional. If this attribute is omitted, Zroute does not check the nearby metal's width.

By default, the `endOfLine1NeighborMod1MaxLength` attribute applies only to a length extending from a convex corner, not a concave corner, as shown in [Figure 2-63](#).

Figure 2-63 Modified One-Nighbor End-of-Line Spacing Rule, Variable Width



To include check for the lengths extending from convex corners:

```
endOfLine1NeighborMod1ConvexCornerIncluded = 1
```

To have the rule apply to lengths extending from convex as well as concave corners, add a statement that sets `endOfLine1NeighborMod1ConvexCornerIncluded` to 1. For example,

```
Layer "M1" {
  endOfLine1NeighborMod1MaxThreshold = 0.288
  endOfLine1NeighborMod1MaxThreshold2 = 0.290
  endOfLine1NeighborMod1MaxLength = 0.380
  endOfLine1NeighborMod1MinSpacing = 0.140
  endOfLine1NeighborMod1ConvexCornerIncluded = 1
}
```

Two-Nighbor End-of-Line Spacing Rule

The two-neighbor end-of-line spacing rule specifies the minimum distance between the end of a narrow metal line and a wider neighboring metal line, and also from the side of the same narrow metal line to another neighboring metal.

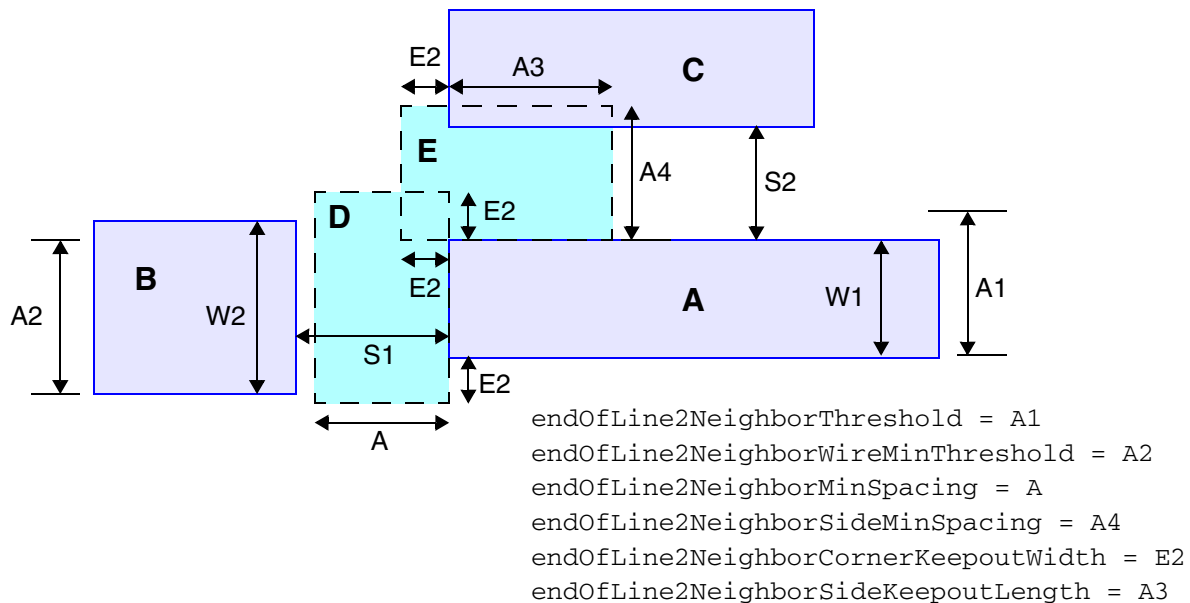
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see [“Stub Mode 4: Two-Edge Spacing With Connected Metal Exclusion”](#) on page 2-87.

In [Figure 2-64](#), if the metal width W_1 is less than or equal to the width threshold A_1 and the metal width W_2 is at least the width threshold A_2 , then at least one of the following conditions must be true:

- The distance between the line end and the first other metal S_1 is at least the minimum spacing value A_2 .
- The distance between the side of the narrow metal line and the second other metal S_2 is at least the minimum side spacing value A_4 .

Figure 2-64 Two-Nighbor End-of-Line Spacing Rule



In the figure, either metal B must be outside of the dashed rectangle D, or metal C must be outside of the dashed rectangle E. In this case, metal B is outside of dashed rectangle D, so the layout passes the rule. The dashed areas extend outward from the line-end corners by the corner keepout width E2. The side keepout area E has a length of A3 plus extension E2.

If the side minimum spacing value A4 is not defined, then the end-of-line minimum spacing value A is used for both the end-of-line and side spacing checks.

Here is an example of the syntax used for this rule:

```

Layer "M2" {
  endOfLine2NeighborThreshold = 0.070
  endOfLine2NeighborWireMinThreshold = 0.072
  endOfLine2NeighborMinSpacing = 0.075
  endOfLine2NeighborSideMinSpacing = 0.080
  endOfLine2NeighborCornerKeepoutWidth = 0.030
  endOfLine2NeighborSideKeepoutLength = 0.075
}

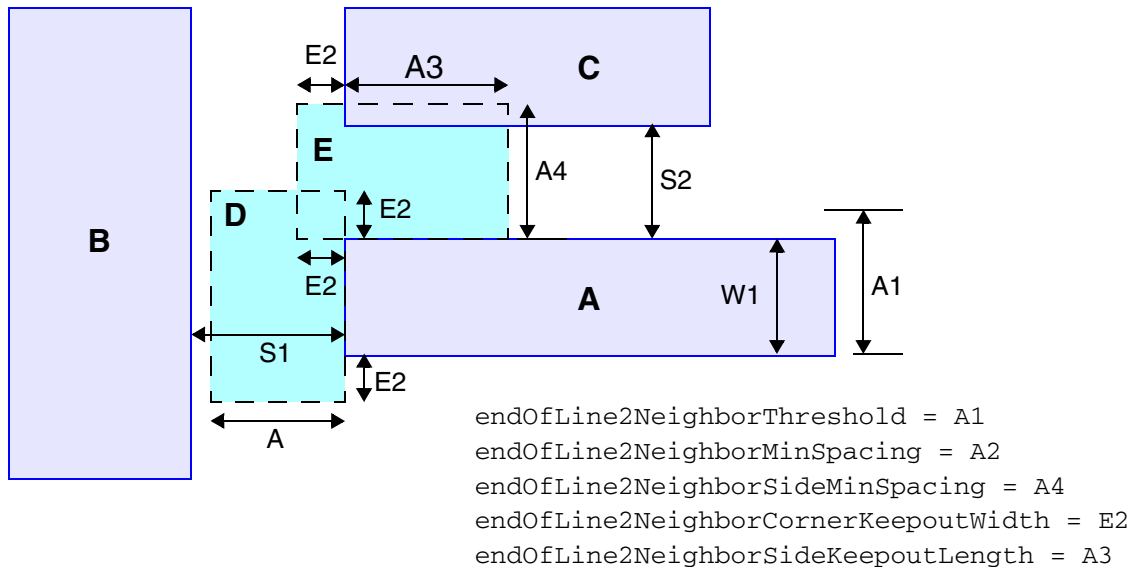
```

In this example, for layer M2, when a metal line with a width less than or equal to 0.070 has neighboring metal at the end with a width of at least 0.072 and also metal at the side near the end, the spacing to the neighboring metal at the end must be at least 0.075 or the spacing to the neighboring metal at the side must be at least 0.080. The keepout region extends 0.030 away from the line-end corners and 0.080 from the corner along the side.

If the neighbor's width is not defined, the rule is triggered by a single width threshold A1. In [Figure 2-65](#), if the metal width W1 is less than or equal to the width threshold A1, then the

distance between the line end and the first other metal S1 must be at least the minimum spacing value A, or the distance between the side of the narrow metal line and the second other metal S2 must be at least the minimum side spacing value A4.

Figure 2-65 Two-Neighbor End-of-Line Spacing Rule



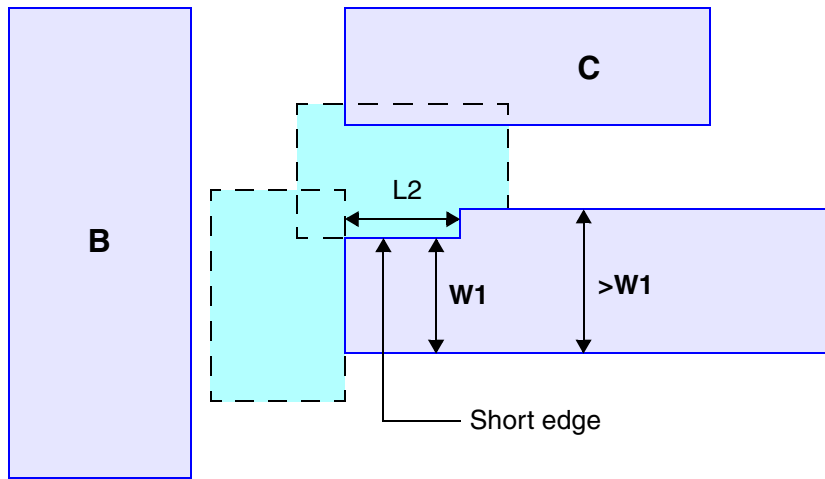
In the figure, either metal B must be outside of the dashed rectangle D, or metal C must be outside of the dashed rectangle E. In this case, metal B is outside of dashed rectangle D, so the layout passes the rule. The dashed areas extend outward from the line-end corners by the corner keepout width E2. The side keepout area E has a length of A3 plus extension E2.

Here is an example of the syntax used for this rule:

```
Layer "M2" {
  endOfLine2NeighborThreshold = 0.070
  endOfLine2NeighborMinSpacing = 0.075
  endOfLine2NeighborSideMinSpacing = 0.080
  endOfLine2NeighborCornerKeepoutWidth = 0.030
  endOfLine2NeighborSideKeepoutLength = 0.075
}
```

In this example, for layer M2, when a metal line with a width of less than 0.070 has neighboring metal at the end and at the side near the end, the spacing to the neighboring metal at the end must be at least 0.075 or the spacing to the neighboring metal at the side must be at least 0.080. The keepout region extends 0.030 away from the line-end corners and 0.080 from the corner along the side.

The rule does not apply when a short edge is adjacent to the end of the line, causing the wire width to be greater than the end-of-line width threshold, as shown in [Figure 2-66](#).

Figure 2-66 Two-Nighbor End-of-Line With Adjacent Short Edge

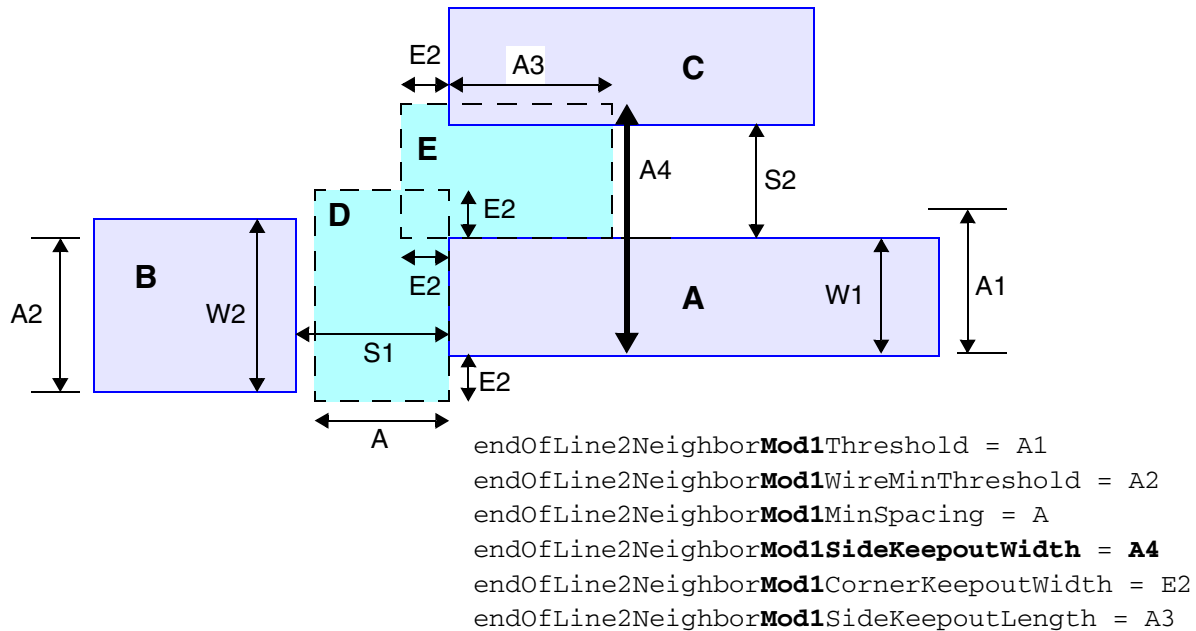
By default, the adjacent edge L2 is considered short if it is less than or equal to the `minWidth` value defined for the metal. To define a different L2 threshold for the two-neighbor end-of-line spacing rule, use the following attribute:

```
endOfLine2NeighborMinLength = L2
```

If the adjacent edge at is less than or equal to the specified value, the rule does not apply.

[Figure 2-67](#) shows an alternative, modified form of the rule that specifies the side spacing requirement A4 as a distance from the neighboring metal to the far edge of the narrow metal line instead of the nearest edge. This is a separate rule with different syntax, but operating in a manner similar to the foregoing rule.

Figure 2-67 Modified Two-Nighbor End-of-Line Spacing Rule



Here is an example of the modified syntax used for this rule:

```

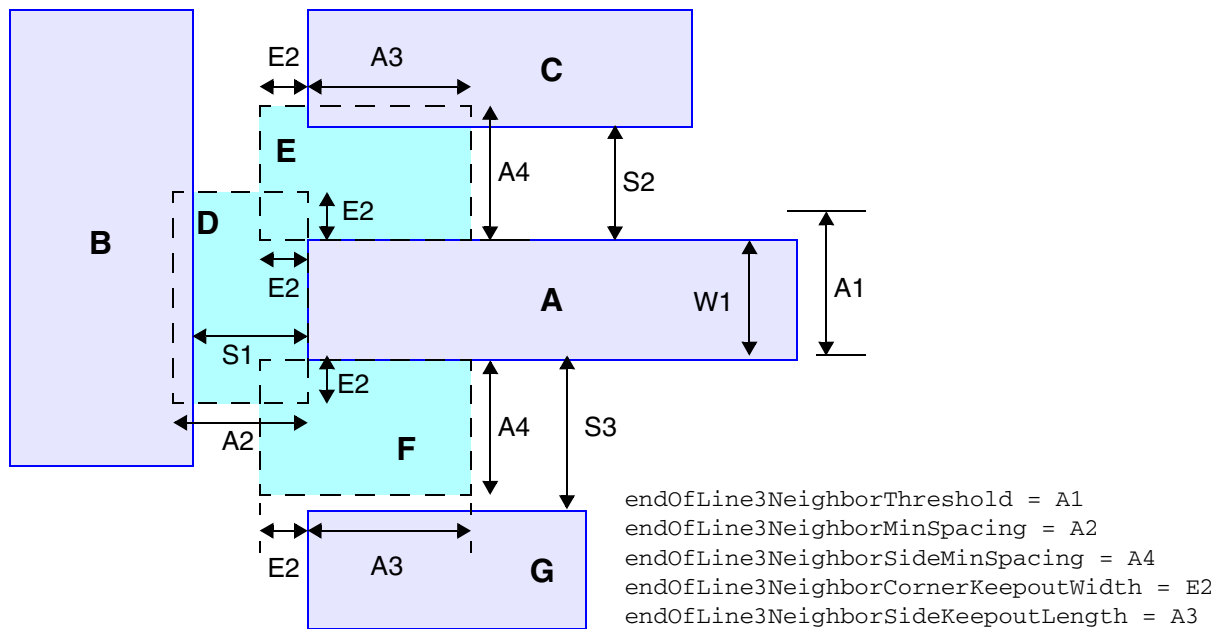
Layer "M2" {
  endOfLine2NeighborMod1Threshold = 0.070
  endOfLine2NeighborMod1WireMinThreshold = 0.072
  endOfLine2NeighborMod1MinSpacing = 0.075
  endOfLine2NeighborMod1SideKeepoutWidth = 0.150
  endOfLine2NeighborMod1CornerKeepoutWidth = 0.030
  endOfLine2NeighborMod1SideKeepoutLength = 0.075
}

```

Three-Nighbor End-of-Line Spacing Rule

The three-neighbor end-of-line spacing rule specifies the minimum distance between the end of a narrow metal line and a neighboring metal, and from the two sides of the same narrow metal line to two other neighboring metals. In [Figure 2-68](#), if the metal width W1 is less than or equal to the width threshold A1, then the distance between the line end and the metal S1 must be at least the minimum spacing value A2. If not, the distance between at least one of the two sides of the narrow metal line and the neighboring metals, S2 or S3, must be at least the minimum side spacing value A4.

Figure 2-68 Three-Nighbor End-of-Line Spacing Rule



In the figure, either metal B must be outside of the dashed rectangle D, or if not, either metal C must be outside of the dashed rectangle E or metal G must be outside of the dashed rectangle F. In this case, metal G is outside of dashed rectangle F, so the layout passes the rule. The dashed areas extend outward from the line-end corners by the corner keepout width E2. The side keepout areas E and F each have a length of A3 plus extension E2.

Here is an example of the syntax used for this rule:

```

Layer "M2" {
  endOfLine3NeighborThreshold = 0.288
  endOfLine3NeighborMinSpacing = 0.14
  endOfLine3NeighborSideMinSpacing = 0.16
  endOfLine3NeighborCornerKeepoutWidth = 0.02
  endOfLine3NeighborSideKeepoutLength = 0.27
}

```

In this example, for layer M2, when a metal line having a width of less than 0.288 has neighboring metal at the end and at two sides near the end, the spacing to the neighboring metal at the end must be at least 0.14 or the spacing to the neighboring metal on at least one of the two sides must be at least 0.16. The two side keepout regions extend 0.02 away from the line-end corners and 0.16 from the corner along the sides.

If `endOfLine3NeighborSideMinSpacing` is omitted, `endOfLine3NeighborMinSpacing` applies to both the end-of-line spacing check and the side spacing check.

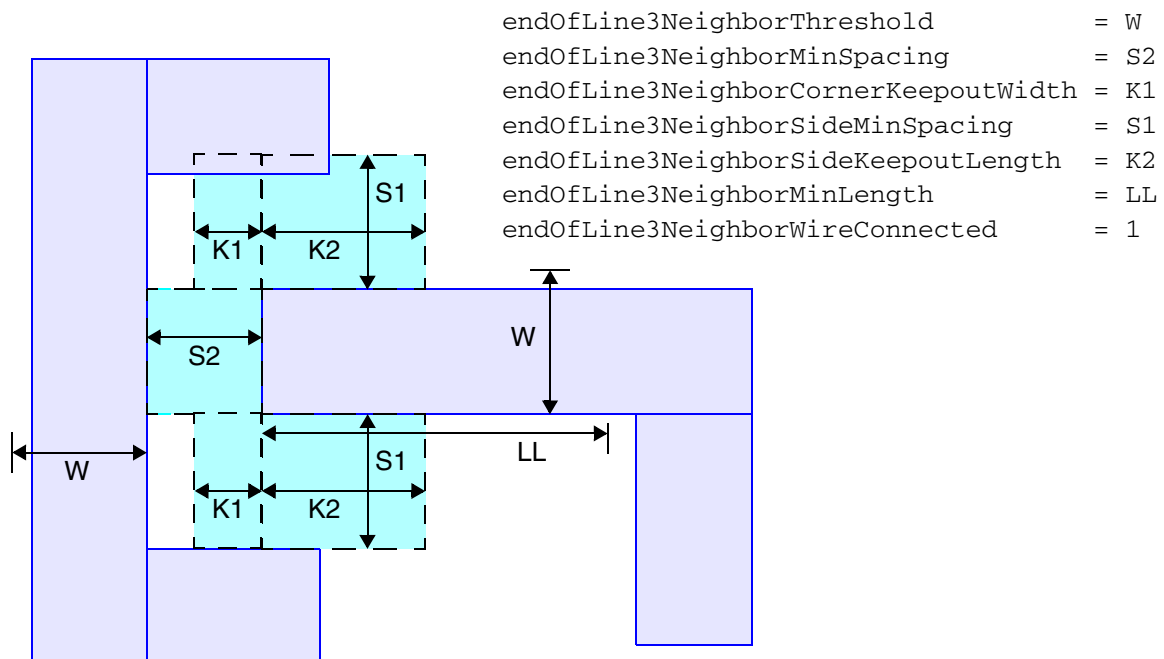
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see [“Stub Mode 3: Three-Edge Spacing at Metal End”](#) on page 2-86.

Three-Neighbor End-of-Line Cap Rule

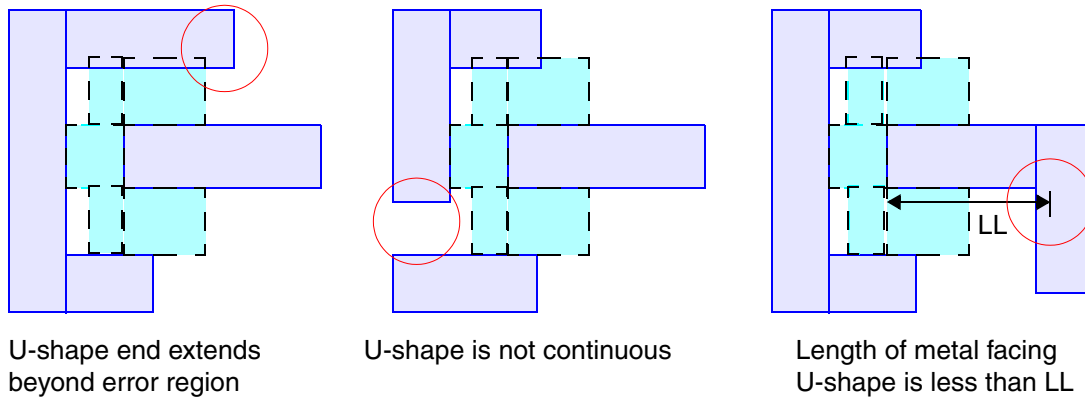
The three-neighbor end-of-line cap rule specifies the minimum distance between the end of a narrow metal line and neighboring metal that forms a U-shaped “cap” facing the wire end. In [Figure 2-69](#), the metal line has a width less than W and a length greater than LL . Specifying the LL attribute is optional. The U-shape opposite the line end also has a width less than W .

Figure 2-69 Three-Neighbor End-of-Line Spacing Rule



Two checking regions extend from the line-end corners, each region measuring $(K1+K2)$ by $S1$, extending the distance $K1$ away from the line-end and $K2$ along the side of the metal line. If the U-shape has a line-end inside one or both checking regions, then the spacing from the line-end to the U-shape must be at least $S2$.

[Figure 2-70](#) shows some examples of configurations that resemble a U-shaped cap, but the rule does not apply.

Figure 2-70 Three-Nighbor End-of-Line Cap Rule Exclusions

This is the general syntax of the rule:

```

Layer "MetalX" {
  endOfLine3NeighborThreshold           = W
  endOfLine3NeighborMinSpacing          = S2
  endOfLine3NeighborCornerKeepoutWidth = K1
  endOfLine3NeighborSideMinSpacing      = S1
  endOfLine3NeighborSideKeepoutLength  = K2
  endOfLine3NeighborMinLength           = LL # optional
  endOfLine3NeighborWireConnected      = 1
}

```

For example,

```

Layer "M2" {
  endOfLine3NeighborThreshold           = 0.061
  endOfLine3NeighborMinSpacing          = 0.127
  endOfLine3NeighborCornerKeepoutWidth = 0.000
  endOfLine3NeighborSideMinSpacing      = 0.068
  endOfLine3NeighborSideKeepoutLength  = 0.136
  endOfLine3NeighborMinLength           = 0.168
  endOfLine3NeighborWireConnected      = 1
}

```

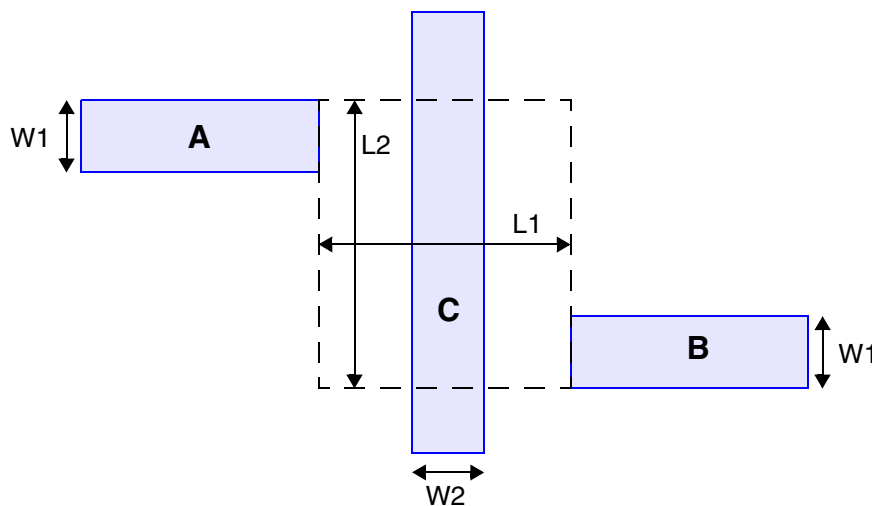
Note:

This rule is supported only by Zroute.

Two-Sided End-of-Line Spacing Rule

The two-sided end-of-line spacing rule specifies the minimum distance between two narrow end-of-line wires approaching a third narrow wire from both sides at 90 degrees. In [Figure 2-71](#), the two narrow wires A and B, both having a width less than or equal to $W1$, approach a third narrow wire C having a width less than $W2$, from opposite directions. The ends of wires A and B must stay outside of the rectangle measuring $L1$ by $L2$ and centered on wire C.

Figure 2-71 Two-Sided End-of-Line Spacing Rule



To invoke this rule in Zroute, use the following syntax in the `Layer` section:

```
tJunctionStubWireMaxThreshold    = W1
tJunctionOrthoWireMaxThreshold   = W2
tJunctionStubKeepoutMinSpacing   = L2
tJunctionStubKeepoutMinWidth     = L1
```

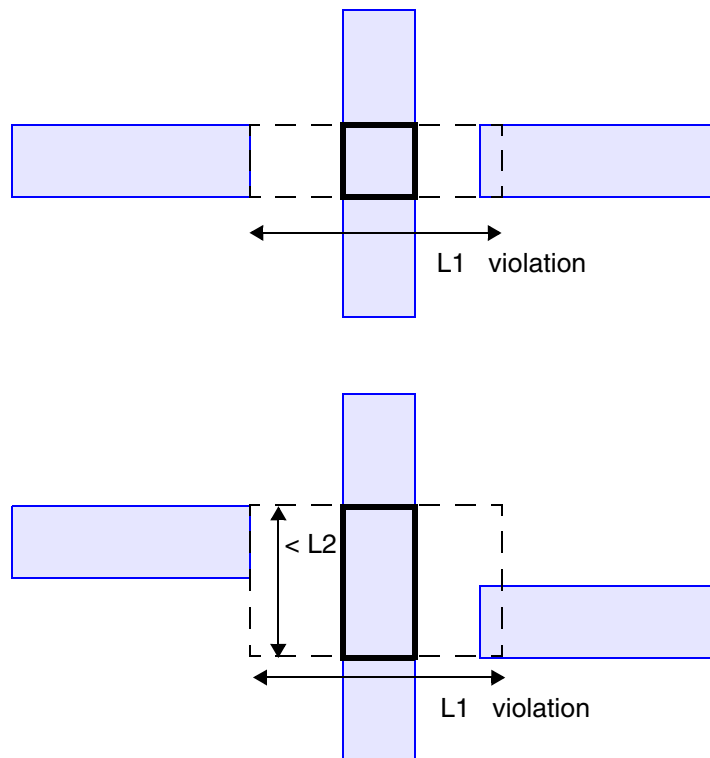
For example,

```
tJunctionStubWireMaxThreshold    = 0.20
tJunctionOrthoWireMaxThreshold   = 0.22
tJunctionStubKeepoutMinSpacing   = 0.80
tJunctionStubKeepoutMinWidth     = 0.70
```

Note:

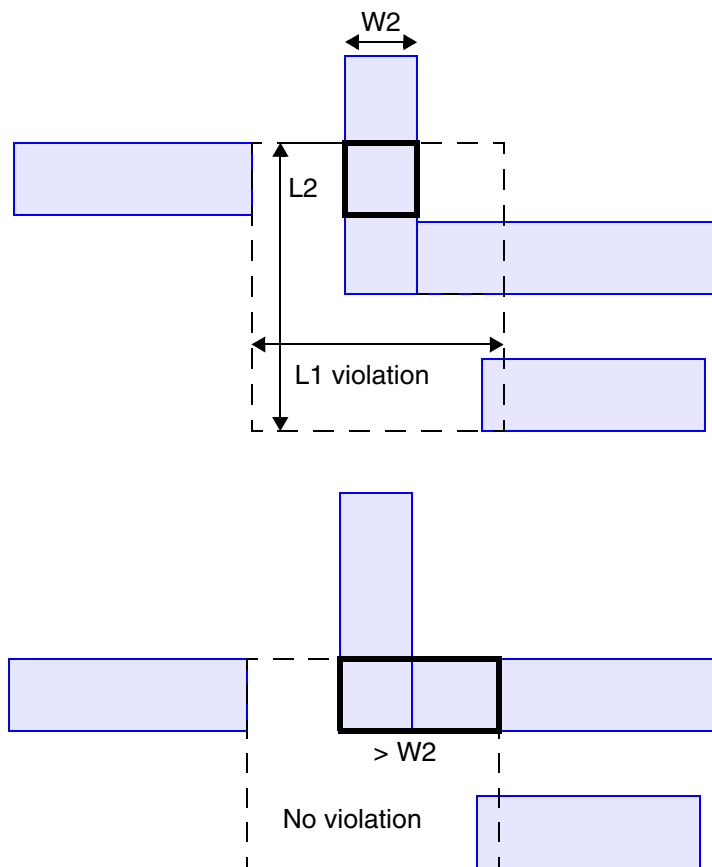
This rule is supported only by Zroute.

[Figure 2-72](#) shows two examples of violations. When the two end-of-line wires approach the third wire directly opposite from each other or offset by as much as $L2$ as measured from their outer edges, both ends must stay outside of the box measuring $L1$ in length. The black rectangle represents the overlap of the two approaching wires with the central wire.

Figure 2-72 Two-Sided End-of-Line Spacing Violation Examples

[Figure 2-73](#) shows two examples of end-of-line wires approaching an L-shaped wire. The black rectangle represents the overlap of the two approaching wires with the L-shaped wire. The first example is a violation. However, the second example is not a violation because the overlap applies to the long section of the L-shaped wire. The L-shaped wire's width is considered to be greater than $W2$, so it is not a narrow wire for this application of the rule, and the rule does not apply.

Figure 2-73 Two-Sided End-of-Line Spacing Violation at “L”



Two-Sided End and Joint Spacing Rule

The two-sided end and joint spacing rule specifies the minimum distance between two wire ends, or two joints, or a wire end and a joint located on opposite sides of another wire.

A wire end is an edge that connects two convex corners and has an edge length (wire width) of at least L_E but no more than W .

A joint is an edge that is not a wire end, has a length of no more than W , has a span of at least P_J , and whose nearest convex corner is no more than the distance D away.

Figure 2-74 shows two wire ends approaching opposite sides of a horizontal wire and two joints located on opposite sides of the same horizontal wire. Figure 2-75 shows examples of a wire end and a joint located on opposite sides of the same horizontal wire.

Figure 2-74 Two Wire Ends and Two Wire Joints on Opposite Sides of a Wire

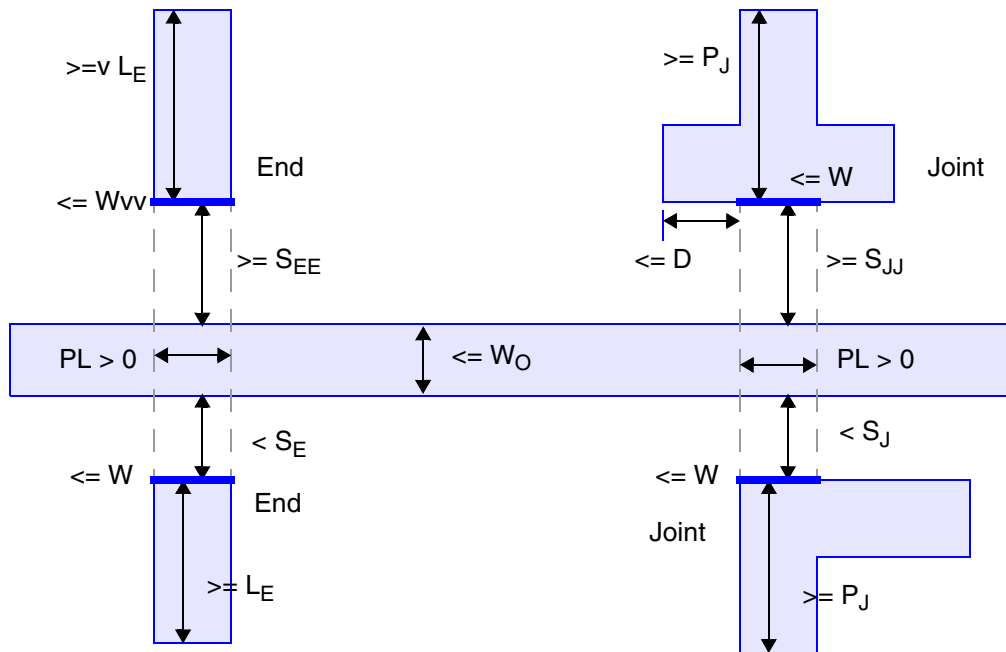
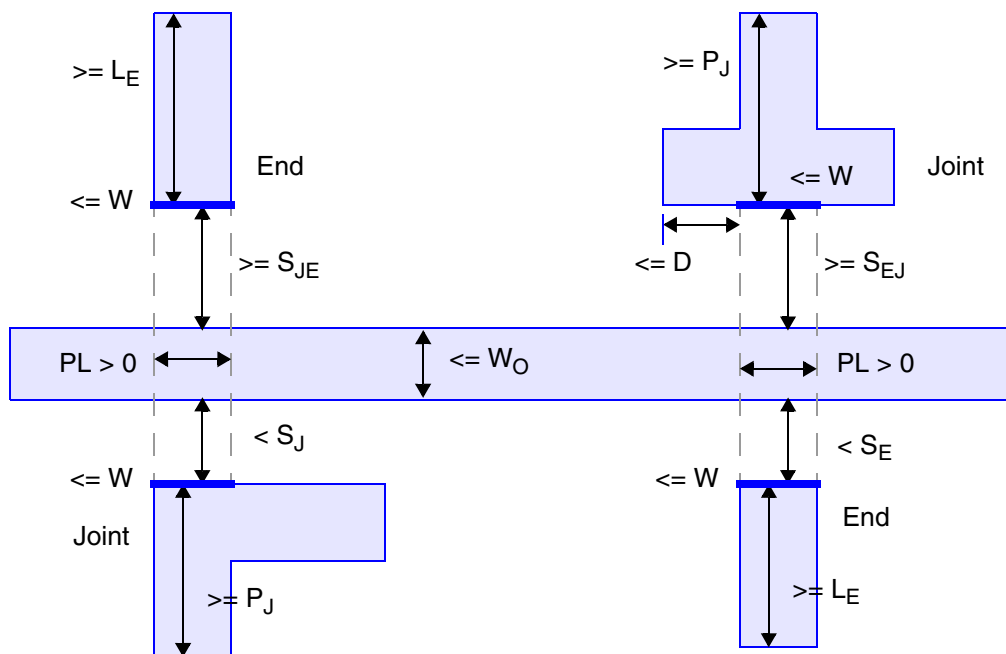


Figure 2-75 A Wire End and a Joint on Opposite Sides of a Wire



A minimum spacing S is required when two ends or two joints approach a wire having a width less than or equal to W_O from opposite sides, and sharing a parallel length PL between the two ends or joints that is larger than zero.

The rule defines four different minimum spacing requirements:

- S_{EE} is the minimum spacing between one end and the wire when the spacing between the wire and the end on the opposite side is less than S_E .
- S_{JJ} is the minimum spacing between one joint and the wire when the spacing between the wire and the joint on the opposite side is less than S_J .
- S_{JE} is the minimum spacing between one end and the wire when the spacing between the wire and the joint on the opposite side is less than S_J .
- S_{EJ} is the minimum spacing between one joint and the wire when the spacing between the wire and the end on the opposite side is less than S_E .

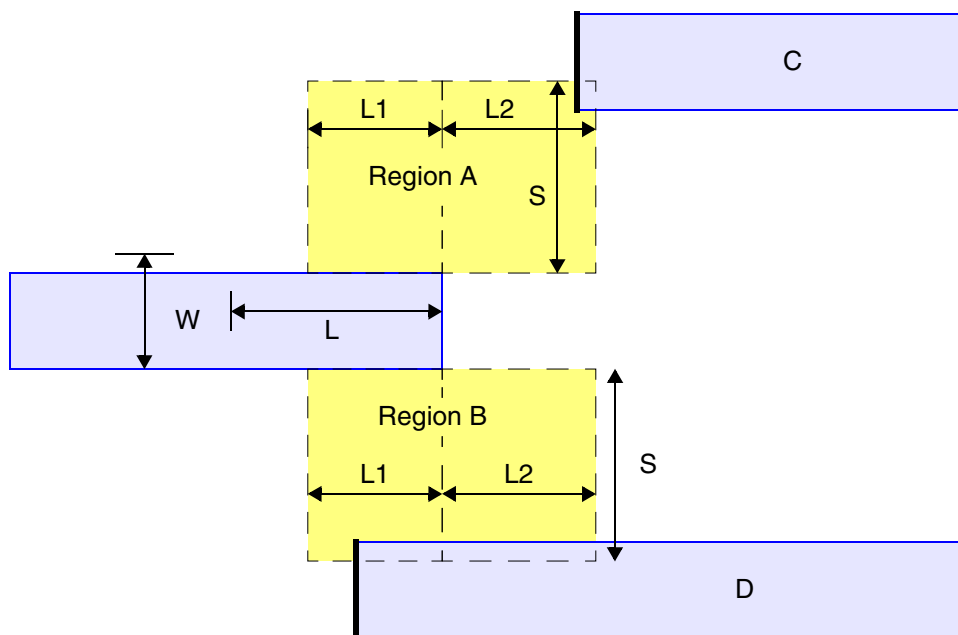
The rule is defined in the metal `Layer` section as follows:

```
Layer "MetalX" {
  endJointOrthoWireMaxThreshold          =  $W_O$ 
  endWireSideEdgeLengthThreshold         =  $L_E$ 
  jointWireSpanThreshold                  =  $P_J$ 
  jointToConvexCornerMaxDist              =  $D$ 
  endJointWireThresholdTblSize           =  $N$ 
  endJointWireWidthMaxThresholdTbl       =  $(W_1, W_2, \dots)$ 
  endJointWireParallelWidthThresholdTbl  =  $(PL_1, PL_2, \dots)$ 
  endWireMaxSpacingThreshold              =  $S_E$ 
  jointWireMaxSpacingThreshold            =  $S_J$ 
  endEndWireMinSpacing                   =  $S_{EE}$ 
  jointJointWireMinSpacing                =  $S_{JJ}$ 
  endJointWireMinSpacing                  =  $S_{EJ}$ 
  jointEndWireMinSpacing                  =  $S_{JE}$ 
}
```

End-of-Line Two-Corner Keepout Rule

The end-of-line two-corner keepout rule specifies a keepout region at the two corners of a metal end-of-line, which prevents other metal end-of-lines from coming close to both corners. In [Figure 2-76](#), the rule applies to a metal having a width less than or equal to the width threshold W . You can optionally specify a minimum length threshold L ; in that case, the rule does not apply for metal lines shorter than L .

Figure 2-76 End-of-Line Two-Corner Keepout Rule



The keepout region is a rectangular area measuring S by $(L1+L2)$. $L1$ is measured back from the corner along the metal line and $L2$ is measured from the corner away from the metal line. A violation occurs if two outside metal lines approaching from the opposite direction have end-of-lines inside region A and region B. For example, horizontal lines C and D would trigger an error.

This is the general syntax of the rule:

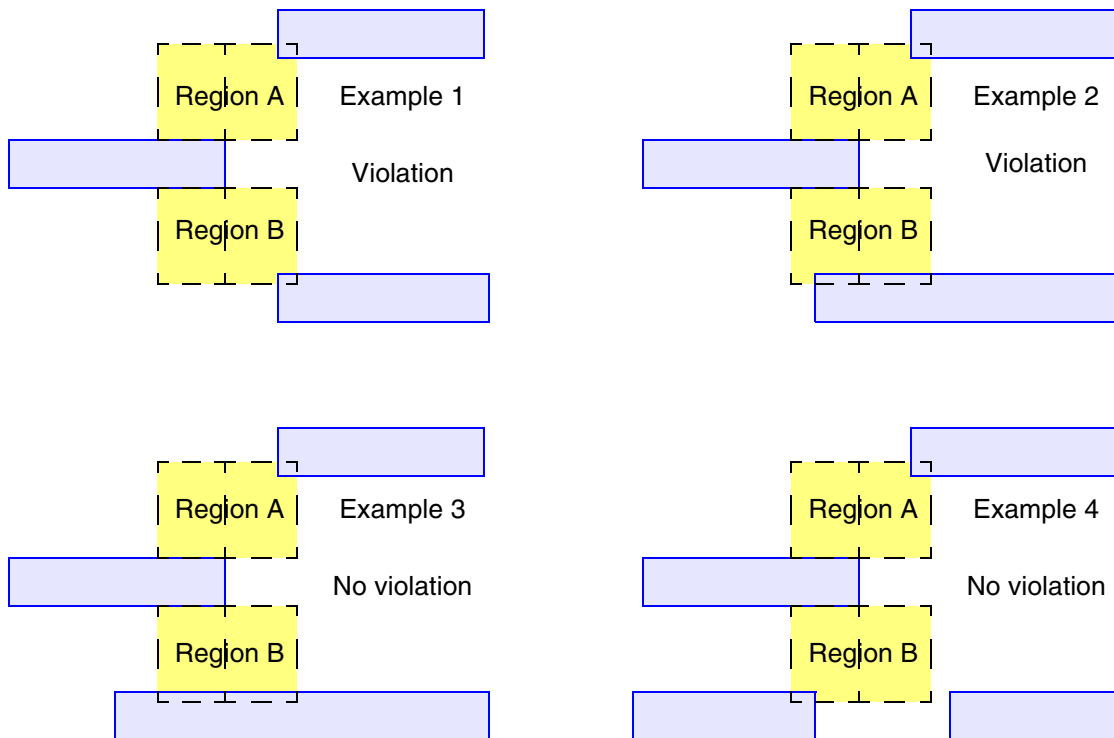
```
Layer "Mx" {
    endOfLine2CornerThreshold           = W
    endOfLine2CornerMinLength          = L
    endOfLine2CornerKeepoutWidth       = S (optional)
    endOfLine2CornerKeepoutLength      = L2
    endOfLine2CornerSideKeepoutLength = L1
}
```

For example,

```
Layer "Mx" {
    endOfLine2CornerThreshold           = 0.22
    endOfLine2CornerKeepoutWidth       = 0.43
    endOfLine2CornerKeepoutLength      = 0.31
    endOfLine2CornerSideKeepoutLength = 0.26
}
```

The examples in [Figure 2-77](#) further demonstrate the rule. The first two examples are violations, but the third one is not because there is no end-of-line in region B. The fourth example has an end-of-line in region B, but it comes from a line approaching from the same direction as the original line, so it is not a violation.

Figure 2-77 End-of-Line Two-Corner Keepout Rule Examples



End-of-Line Spacing Rule and Stub Modes

The end-of-line spacing rule defines a larger minimum spacing requirement where the end of a wire is perpendicular to another wire. You define this rule by setting the following attributes in a metal `Layer` section of the technology file:

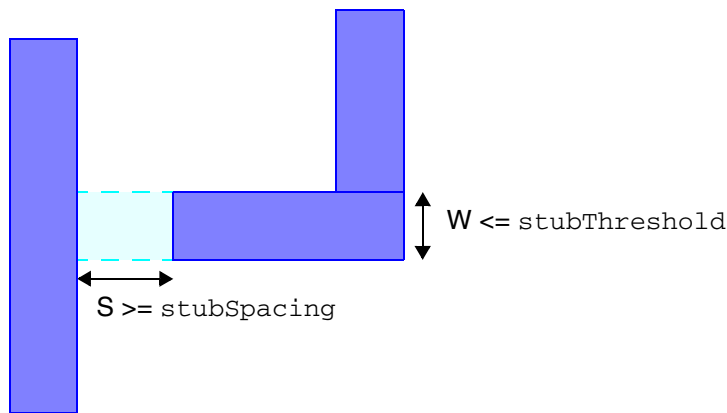
- The `stubSpacing` attribute defines the minimum spacing between the end-of-line metal segment and its adjacent wire.
- The `stubThreshold` attribute defines the maximum width of the end-of-line metal segment to which this rule applies. If the wire is wider than the `stubThreshold` value, the default minimum spacing applies.

Note:

The stub modes are still supported. However, for any new technology file, you should use the more advanced rules described in [“End-of-Line Spacing Rules” on page 2-60](#). Zroute does not support usage of both old and new syntax in the same technology file.

Figure 2-78 illustrates the application of the stub mode rule. The rule applies only to the lightly shaded area between the wire end and the adjacent wire. When the end-of-line wire width W is less than `stubThreshold`, the spacing S must be at least `stubSpacing`.

Figure 2-78 End-of-Line Spacing Rule



Depending on the foundry process specification, you should specify a value of 1, 2, 3, or 4 for the `stubMode` attribute in the `Technology` section of the technology file. The stub mode determines in detail the types of checking performed in different end-of-line situations.

For example,

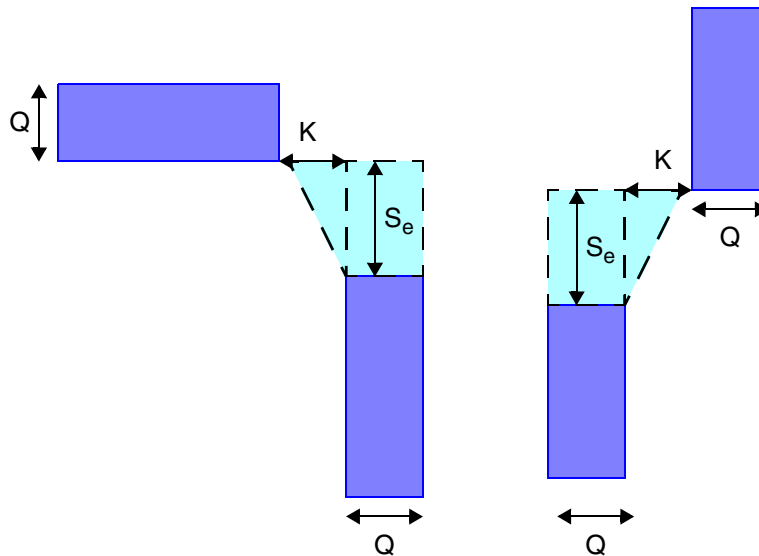
```
Technology {
    stubMode = 1
}

Layer "M1" {
    stubSpacing    = 0.14
    stubThreshold  = 0.20
}
```

Stub Mode 1: Single-Edge Spacing at Metal End

With `stubMode` set to 1, the minimum spacing between two narrow metal end-of-lines is S_e (`stubSpacing`) when the metal widths are both less than or equal to Q (`stubThreshold`). This rule applies to metal end-of-lines that are offset by as much as the distance K (`endOfLineCornerKeepoutWidth`). In each of the two examples shown in [Figure 2-79](#), the upper blue metal line must stay outside of the lightly shaded regions between it and the lower-metal line.

Figure 2-79 Stub Mode 1



This is the syntax for specifying the end-of-line rule in stub mode 1:

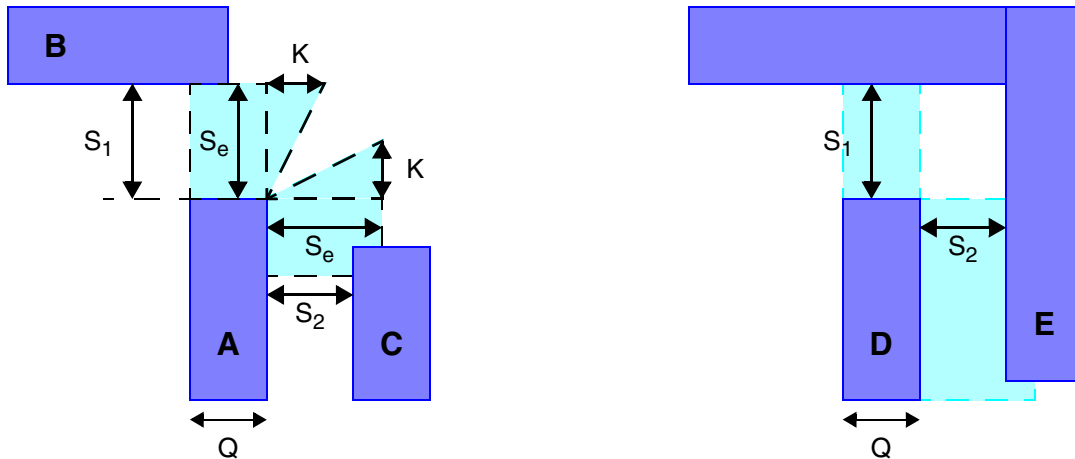
```
Technology {
  stubMode = 1
}

Layer "Metal1" {
  stubSpacing = Se
  stubThreshold = Q
  endOfLineCornerKeepoutWidth = K
}
```

Stub Mode 2: Two-Edge Spacing at Metal End

With `stubMode` set to 2, when two adjacent edges of a narrow metal end-of-line are close to two neighboring metal edges, or when any one short edge is close to an inside corner between two neighboring edges, then at least one of the two spacing values must be greater than the specified stub spacing value. Two examples are illustrated in [Figure 2-80](#).

Figure 2-80 Stub Mode 2



In the first example, the top and right edges of metal A are close to the edges of metal B and metal C. The width of metal A is less than or equal to Q (`stubThreshold`). Either the spacing S_1 or the spacing S_2 must be at least S_e (`stubSpacing`). One of the two spacings (metal C in this example) can be less than S_e as long as it meets the general metal-to-metal spacing requirement and the other spacing (metal B in this example) meets the S_e spacing requirement. Metal that is offset by as much as K (`endOfLineCornerKeepoutWidth`) is still considered by this rule.

In the second example, the width of metal D is less than or equal to Q (`stubThreshold`) and the metal is close to the inside corner of metal E. Of the two spacing values S_1 and S_2 , at least one of them must be at least S_e (`stubSpacing`). In this example, S_1 meets this requirement. S_2 is less than S_e but still meets the general metal-to-metal minimum spacing requirement.

This is the syntax for specifying the end-of-line rule in stub mode 2:

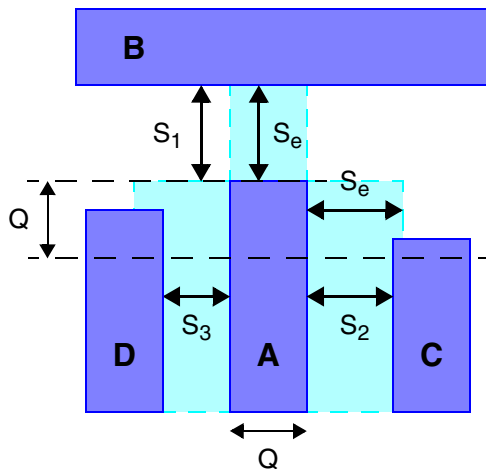
```
Technology {
  stubMode = 2
}

Layer "Metal1" {
  stubSpacing = Se
  stubThreshold = Q
  endOfLineCornerKeepoutWidth = K
}
```

Stub Mode 3: Three-Edge Spacing at Metal End

With `stubMode` set to 3, when three adjacent edges of a narrow metal end-of-line are close to three neighboring metal edges, and two of the neighboring metals are end-of-lines ending within the stub threshold, then at least one of the three spacing values must be greater than the specified stub spacing value. An example is shown in [Figure 2-81](#).

Figure 2-81 Stub Mode 3



In this example, metal A has a width less than or equal to Q (`stubThreshold`) and has three neighboring metals, B, C, and D. The ends of metals C and D are offset from the end of metal A by no more than Q . At least one of the three metal-to-metal spacing values S_1 , S_2 , and S_3 must be at least the stub spacing S_e (`stubSpacing`). In this example, spacing S_1 meets this requirement. S_2 and S_3 must still meet the general metal-to-metal minimum spacing requirement.

This is the syntax for specifying the end-of-line rule in stub mode 3:

```
Technology {
  stubMode = 3
}

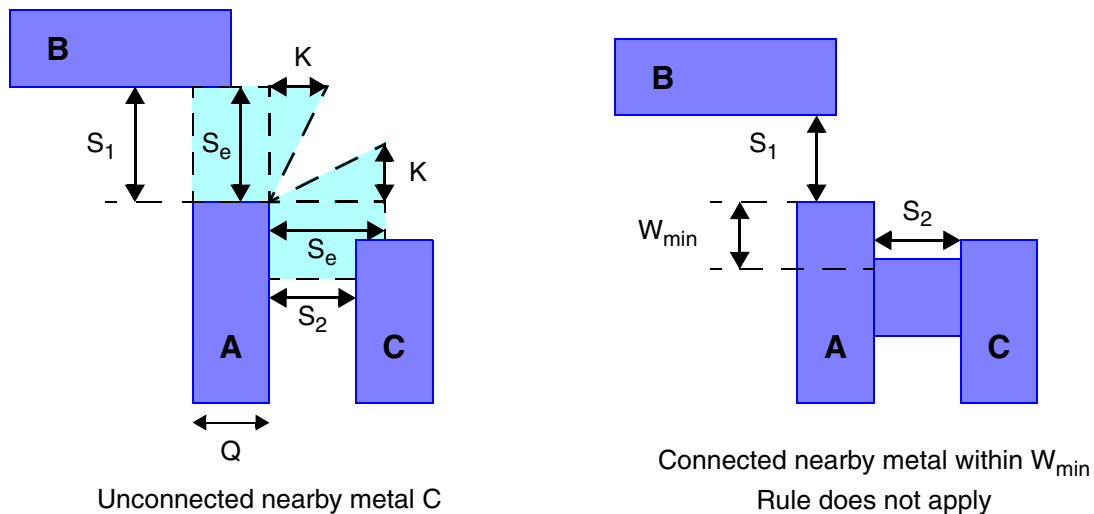
Layer "Metall" {
  stubSpacing = Se
  stubThreshold = Q
}
```


Stub Mode 4: Two-Edge Spacing With Connected Metal Exclusion

Stub mode 4 is similar to stub mode 2. The only difference is that the rule does not apply to the case where the nearby metal is connected with a jog or displacement of less than the default minimum width, W_{\min} (`minWidth`).

In the first example shown in [Figure 2-82](#), there is no connection between metal A and metal C, so stub mode 4 is the same as stub mode 2. However, in the second example, metal A is connected to the nearby metal and the size of the jog is less than W_{\min} , so the rule does not apply. Even if S_1 and S_2 are both less than S_e , it is not a violation of the stub mode 4 rule, whereas it would be a violation with the stub mode 2 rule. If the size of the jog is greater than W_{\min} , then the rule still applies and stub mode 4 still works like stub mode 2.

Figure 2-82 Stub Mode 4



This is the syntax for specifying the end-of-line rule in stub mode 4:

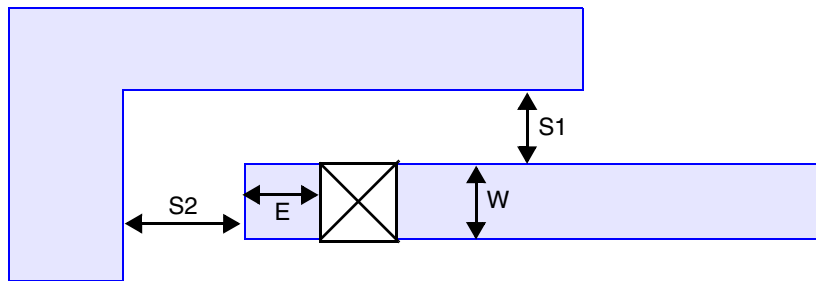
```
Technology {
  stubMode = 4
}

Layer "Metall" {
  minWidth = Wmin
  stubSpacing = Se
  stubThreshold = Q
  endOfLineCornerKeepoutWidth = K
}
```

Enhanced Dense End-of-Line Spacing Rule

The enhanced dense end-of-line spacing rule defines the minimum end-of-line spacing for wire ends with a via dropped to the upper or lower via layer when the width (W) of the wire is less than the stub threshold as defined in the `stubThreshold` attribute in the associated `Layer` section in the technology file. [Figure 2-83](#) shows the attributes associated with this rule.

Figure 2-83 Enhanced Dense End-of-Line Spacing Rule



This rule is defined in the `DesignRule` section of the technology file. The minimum spacing requirement depends on the width of the via enclosure (E). The enclosure width thresholds are defined in the `endOfLineEncTbl` attribute and the end-of-line minimum spacing requirements (S2) are specified in the `endOfLineEncSpacingTbl` attribute. The S1 spacing requirement is specified as the largest value in the `endOfLineEncSpacingTbl` attribute.

For example,

```
DesignRule {
  layer1 = "M2"
  layer2 = "VIA1"
  endOfLineEncTblSize = 5
  endOfLineEncSpacingTbl = (0.1, 0.105, 0.11, 0.115, 0.12)
  endOfLineEncTbl = (0.05, 0.045, 0.04, 0.035, 0.03)
}
```

In this example, S1 spacing requirement is 0.12.

End-of-Line to End-of-Line Spacing Rule (Classic Router)

Note:

This rule is supported only by the classic router. For an equivalent rule supported by Zroute, see [“End-of-Line to End-of-Line Spacing Rule” on page 2-61](#).

Use the `stubToStubSpacing` and `endOfLineCornerKeepoutWidth` attributes as well as the `stubSpacing` attribute to specify the end-of-line to end-of-line spacing rule, also known as the tip-to-tip spacing rule. Define these attributes in a metal `Layer` section of the technology file.

The `stubToStubSpacing` specifies the spacing between two end-of-line wire segments. The `endOfLineCornerKeepoutWidth` specifies the parallel projection between the two end-of-line edges, defining a keepout region.

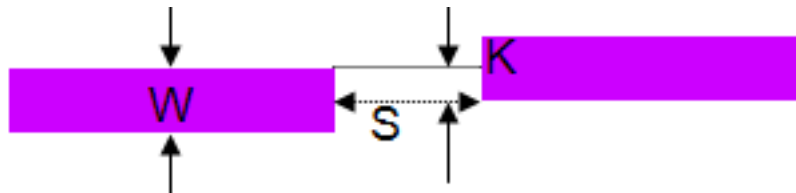
Also, you must specify a value of 1 for the `stubMode` attribute in the `Technology` section of the technology file.

For example,

```
Technology {
    stubMode    = 1
}
Layer "M2" {
    layerNumber      = 3
    maskName         = "metal2"
    stubSpacing      = 0.088
    stubToStubSpacing = 0.104
    stubThreshold    = 0.288
    endOfLineCornerKeepoutWidth = -0.002
}
```

Figure 2-84 shows how this syntax is interpreted.

Figure 2-84 End-of-Line to End-of-Line Spacing Rule (Classic Router)



- When the wire Mx width (W) is less than 0.288, the minimum space (S) between an end-of-line wire and an end-of-line wire (tip-to-tip spacing) must be greater than or equal to 0.104.
- The wire must have ends overlapping projection (K). In this example, K is greater than or equal to 0.002.

L-Shaped End-of-Line Spacing Rule

L-shaped end-of-line spacing is applied when the end-of-line segment is part of an L-shape geometry and its length meets the given length threshold. The L-shape definition includes any shapes that have a partial L shape, such as a T shape.

Use the `stubLengthThreshold` attribute as well as the `stubSpacing` and `stubThreshold` attributes to specify this rule. Define these attributes in a metal `Layer` section of the technology file.

Also, you must specify a value of 1 for the `stubMode` attribute in the `Technology` section of the technology file.

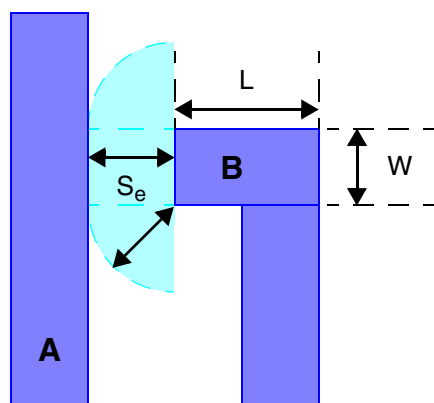
The `stubSpacing` attribute defines the end-of-line edge or corner of the metal whose edge width is less than or equal to the given `stubThreshold`. An L shape with an adjacent segment on the same layer is required, and at least one of the side edge lengths must be less than or equal to the given `stubLengthThreshold`. Otherwise, the default minimum spacing specified with `minSpacing` applies.

The L-shape rule is applied to the end-of-line segment edge as well as to the diagonal corner spacing, provided that the L-shape geometry and its length meet the given length threshold.

For example,

```
Technology {
    stubMode = 1
}
Layer "M7" {
    minSpacing           = 0.20
    stubSpacing          = 0.22
    stubThreshold        = 0.24
    stubLengthThreshold  = 0.28
}
```

With the foregoing rule definitions, in [Figure 2-85](#), the L-shaped end-of-line spacing rule applies only when the length of the segment L is less than 0.28 (`stubLengthThreshold`) and the width of the segment W is less than 0.24 (`stubThreshold`). If both of these conditions are true, then the minimum spacing is S_e (`minSpacing`); otherwise, it is the default metal-to-metal minimum spacing.

Figure 2-85 L-Shaped End-of-Line Spacing Rule

This rule is typically set to be less restrictive than the ordinary end-of-line minimum spacing rule, thus conserving routing resources for the L-shaped case versus the normal case.

End-of-Line Depth Rule

Note:

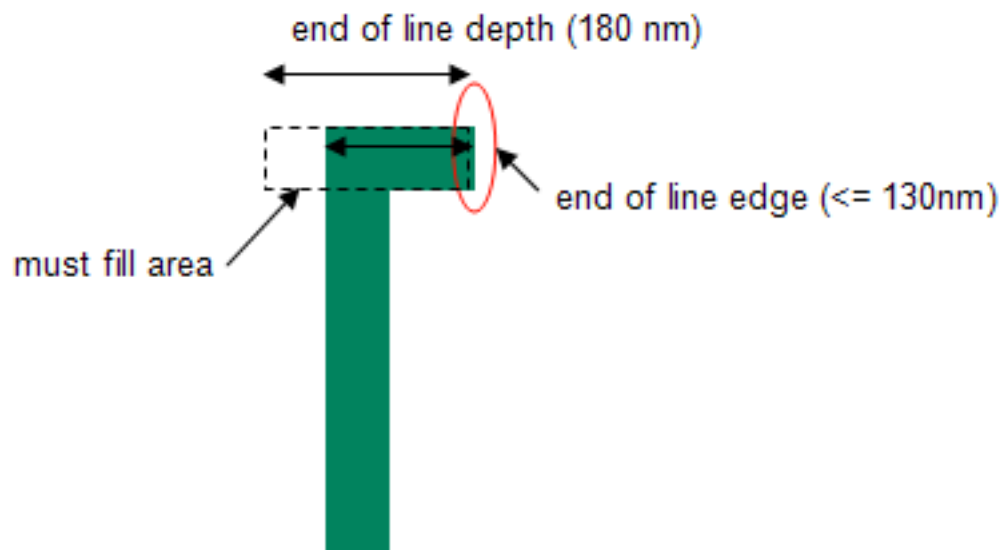
This rule is supported only by the classic router; Zroute does not support this rule.

Short segments near line ends cannot always be resolved on the wafer and can create problems for optical control systems. To avoid this situation, use the end-of-line depth rule. This rule specifies that the distance from the end-of-line edge to the opposite internal edge is to be greater than or equal to the value you define with the `stubMinLength` attribute for a stub with a line end width less than or equal to the value you define with the `stubThreshold` attribute. Define these attributes in a metal `Layer` section of the technology file.

For example,

```
Layer "Metal1" {
    stubThreshold = 0.130
    stubMinLength = 0.180
}
```

[Figure 2-86](#) shows the attributes associated with the end-of-line depth rule.

Figure 2-86 End-of-Line Depth Rule

- The `stubThreshold` value defines the end-of-line edge width. The end-of-line edge is any edge less than or equal to 0.130 connected to two convex vertices.
- The `stubMinLength` value defines a “must fill” area, starting from the end-of-line edge. If the must fill area is not filled, a violation is reported.

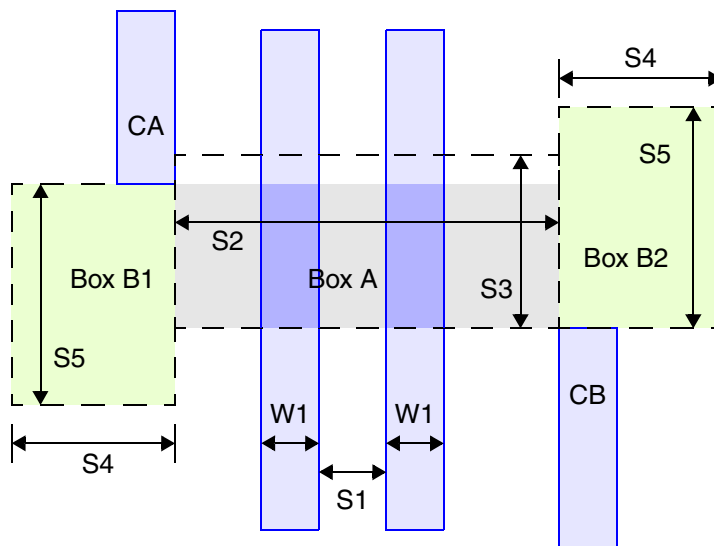
Bridge Rules

The bridge rules specify the minimum spacing between metal geometries near two closely spaced parallel metal lines and line-ends, with no other nearby metal geometries. The rules apply only when the parallel lines have specific width and spacing values. There are three different bridge rules: Bridge Rule 1, Bridge Rule 2, and Bridge Rule 3.

Bridge Rule 1

Bridge rule 1 specifies the minimum offset between two nearby metal line-ends on either side of two parallel metal lines, as shown in [Figure 2-87](#).

Figure 2-87 Bridge Rule 1

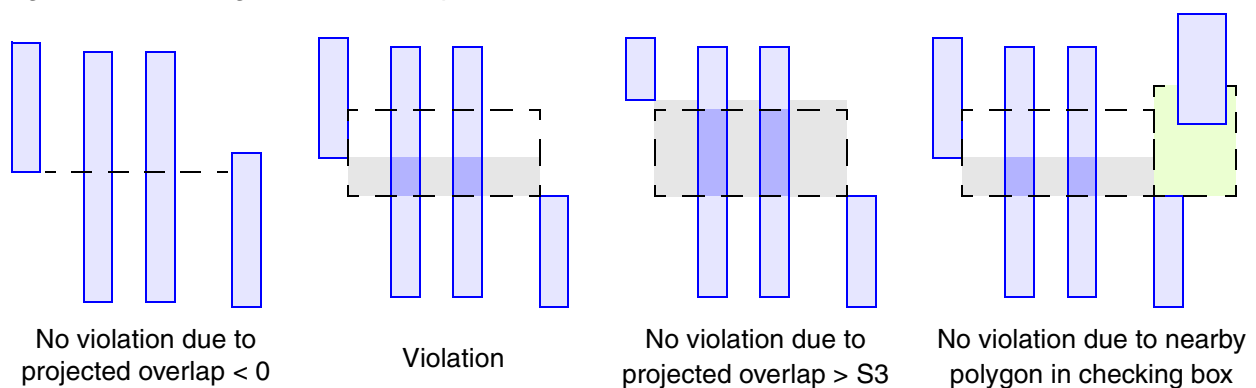


The two parallel metal lines have a width of exactly $W1$ and a spacing of exactly $S1$, and they fully cross the projection area between the two outside line-ends, shown as Box A in the figure. The two line-ends of metal CA and CB, if extended longer, would have a spacing of exactly $S2$.

A violation occurs when the projected overlap between the line-ends is greater than zero but less than $S3$. In other words, a violation occurs when the projections of the two line-ends are close together but do not overlap. The rule does not apply if there is any metal polygon that covers or touches the green checking boxes, Box B1 and Box B2, each measuring $S4$ by $S5$.

Figure 2-88 shows some examples of where the rule does and does not apply.

Figure 2-88 Bridge Rule 1 Examples



This is the general syntax of the rule:

```
Layer "MetalX" {  
    diffSideKeepoutMidWireExactWidth    = W1  
    diffSideKeepoutMidWireExactSpacing  = S1  
    diffSideKeepoutNumMidWires           = 2  
    diffSideKeepoutSideExactSpacing     = S2  
    diffSideKeepoutEndMinSpacing        = S3  
    diffSideKeepoutWidth                 = S4  
    diffSideKeepoutLength                = S5  
}
```

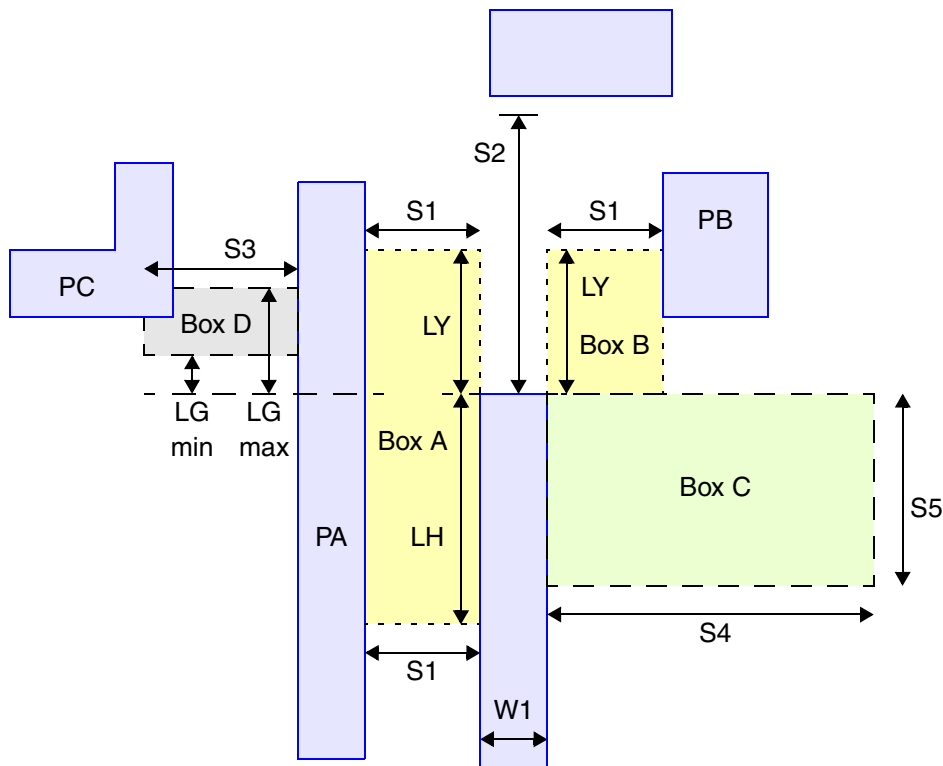
For example,

```
Layer "M1" {  
    diffSideKeepoutMidWireExactWidth    = 0.056  
    diffSideKeepoutMidWireExactSpacing  = 0.056  
    diffSideKeepoutNumMidWires           = 2  
    diffSideKeepoutSideExactSpacing     = 0.280  
    diffSideKeepoutEndMinSpacing        = 0.260  
    diffSideKeepoutWidth                 = 0.180  
    diffSideKeepoutLength                = 0.270  
}
```

Bridge Rule 2

Bridge rule 2 specifies the minimum spacing between a metal polygon and a line-end with another parallel line between them, as shown in [Figure 2-89](#).

Figure 2-89 Bridge Rule 2



The two parallel metal lines in the center have a width of exactly $W1$ and a spacing of exactly $S1$. One of the lines has an end and there is no other polygon found in the path of that line from the line-end within a distance of $S2$.

Box A in the diagram is fully butted against the side of the metal with the line-end and a nearby polygon, PA. Box A measures exactly $S1$ by $(LH+LY)$.

Box B in the diagram is formed at the line-end corner, away from Box A and away from the metal line, and touches the corner of another polygon, PB. Box B measures $S1$ by LY .

Box C in the diagram is formed at the line-end corner along the side of the metal line, measuring $S4$ by $S5$. There is no other metal covering or touching this box.

When all of these conditions are met, the rule checks for the presence of a metal polygon PC on the far side of PA, which enters into an area shown as Box D in the diagram. The presence of such a metal polygon in Box D is a violation of the rule. This violation occurs if the distance between the polygon PC and metal PA is no more than $S3$ and the projected line-end distance is greater than $LG\ min$ and not greater than $LG\ max$.

This is the general syntax of the rule:

```

Layer "MetalX" {
  endSideKeepoutMidWireExactWidth      = W1
  endSideKeepoutMidWireExactSpacing    = S1
  endSideKeepoutMidWireEndSpacing      = S2
  endSideKeepoutMidWireMinLength       = LL
  endSideKeepoutNeighborWireMaxWidth   = PA,
  endSideKeepoutParallelLength         = LH
  endSideKeepoutParallelLengthExtension = LY
  endSideKeepoutEndMinSpacing          = LG min
  endSideKeepoutEndMaxSpacing          = LG max
  endSideKeepoutSideMinSpacing         = S3
  endSideKeepoutWidth                  = S4
  endSideKeepoutLength                 = S5
}

```

For example,

```

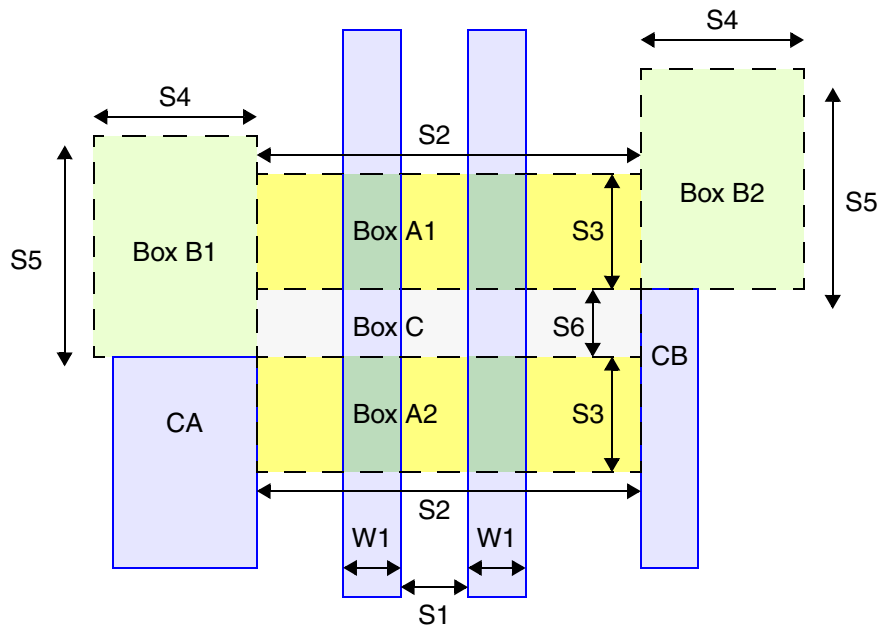
Layer "M1" {
  endSideKeepoutMidWireExactWidth      = 0.056
  endSideKeepoutMidWireExactSpacing    = 0.056
  endSideKeepoutMidWireEndSpacing      = 0.340
  endSideKeepoutMidWireMinLength       = 0.340
  endSideKeepoutNeighborWireMaxWidth   = 0.062
  endSideKeepoutParallelLength         = 0.133
  endSideKeepoutParallelLengthExtension = 0.090
  endSideKeepoutEndMinSpacing          = 0.035
  endSideKeepoutEndMaxSpacing          = 0.112
  endSideKeepoutSideMinSpacing         = 0.062
  endSideKeepoutWidth                  = 0.260
  endSideKeepoutLength                 = 0.084
}

```

Bridge Rule 3

Bridge rule 3 is similar to bridge rule 1, except that the two line-ends on either side of the parallel lines face the same direction rather than opposite directions. The rule specifies the minimum offset between two metal line-ends, as shown in [Figure 2-90](#).

Figure 2-90 Bridge Rule 3



The two straight, parallel metal lines in the center have a width of exactly $W1$ and a spacing of exactly $S1$, and they fully cross the regions shown as Box A1, Box C, and Box A2. Box A1 and Box A2 extend along the lengths of the metal lines by the amount $S3$. The two outside metal lines along the sides, CA and CB, have a spacing of exactly $S2$. There are no other metal polygons touching or covering Box B1 and Box B2, each box measuring $S4$ by $S5$. If all these conditions are met, the CA and CB line-ends must be offset by more than $S6$.

This is the general syntax of the rule:

```

Layer "MetalX" {
  sameSideKeepoutMidWireExactWidth      = W1
  sameSideKeepoutMidWireExactSpacing    = S1
  sameSideKeepoutNumMidWires             = 2
  sameSideKeepoutSideExactSpacing        = S2
  sameSideKeepoutEndExtensionRange       = S3
  sameSideKeepoutEndMinOffset            = S6
  sameSideKeepoutWidth                   = S4
  sameSideKeepoutLength                   = S5
}

```

For example,

```

Layer "M1" {
  sameSideKeepoutMidWireExactWidth      = 0.06
  sameSideKeepoutMidWireExactSpacing    = 0.06
  sameSideKeepoutNumMidWires             = 2
  sameSideKeepoutSideExactSpacing        = 0.30
  sameSideKeepoutEndExtensionRange       = 0.11
}

```

```
sameSideKeepoutEndMinOffset      = 0.06
sameSideKeepoutWidth             = 0.20
sameSideKeepoutLength            = 0.14
}
```

Fat Metal Contact Rules

The fat metal contact rules are described in the following sections:

- [Fat Metal Contact Rule](#)
- [Area-Based Fat Metal Contact Rule](#)
- [Fat Metal Extension Contact Rule](#)
- [Area-Based Fat Metal Extension Contact Rule](#)
- [Alternative Syntax for Fat Metal Contact and Extension Rules](#)
- [Fat Poly Contact Rule](#)

Fat Metal Contact Rule

The fat metal contact rule specifies the minimum number of vias required to connect between two metal layers when either of the metal segments is a fat wire. You can specify a fat metal contact rule by using either a one-dimensional or two-dimensional table rule. When you use a one-dimensional table rule, the thresholds are determined by the maximum width of the upper and lower layer metal segments. When you use a two-dimensional rule, you specify separate thresholds for the upper-metal and lower-metal layers. You can define a fat metal extension contact rule by specifying an extension threshold for the fat metal contact rule.

One-Dimensional Table Rule

Note:

The newer rule described in [“Area-Based Fat Metal Contact Rule” on page 2-101](#) is preferred to this older rule. Support for this older rule will end in a future release.

To define a one-dimensional fat metal contact rule, specify the following attributes in a via `Layer` section of the technology file:

- `fatTblDimension`

Specifies the size of the one-dimensional fat table.

- `fatTblThreshold`

Specifies the thresholds used to determine which rules apply, based on the maximum width of the metal segments on the upper-metal and lower-metal layers. The width is the smaller of the x-length or y-length of the metal segment. You must specify n values, where n is the table dimension.

- `fatTblFatContactNumber`

Specifies the contact code numbers for the fat wires. You must specify n values, where n is the table dimension.

- `fatTblFatContactMinCuts`

Specifies the minimum number of cuts. You must specify n values, where n is the table dimension.

In the following example, if either of the metal widths connected by the VIA1 layer is greater than or equal to 0.155, then a via whose `contactCode` value is defined as 21 will be used and at least two cuts of this via will be necessary.

```
layer    "VIA1" {
    fatTblDimension      = 3
    fatTblThreshold      = (0, 0.155, 1.605)
    fatTblFatContactNumber = (1, 21, 31)
    fatTblFatContactMinCuts = (1, 2, 4)
}
```

Two-Dimensional Table Rule

Note:

The newer rule described in [“Area-Based Fat Metal Contact Rule” on page 2-101](#) is preferred to this older rule. Support for this older rule will end in a future release.

Both the power and ground routing and the detail routing operations honor the two-dimensional table rule. A rule violation is flagged as a DRC error.

To define a two-dimensional fat metal contact rule, specify the following attributes in a via `Layer` section of the technology file:

- `fatTblDimension`

Specifies the size of the two-dimensional fat table.

- `fatTblThreshold`

Specifies the thresholds used to determine which rules apply, based on the width of the metal segment on the lower layer. The width is the smaller of the x-length or y-length of the metal segment. You must specify n values, where n is the table dimension.

- `fatTblThreshold2`

Specifies the thresholds used to determine which rules apply, based on the width of the metal segment on the upper layer. The width is the smaller of the x-length or y-length of the metal segment. You must specify n values, where n is the table dimension.

- `fat2DTblFatContactNumber`

Specifies the contact code numbers used to select vias from the library. You must specify an $n \times n$ table, where n is the table dimension.

- `fat2DTblFatContactMinCuts`

Specifies the minimum number of cuts. You must specify an $n \times n$ table, where n is the table dimension.

Note:

In the two-dimensional fat contact tables, the horizontal index is for the lower-metal layer (`fatTblThreshold`) and the vertical index is for the upper-metal layer (`fatTblThreshold2`).

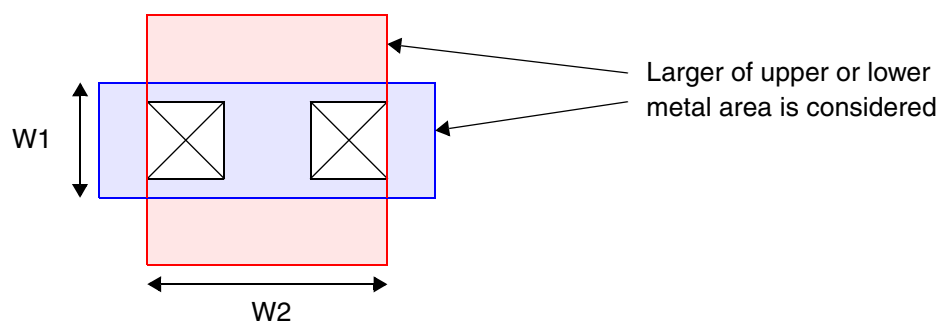
In the following example, if the lower-metal width value is greater than or equal to 0.421 and less than 0.701, the tool considers that value as element 2 of `fatTblThreshold`. If the upper-metal width value is greater than or equal to 0.501 and less than 0.781, then the tool considers that value as element 3 of `fatTblThreshold2`. Consequently, the tool picks up the (2,3) element value from `fat2DTblFatContactNumber`, which is 18, and the (2,3) element from `fat2DTblFatContactMinCuts`, which is 3. So, IC Compiler uses the via whose `contactCode` value is 18 and uses at least three cuts of the via to connect the lower and upper-metal layers.

```
"Layer "VIA23" {
  fatTblDimension = 4
  /* Threshold for lower layer metal width */
  fatTblThreshold = (0.0, 0.421, 0.701, 0.981)
  /* Threshold for upper layer metal width */
  fatTblThreshold2 = (0.0, 0.221, 0.501, 0.781)
  fat2DTblFatContactNumber = (1, 18, 17, 20,
                              20, 17, 18, 20,
                              21, 18, 18, 19,
                              20, 20, 20, 20)
  fat2DTblFatContactMinCuts = (1, 2, 2, 2,
                              2, 3, 3, 4,
                              4, 4, 4, 6,
                              4, 4, 4, 4)
}
```

Area-Based Fat Metal Contact Rule

The area-based fat metal contact rule specifies the contact code numbers of the allowed vias and the minimum number of vias required to connect between two metal layers when either of the metal segments is a fat wire and the number of vias depends on the area. The table dimensions determine the number of width ranges considered for each of the two metal layers. The larger of the upper-metal layer area or the lower-metal layer area is compared to the area threshold. See [Figure 2-91](#).

Figure 2-91 Area-Based Fat Metal Contact Rule



The following example demonstrates usage of the rule.

```

Layer "V1" {
  ...
  fatTblDimension           = 5
  fatTblThreshold           = (0.0,0.09,0.16,0.35,0.48)
  fatTblDimension2         = 3
  fatTblThreshold2         = (0,0.28,0.50)
  fatTblAreaDimension       = 2
  fatTblAreaThreshold       = (0.0,0.78)
  fatTblFatContactNumber    = ( "7,17,77,37", "17,77,37", "127,77,87",
                                47           , "127,77,87", "127,77,87",
                                107          , "127,77,87", "127,77,87",
                                "127,77,87", "127,77,87", "127,77,87",
                                "127,77,87", "127,77,87", "127,77,87",
                                "7,17,77,37", "127,77,87", "127,77,87",
                                47           , "127,77,87", "127,77,87",
                                107          , "127,77,87", "127,77,87",
                                "127,77,87", "127,77,87", "127,77,87",
                                "127,77,87", "127,77,87", "127,77,87")
  fatTblFatContactMinCuts   = ( "1,1,1,1", "2,1,1", "4,3,3",
                                1           , "2,1,1", "4,3,3",
                                1           , "2,1,1", "4,3,3",
                                "3,2,2", "3,2,2", "3,2,2", "4,3,3",
                                "4,3,3", "4,3,3", "4,3,3", "4,3,3",
                                "2,2,2", "3,2,2", "4,3,3")
}

```

```

2      , "3,2,2", "4,3,3"
2      , "4,3,3", "4,3,3"
"4,3,3", "4,3,3", "4,3,3"
"5,4,4", "5,4,4", "5,4,4")

```

The `fatTblDimension` attribute is the number of lower-metal width ranges considered, whereas `fatTblDimension2` is the number of upper-metal width ranges considered. In the previous example, the five width ranges for the lower-metal layer and three width ranges for the upper-metal layer require a 5-by-3 table of fat contact numbers. The two area ranges require two of each 5-by-3 table, one for areas below the area threshold and another for areas greater than or equal to the area threshold. The same is true for the table that defines the minimum number of cuts.

If the area dimension and area threshold are not specified, the rule does not consider area and it becomes a 2-dimensional fat metal contact rule. For example,

```

Layer "V1" {
  ...
  fatTblDimension      = 5
  fatTblThreshold      = (0.0,0.09,0.16,0.35,0.48)
  fatTblDimension2     = 3
  fatTblThreshold2     = (0,0.28,0.50)
  fatTblFatContactNumber = ( "7,17,77,37", "17,77,37", "127,77,87",
                             47          , "127,77,87", "127,77,87",
                             107         , "127,77,87", "127,77,87",
                             "127,77,87", "127,77,87", "127,77,87",
                             "127,77,87", "127,77,87", "127,77,87")
  fatTblFatContactMinCuts = ( "1,1,1,1", "2,1,1", "4,3,3",
                              1          , "2,1,1", "4,3,3",
                              1          , "2,1,1", "4,3,3",
                              "3,2,2", "3,2,2", "3,2,2", "4,3,3",
                              "4,3,3", "4,3,3", "4,3,3", "4,3,3")

```

For any new design rule sets, this simpler form of the general area-based rule should be used instead of the older rule described in [“Two-Dimensional Table Rule” on page 2-99](#).

If the area dimension, area threshold, second-metal width dimension, and second-metal width thresholds are not specified, the rule becomes a conventional 1-dimensional fat metal contact rule that depends only on the widths of the two metal layers. For example,

```

Layer "V1" {
  ...
  fatTblDimension      = 5
  fatTblThreshold      = (0.0,0.09,0.16,0.35,0.48)
  fatTblFatContactNumber = ("7,17,77,37",47,107,"127,77,87","127,77,87")
  fatTblFatContactMinCuts = ("1,1,1,1",1,1,"3,2,2","4,3,3")

```

For any new design rule sets, this simpler form of the general area-based rule should be used instead of the older rule described in [“One-Dimensional Table Rule” on page 2-98](#).

To have the rule consider the via's enclosure shape when computing the metal width for the fat metal via rule, set the `fatTblThresholdIncludeEnclosure` attribute to 1, as shown in the following example:

```
Layer "V1" {
  ...
  fatTblThresholdIncludeEnclosure = 1
  fatTblDimension           = 5
  fatTblThreshold           = (0.0,0.09,0.16,0.35,0.48)
  fatTblDimension2          = 3
  fatTblThreshold2          = (0,0.28,0.50)
  fatTblAreaDimension       = 2
  fatTblAreaThreshold       = (0.0,0.78)
  ...
}
```

When the `fatTblThresholdIncludeEnclosure` attribute is set to 1, the router considers both the metal wire shape and the via's enclosure shape for computing metal width for a 1-dimensional, 2-dimensional, or 3-dimensional area-based fat metal rule. If this attribute is set to 0 or omitted entirely from the technology file, the router considers only the metal wire's shape.

Fat Metal Extension Contact Rule

Note:

The newer rule described in [“Area-Based Fat Metal Extension Contact Rule” on page 2-105](#) is preferred to this older rule. Support for this older rule will end in a future release.

The fat metal extension contact rule determines when the fat metal contact rule applies to normal wires that extend from a fat wire. In [Figure 2-44 on page 2-51](#), the portion of the extension wire within the extension threshold, *L*, uses the fat metal contact rules.

To define a fat metal extension contact rule, specify the following attributes in a via `Layer` section of the technology file:

- `fatTblDimension`
Specifies the size of the one-dimensional fat table.
- `fatTblThreshold`
Specifies the thresholds used to determine which rules apply, based on the maximum width of the metal segments on the upper-metal and lower-metal layers. The width is the smaller of the x-length or y-length of the metal segment. You must specify *n* values, where *n* is the table dimension.

- `fatTblExtensionRange`
Specifies the extension range thresholds. You must specify n values, where n is the table dimension.
- `fatTblExtensionContactNumber`
Specifies the contact code numbers used to select vias from the library. You must specify n values, where n is the table dimension.
- `fatTblExtensionContactMinCuts`
Specifies the minimum number of vias for the wires within the extension range of fat wires. You must specify n values, where n is the table dimension.

For example,

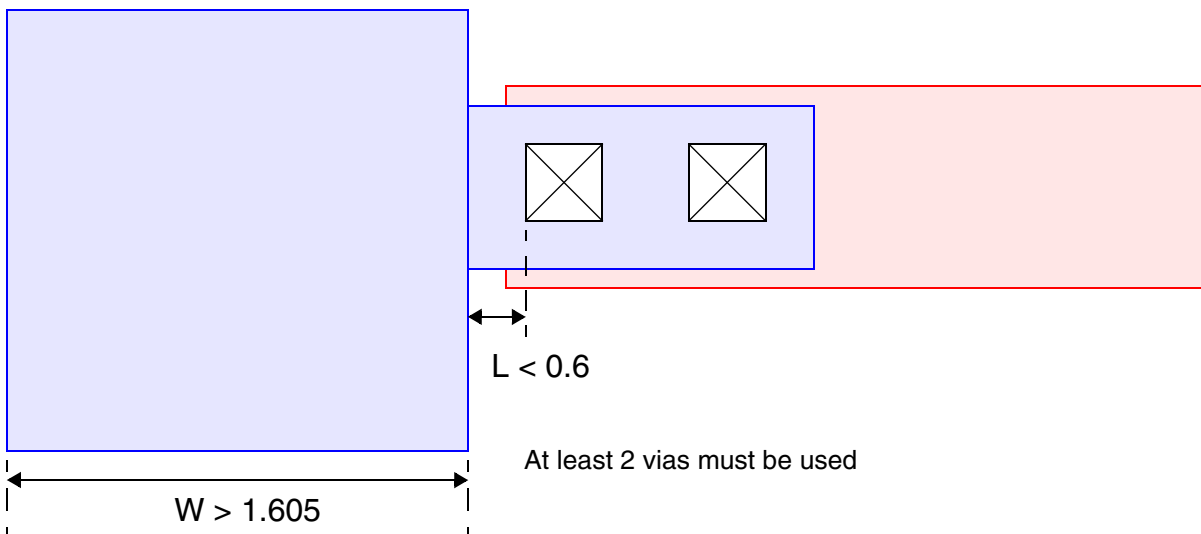
```

Layer "VIA1" {
  fatTblDimension           = 3
  fatTblThreshold           = (0, 0.155, 1.605)
  fatTblExtensionRange      = (0, 0, 0.6)
  fatTblExtensionContactNumber = (1, 21, 31)
  fatTblExtensionMinCuts    = (1, 1, 2)
}

```

In [Figure 2-92](#), if the fat metal width is greater than 1.605 and the fat metal extension range is less than 0.6, contact code number 31 is used with at least 2 via cuts for a connection between the upper-metal and lower-metal layers.

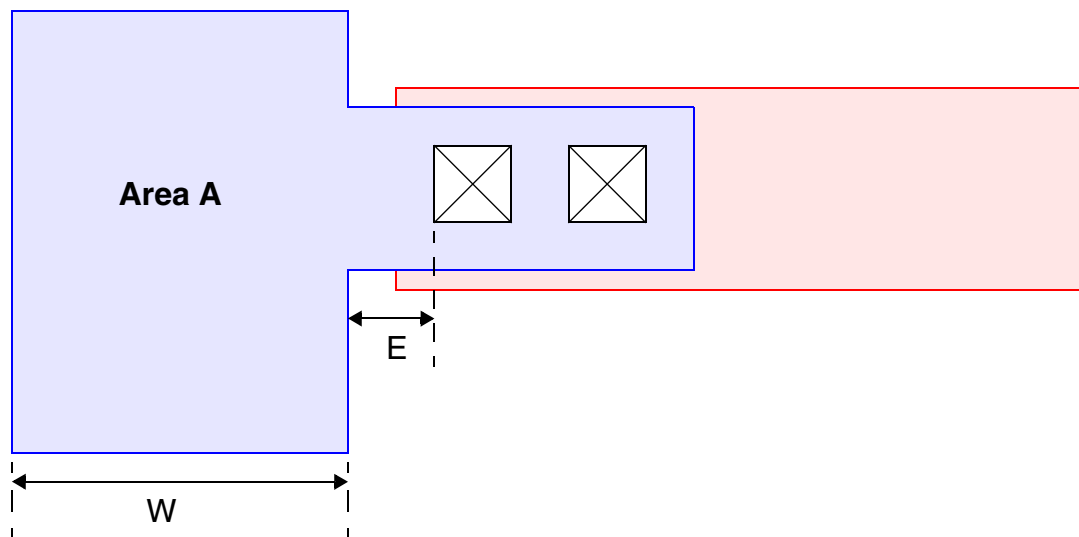
Figure 2-92 Fat Metal Extension Contact Rule



Area-Based Fat Metal Extension Contact Rule

The area-based fat metal extension contact rule considers the area of the wide metal A as well as the metal's width W and extension range E, shown in [Figure 2-93](#).

Figure 2-93 Area-Based Fat Metal Extension Contact Rule



The contact code numbers of the allowable vias and the minimum number of cuts used for connecting the fat metal shape and the extended shape depend on the fat metal's width W, area A, and extension range E. Here is an example:

```
Layer "V1" {
  ...
  fatTblExtensionDimension = 3
  fatTblExtensionThreshold = (0.20, 1.10, 1.90 )
  fatTblExtensionRangeDimension = 3
  fatTblExtensionRange = ( 1.1, 2.1, 5.6 )
  fatTblExtensionAreaDimension = 4
  fatTblExtensionAreaThreshold = ( 0, 0.046, 1.05, 9.20 )
  fatTblExtensionContactNumber =
    ("7,17,27,37,47", "7,17,27,37,47", "7,17,27,37,47",
     "7,17,27,37,47", "7,17,27,37,47", "7,17,27,37,47",
     "7,17,27,37,47", "7,17,27,37,47", "7,17,27,37,47",
     "57,67", "7,17,27,37,47", "7,17,27,37,47",
     "57,67", "7,17,27,37,47", "7,17,27,37,47",
     "57,67", "7,17,27,37,47", "7,17,27,37,47",
     "57,67", "7,17,27,37,47", "7,17,27,37,47",
     "57,67", "57,67", "7,17,27,37,47",
     "57,67", "57,67", "7,17,27,37,47",
     "57,67", "7,17,27,37,47", "7,17,27,37,47",
     "57,67", "57,67", "7,17,27,37,47",
     "57,67", "57,67", "57,67" )
}
```

```

; 1st 3x3 table is for area < 0.046
; 2nd 3x3 table is for 0.046 <= area < 1.05
; 3rd 3x3 table is for 1.05 <= area < 9.20
; 4th 3x3 table is for 9.20 <= area

fatTblExtensionMinCuts =
( "1,1,1,1,1", "1,1,1,1,1", "1,1,1,1,1",
  "1,1,1,1,1", "1,1,1,1,1", "1,1,1,1,1",
  "1,1,1,1,1", "1,1,1,1,1", "1,1,1,1,1",
    "2,2", "1,1,1,1,1", "1,1,1,1,1",
    "2,2", "1,1,1,1,1", "1,1,1,1,1",
    "2,2", "1,1,1,1,1", "1,1,1,1,1",
    "2,2", "1,1,1,1,1", "1,1,1,1,1",
    "2,2", "2,2", "1,1,1,1,1",
    "2,2", "2,2", "1,1,1,1,1",
    "2,2", "1,1,1,1,1", "1,1,1,1,1",
    "2,2", "2,2", "1,1,1,1,1",
    "2,2", "2,2", "2,2")
; 1st 3x3 table is for area < 0.046
; 2nd 3x3 table is for 0.046 <= area < 1.05
; 3rd 3x3 table is for 1.05 <= area < 9.20
; 4th 3x3 table is for 9.20 <= area

```

The three width ranges and three extension ranges require a 3-by-3 table of allowed contact code numbers. The four area ranges require four of each 3-by-3 table. The same is true for the table defining the minimum number of cuts.

A single entry of 0 in the contact number table is interpreted to mean that all contact numbers are acceptable for that combination of width, extension, and area. In that case, the corresponding entry in the minimum-cuts table is ignored. This feature is useful when you do not want area-based contact checking to be performed on default-width metal.

In the foregoing example, if the list 7, 17, 27, 37, 47 represents the entire list of all contact numbers, then each occurrence of that list in the contact number table can be replaced with a single "0" and each corresponding entry in the minimum-cuts table can be replaced with a single "1", which is ignored:

```

fatTblExtensionContactNumber =
(0,0,0,
 0,0,0
 0,0,0,
  "57,67",0,0,
  ...
fatTblExtensionMinCuts =
(1,1,1,
 1,1,1
 1,1,1,
  "2,2",1,1,
  ...

```

If the rule does not have an area threshold, specify the area dimension as one and the area threshold as zero, as shown in the following example:

```
Layer "V1" {
  ...
  fatTblExtensionDimension = 3
  fatTblExtensionThreshold = (0.20, 1.10, 1.90 )
  fatTblExtensionRangeDimension = 3
  fatTblExtensionRange = ( 1.1, 2.1, 5.6 )
  fatTblExtensionAreaDimension = 1
  fatTblExtensionAreaThreshold = (0)
  fatTblExtensionContactNumber =
    ("7,17,27,37,47", "7,17,27,37,47", "7,17,27,37,47",
     "57,67",          "7,17,27,37,47", "7,17,27,37,47",
     "57,67",          "57,67",          "57,67"          )
  fatTblExtensionMinCuts =
    ( "1,1,1,1,1", "1,1,1,1,1", "1,1,1,1,1",
      "2,2",       "1,1,1,1,1", "1,1,1,1,1",
      "2,2",       "2,2",       "2,2"          )
}
```

For any new design rule sets, this simpler form of the general area-based rule should be used instead of the older rule described in [“Fat Metal Extension Contact Rule” on page 2-103](#).

To restrict the selection of contacts to only those contacts having a width that exactly matches the specified metal width, add the following line:

```
fatTblSelectExactContactCodeNumber = 1
```

If the router cannot find a contact in the allowed list that has the exact width, it reports this condition as a DRC error.

Alternative Syntax for Fat Metal Contact and Extension Rules

You can optionally use an alternative form of syntax similar to Library Exchange Format (LEF) to specify the attributes for the [Area-Based Fat Metal Contact Rule](#) and the [Fat Metal Extension Contact Rule](#). Using this form of syntax, you do not need to define extra contact codes or fill out the long cut tables. All of this information is derived automatically from data provided in LEF-like format.

To define the rules, you need to specify a minimum set of preliminary data, including the cut sizes, via enclosures, minimum number of cuts for each specified metal width/area/extension range, and the contact codes equivalent to LEF via definitions that are already defined.

The alternative syntax has two tables in the cut layer section: an enclosure table and a minimum cuts table:

```

Layer "ViaX" {

    cutNameTbl    = (VX, VXBAR, VXLRG)    # cut names
    cutWidthTbl   = (0.05, 0.05, 0.10)    # cut widths
    cutHeightTbl  = (0.05, 0.10, 0.10)    # cut heights

    enclosureTblSize = 1    # number of enclosureTbl rows
    enclosureTbl     = (cutName, lowEncWidth, lowEncHeight, upEncWidth,
                        upEncHeight, lowMetalWidth, upMetalWidth)

    minCutsTblSize  = 1    # number of minCutsTbl rows
    minCutsTbl      = (numCuts, cutName, lowMetalWidth, upMetalWidth,
                        area, extension)
}

```

The width and height values can be swapped, but they must be defined consistently throughout the rule definition. The `enclosureTbl` definition is similar to the `ENCLOSURE` rule in LEF and the `minCutsTbl` definition is similar to the `MINIMUMCUT` rule in LEF.

Alternative Syntax Example

Here is an example of a fat metal contact definition in LEF format:

```

LAYER V1
    CUTCLASS VX          WIDTH 0.05 ;
    CUTCLASS VXBAR       WIDTH 0.05 LENGTH 0.10 ;
    PROPERTY LEF58_ENCLOSURE "
        ENCLOSURE CUTCLASS VX BELOW 0.014 0.000 WIDTH 0.051 ;
        ENCLOSURE CUTCLASS VX ABOVE 0.032 0.000 ;
LAYER M1
    MINIMUMCUT 2 WIDTH 1.800 FROMABOVE AREA 8.999 WITHIN 5.501 ;

```

Here is the corresponding technology file definition using the alternative form of syntax:

```

Layer "V1" {
    cutNameTbl    = (VX, VXBAR)
    cutWidthTbl   = (0.05,0.05)
    cutHeightTbl  = (0.05,0.10)
    enclosureTblSize = 2
    enclosureTbl   = (VX,0.014,0.000,-1,-1,0.051,-1,
                    VX,-1,-1,0.032,0.000,-1,0.051)
    minCutsTblSize = 1
    minCutsTbl     = (2,VX,1.801,1.801,9.0,5.5)
}

```

For exact equivalence, the `MINIMUMCUT WIDTH` and `AREA` values must be increased by one database unit in the technology file definition. Similarly, `WITHIN` values must be decreased by one database unit in the technology file definition.

Conversion to Standard Syntax

When you specify a rule using this alternative syntax, IC Compiler internally converts the rule to standard syntax described in the foregoing sections, [Area-Based Fat Metal Contact Rule](#) and [Fat Metal Extension Contact Rule](#). IC Compiler automatically combines input entries in the alternative syntax and fills in the tables using default data, if applicable.

To verify the accuracy of the converted rules, you can optionally write them out from the IC Compiler tool. To do so, set the conversion control variable, `write_converted_tf_syntax`, to `true` before you write out the technology file:

```
icc_shell> set_app_var write_converted_tf_syntax true
...
icc_shell> write_mw_lib_files -technology -output my_tech.tf my_mw_lib.mw
...
```

By default, the control variable is set to `false`, which causes the technology file to be written out using the same syntax used to read it in.

The following example demonstrates the conversion process. Suppose that you specify the area-based fat metal contact rule and extension contact rule using LEF-like syntax, as follows:

```
Layer "V2" {
  cutTblSize      = 4
  cutNameTbl      = (Vsm,Vv,Vh,Vlg)
  cutWidthTbl     = (0.06,0.06,0.12,0.12)
  cutHeightTbl    = (0.06,0.12,0.06,0.12)
  cutDataTypeTbl  = (0,1,1,15)
  enclosureTblSize = 6
  enclosureTbl    = (Vsm,-1.000,-1.000, 0.006, 0.006,-1.000, 0.226,
                    Vh, -1.000,-1.000, 0.021,-1.000,-1.000, 0.226,
                    Vsm,-1.000,-1.000, 0.010, 0.010,-1.000, 0.402,
                    Vsm, 0.010, 0.010,-1.000,-1.000, 0.130,-1.000,
                    Vsm, 0.018, 0.018,-1.000,-1.000, 0.180,-1.000,
                    Vh, 0.021,-1.000,-1.000,-1.000, 0.180,-1.000)
  minCutsTblSize  = 8
  minCutsTbl      = (2,Vsm,-1.000,0.226,-1.000,-1.000,
                    1,Vh,-1.000,0.226,-1.000,-1.000,
                    3,Vsm,-1.000,0.402,-1.000,-1.000,
                    4,Vsm,-1.000,0.550,-1.000,-1.000,
                    1,*,0.130,-1.000,-1.000,2.000,
                    2,Vsm,0.180,-1.000,-1.000,3.000,
                    1,Vh, 0.180,-1.000,-1.000,-1.000)
}
```

IC Compiler selects the equivalent contact codes from the technology file and obtains the attributes for these contacts:

<pre>ContactCode "VIA02F" { contactCodeNumber = 23 cutLayer = "V2"</pre>	<pre>ContactCode "VIA02B" { contactCodeNumber = 24 cutLayer = "V2"</pre>
---	---

```

lowerLayer      = "M2 "
upperLayer      = "M3 "
cutWidth        = 0.06
cutHeight       = 0.06
upperLayerEncWidth = 0.021
upperLayerEncHeight = 0.021
lowerLayerEncWidth = 0.021
lowerLayerEncHeight = 0.021
}
ContactCode "VIA02U" {
  contactCodeNumber = 89
  cutLayer          = "V2 "
  lowerLayer        = "M2 "
  upperLayer        = "M3 "
  isDefaultContact  = 1
  cutWidth          = 0.06
  cutHeight         = 0.06
  upperLayerEncWidth = 0.010
  upperLayerEncHeight = 0.010
  lowerLayerEncWidth = 0.021
  lowerLayerEncHeight = 0
}
lowerLayer      = "M2 "
upperLayer      = "M3 "
cutWidth        = 0.12
cutHeight       = 0.06
upperLayerEncWidth = 0.021
upperLayerEncHeight = 0
lowerLayerEncWidth = 0.021
lowerLayerEncHeight = 0
}

```

The `minCutsTbl` section and part of the `enclosureTbl` section are converted to via rules in standard format:

```

Layer "V2" {
  fatTblDimension          = 2
  fatTblThreshold          = (0,0.180)
  fatTblDimension2         = 4
  fatTblThreshold2         = (0,0.226,0.402,0.550)
  fatTblFatContactNumber   = (0, "89, 23, 24", "23, 24", "23, 24",
                              24, "23, 24", "23, 24", "23, 24")
  fatTblFatContactMinCuts  = (1, "2, 2, 1", "3, 1", "4, 1",
                              1, "2, 1", "3, 1", "4, 1")
  fatTblExtensionDimension = 5
  fatTblExtensionThreshold = (0,0.130,0.180,0.226,0.402)
  fatTblExtensionRangeDimension = 3
  fatTblExtensionRange     = (0,2,3)
  fatTblExtensionAreaDimension = 1
  fatTblExtensionAreaThreshold = (0)
  fatTblExtensionContactNumber = (0, 0, 0,
                                   "89, 24, 23", "89, 24, 23", 0,
                                   "89, 23, 24", "89, 23, 24", "89, 23",
                                   "89, 23, 24", "89, 23, 24", "89, 23",
                                   "89, 23, 24", "89, 23, 24", "89, 23")
  fatTblExtensionMinCuts    = (1, 1, 1,
                              "1, 1, 1", "1, 1, 1", 1,
                              "2, 2, 1", "2, 2, 1", "2, 2",
                              "2, 2, 1", "2, 2, 1", "2, 2",
                              "2, 2, 1", "2, 2, 1", "2, 2")
}

```


The default contact code is used for the `fatTblExtensionContactNumber` definition.

Fat Poly Contact Rule

The fat poly contact rule specifies whether IC Compiler allows wires and vias that are connecting to pins to create fat shapes that would otherwise trigger fat via rule violations. The fat via rules are defined in the poly contact `Layer` section that connects the pin and metal layers.

To control the scope of the check for the fat poly contact rule, set the `dontMakePinFat` detail route option. You set this option with the `set_droute_options` command. The option setting is stored with the cell.

```
icc_shell> set_droute_options -name dontMakePinFat -value value
```

Setting the value to 0 causes only normal checking to be performed. Setting the value to 1 causes checking of all pin connections to avoid creating new fat shapes that could cause either `fatVia` or `minEnclosedArea` rule violations.

Via Spacing Rules

The via spacing rules are described in the following sections:

- [Adjacent Via Rule](#)
- [Enclosed Via Spacing Rule](#)
- [Isolated Via Rule](#)
- [Misaligned Via Spacing Rule](#)

Adjacent Via Rule

The adjacent via rule specifies the maximum number of adjacent vias within a specified range of a via.

You define the adjacent via rule by setting the following attributes in a via `Layer` section of the technology file:

- The `maxNumAdjacentCut` attribute defines the maximum number of adjacent vias.
- The `adjacentCutRange` attribute defines the range within which to check for adjacent vias.

For example,

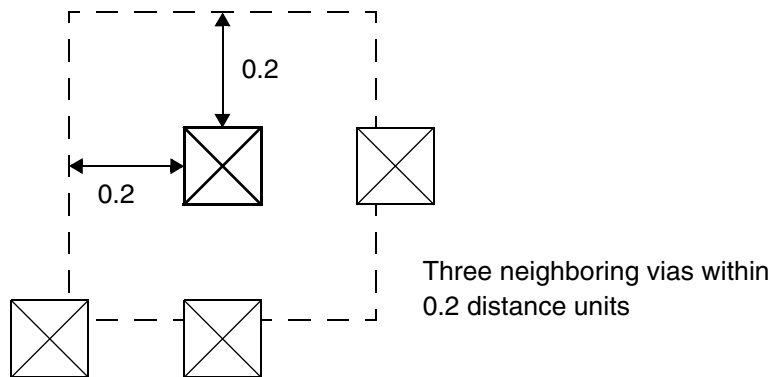
```

layer    "VIA1" {
  maxNumAdjacentCut    = 2
  adjacentCutRange     = 0.2
}

```

In [Figure 2-94](#), there are more than two adjacent vias within the specified distance range, thereby triggering a rule violation. It does not matter whether the adjacent vias belong to the same net or a different net.

Figure 2-94 Adjacent Via Rule



Enclosed Via Spacing Rule

A via surrounded by at least a specified number of vias within a specified distance range is called an enclosed via. You can specify the minimum distance between enclosed vias and the minimum distance between an enclosed via and a neighboring via. A neighboring via is a nearby via that is not an enclosed via.

Use the following attributes in a via `Layer` section of the technology file to specify the enclosed via spacing rule:

`enclosedCutNumNeighbor`

Specifies the minimum number of neighboring vias allowed for defining an enclosed via.

`enclosedCutNeighborRange`

Specifies the range of neighboring vias for defining an enclosed via.

`enclosedCutMinSpacing`

The minimum spacing between two enclosed vias.

`enclosedCutToNeighborMinSpacing`

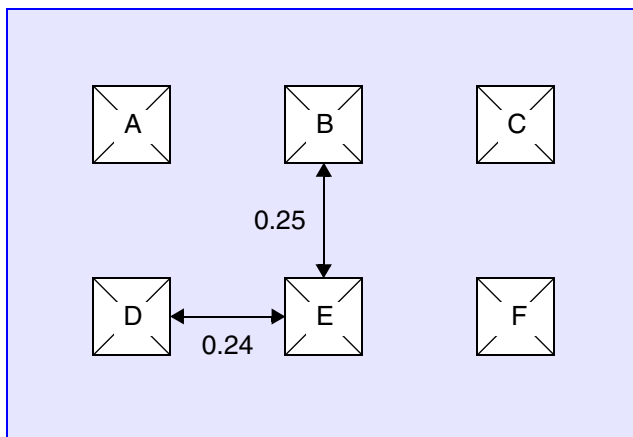
Specifies the minimum spacing allowed between an enclosed via and surrounding vias.

For example,

```
Layer "V3" {
  enclosedCutNumNeighbor      = 3
  enclosedCutNeighborRange    = 0.4
  enclosedCutMinSpacing      = 0.25
  enclosedCutToNeighborMinSpacing = 0.24
}
```

In this example, a via is defined to be an enclosed via when it has three or more nearby vias within a distance of 0.4. The minimum allowed spacing between two enclosed vias is 0.25. The minimum allowed spacing between an enclosed via and a neighboring, non-enclosed via is 0.24. In [Figure 2-95](#), vias B and E in the 3-by-2 via array are enclosed vias because they each are surrounded by three nearby vias, whereas vias A, C, D, and F are not enclosed vias. The minimum spacing between the two enclosed vias is 0.25, whereas the minimum spacing between an enclosed and neighboring, non-enclosed via is 0.24.

Figure 2-95 Enclosed Via Spacing Rule Example



Isolated Via Rule

The isolated via rule defines the maximum distance between vias. It is specified by setting detail route options. You set these options with the `set_droute_options` command. These option settings are stored with the cell.

An isolated via is a via that does not have neighboring vias close enough to meet the requirements of the technology. To pass this rule, a via must meet at least one of the following two requirements:

- An adjacent via is located within the distance in microns that is specified in the `isolatedViaSpacing` detail route option.

The value of this option must be between 0.0 and 20.0. If you do not set this option, a distance of 1.0 micron is used.

- Adjacent vias exist in all four surrounding quadrants within the distance in microns that is specified in the `isolatedViaQuadrantSpacing` detail route option.

The value of this option must be between 0.0 and 50.0, and it must be greater than the value specified for `isolatedViaSpacing`. If you do not set this option, a distance of 10.0 microns is used.

The values that you set for these detail route options are stored with the design when you save the design in Milkyway format.

To check for isolated via violations in your design, run the `report_isolated_via` command. The tool measures the spacing between vias as follows:

- If the vias are not aligned, the tool measures corner-to-corner spacing.
- If the vias are aligned, the tool measures edge-to-edge spacing.

The tool checks the isolated via spacing first and then checks the isolated via quadrant spacing. If a via fails both tests, it is flagged as an isolated via violation.

To fix isolated via violations, run the `fix_isolated_via` command. By default, the tool fixes the isolated via violations for all unfixed signal, clock, power, and ground nets by inserting a hang-on via (a via that is connected to only one metal layer) on the net.

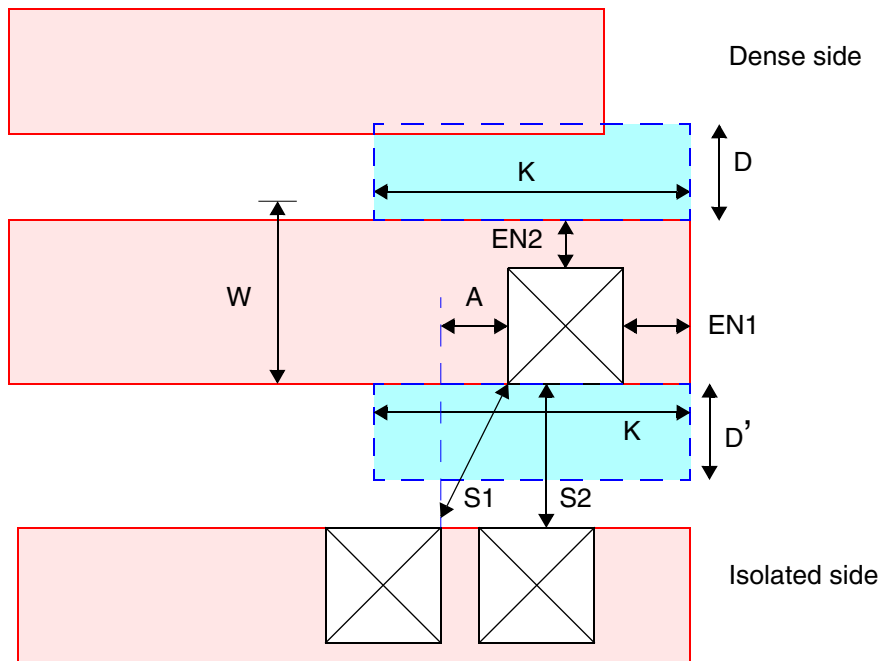
To check for and fix isolated via violations on fixed nets, set the `droute_checkFixedDRC` variable to 1. In this case, the violations are fixed by adding a via to a unfixed routing segment.

To fix isolated via locations that occur on clock nets without touching the clock net itself, set the `droute_fixIsoViaTouchClock` variable to 0. To fix isolated via violations on power and ground nets without touching the power or ground net itself, set the `droute_fixIsoViaTouchPG` variable to 0.

Misaligned Via Spacing Rule

The misaligned via rule specifies the minimum spacing between vias when the connected upper-layer metal lines are spaced closely and a via is near the metal line-end. [Figure 2-96](#) shows how the rule specifies the minimum spacing between a via in a metal line near the line-end and other vias. The rule applies only to a via layer and the via's upper-level metal.

Figure 2-96 Misaligned Via Minimum Spacing Rule, Example 1



This is the general syntax of the rule:

```
DesignRule {
  layer1 = "Mx"
  layer2 = "VIA(x-1)"
  misalignedViaWireTblSize = n
  misalignedViaWireThresholdTbl = (W1,W2, ...)
  misalignedViaWireMaxSpacingThresholdTbl = (D1,D2, ...)
  misalignedViaWireMinSpacingThreshold2Tbl = (D'1,D'2, ...)
  misalignedViaWireKeepoutLengthTbl = (K1,K2, ...)
  misalignedViaEndEnclosure = EN1
  misalignedViaSideEnclosure = EN2
  misalignedViaCornerKeepoutWidth = A
  misalignedViaMinSpacing = S2
  misalignedViaCornerMinSpacing = S1
}
```

This rule applies to a square via located in a metal line of width less than W when the line-end metal enclosure is less than or equal to $EN1$, the side metal enclosure is less than or equal to $EN2$, and there are nearby parallel metal lines on both sides.

The metal line containing the via has two keepout regions along the sides. Each keepout region extends along the metal line for a length K from the line-end and a distance D away from one side of the metal line and the distance D' away from the other side of the metal.

The line-end of one nearby line touches or partially covers the keepout D-by-K region on that side; this is called the “dense” side. On the other side, the nearby parallel line is more than the distance D’ away; this is called the “isolated” side. The rule specifies the minimum distance between a via in the metal near the line-end and the via in the nearby metal on the isolated side.

The minimum spacing is specified either as a via edge-to-edge spacing S2 or a via corner-to-corner spacing S1, depending on the extent of the shared parallel overlap between the original via near the line-end and the via in the nearby metal on the isolated side. If the overlap is less than negative A, the corner-to-corner spacing value S1 applies. Otherwise, the edge-to-edge spacing S2 applies.

The rule conditions are table-based. Here is an example of a rule using a table size of 2:

```
DesignRule {
  layer1 = "M6"
  layer2 = "VIA5"
  misalignedViaWireTblSize = 2
  misalignedViaWireThresholdTbl = (0.000,0.061)
  misalignedViaWireMaxSpacingThresholdTbl = (0.073,0.073)
  misalignedViaWireMinSpacingThreshold2Tbl = (0.073,0.000)
  misalignedViaWireKeepoutLengthTbl = (0.133,0.133)
  misalignedViaEndEnclosure = 0.045
  misalignedViaSideEnclosure = 0.000
  misalignedViaCornerKeepoutWidth = 0.048
  misalignedViaMinSpacing = 0.100
  misalignedViaCornerMinSpacing = 0.095
}
```

In this example, for a square via under a metal line with a width less than 0.061, the via has a line-end overlap of less than 0.045 and a side overlap of zero. Each keepout region measures 0.133 by 0.073. In [Figure 2-96](#), one nearby metal ends inside the upper keepout region, so the rule specifies the minimum spacing from the enclosed via to any vias in the metal line on the isolated side.

If the shared parallel overlap is less than –0.048, the via-to-via corner-to-corner spacing must be at least 0.095. If the shared parallel overlap is greater than –0.048, the via-to-via edge-to-edge spacing must be at least 0.100.

In [Figure 2-97](#), the metal line at the bottom touches the bottom keepout region, so the rule specifies the minimum distance between the original via and the vias in the top line. The rule also applies to vias within the same metal geometry, as shown in [Figure 2-98](#), when the table value of `misalignedViaWireMinSpacingThreshold2Tbl` is zero.

Figure 2-97 Misaligned Via Minimum Spacing Rule, Example 2

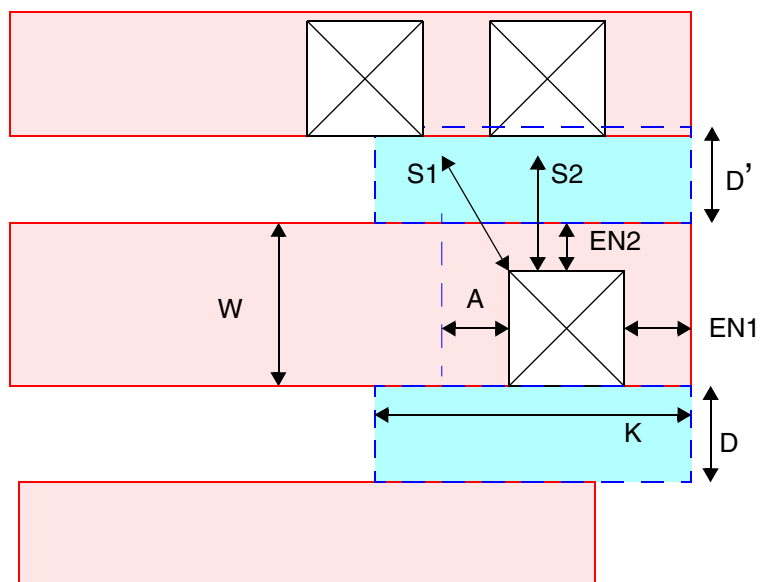
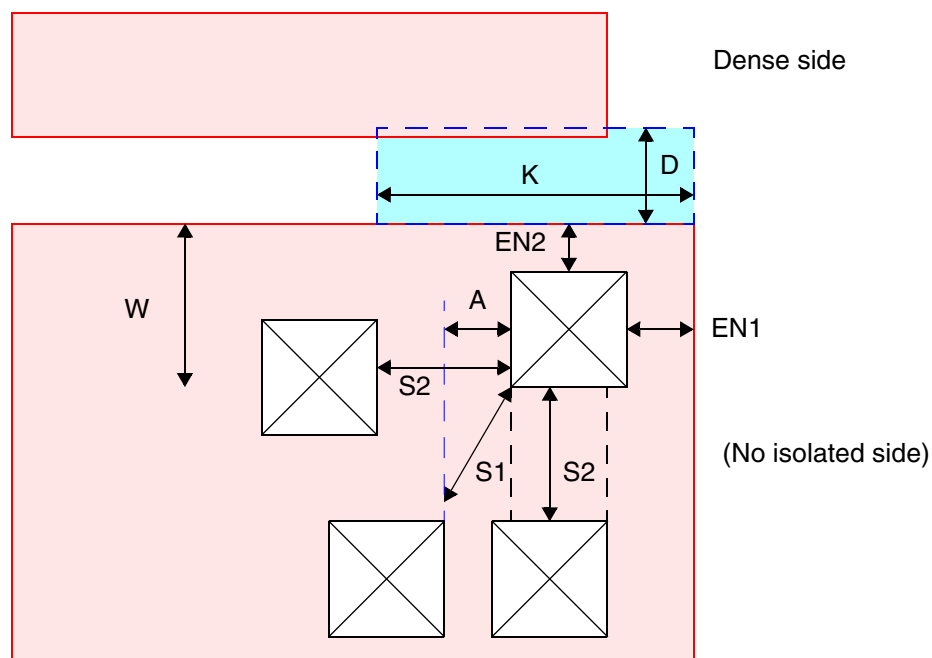


Figure 2-98 Misaligned Via Minimum Spacing Rule, Example 3



If the minimum spacing between vias depends on the via sizes, you can define the minimum spacing values in the form of a table using the following syntax:

```
misalignedViaCut1TblSize      = 1
misalignedViaCut1NameTbl     = (Vsm)
misalignedViaCut2TblSize     = 3
misalignedViaCut2NameTbl     = (Vsm, Vh, Vv)
misalignedViaMinSpacingTbl   = (0.110, 0.110, 0.110) # S2, 1x3 table
misalignedViaCornerMinSpacingTbl = (0.090, 0.090, 0.097) # S1, 1x3 table
```

There are two table size attributes and two name tables, one for each of the vias that have a minimum spacing requirement. The name tables list the names of the vias as defined in the general cut spacing rule; see [“General Cut Spacing Rule” on page 2-33](#).

The minimum spacing table attribute is a table of spacing values, which specifies one value for each possible combination of vias checked for adequate spacing. This is the spacing S2 in the foregoing diagrams. The number of spacing values is equal to the product of the two via table sizes.

Similarly, the corner minimum spacing table attribute specifies the corner-to-corner spacing between each possible combination of vias sizes checked for adequate spacing. This is spacing S1 in the foregoing diagrams.

If you use the table-type cut1 and cut2 table size and name table attributes, then you must also use the table-type minimum spacing and corner minimum spacing attributes. In that case, the plain minimum spacing attributes `misalignedViaMinSpacing` and `misalignedViaCornerMinSpacing` are ignored.

Via Enclosure Rules

The via enclosure rules are described in the following sections:

- [Minimum Enclosure Rule](#)
- [End-of-Line Via Enclosure Rules](#)
- [Two-Neighbor End-of-Line Via Enclosure Rule](#)
- [Three-Neighbor End-of-Line Via Enclosure Rule](#)
- [Enclosed Via Metal Minimum Length Rule](#)
- [Fat Wire Via Enclosure Rule](#)
- [Fat Metal Via Keepout Rules](#)
- [Concave Metal Corner Via Enclosure Rule](#)
- [Convex Metal Corner Via Enclosure Rule](#)

Minimum Enclosure Rule

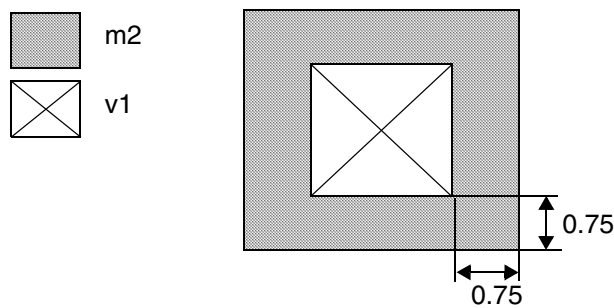
The minimum enclosure rule specifies the minimum distance at which layer1 must enclose layer2 when the two layers overlap.

This rule is defined in a `DesignRule` section of the technology file. To specify the minimum enclosure, use the `minEnclosure` attribute. For example, to specify that the metal 2 layer must enclose the via 1 layer by a minimum of 0.75 user units when the layers overlap, as shown in [Figure 2-99](#), enter

```
DesignRule {
  layer1      = "M2 "
  layer2      = "V1 "
  minEnclosure = 0.05
}
```

[Figure 2-99](#) represents the minimum enclosure rule, which says that when the metal 2 and via 1 layers overlap, the metal 2 layer must enclose the via 1 layer at a distance of 0.75 user units.

Figure 2-99 minEnclosure Rule Example



End-of-Line Via Enclosure Rules

The end-of-line via enclosure rule applies when the via is in an end-of-line orientation and a rule that applies when the via is in a T-shape orientation. Both rules apply to single vias only, and not to double vias.

To specify whether the router checks or ignores these two rules, set the `ignoreMetalExtensionRule` detail route option. You set this option with the `set_droute_options` command. The option setting is stored with the cell.

```
icc_shell> set_droute_options -name ignoreMetalExtensionRule -value value
```

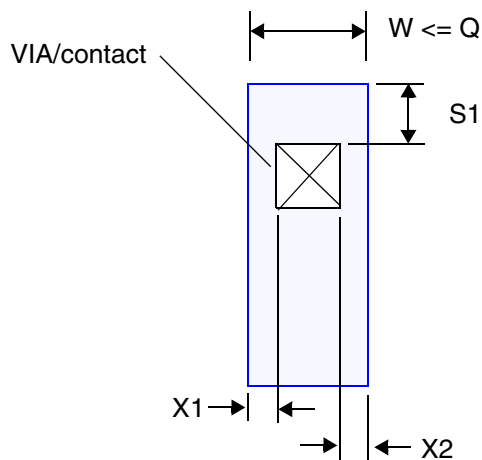
The following table lists the valid values for the `ignoreMetalExtensionRule` detail route option.

Value	Description
0	Checks the rules (the default)
1	Ignores the rules, even when they are set in the technology file

Zero-Nighbor End-of-Line Via Enclosure Rule

The zero-neighbor end-of-line via enclosure rule specifies the minimum amount of metal overlap around a via for both metal layers that are connected through the via, in the absence of any nearby metal. [Figure 2-100](#) shows the attributes associated with this rule.

Figure 2-100 Metal Extension for End-of-Line Via Enclosure Rule



For example,

```
DesignRule      {
  layer1          = "VIA4"
  layer2          = "M5"
  endOfLineEncTblSize      = 3
  endOfLineEncViaArrayExcluded = 1
  endOfLineEncWidthThreshold = 0.28
  endOfLineEncSideThreshold  = (0.000, 0.015, 0.025)
  endOfLineEncTbl          = (0.081, 0.061, 0.000)
}
```

Note:

The `endOfLineEncWidthThreshold` attribute is used only by Zroute, not the classic router.

When a single via is in the end-of-line of upper metal or lower metal and the metal width is no more than Q , the extension must be greater than $S1$. The extension depends on $X1$ and $X2$; if $X1$ is different from $X2$, the smaller value is applied. Therefore, when the metal width is no more than 0.28 ,

- If $0.005 \text{ nm} \leq X1/X2 < 0.015 \text{ nm}$, $S1$ is greater than 0.08 nm for the via
- If $0.015 \text{ nm} \leq X1/X2 < 0.025 \text{ nm}$, $S1$ is greater than 0.06 nm for the via

The `endOfLineEncViaArrayExcluded` attribute, when set to 1, causes the rule to be waived for any via array, such as a double via. To have the rule apply to both via arrays and single vias, omit this statement from the rule.

T-Shape Metal Extension for End-of-Line Via Enclosure Rule

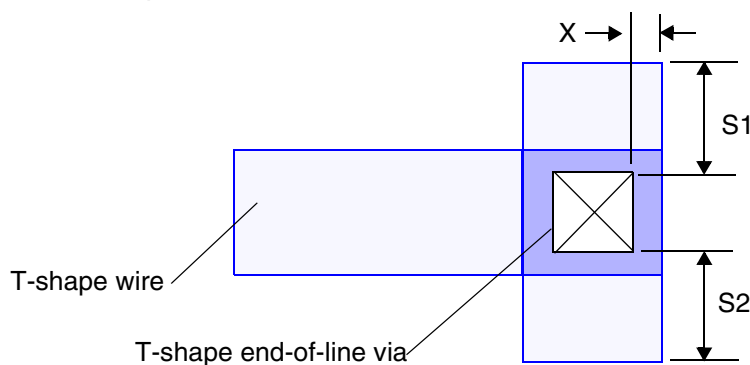
Use the `endOfLineTShapeEncTblSize`, `endOfLineTShapeCornerMinSpacing`, `endOfLineTShapeEncSideThreshold`, and `endOfLineTShapeEncTbl` attributes to specify the T-shape end-of-line via enclosure rule. Define these attributes in the `DesignRule` section of the technology file.

For example,

```
DesignRule {
  layer1                = "VIA4"
  layer2                = "M5"
  endOfLineTShapeEncTblSize = 4
  endOfLineTShapeCornerMinSpacing = 0.035
  endOfLineTShapeEncSideThreshold = (0.005, 0.010, 0.020, 0.035)
  endOfLineTShapeEncTbl      = (0.081, 0.061, 0.051, 0.041)
}
```

[Figure 2-101](#) shows the attributes associated with the metal extension for end-of-line via enclosure rule.

Figure 2-101 T-Shape Metal Extension for End-of-Line Via Enclosure Rule



When a single via is in the end-of-line T-shape metal, the extension is greater than S1 and S2. The extension depends on X. Therefore,

- If X is less than or equal to 0.005 nm and less than 0.010 nm, S1 and S2 are greater than 0.08 nm for the via
- If X is less than or equal to 0.010 nm and less than 0.020 nm, S1 and S2 are greater than 0.06 nm for the via
- If X is less than or equal to 0.020 nm and less than 0.035 nm, S1 and S2 are greater than 0.05 nm for the via
- If X is less than or equal to 0.035 nm, S1 and S2 are greater than 0.04 nm for the via

This rule applies only to a single via enclosed within a T-shaped wire. It does not apply to a double via. The rule can be disabled entirely by setting `ignoreMetalExtensionRule` to 1.

Two-Neighbor End-of-Line Via Enclosure Rule

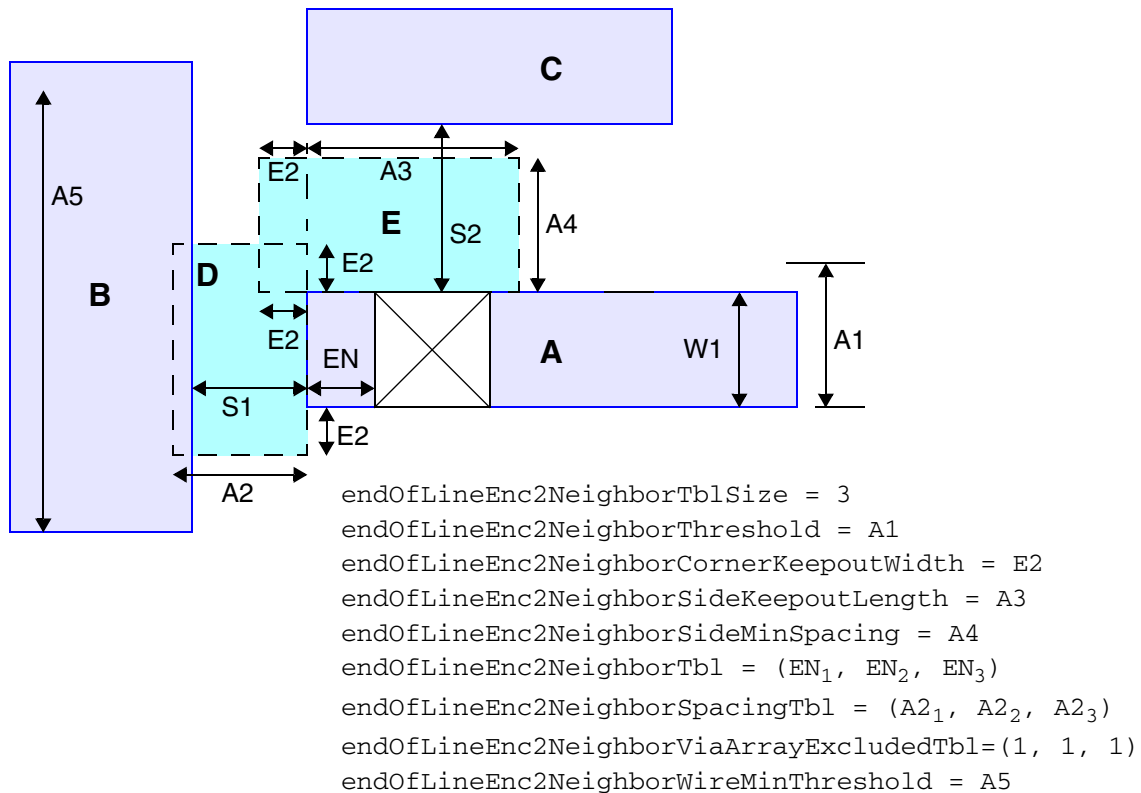
The two-neighbor end-of-line via enclosure rule specifies the minimum spacing between an enclosed via in a narrow metal line and two metals near the line-end and along one side.

Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see [“Enhanced Dense End-of-Line Spacing Rule” on page 2-88](#).

In [Figure 2-102](#), if the metal width W1 is less than or equal to the width threshold A1, then the distance between the line end and the first other metal S1 must be at least the minimum spacing value A2, or the distance between the side of the narrow metal line and the second other metal S2 must be at least the minimum side spacing value A4. Edges facing the end of line edge are checked only if the edge length exceeds the wire minimum threshold A5. The line-end spacing value A4 is specified as an n-dimensional array, depending on the via enclosure value EN.

Figure 2-102 Two-Neighbor End-of-Line Via Enclosure Rule



In the figure, either metal B must be outside of the dashed rectangle D or metal C must be outside of the dashed rectangle E. In this case, metal C is outside of dashed rectangle E, so the layout passes the rule. The dashed areas extend outward from the line-end corners by the corner keepout width E2. The side keepout area E has a length of A3 plus extension E2.

The rule is enforced only for a single via, and not for a via array, if the corresponding entry in the `endOfLineEnc2NeighborViaArrayExcludedTbl` table is set to 1. If the table entry is 0, the rule is enforced for both single vias and via arrays.

Here is an example of the syntax used for this rule:

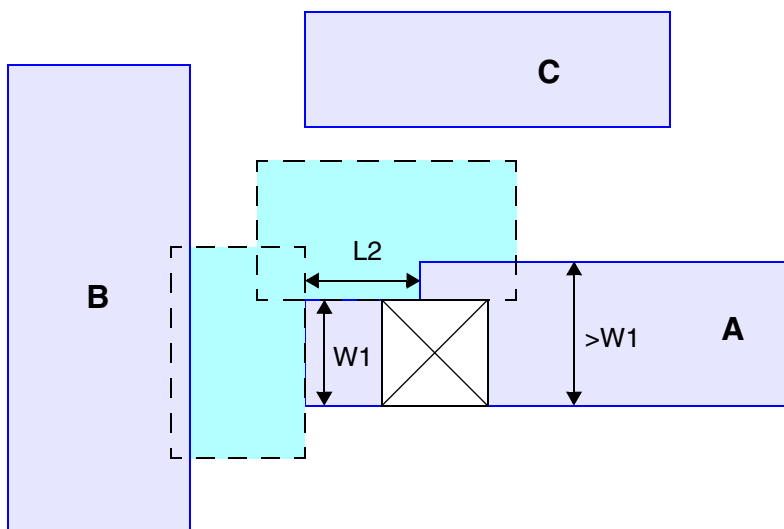
```
DesignRule {
  layer1 = "M2"
  layer2 = "VIA1"
  endOfLineEnc2NeighborTblSize = 2
  endOfLineEnc2NeighborThreshold = 0.07
  endOfLineEnc2NeighborCornerKeepoutWidth = 0.03
  endOfLineEnc2NeighborSideKeepoutLength = 0.08
  endOfLineEnc2NeighborSideMinSpacing = 0.068
  endOfLineEnc2NeighborTbl = (0.038, 0.058)
  endOfLineEnc2NeighborSpacingTbl = (0.15, 0.13)
  endOfLineEnc2NeighborViaArrayExcludedTbl = (1, 1)
  endOfLineEnc2NeighborWireMinThreshold = .08
}
```

In this example, when a metal line having a width of less than 0.07 has neighboring metal at the end and at the side near the end, the spacing to the neighboring metal at the end must be at least the table-specified value of A2, which depends on the amount of via enclosure EN; or the spacing to the neighboring metal at the side must be at least 0.068. The keepout region extends 0.03 away from the line-end corners and 0.08 from the corner along the side.

If `endOfLineEnc2NeighborWireMinThreshold` (the width of the neighboring wire) is not defined, the rule is triggered only by `endOfLineEnc2NeighborThreshold` (the width of the end-of-line wire).

The rule does not apply when a short edge is adjacent to the end of the line, causing the wire width to be greater than the end-of-line width threshold, as shown in [Figure 2-103](#).

Figure 2-103 Two-Nighbor End-of-Line Via With Adjacent Short Edge



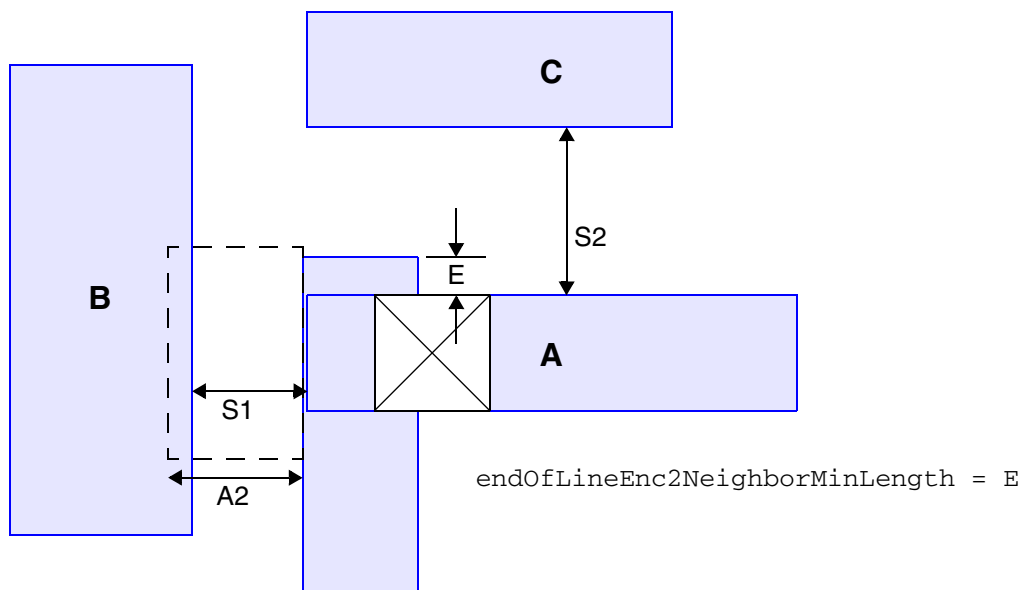
By default, the adjacent edge L2 is considered short if it is less than or equal to the `minWidth` value defined for the metal. To define a different L2 threshold for the two-neighbor end-of-line via enclosure rule, use the following attribute:

```
endOfLineEnc2NeighborMinLength = L2
```

If the adjacent edge length is less than or equal to the specified value, the rule does not apply.

You can also specify a minimum via enclosure on the sides perpendicular to the closer spacing, as shown in [Figure 2-104](#).

Figure 2-104 Two-Neighbor End-of-Line Via Enclosure Rule

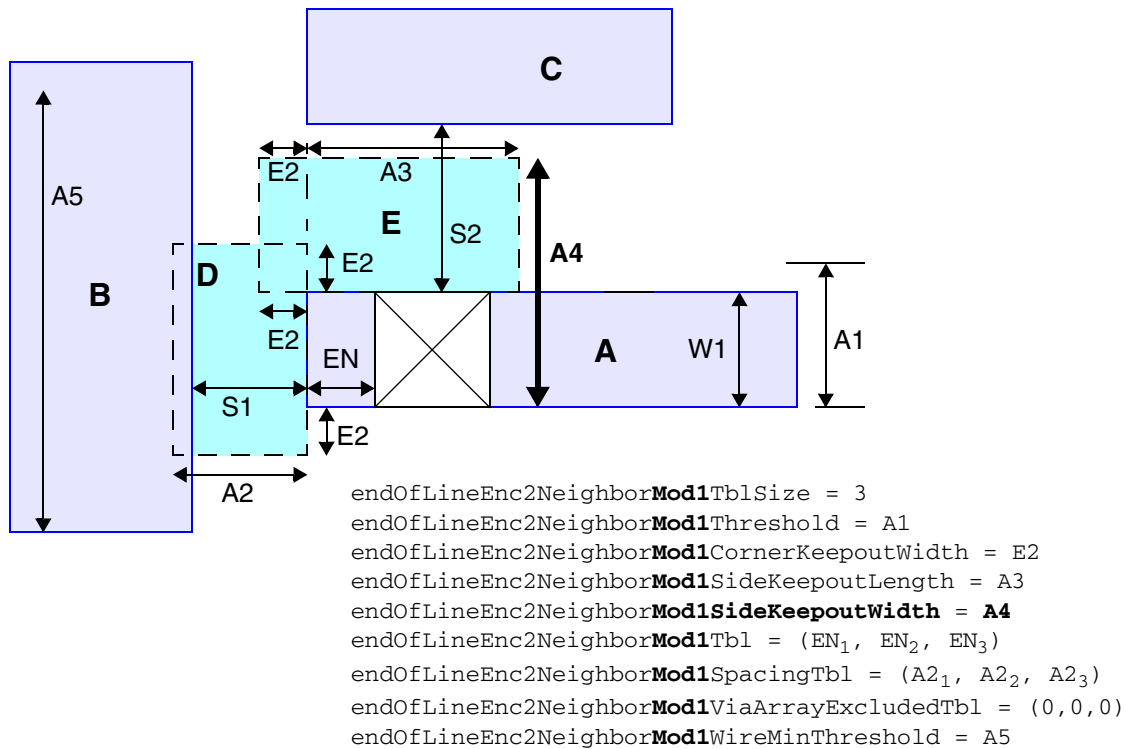


For example,

```
DesignRule {
  layer1 = "M2"
  layer2 = "VIA1"
  endOfLineEnc2NeighborTblSize = 2
  endOfLineEnc2NeighborThreshold = 0.07
  endOfLineEnc2NeighborCornerKeepoutWidth = 0.03
  endOfLineEnc2NeighborSideKeepoutLength = 0.08
  endOfLineEnc2NeighborSideMinSpacing = 0.068
  endOfLineEnc2NeighborMinLength = 0.04
  endOfLineEnc2NeighborTbl = (0.038, 0.058)
  endOfLineEnc2NeighborSpacingTbl = (0.15, 0.13)
  endOfLineEnc2NeighborViaArrayExcludedTbl = (1, 1)
  endOfLineEnc2NeighborWireMinThreshold = .08
}
```

Figure 2-105 shows an alternative, modified form of the rule that specifies the side spacing requirement A4 as a distance from the neighboring metal to the far edge of the narrow metal line instead of the nearest edge. This is a separate rule with different syntax, but operating in a manner similar to the foregoing rule.

Figure 2-105 Modified Two-Nighbor End-of-Line Via Enclosure Rule



Here is an example of the modified syntax used for this rule:

```

DesignRule {
  endOfLineEnc2NeighborMod1TblSize = 2
  endOfLineEnc2NeighborMod1Threshold = 0.07
  endOfLineEnc2NeighborMod1CornerKeepoutWidth = 0.03
  endOfLineEnc2NeighborMod1SideKeepoutLength = 0.08
  endOfLineEnc2NeighborMod1SideKeepoutWidth = 0.18
  endOfLineEnc2NeighborMod1Tbl = (0.038, 0.058)
  endOfLineEnc2NeighborMod1SpacingTbl = (0.15, 0.13)
  endOfLineEnc2NeighborMod1ViaArrayExcludedTbl = (0, 0)
  endOfLineEnc2NeighborMod1WireMinThreshold = 0.08
}

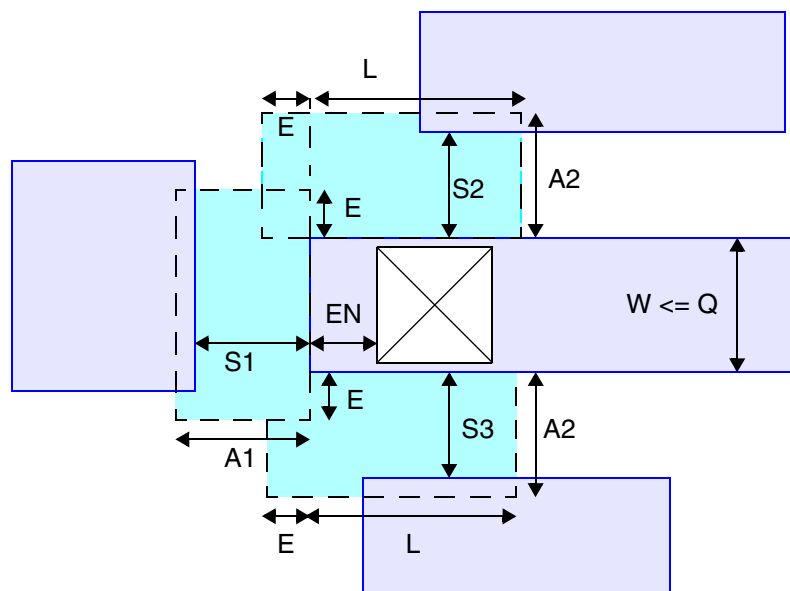
```


Three-Neighbor End-of-Line Via Enclosure Rule

The three-neighbor end-of-line via enclosure rule specifies the minimum overlap of a line-end over an enclosed via when there are three neighboring metal wires near the line-end and along two sides. This rule is supported only by Zroute, not the classic router.

In [Figure 2-106](#), if a metal wire encloses a via (in the layer above or below the via) near the wire line-end, and there are three neighboring wires in the same layer less than the distance A1 from the end and less than the distance A2 from the two sides, then the end of the wire must enclose the via by at least the distance EN.

Figure 2-106 Two-Neighbor End-of-Line Via Enclosure Rule



```

endOfLineEnc3NeighborThreshold           = Q
endOfLineEnc3NeighborCornerKeepoutWidth  = E
endOfLineEnc3NeighborSideKeepoutLength  = L
endOfLineEnc3NeighborSideMinSpacing     = A2
endOfLineEnc3NeighborMinSpacing         = A1
endOfLineEnc3NeighborMinEnclosure        = EN
endOfLineEnc3NeighborViaArrayExcluded    = 1

```

In the figure, if there is neighboring metal overlapping all three light-blue rectangles, then the minimum enclosure at the line-end EN must be satisfied. However, the rule is waived and the requirement does not apply when there is a via array (two or more vias). If you want the rule to apply to via arrays, set the `endOfLineEnc3NeighborViaArrayExcluded` attribute to zero or omit the statement entirely.

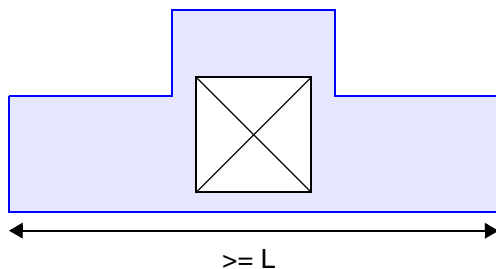
Here is an example of the syntax used for this rule:

```
DesignRule {
  layer1 = "M2"
  layer2 = "VIA1"
  endOfLineEnc3NeighborThreshold = 0.07
  endOfLineEnc3NeighborCornerKeepoutWidth = 0.03
  endOfLineEnc3NeighborSideKeepoutLength = 0.08
  endOfLineEnc3NeighborSideMinSpacing = 0.068
  endOfLineEnc3NeighborMinSpacing = 0.038
  endOfLineEnc3NeighborMinEnclosure = 0.015
  endOfLineEnc3NeighborViaArrayExcluded = 1
}
```

Enclosed Via Metal Minimum Length Rule

The enclosed via metal minimum length rule specifies a minimum length L for a metal segment that completely encloses a via, as shown in [Figure 2-107](#).

Figure 2-107 Enclosed Via Minimum Length Rule



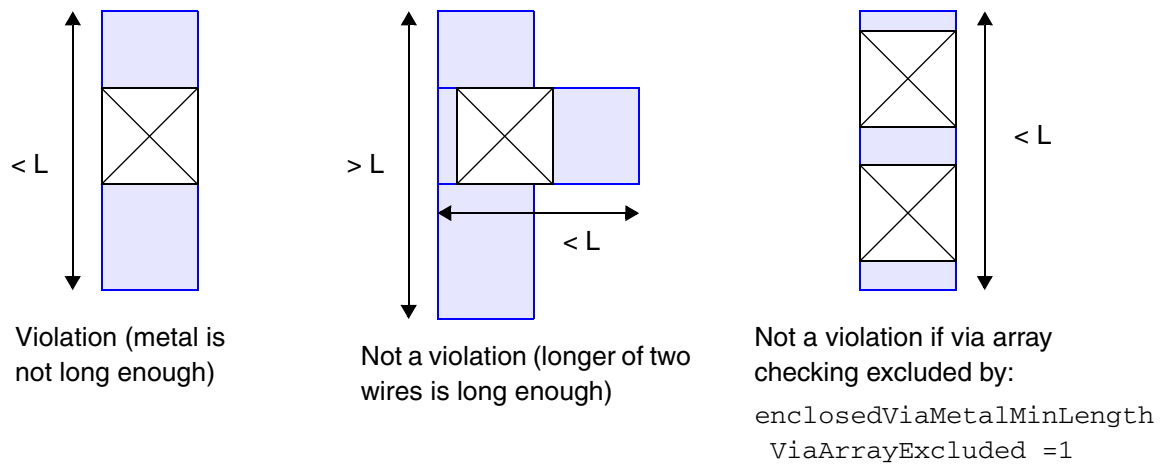
If a via is connected to more than one segment on the same layer, the minimum length constraint applies only to the longest wire that completely encloses the via. This rule can be optionally waived for via arrays.

This is the syntax of the rule:

```
DesignRule {
  layer1 = "via1"
  layer2 = "metal2"
  enclosedViaMetalMinLength = L
  enclosedViaMetalMinLengthViaArrayExcluded = 1
}
```

To have the rule apply to via arrays as well as single vias, omit the `enclosedViaMetalMinLengthViaArrayExcluded` statement from the rule.

The examples shown in [Figure 2-107](#) demonstrate how the rule works.

Figure 2-108 Enclosed Via Minimum Length Rule

Fat Wire Via Enclosure Rule

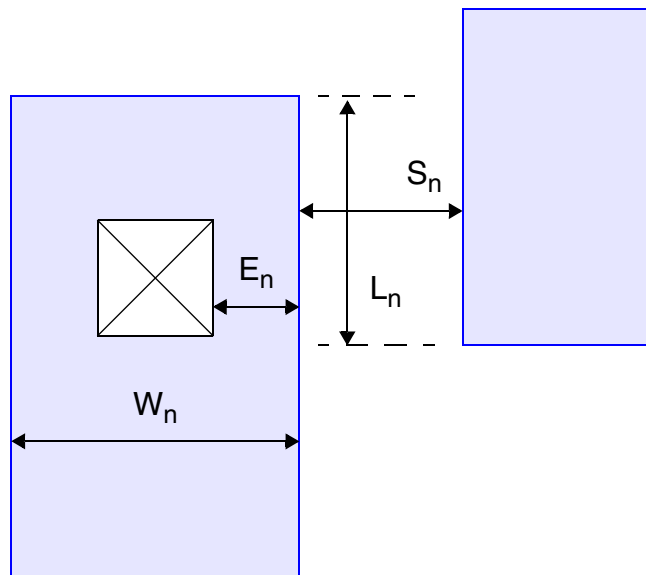
The fat wire via enclosure rule specifies the minimum amount of fat metal overlap over a via for the metal layer either above or below the via. The minimum overlap requirement depends on the width of the fat metal enclosing the via, the spacing to any nearby metal, and the parallel length of that nearby metal, as specified in a table.

Here is the syntax of the rule:

```
DesignRule
{
  layer1 = "MetalX"
  layer2 = "ViaX"
  fatWireViaEncTblSize = 4
  fatWireViaEncWidthThresholdTbl = (W1, W2, W3, W4)
  fatWireViaEncParallelLengthThresholdTbl = (L1, L2, L3, L4)
  fatWireViaEncMaxSpacingThresholdTbl = (S1, S2, S3, S4)
  fatWireViaEnclosureTbl = (E1, E2, E3, E4)
  fatWireViaArrayExcludedTbl = (0, 0, 0, 1)
}
```

Figure 2-109 shows how the minimum enclosure depends on the rule attributes.

Figure 2-109 Fat Wire Via Enclosure Rule



Here is an example of the rule:

```
DesignRule {
  layer1 = "Mx"
  layer2 = "VIAx"
  fatWireViaEncTblSize = 4
  fatWireViaEncWidthThresholdTbl = (0.055, 0.060, 0.075, 0.170)
  fatWireViaEncParallelLengthThresholdTbl = (0.12, 0.12, 0.14, 0.14)
  fatWireViaEncMaxSpacingThresholdTbl = (0.062, 0.068, 0.090, 0.134)
  fatWireViaEnclosureTbl = (0.008, 0.009, 0.012, 0.015)
  fatWireViaArrayExcludedTbl = (0, 0, 1, 0)
}
```

In this example, if the fat wire width W is greater than or equal to 0.055 but less than 0.060, the parallel length L of the nearby metal is greater than 0.12, and the spacing S to the nearby metal is less than 0.062, then the minimum enclosure E is 0.008. The last attribute, `fatWireViaArrayExcludedTbl`, is set to 1 to waive the rule for a double via for the corresponding set of values for W , L , S , and E .

To specify the minimum rather than maximum value of the spacing to the nearby metal for applying the rule, use the `fatWireViaEncMinSpacingThresholdTbl` attribute.

Instead of specifying width thresholds using the `fatWireViaEncWidthThresholdTbl` attribute, you can specify separate minimum and maximum width thresholds using the `fatWireViaEncMaxWidthThresholdTbl` and `fatWireViaEncMinWidthThresholdTbl` attributes. For example,

```
DesignRule {
  layer1 = "Mx"
  layer2 = "VIAx"
  fatWireViaEncTblSize = 4
  fatWireViaEncWidthThresholdTbl = (0.055, 0.060, 0.075, 0.170)
  fatWireViaEncMinWidthThresholdTbl = (0.000, 0.055, 0.060, 0.075)
  fatWireViaEncMaxWidthThresholdTbl = (0.055, 0.060, 0.075, 0.170)
  fatWireViaEncMaxSpacingThresholdTbl = (0.062, 0.068, 0.090, 0.134)
  fatWireViaEnclosureTbl = (0.008, 0.009, 0.012, 0.015)
  fatWireViaArrayExcludedTbl = (0, 0, 1, 0)
```

By default, width is measured along the smaller dimension of a metal rectangle. To always measure the width in the direction perpendicular to the direction of the parallel run, whether or not that direction is the smaller dimension, use the following attribute setting:

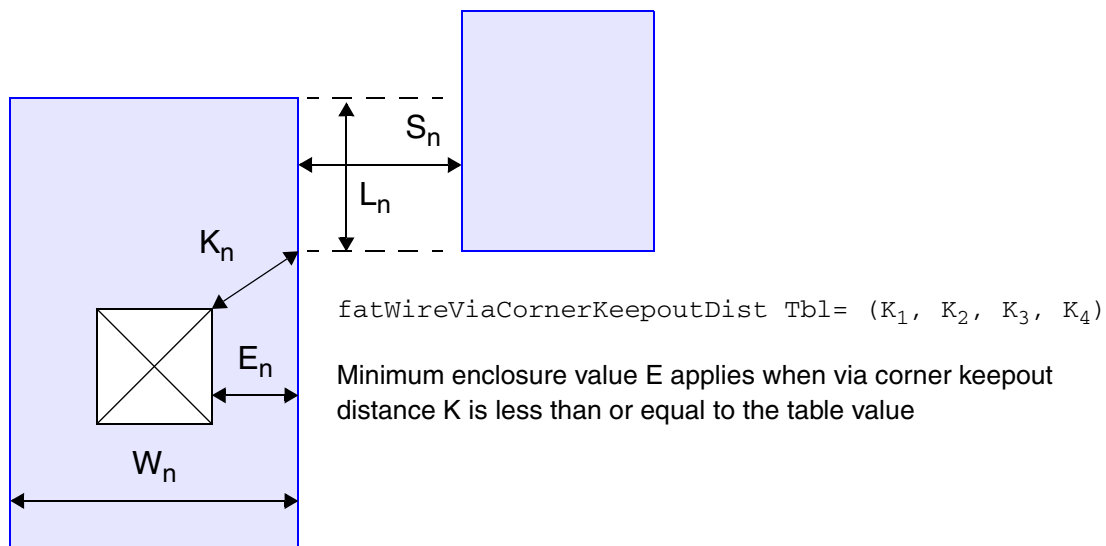
```
fatWireViaEncWidthIsOrthoTbl = 1
```

The fat wire via enclosure rule applies only to the case where the via resides in the fat metal within the parallel overlap region of the enclosing metal area and nearby metal area. To have the rule apply even when the via is some distance outside of the parallel overlap region, specify a via corner keepout value, as shown in the following example.

```
DesignRule {
  layer1 = "Mx"
  layer2 = "VIAx"
  fatWireViaEncTblSize = 4
  fatWireViaEncWidthThresholdTbl = (0.055, 0.060, 0.075, 0.170)
  fatWireViaEncParallelLengthThresholdTbl = (0.12, 0.12, 0.14, 0.14)
  fatWireViaEncMaxSpacingThresholdTbl = (0.062, 0.068, 0.090, 0.134)
  fatWireViaEnclosureTbl = (0.008, 0.009, 0.012, 0.015)
  fatWireViaArrayExcludedTbl = (0, 0, 1, 0)
  fatWireViaCornerKeepoutDistTbl = (0.016, 0.018, 0.019, 0.019)
}
```

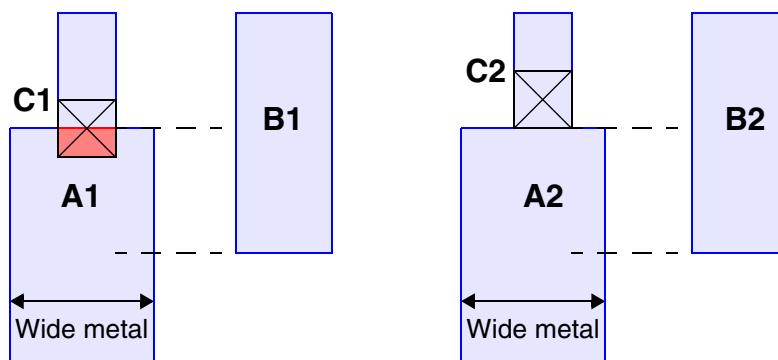
In this example, the rule still applies as long as the distance from the corner of the via to the parallel overlap region is less than or equal to the specified keepout distance K, as shown in [Figure 2-110](#).

Figure 2-110 Fat Wire Via Enclosure Rule With Corner Keepout Distance

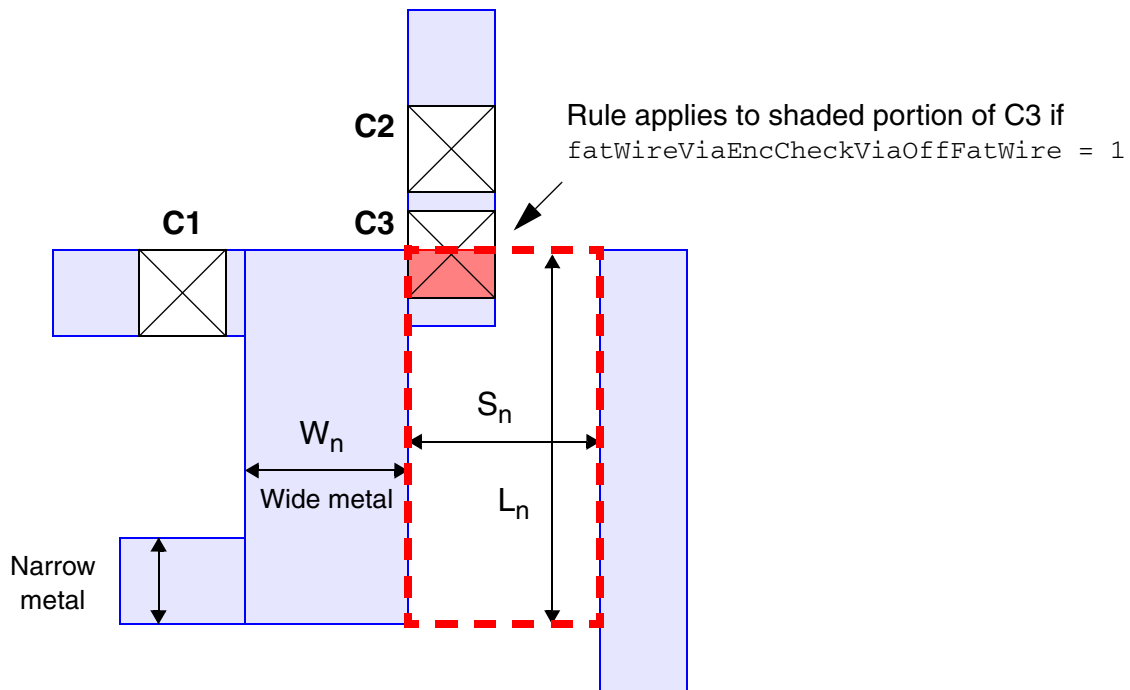


In [Figure 2-111](#), the corner keepout attribute is not used, and via C1 is half inside and half outside the parallel overlap region of the wide metal, so the fat wire via enclosure rule applies only to the lower portion of the via, highlighted in red in the figure. Via C2 is entirely outside the parallel overlap region of the wide metal, so the fat wire via enclosure rule does not apply at all to that via.

Figure 2-111 Via Inside and Outside Fat Metal Parallel Length Region



In [Figure 2-112](#), vias C1, C2, and C3 are all outside the parallel overlap region of the wide metal. By default, the fat wire via enclosure rule does not apply to these vias.

Figure 2-112 Exception to Via Outside Fat Metal Parallel Length Region

However, you can optionally have the rule apply to the portion of via C3 that covers the parallel-length region between the wide metal and the nearby metal. This is the region marked by the dashed red rectangle in the diagram. To apply the rule to the portion of any via inside this region, use the following attribute setting:

```
fatWireViaEncCheckViaOffFatWire = 1
```

When this attribute is set to 1, the fat wire via enclosure rule applies to vias that are off the fat wire but partially or completely inside the region between the fat metal and nearby metal. If this attribute is set to 0 or omitted from the design rule definition, the rule does not apply to any vias that are off the fat wire.

Fat Metal Via Keepout Rules

The fat metal via keepout rules are specified in the `DesignRule` section of the technology file by using the `fatWireViaKeepoutTblSize`, `fatWireViaKeepoutWidthThreshold`, `fatWireViaKeepoutMinSize`, `fatWireViaKeepoutEnclosure`, `fatWireViaKeepoutParallelLengthThreshold`, and `fatWireViaKeepoutMaxSpacingThreshold` attributes.

You can use these attributes to define the following rules:

- The fat metal via keepout area rule
- The via enclosure rule
- The poly contact enclosure rule

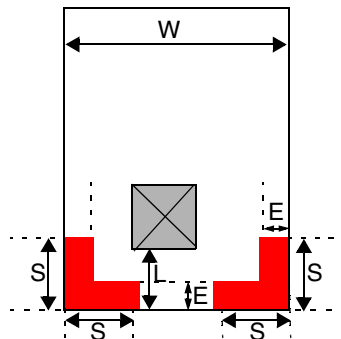
Note:

The fat wire via keepout rules are still supported. However, for any new technology file, you should use the more advanced rules described in [“Fat Wire Via Enclosure Rule” on page 2-129](#). Zroute does not support usage of both old and new syntax in the same technology file.

Fat Metal Via Keepout Area Rule

When you do not want the via at the end-of-line upper-fat wire or lower-fat wire to be placed too close to the corner of the fat wire, use this rule to define a via keepout region at the end-of-line corners (shown in red in [Figure 2-113](#)). A single via must be placed outside of the keepout area. In the case of a double via, at least one of the vias must be placed outside of the keepout area.

Figure 2-113 Fat Metal Via Keepout Area Rule



When the width, W , of the fat wire is greater than or equal to the `fatWireViaKeepoutWidthThreshold` value, the distance, L , between the via and the corner edges of the fat wire must be in greater than or equal to the `fatWireViaKeepoutEnclosure` value, E . The length of the keepout area in each direction from the corner, S , is specified by the `fatWireViaKeepoutMinSize` attribute.

For example,

```
DesignRule      {
    layer1              = "V1 "
    layer2              = "M2 "
    fatWireViaKeepoutTblSize      = 2
    fatWireViaKeepoutWidthThreshold = (0.5,1.0)
    fatWireViaKeepoutMinSize      = (0.18,0.36)
    fatWireViaKeepoutEnclosure    = (0.05,0.10)
}

DesignRule      {
    layer1              = "V1 "
    layer2              = "M1 "
    fatWireViaKeepoutTblSize      = 1
    fatWireViaKeepoutWidthThreshold = (0.5)
    fatWireViaKeepoutMinSize      = (0.18)
    fatWireViaKeepoutEnclosure    = (0.05)
}
```

Fat Via Enclosure Rule

The fat via enclosure rule defines the minimum width, *G*, of the metal enclosure of a via under the following conditions:

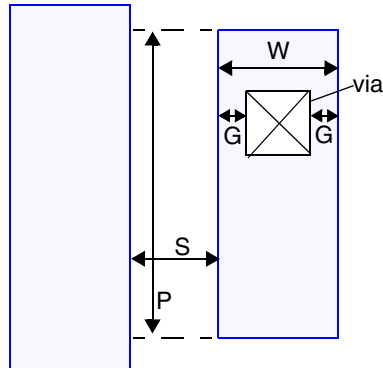
- The width, *W*, of the enclosing metal segment is between the widths specified in the `fatWireViaKeepoutWidthThreshold` attribute.
- The space, *S*, between the enclosing metal segment and the neighboring metal segment is less than the value specified in the `fatWireViaKeepoutMaxSpacingThreshold` attribute.
- The parallel length, *P*, between the enclosing metal segment and the neighboring metal segment is greater than the value specified in the `fatWireViaKeepoutParallelLengthThreshold` attribute and the parallel run covers the poly contact edge partially or completely.
- The enclosing metal does not contain a double-via array.

Use the `fatWireViaKeepoutEnclosure` attribute to specify the minimum width of the metal enclosure.

The `pinRectangleMerge` detail route option controls how pin and obstacle rectangles are merged when checking this rule. When set to 0 (the default), neither pin nor obstacle rectangles are merged. When set to 1, pin rectangles, but not obstacle rectangles, are merged. When set to 2, both pin and obstacle rectangles are merged.

Figure 2-114 shows the measurements involved in the fat via enclosure rule.

Figure 2-114 Via Enclosure Rule



For example, the following `DesignRule` section defines a minimum enclosure width of 0.015 microns when the enclosing metal 2 segment is

- Between 0.11 and 0.21 microns wide
- Closer than 0.08 microns to its neighboring metal 2 segment
- Parallel to its neighboring metal 2 segment for at least 0.27 microns

```
DesignRule {
    layer1                = "M2 "
    layer2                = "VIA "
    fatWireViaKeepoutParallelLengthThreshold = 0.27
    fatWireViaKeepoutMaxSpacingThreshold    = 0.08
    fatWireViaKeepoutTblSize                = 2
    fatWireViaKeepoutWidthThreshold         = (0.11,0.21)
    fatWireViaKeepoutMinSize                = (2,2)
    fatWireViaKeepoutEnclosure              = (0.015,0)
}
```

```
icc_shell> set_droute_options -name pinRectangleMerge -value 2
```

Fat Poly Contact Enclosure Rule

The fat poly contact enclosure rule is the same as the fat via enclosure rule, but it defines the enclosure requirements between a poly contact and a metal segment, rather than between a via and a metal segment. This rule does not apply if the enclosing metal contains a double poly contact array.

For example, the following `DesignRule` section and option settings define a minimum enclosure width of 0.015 μm when the enclosing metal 1 segment is between 0.11 μm and 0.21 μm wide, is closer than 0.08 μm to its neighboring metal segment, and is parallel to it for at least 0.27 μm .

```

DesignRule {
  layer1          = "M1"
  layer2          = "CO"
  fatWireViaKeepoutParallelLengthThreshold = 0.27
  fatWireViaKeepoutMaxSpacingThreshold    = 0.08
  fatWireViaKeepoutTblSize                = 2
  fatWireViaKeepoutWidthThreshold         = (0.11,0.21)
  fatWireViaKeepoutMinSize                = (2,2)
  fatWireViaKeepoutEnclosure              = (0.015,0)
}

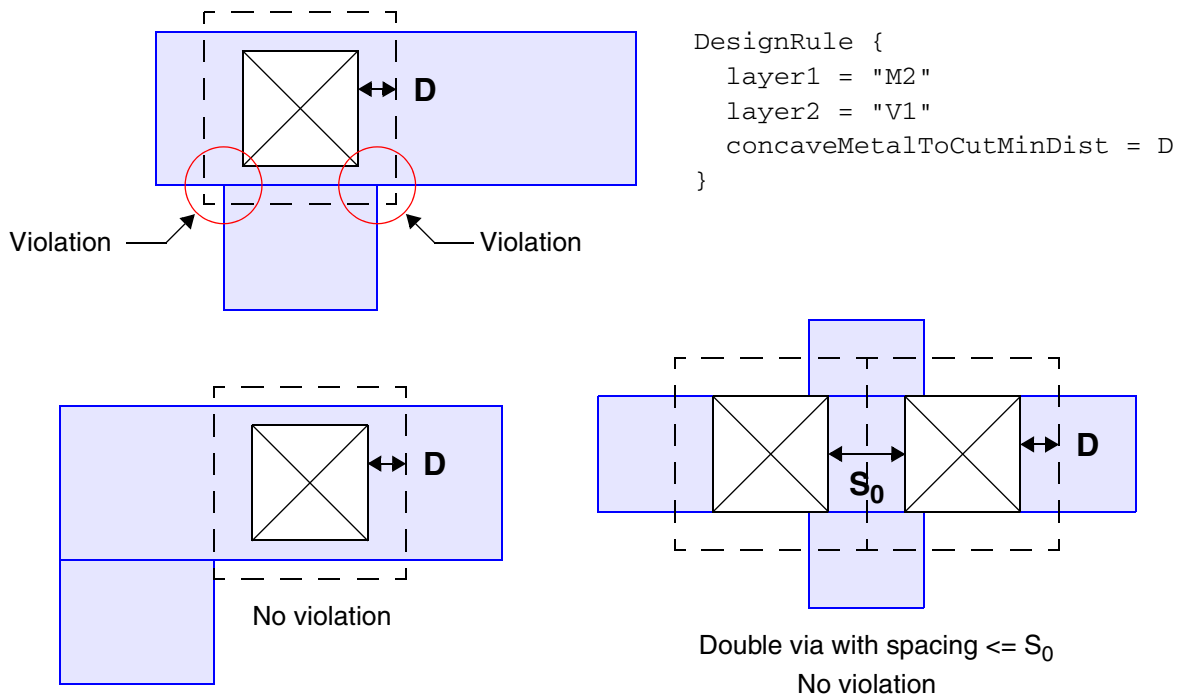
```

```
icc_shell> set_droute_options -name pinRectangleMerge -value 2
```

Concave Metal Corner Via Enclosure Rule

The concave metal corner via enclosure rule specifies a minimum allowed distance between a concave metal corner and a via enclosed by the metal. In [Figure 2-115](#), the rule prohibits any concave metal corner within a rectangular exclusion area surrounding the via by a distance D . By default, the rule does not apply to via arrays, where two or more vias are enclosed by the same metal segments with a spacing between the vias of no more than S_0 , the `minCutSpacing` value specified in the `ContactCode` section of the technology file.

Figure 2-115 Concave Metal Corner to Via Enclosure Rule



Here is an example of the syntax used for this rule:

```
DesignRule {  
    layer1 = "M2"  
    layer2 = "V1"  
    concaveMetalToCutMinDist = 0.30  
}
```

In this example, any concave metal corner must be at least 0.30 distance units away from a single enclosed via.

To enforce the rule for via arrays as well as single vias, add the following line:

```
concaveMetalToViaArrayIncluded = 1
```

In that case, each via in a via array must meet the concave metal corner enclosure requirement.

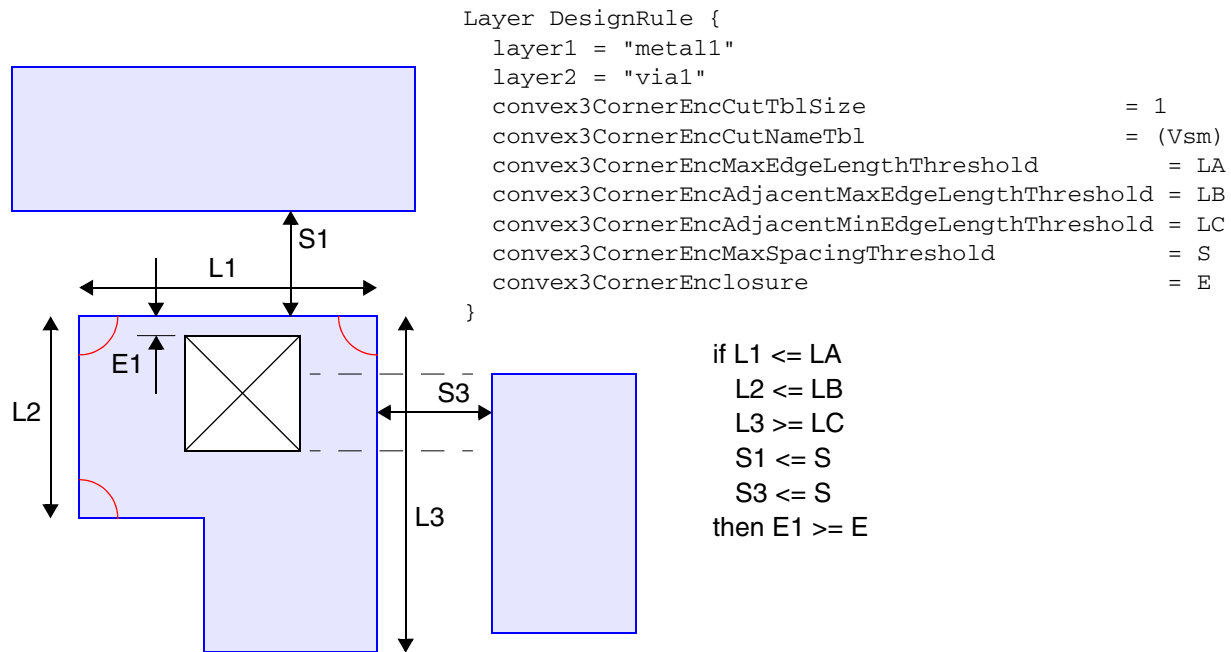
This rule is supported by Zroute only, not the classic router.

Convex Metal Corner Via Enclosure Rule

The convex metal corner via enclosure rule specifies the minimum amount of enclosure of metal around a via when the via is surrounded by three 90-degree convex corners and nearby metal. It does not matter whether the nearby metal geometries belong to the same net or a different net from the metal containing the via.

In [Figure 2-116](#), a via is surrounded by three 90-degree convex corners marked in red. The rule applies when L1 is less than or equal to the threshold LA, L2 is less than or equal to the threshold L2, L3 is at least the threshold L3, two nearby metal shapes have a parallel overlap with the via greater than zero, and the two nearby shapes are no more than the distance S away from the enclosing metal edges. When all of these conditions are true, the minimum enclosure of the L1 edge must be at least the value E1.

Figure 2-116 Convex Metal Corner Via Enclosure and Spacing Rule



Here is an example of the rule:

```

Layer DesignRule {
  layer1 = "metal1"
  layer2 = "via1"
  convex3CornerEncCutTblSize = 2
  convex3CornerEncCutNameTbl = (Vsm,Vlg)
  convex3CornerEncMaxEdgeLengthThreshold = (0.15,0.20)
  convex3CornerEncAdjacentMaxEdgeLengthThreshold = (0.08,0.13)
  convex3CornerEncAdjacentMinEdgeLengthThreshold = (0.13,0.15)
  convex3CornerEncMaxSpacingThreshold = (0.06,0.06)
  convex3CornerEnclosure = (0.012,0.015)
}

```

The example specifies the minimum enclosure rule for the via shapes named Vsm and Vlg, as defined in the general cut spacing rule. For information about naming via shapes, see [“General Cut Spacing Rule” on page 2-33](#).

Note:

This rule is supported only by Zroute.

Via Placement Rules

The following types of rules restrict the placement of vias:

- [Via Stack Rule](#)
- [Via Array \(Via Farm\) Rule](#)
- [Via Density Rule](#)
- [Vias on Nonpreferred Route Direction Rule](#)

Via Stack Rule

The `stackable` attribute in the `DesignRule` section specifies whether vias in successive layers are stackable. When no spacing rule exists between the two layers, the `stackable` attribute does not need to be specified and the router allows vias to overlap arbitrarily. When a spacing rule exists between the two layers and `stackable` is set to 1, Zroute either separates the vias according to the spacing rule or stacks one via over the other, with arbitrary overlap by default.

The `maxStackLevel` attribute specifies the maximum number of vias in successive layers that can be stacked upon each other. If the number of successive stacked vias exceeds the specified limit, it is a rule violation.

There are four operating modes that determine the scope of stacked via checking, which can depend on whether the vias are single or belong to an array. The `maxStackLevelMode` attribute in the `Technology` section of the technology file specifies the checking mode. You can set this attribute to a number from 0 to 3, defined as follows:

- 0 (the default): The rule applies only to a set of stacked vias when some or all of the vias are single. The rule is ignored when all of the stacked vias belong to arrays.
- 1: The rule applies to all stacked vias, whether single or belonging to an array.
- 2: The rule applies only to a set of stacked vias when all of them are single vias. The rule is ignored if any of the stacked vias belong to an array.
- 3: This rule applies to all stacked vias except in the case where the stacked vias all belong to via arrays and the arrays are aligned in the stack.

For example,

```
Technology {
    maxStackLevelMode = 1
}

Layer "Via12" {
    maxStackLevel = 4
}
```

Figure 2-117 and Table 2-2 show the scope of the maximum stack level check for each checking mode when you specify a `maxStackLevel` value of 4. The figure shows a cross-section view of the metal layers and black vias between them. The presence of two side-by-side vias indicates a via array. The table indicates whether a violation is detected for each checking mode, 0 through 3, and each case, A through E, based on the number of stacked vias, the alignment of the vias, and whether the vias are single or arrayed.

Figure 2-117 Via Stack Diagram

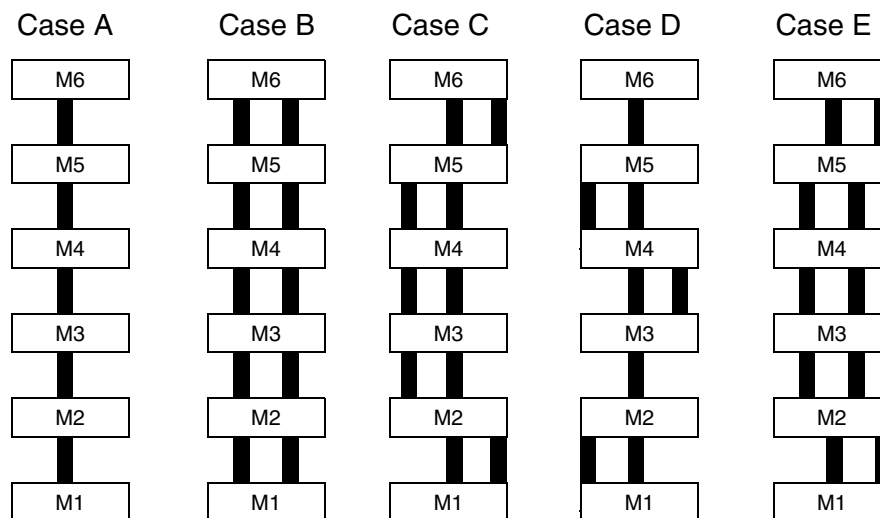


Table 2-2 Via Stack Violation Conditions

checkViaArrayMaxStackLevel	Case A	Case B	Case C	Case D	Case E
0	Violation			Violation	
1	Violation	Violation	Violation	Violation	
2	Violation				
3	Violation			Violation	

You can optionally specify which named via types are considered via arrays for the stack level rule by using the `maxStackLevelCutAsViaArrayTbl` attribute. For example,

```
Technology {
    maxStackLevelMode = 2
}

Layer "Via12" {
    maxStackLevel           = 4
    cutNameTbl              = (Vsm, Vh, Vv, Vlg)
    maxStackLevel           = 3
    maxStackLevelCutAsViaArrayTbl = (0, 1, 1, 1)
}
```

A value of 1 in the table identifies the corresponding via type as a via array for purposes of applying the `maxStackLevel` rule. In this example, `Vsm` is considered a single via and `Vh`, `Vv`, and `Vlg` are considered via arrays for applying the `maxStackLevel` rule.

When vias are stackable, the Zroute router can place one via over the other with their centers exactly aligned. You can optionally specify an allowable amount of offset between the centers of the stacked vias by using the `stackViaCenterSpacingThreshold` attribute, as in the following example:

```
DesignRule {
    layer1           = "VIA1"
    layer2           = "VIA2"
    stackable        = 1
    stackViaCenterSpacingThreshold = 0.01
}
```

In this example, Zroute can stack a via in layer VIA2 over a via in layer VIA1 with a center-to-center offset of up to 0.01.

The classic router's via stack rule behavior is similar, except that it ignores the `maxStackLevelCutAsViaArrayTbl` attribute and treats all cut types as single vias when applying the rule.

Via Array (Via Farm) Rule

The via farm rule specifies the maximum number of rows in a via array and the minimum spacing between two via arrays. This rule is used only by the power and ground router.

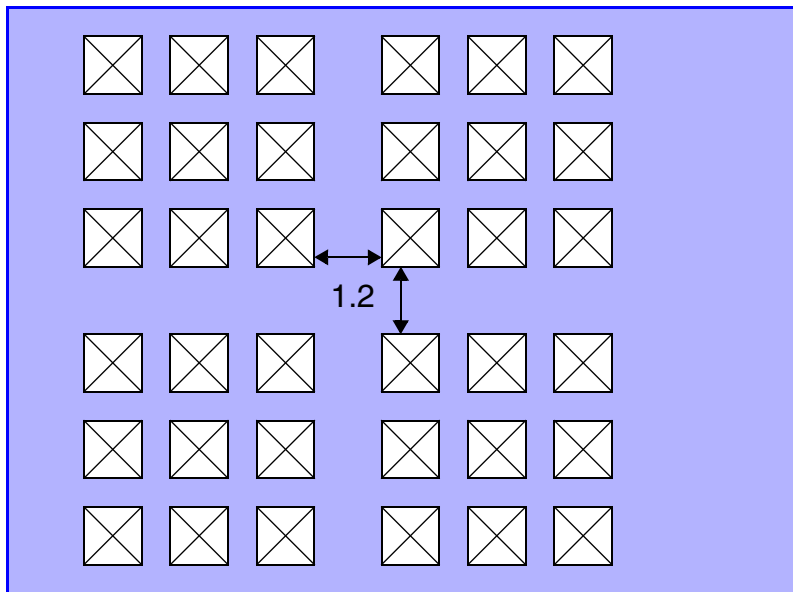
This rule is defined in the `ContactCode` section of the technology file. Each via array requires a `ContactCode` definition. To specify the via array size, use the `maxNumRows` attribute. To specify the spacing between two via arrays, use the `viaFarmSpacing` attribute.

For example,

```
ContactCode "VIA12_fat" {
  maxNumRows      = 3
  viaFarmSpacing = 1.2
}
```

In this example, the maximum via array size is 3 by 3 and the minimum spacing between via arrays is 1.2, as shown in [Figure 2-118](#).

Figure 2-118 Via Farm Rule



By default, the maximum number of rows and via farm spacing constraints apply to both the horizontal and vertical directions. To apply these rules only in the direction of the longer wire intersections, set the `viaFarmLongDirection` attribute to 1. In this case, the number of rows and spacing constraints in the shorter direction are determined by the intersection boundaries and minimum cut spacing.

For example,

```
ContactCode "VIA12_fat" {
  maxNumRows      = 3
  viaFarmSpacing = 1.2
  viaFarmLongDirection = 1
}
```

Via Density Rule

Use the following attributes to specify the via density rule:

`layer`

Defines the via layer for which the rule is specified.

`windowSize`

Defines the size of the window for the density check. By default, the window step size is half of the defined `windowSize`.

`minDensity`

Defines the minimum percentage of metal allowed in the window.

`maxDensity`

Defines the maximum percentage of metal allowed in the window.

These attributes are defined in the `DensityRule` section of the technology file. Each via layer requires a `DensityRule` definition. For example,

```
DensityRule {  
    layer = "VIA1"  
    windowSize = 200  
    minDensity = 2  
    maxDensity = 5  
}
```

Vias on Nonpreferred Route Direction Rule

By default, vias can be placed on metal routes running in the preferred or nonpreferred direction. To restrict the placement of vias to routes running in the preferred direction only, set the `nonPreferredRouteMode` attribute to 1 in the via's `Layer` section. For example,

```
layer "VIA12" {  
    nonPreferredRouteMode = 1  
}
```

In this mode, the router avoids placing a via on any route segment that runs in the nonpreferred direction. The design rule checker checks for any via cut that overlaps a route in the nonpreferred direction and flags such an occurrence as a DRC error. The rule applies during global routing, track assignment, and detail routing. If a via surrounds, overlaps, or abuts a pin, checking of the rule is waived on the pin layer.

Metal and Via on Wire Track Rules

To enforce the alignment of metal wire centerlines to the wire track grid, use the following syntax in the metal layer section of the technology file:

```
Layer "M2" {  
    onWireTrack = 1  
}
```

In this example, Zroute aligns the centerlines of M2 metal routes to the wire track grid. This alignment applies to both M2 metal routes and M2 metal enclosures associated with vias.

Similarly, to enforce the alignment of via centers to the wire track grid, use the following syntax in the via layer section of the technology file:

```
Layer "V23" {  
    onWireTrack = 1  
}
```

In this example, Zroute aligns the centers of V23 vias to the wire track grid. This alignment applies only to the cut layer, not to the M2 and M3 metal enclosures associated with the vias.

Setting the `onWireTrack` attribute to 0 allows wires or vias to fall anywhere, irrespective of the wire track grid.

Metal and Via Alignment Using the onGrid Attribute

The technology file syntax and Zroute router also support an older layer attribute, `onGrid`. The `onGrid` attribute has the same behavior as `onWireTrack`, except that the alignment characteristic specified for a metal layer applies only to metal wires and not via enclosures, and the alignment characteristic specified for a via layer applies to the cut layer and also the associated metal enclosures. Technology files using the older `onGrid` attribute should be updated to use the `onWireTrack` attribute. If both the `onGrid` and `onWireTrack` attributes are used in the same technology file, the `onGrid` attribute settings are ignored.

Note that the `onGrid` attribute, like `onWireTrack`, controls the alignment of routes and vias to the wire track grid, not the manufacturing database grid.

Metal and Via Alignment Using the Classic Router

For the classic router, you can use the `noOffGridRouting` detail route option to specify that all vias above metal layer M2 must be on grid. Enter

```
icc_shell> set_droute_options -name noOffGridRouting -value 4
```

Keep the following points in mind:

- Only vias must be on grid. Wires can be routed off grid to allow them to touch off-grid pins.
- V12 can be off grid to allow wires to touch off-grid pins.

For the classic router only, you can use the `noOffGridRouting` detail route option and the `VnNoOffGridRouting` detail route option to specify a layer-by-layer control for checking that vias are on grid, where *n* specifies the layer. You must set `noOffGridRouting` to 4 and set `VnNoOffGridRouting` to 1 for each layer that requires the vias to be on grid.

The syntax is

```
set_droute_options -name VnNoOffGridRouting -value value
```

Value	Description
0	Ignores the via-on-grid check for the specified layers
1	Checks that vias are on grid for the specified layers (the default)

For example, when V2 through V5 are required to be on grid, use the following commands:

```
icc_shell> set_droute_options -name noOffGridRouting -value 4
icc_shell> set_droute_options -name V1NoOffGridRouting -value 0
icc_shell> set_droute_options -name V2NoOffGridRouting -value 1
icc_shell> set_droute_options -name V3NoOffGridRouting -value 1
icc_shell> set_droute_options -name V4NoOffGridRouting -value 1
icc_shell> set_droute_options -name V5NoOffGridRouting -value 1
icc_shell> set_droute_options -name V6NoOffGridRouting -value 0
...
icc_shell> set_droute_options -name V12NoOffGridRouting -value 0
```

Keep the following points in mind:

- The grid must be defined as a preferred-direction-to-preferred-direction grid. The rule does not consider nonpreferred direction grids.
- The via-on-grid rule checks the upper-layer preferred tracks and lower-layer preferred tracks only.

Wire Shape Rules

The following types of rules control the allowed wire shapes:

- [Jog Wire Rules](#)
- [Dog Bone Rule](#)
- [Protrusion Length Rule](#)

Jog Wire Rules

The jog wire rules are described in the following sections:

- [Small Jog Rule](#)
- [Jog Wire End-of-Line Via Rule](#)
- [Jog Wire Via Keepout Region Rule](#)
- [Line-End to Jog Distance Rule](#)

Small Jog Rule

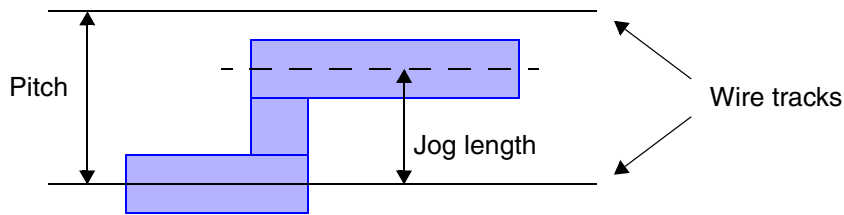
Use the `droute_smallJogMinLength` variable to specify the minimum length allowed for a small jog. The following table shows the valid values for this variable and their meanings.

Value	Description
0	The small jog rule is not checked.
1	The jog length needs to be greater than or equal to 1/4 pitch of the wire tracks for the routing layer.
2	The jog length needs to be greater than or equal to 1/2 pitch of the wire tracks for the routing layer.

When you specify a value of 1 or 2, the small jog rule is checked, and then fixed by the search-and-repair operation. A violation occurs when the length of the jog is shorter than 1 or 2 times the quarter pitch of the wire tracks.

[Figure 2-119](#) shows the attributes associated with the small jog rule.

Figure 2-119 Small Jog Rule



For example, when you enter the following:

```
icc_shell> set droute_smallJogMinLength 2
```

a violation occurs when the jog is shorter than half the pitch of the wire track.

Jog Wire End-of-Line Via Rule

Use the `endOfLineViaJogWidth`, `endOfLineViaEncWidth` and `endOfLineViaJogLength` attributes to specify the jog wire end-of-line via rule. These attributes let you define additional location constraints for vias that connect to short jog wires. Define these attributes in a `DesignRule` section of the technology file.

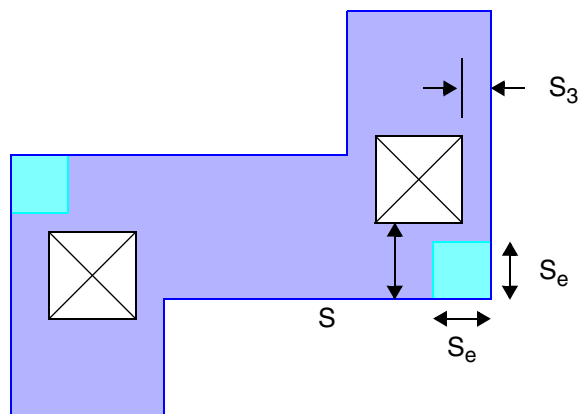
Jog Wire Via Keepout Region Rule

Use the `jogWireViaKeepoutTblSize`, `jogWireViaKeepoutEncThreshold`, and `jogWireViaKeepoutMinSize` attributes to specify the jog wire via keepout region rule. When you do not want a single via on a jog wire to be placed too close to the outside corner of the jog wire, use this rule to define a via keepout region at the jog wire corner. Define these attributes in the `DesignRule` section of the technology file, by specifying different values between the metal and via layer. For example,

```
DesignRule {
  layer1           = "M1"
  layer2           = "V1"
  minEnclosure     = 0
  jogWireViaKeepoutTblSize = 3
  jogWireViaKeepoutEncThreshold = (0.005,0.015,0.025)
  jogWireViaKeepoutMinSize = (0.08,0.06,0)
}
```

Figure 2-120 shows the attributes associated with the jog wire via keepout rule. In this example, when the via enclosure S_3 is less than 0.005, then the spacing to the corner S_e must be at least 0.08; the keepout region at the corner measures 0.08 by 0.08. When the via enclosure S_3 is greater than 0.005 but less than 0.015, then the spacing to the corner S_e must be at least 0.06.

Figure 2-120 Jog Wire Via Keepout Region Rule

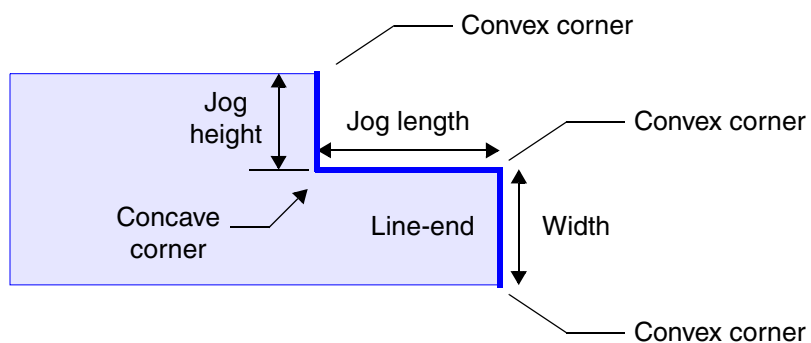


This rule applies only to a single via enclosed within a jog wire. It does not apply to a double via in a given jog wire.

Line-End to Jog Distance Rule

The line-end jog distance rule specifies the minimum lengths of segments at a jog near a line-end. The rule applies to any three consecutive polygon edges that meet at a sequence of convex, convex, concave, and convex corners, as shown in [Figure 2-121](#).

Figure 2-121 Jog Wire Via Keepout Region Rule



The segment between two convex corners is a line-end and the two adjacent segments form a jog. When the line-end width is less than the minimum width attribute W , then either the jog height must be at least the minimum height attribute H or the jog length must be at least the minimum length attribute L . In other words, if the jog height is less than H , then the jog length must be at least L .

This is the general form of the rule:

```
Layer "MetalX" {
  minEdgeJogWireMinWidth  = W
  minEdgeJogWireMinLength = L
  minEdgeJogMinHeight     = H
}
```

For example,

```
Layer "M1" {
  minEdgeJogWireMinWidth  = 0.068
  minEdgeJogWireMinLength = 0.136
  minEdgeJogMinHeight     = 0.058
}
```

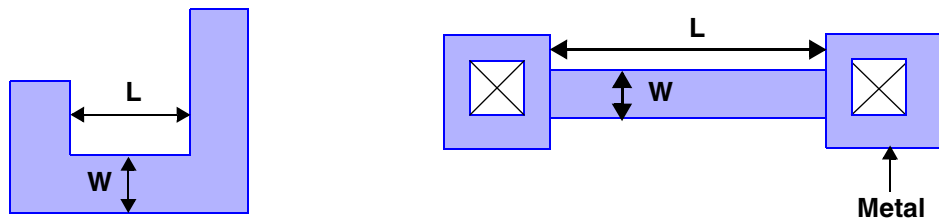
Dog Bone Rule

The dog bone rule applies during wire jogging and pin access when notches can appear next to narrow wires. A violation occurs when both of the following conditions exist:

- The metal width is less than the value specified with the `sameNetWidthThreshold` attribute ($W < X$).
- The notch spacing is less than the value specified with the `sameNetMinSpacing` attribute ($L < Y$).

Figure 2-122 shows the attributes associated with the dog bone rule.

Figure 2-122 Dog Bone Rule



Use the following attributes to specify the dog bone rule:

`sameNetWidthThreshold`

Specifies a number that represents the threshold value for the thin wire width.

`sameNetMinSpacing`

Specifies the minimum spacing between inside edges of the wire.

Define these attributes in a metal `Layer` section of the technology file.

For example,

```
Layer "M5" {
    sameNetWidthThreshold    = 0.5
    sameNetMinSpacing        = 1.0
}
```

Protrusion Length Rule

The protrusion length rule applies when a fat wire has both of the following:

- A width larger than a threshold value `T` specified with the `protrusionFatThresholdTbl` attribute
- A connected thin wire whose length is less than a value `L` specified with the `protrusionLengthLimitTbl` attribute

In such cases, the thin wire must have a width of at least the value `W` specified with the `protrusionMinWidthTbl` attribute.

Use the following attributes to specify the protrusion length rule:

`protrusionFatThresholdTbl`

Specifies a floating-point number that represents the threshold value for the fat wire.

`protrusionLengthLimitTbl`

Specifies a floating-point number that represents the length value for the connected thin wire.

`protrusionMinWidthTbl`

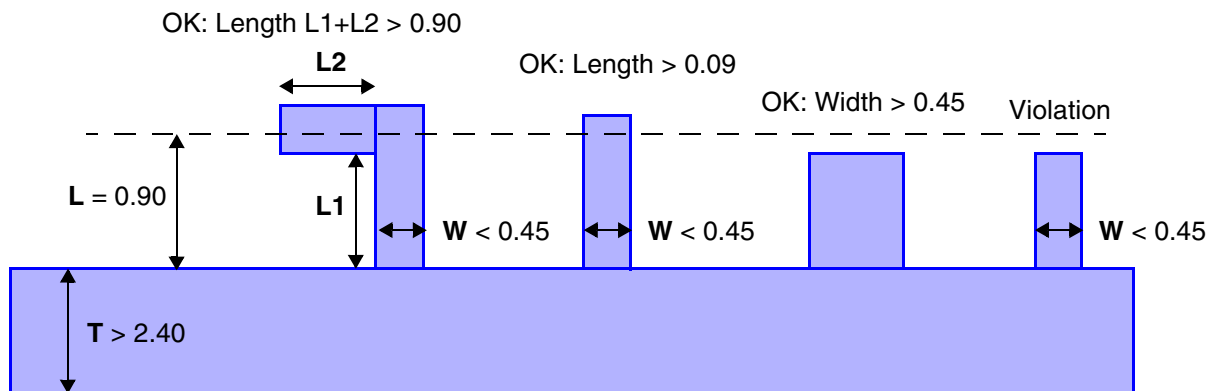
Specifies a floating-point number that represents the minimum width value for the connected thin wire.

Define these attributes in a metal `Layer` section of the technology file. For example,

```
Layer "M2" {
    protrusionTblDim          = 2
    protrusionFatThresholdTbl = (1.20,2.40)
    protrusionLengthLimitTbl  = (0.60,0.90)
    protrusionMinWidthTbl     = (0.30, 0.45)
}
```

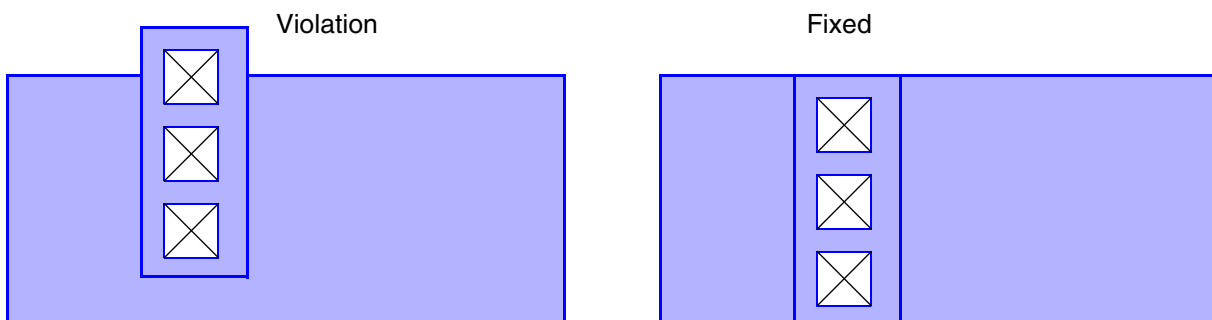
[Figure 2-123](#) shows the attributes associated with the protrusion length rule.

Figure 2-123 Protrusion Length Rule



A protrusion length violation can occur where a wire is connected to a fat power or ground net by dropping a via or via array, as shown in [Figure 2-124](#). A violation such as this can be fixed by adding metal stubs or rerouting the wire.

Figure 2-124 Protrusion Length Rule Violation and Correction



Metal Density Rules

This section describes the metal density rule and the metal density gradient rule.

Metal Density Rule

Use the following attributes to specify the metal density rule:

`layer`

Defines the metal layer for which the rule is specified.

`windowSize`

Defines the size of the window for the density check. By default, the window step size is half of the defined `windowSize`.

`minDensity`

Defines the minimum percentage of metal allowed in the window.

`maxDensity`

Defines the maximum percentage of metal allowed in the window.

These attributes are defined in the `DensityRule` section of the technology file. Each metal layer requires a `DensityRule` definition. For example,

```
DensityRule {  
    layer = "M1"  
    windowSize = 200  
    minDensity = 20  
    maxDensity = 80  
}
```

The metal density rule is used by the `signoff_metal_fill`, `report_metal_density`, and `insert_metal_filler` commands. A DRC violation occurs if the total percentage of the named metal layer in the window size is not within the specified minimum and maximum density limits.

Metal Density Gradient Rule

In the `DensityRule` section of the technology file, the `densityGradient` attribute specifies the maximum allowable the change in fill density between adjacent windows. The density gradient information is used by the `signoff_metal_fill`, `report_metal_density`, and `insert_metal_filler` commands.

The fill density of each window is compared with the corresponding density of its neighbors. If the percentage difference between them exceeds the density gradient of that layer, the fill is trimmed to satisfy the density gradient rule.

Parallel Length-Based Floating Wire Antenna Rule

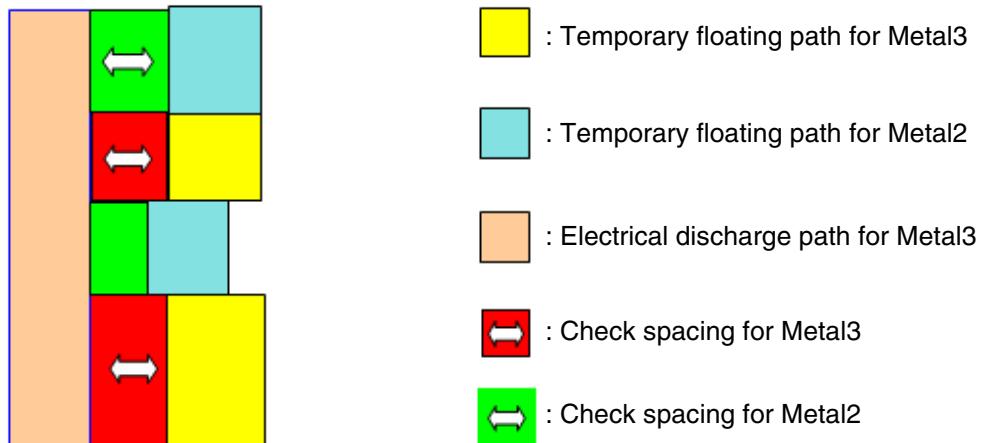
IC Compiler checks floating wire antennas by checking the maximum allowable value for metal areas and the minimum spacing to the adjacent wire. The parallel length-based floating wire antenna rule also considers the parallel length between the floating wire and the adjacent wire. Use the following options to specify the parallel length-based floating wire antenna rule:

```
M1FloatingWirePLength1-5
M2FloatingWirePLength1-5
...
M12FloatingWirePLength1-5
M1FloatingWirePLMinSpcl-5
M2FloatingWirePLMinSpcl-5
...
M12FloatingWirePLMinSpcl-5
```

The floating wires will be merged according to the values you specify.

For example, in [Figure 2-125](#), the Metal2 and Metal3 floating wires will be merged.

Figure 2-125 Parallel Length-Based Floating Wire Antenna Rule



3

Routing Rules for Advanced Geometries

IC Compiler, when used with the ICC-AG package (which includes the Galaxy-AdvRules license), supports routing design rules for advanced emerging technologies, extending down to the 20 nm geometry node and below. These emerging technologies often require complex rules, as described in the following sections:

- [Double-Patterning Spacing Rules](#)
- [Self-Aligned Via Rules](#)
- [Minimum Edge, Width, and Area Rules](#)
- [Metal Spacing Rules](#)
- [Via Spacing Rules](#)
- [Via Enclosure Rules](#)
- [Contact Code Rules](#)
- [IC Compiler Zroute Error Messages](#)

Double-Patterning Spacing Rules

The double-patterning spacing rules specify the minimum spacing between the sides, ends, and corners of metal route geometries in odd-cycle configurations.

In a double-patterning technology, the geometries of a metal layer are decomposed into two mutually exclusive sets, with each set implemented in a separate photomask. The photomasks are used to make successive exposures on the same photoresist, so that the metal layer is fabricated in a single process step after the two exposures.

The purpose of double-patterning is to reduce the effects of optical interference between adjacent geometries. The decomposition of the geometries of a metal layer into two sets, a process called “coloring,” attempts to separate adjacent geometries by assigning them alternating “colors.” Each “color” is implemented as a separate photomask.

In cases where an odd number of geometries are adjacent to each other in a closed-loop pattern, the geometries cannot be entirely decomposed into separated sets. In this situation, the double-patterning spacing rules apply to the set of geometries.

The Zroute router in IC Compiler does not perform coloring. Instead, it attempts to perform routing in a manner that makes the patterns “colorable” by avoiding odd-cycle patterns. If it detects a double-patterning spacing violation, it does not report the violation, but instead fixes the spacing or the odd cycle leading to the violation.

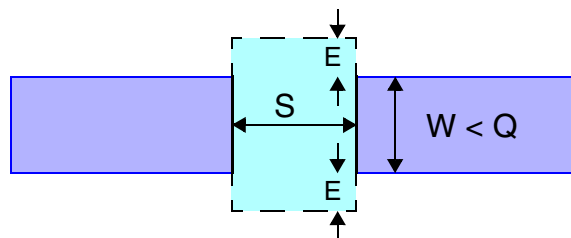
These are the double-patterning spacing rules:

- [Double-Patterning Line-End to Line-End Spacing Rule](#)
- [Double-Patterning Line-End to Line-Side Spacing Rule](#)
- [Double-Patterning Line-Side to Line-Side Spacing Rule](#)
- [Double-Patterning Corner-to-Corner Spacing Rule](#)
- [Double-Patterning Fat Metal Spacing Rule](#)

Double-Patterning Line-End to Line-End Spacing Rule

The double-patterning line-end to line-end spacing rule specifies the minimum spacing between two line-ends facing each other. A line-end is an edge that connects two convex corners and has a length less than a width threshold Q . The keepout area extends for a length E outside the end of the line, as shown in [Figure 3-1](#).

Figure 3-1 Double-Patterning Line-End to Line-End Spacing Rule



```

Layer "MetalX" {
  doublePatternEndMaxWidthThreshold = Q
  doublePatternEndToEndKeepoutLength = 2*E - 0.001
  doublePatternEndToEndMinSpacing = A
}

```

For example,

```

Layer "MetalX" {
  doublePatternEndMaxWidthThreshold = 0.100
  doublePatternEndToEndKeepoutLength = 0.049
  doublePatternEndToEndMinSpacing = 0.120
}

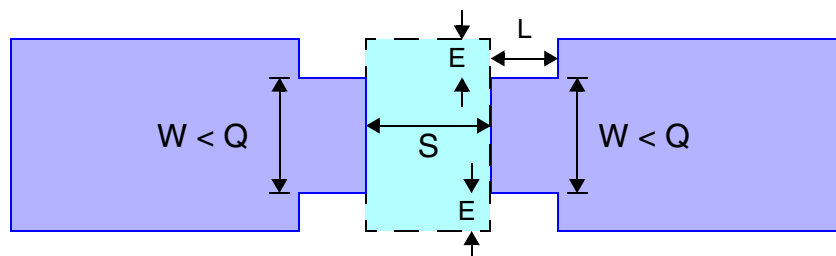
```

In this example, the line-ends have a length of no more than 0.100, and the minimum spacing is 0.120. The keepout area extends for 0.025 outside the end of the line. The keepout attribute is specified as $2 \cdot E - 0.001$, where E (keepout attribute) is 0.025. Thus the attribute is specified as $2 \cdot 0.025 - 0.001 = 0.049$.

You can optionally specify a minimum run length L of the line width away from the line-end for the rule to apply. In that case, specify the additional attribute

`doublePatternEndMinLength`, as shown in Figure 3-2.

Figure 3-2 Double-Patterning Line-End to Line-End Spacing Rule



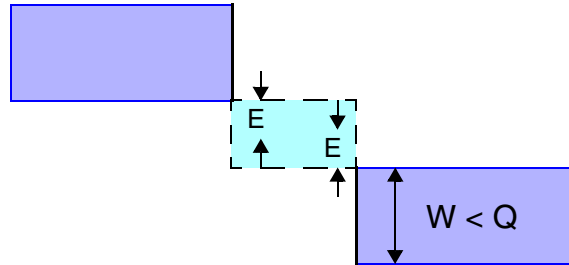
```

Layer "MetalX" {
  doublePatternEndMaxWidthThreshold = Q
  doublePatternEndMinLength = L
  doublePatternEndToEndKeepoutLength = 2*E - 0.001
  doublePatternEndToEndMinSpacing = A
}

```

The spacing check region extends by the distance E from both line-ends, so when the line-ends have a negative parallel run, as shown in [Figure 3-3](#), the check still applies as long as the magnitude of the negative run is less than 0.049.

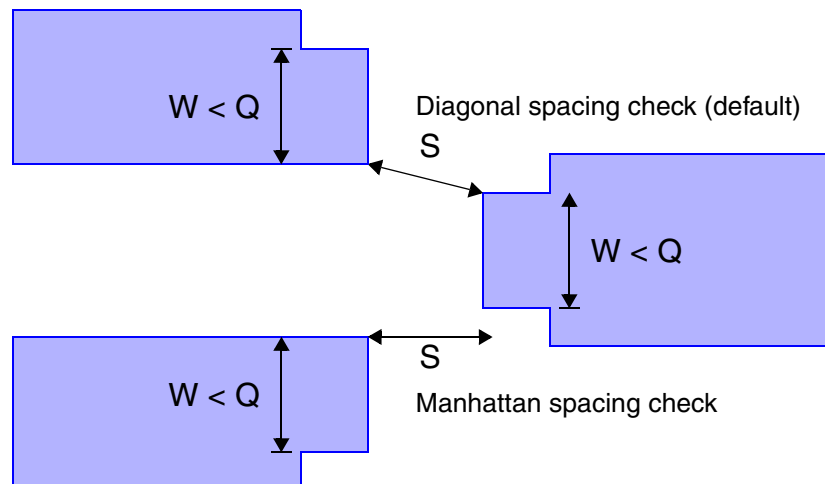
Figure 3-3 Line-End to Line-End Spacing Rule With Negative Parallel Run



With a negative parallel run, the check from corner to corner is performed diagonally by default, as shown in the top portion of [Figure 3-4](#). To perform the check using Manhattan measurement instead, as shown in the bottom portion of the figure, add the following statement:

```
doublePatternCheckManhattanSpacing = 1
```

Figure 3-4 Diagonal and Manhattan Line-End to Line-End Spacing Checks

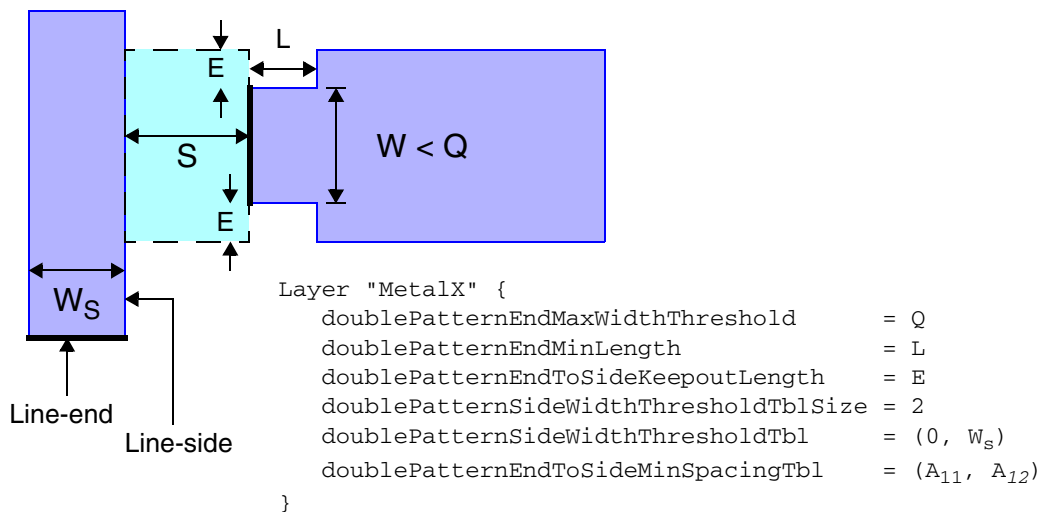


Double-Patterning Line-End to Line-Side Spacing Rule

A line-end is an edge that connects two convex corners and has a length less than a width threshold Q . Any metal edge that is not a line-end is a line-side.

The double-patterning line-end to line-side spacing rule specifies the minimum spacing between a line-end and the line-side of a nearby wire. The rule applies when the line-side at the corner of the line-end runs at least a length L away from the line-end. The keepout area extends for a length E outside the end of the line, as shown in [Figure 3-5](#).

Figure 3-5 Double-Patterning Line-End to Line-Side Spacing Rule



The minimum spacing depends on the width of line whose side is facing the line-end, as expressed in a table. For example,

```

Layer "MetalX" {
    doublePatternEndMaxWidthThreshold      = 0.10
    doublePatternEndMinLength              = 0.04
    doublePatternEndToSideKeepoutLength    = 0.025
    doublePatternSideWidthThresholdTblSize = 2
    doublePatternSideWidthThresholdTbl     = (0.00, 0.05)
    doublePatternEndToSideMinSpacingTbl    = (0.10, 0.07)
}

```

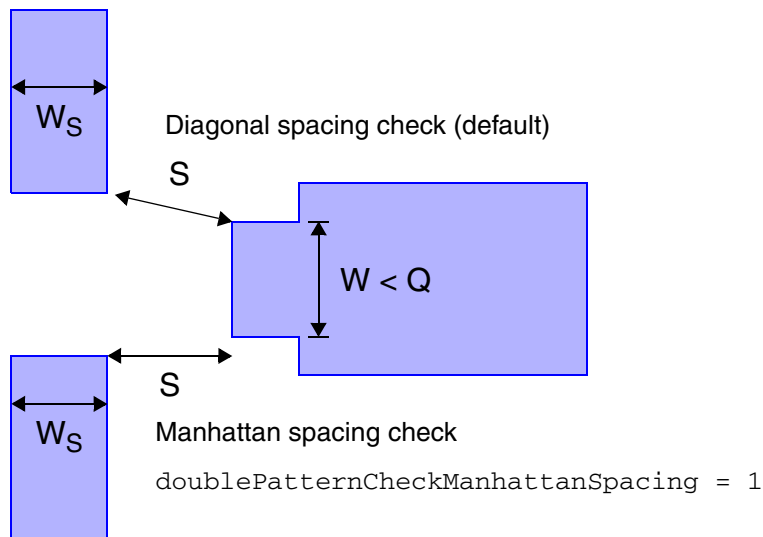
In this example, when the width at the line-end is less than 0.10 and extends at that width for at least 0.04 away from the line-end, and the width of the nearby line is at least 0.00 but less than 0.05, the minimum spacing is 0.10. If the width of the nearby line is at least 0.05, the minimum spacing is 0.07. The keepout area extends for 0.025 outside the end of the line.

If the width of the line-side metal W_S does not matter, you can use the non-table-based form of the rule, as in the following example:

```
Layer "MetalX" {
  doublePatternEndMaxWidthThreshold = 0.10
  doublePatternEndMinLength         = 0.04
  doublePatternEndToSideKeepoutLength = 0.03
  doublePatternEndToSideMinSpacing   = 0.10
}
```

When the parallel overlap between the line-end and line-side is negative, the check from corner to corner is performed diagonally by default, as shown in the top portion of [Figure 3-6](#).

Figure 3-6 Diagonal and Manhattan Line-End to Line-Side Spacing Checks



To perform the check using Manhattan measurement, as shown in the bottom portion of the figure, add the following statement:

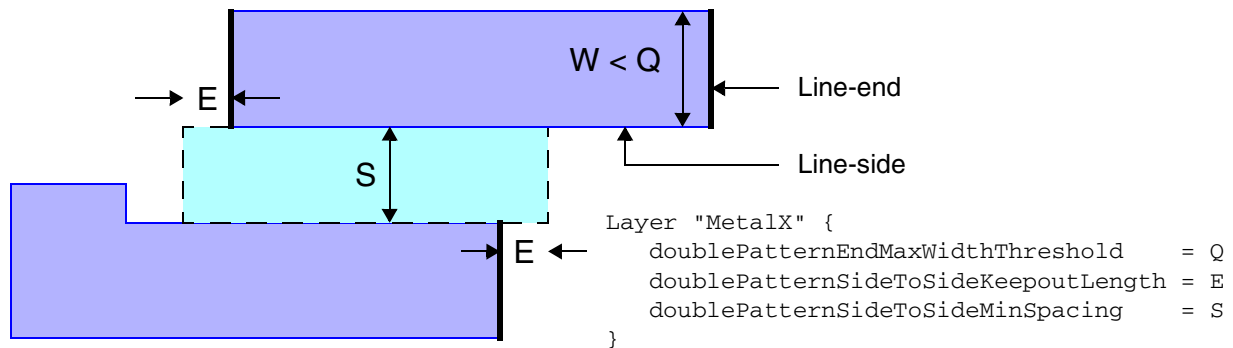
```
doublePatternCheckManhattanSpacing = 1
```

Double-Patterning Line-Side to Line-Side Spacing Rule

A line-end is an edge that connects two convex corners and has a length less than a width threshold Q . Any metal edge that is not a line-end is a line-side.

The double-patterning line-side to line-side spacing rule specifies the minimum spacing S between adjacent line-sides belonging to different wires. The rule applies for a keepout length E beyond the ends of the lines, as shown in [Figure 3-7](#).

Figure 3-7 Double-Patterning Line-Side to Line-Side Spacing Rule



For example,

```

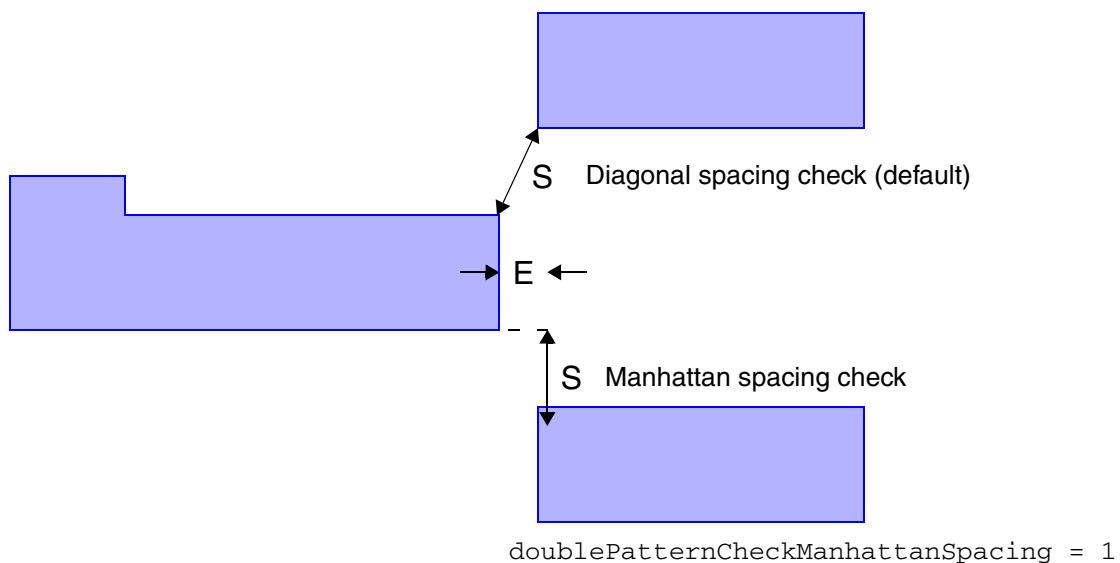
Layer "MetalX" {
    doublePatternEndMaxWidthThreshold    = 0.100
    doublePatternSideToSideKeepoutLength = 0.025
    doublePatternSideToSideMinSpacing    = 0.080
}

```

In this example, the spacing between the line-sides must be at least 0.080. The rule applies within the keepout length 0.025 beyond the line-ends.

When the parallel overlap between the line-sides is negative, the check from corner to corner is performed diagonally by default, as shown in the top portion of [Figure 3-8](#).

Figure 3-8 Diagonal and Manhattan Line-Side to Line-Side Spacing Checks



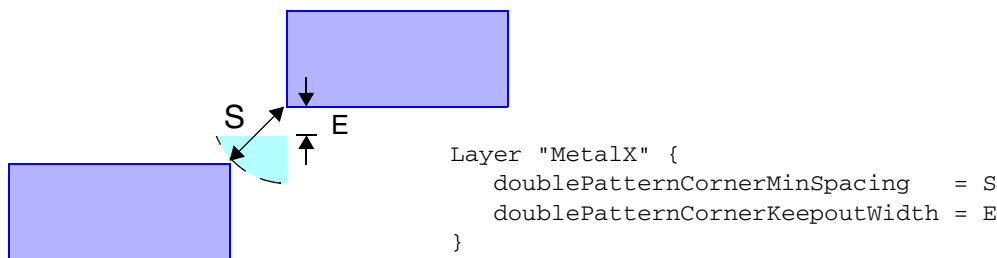
To perform the check using Manhattan measurement instead, as shown in the bottom portion of the figure, add the following statement:

```
doublePatternCheckManhattanSpacing = 1
```

Double-Patterning Corner-to-Corner Spacing Rule

The double-patterning corner-to-corner spacing rule specifies the minimum spacing between two corners, irrespective of other double-patterning rules that might apply. The check is always performed diagonally and is only performed outside of a keepout width away from the corner, as shown in [Figure 3-9](#).

Figure 3-9 Double-Patterning Corner-to-Corner Spacing Rule



For example,

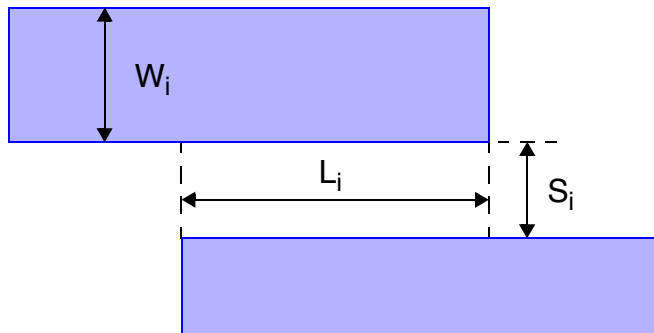
```
Layer "MetalX" {
    doublePatternCornerMinSpacing = 0.10
    doublePatternCornerKeepoutWidth = 0.02
}
```

When the corners lie within the specified keepout width, the rule does not apply, but other rules might apply such as line-side to line-side, line-side to line-end, and line-end to line-end.

Double-Patterning Fat Metal Spacing Rule

The double-patterning spacing rule can depend on width, with wider metal lines requiring larger minimum spacing values. In that case, the fat metal spacing rule provides a table of width threshold, parallel length, and spacing values, as shown in [Figure 3-10](#). This rule applies to both line-ends and line-sides, as long as the metal widths meet the threshold requirements.

Figure 3-10 Double-Patterning Fat Metal Spacing Rule



```

Layer "MetalX" {
  doublePatternFatWireTblSize           = n
  doublePatternFatWireThresholdTbl      = (W1, W2, ..., Wn)
  doublePatternFatWireParallelLengthTbl = (L1, L2, ..., Ln)
  doublePatternFatWireSpacingTbl        = (S1, S2, ..., Sn)
}

```

For example,

```

Layer "MetalX" {
  doublePatternFatWireTblSize           = 2
  doublePatternFatWireThresholdTbl      = (0.08, 0.10)
  doublePatternFatWireParallelLengthTbl = (0.10, 0.11)
  doublePatternFatWireSpacingTbl        = (0.10, 0.12)
}

```

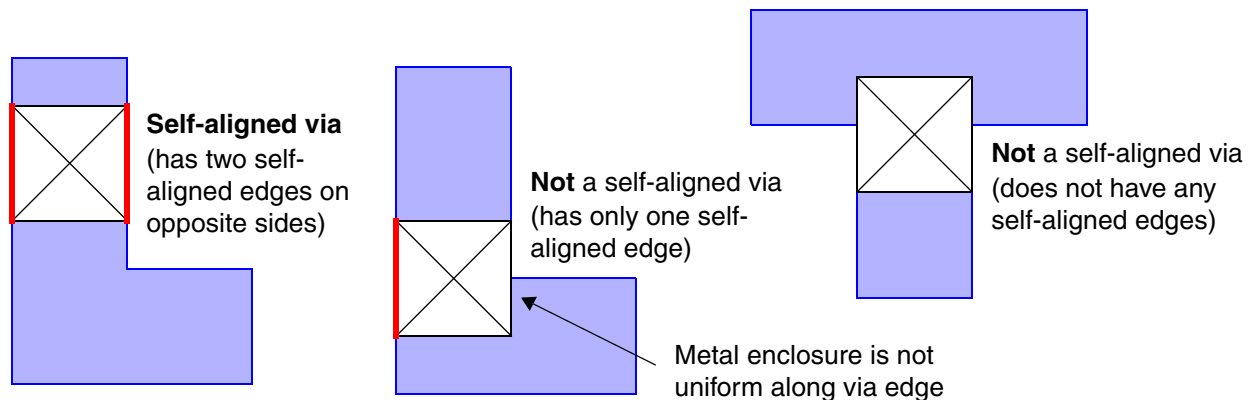
In this example, when the wire width is at least 0.08 but less than 0.10, and the parallel overlap is at least 0.10, the minimum spacing is 0.10. When the wire width is at least 0.10, and the parallel overlap is at least 0.11, the minimum spacing is 0.12.

Self-Aligned Via Rules

A *self-aligned edge* is an edge of a via that coincides with or is uniformly enclosed by an edge of the upper metal for the full length of the edge, where the amount of enclosure is exactly one of a number of specified values. A *self-aligned via* is a via with two self-aligned edges on opposite sides of the via.

In [Figure 3-11](#), the red line segments mark the self-aligned edges of the vias. The via on the left is a self-aligned via, but the other two are not.

Figure 3-11 Self-Aligned Edges and Self-Aligned Vias



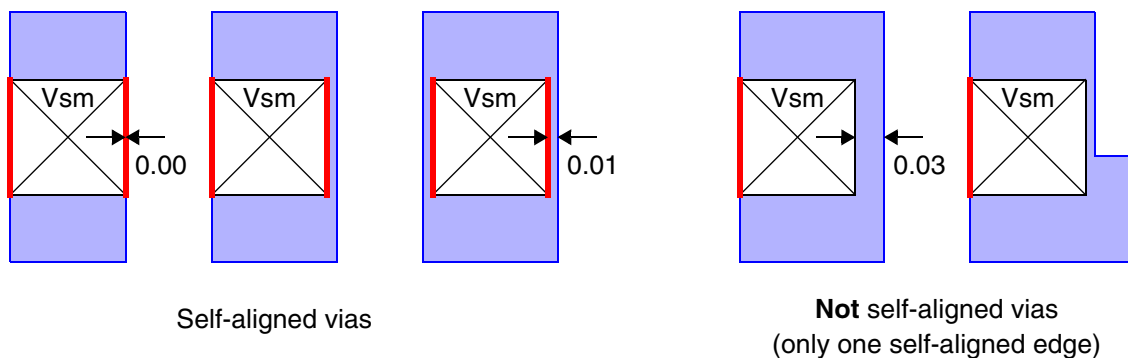
The self-aligned via enclosure rule specifies the amount of upper-metal enclosure that determines whether a via edge is a self-aligned edge. In the rule, you specify a table of cut layer names and cut names, and a table of enclosure values. For example,

```
ViaRule SAV1 {
  cutLayerNameTblSize      = 1
  cutLayerNameTbl          = (VIA1) # via layer names
  cutNameTbl               = (Vsm)  # cut names
  selfAlignedViaUpperLayerEncTblSize = 2
  selfAlignedViaUpperLayerEncTbl   = (0.00, 0.01)
}
```

This example specifies the enclosure values 0.00 and 0.01 for the via called `Vsm` in the cut layer `VIA1`. The cut name `Vsm` must be defined in the `Layer` section for the via layer earlier in the technology file. For each instance of this via, if two opposite sides of the via uniformly coincide with upper-metal edges, or are uniformly enclosed by the upper-metal geometry by a distance of 0.01, the via is a self-aligned via. See the additional examples in [Figure 3-12](#).

Figure 3-12 Self-Aligned Via Examples

```
selfAlignedViaUpperLayerEncTbl = (0.00, 0.01)
```



Several different rules apply specifically to self-aligned vias:

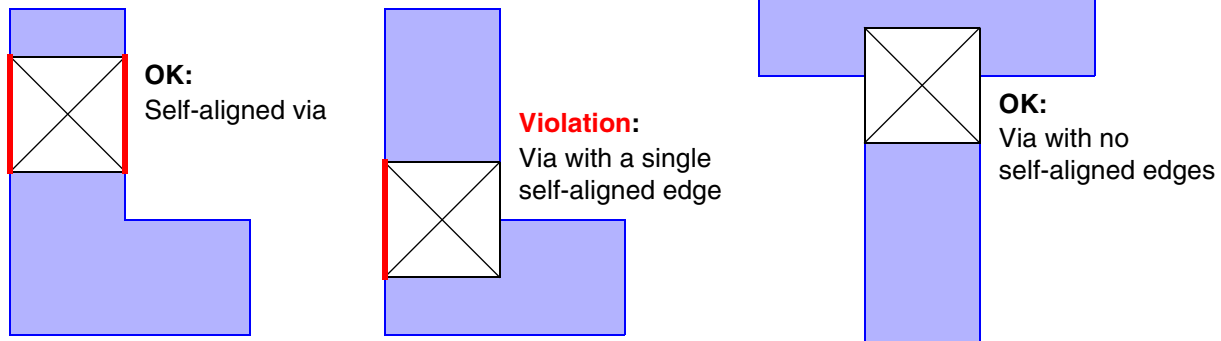
- [Single Self-Aligned Edge Forbidden Rule](#)
- [Self-Aligned Via Spacing Rules](#)
- [Self-Aligned Via Cluster Rules](#)
- [Self-Aligned Via Array Rule](#)
- [Self-Aligned Via Diagonal Spacing Rule](#)
- [Self-Aligned Via Zigzag Spacing Rule](#)

Single Self-Aligned Edge Forbidden Rule

The single self-aligned edge forbidden rule prevents the router from placing a via under metal such that exactly one edge of the via is a self-aligned edge. Two opposite self-aligned edges are allowed, as well as no self-aligned edges, as demonstrated in [Figure 3-13](#).

Figure 3-13 Single Self-Aligned Edge Forbidden Rule

```
selfAlignedViaOneEdgeNotAllowed = 1
```



To invoke this rule, add a line to the self-aligned via enclosure rule as in the following example:

```
ViaRule SAV1 {
    cutLayerNameTblSize      = 1
    cutLayerNameTbl          = (VIA1) # via layer names
    cutNameTbl                = (Vsm) # cut names
    selfAlignedViaUpperLayerEncTblSize = 2
    selfAlignedViaUpperLayerEncTbl    = (0.00, 0.01)
    selfAlignedViaOneEdgeNotAllowed    = 1
}
```

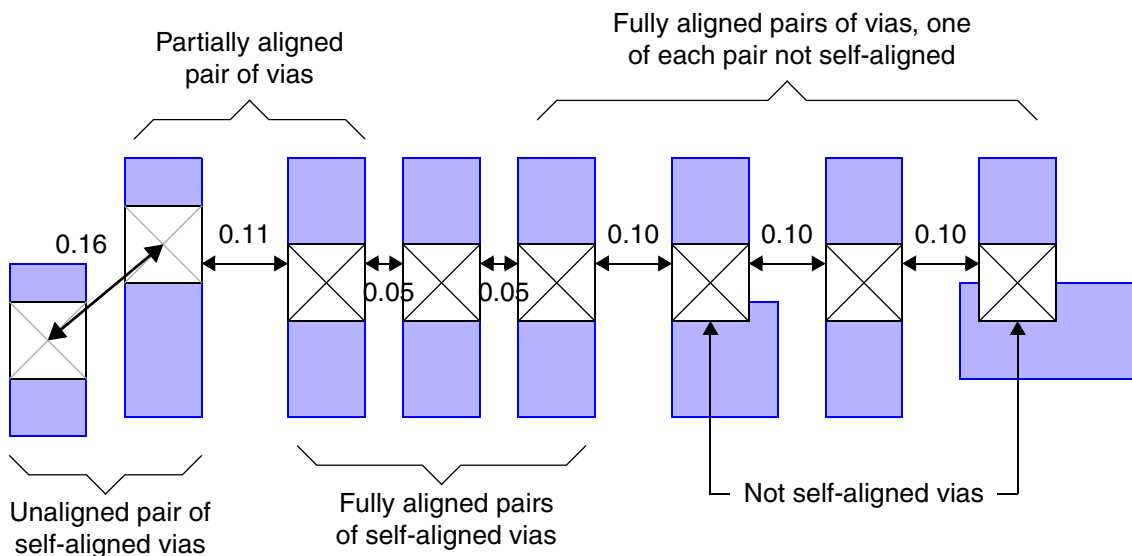
Self-Aligned Via Spacing Rules

The minimum spacing between vias can depend on their alignment to each other and their alignment to the upper-level metal. These are the types of via alignment configurations:

- Unaligned vias – The parallel overlap between the edges of the two vias is less than or equal to zero; there is no parallel overlap.
- Partially aligned vias – The parallel overlap between the edges of the two vias is greater than zero but less than the via edge length.
- Aligned vias – The parallel overlap between the edges of the two vias is equal to the edge length; the vias are exactly aligned.
- Self-aligned via – A via has two opposite edges that coincide with, or are uniformly enclosed by, the upper metal edges by a defined amount (see [“Self-Aligned Via Rules” on page 3-9](#)).

[Figure 3-14](#) shows examples of these alignment configurations.

Figure 3-14 Self-Aligned Via Spacing Rule Examples



The following example shows how to set the various spacing requirements:

```
DesignRule {
  layer1 = "VIA1"
  layer2 = "VIA1"
  cut1TblSize = 1
  cut2TblSize = 1
  cut1NameTbl = (Vsm)
```



```

cut2NameTbl = (Vsm)
unalignedViaCenterMinSpacingTbl = (0.16)
partiallyAlignedViaMinSpacingTbl = (0.11)
alignedViaMinSpacingTbl = (0.10)
selfAlignedViaMinSpacingTbl = (0.05)
}

```

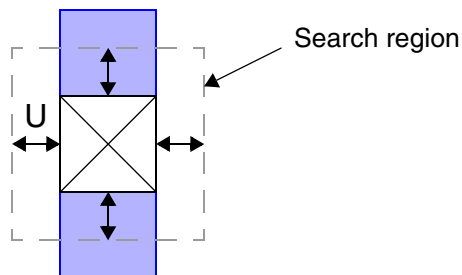
The rule is table-based. This simple example specifies the minimum spacing between Vsm cuts and other Vsm cuts in the VIA1 layer:

- A fully aligned pair of self-aligned vias: 0.05 (edge-to-edge spacing)
- A fully aligned pair of vias, one self-aligned and one not: 0.10 (edge-to-edge spacing)
- A partially aligned pair of vias (whether or not self-aligned): 0.11 (edge-to-edge spacing)
- An unaligned pair of self-aligned vias: 0.16 (separation measured center-to-center)

Self-Aligned Via Cluster Rules

A self-aligned via cluster is a single self-aligned via or a grouping of multiple self-aligned vias that meet specified spacing relationships. The self-aligned via cluster rules define a search region around each self-aligned via, made by expanding of the via edges outward by a specified amount U , as shown in [Figure 3-15](#).

Figure 3-15 Self-Aligned Via Cluster Search Region



When the search regions of two or more self-aligned vias touch or overlap, they form a single combined search region. The grouped vias can form the following types of clusters:

- Rectangular self-aligned via cluster – The individual search regions are exactly aligned on two edges, so that the merged region is rectangular, with the lengths of the each of the merged edges having a length of at least R_L .
- Diagonal self-aligned via cluster – The individual search regions overlap at a corner, producing a merged region with every edge having a length of at least D_L .

- Other type of self-aligned via cluster – Other types of merged shapes can be formed, such as a notch cluster or other odd shape. These other shapes can be forbidden by the self-aligned via cluster rules.

Figure 3-16 shows an example of a rectangular self-aligned via cluster containing two vias. The search regions are exactly aligned, and the length of the merged search region is at least R_L .

Figure 3-16 Rectangular Self-Aligned Via Cluster

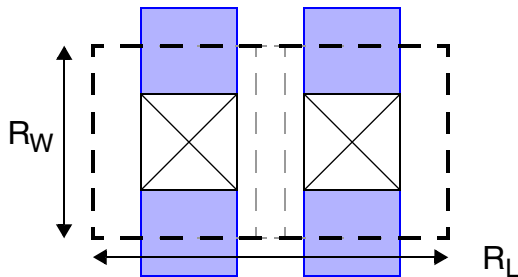


Figure 3-17 shows an example of a diagonal self-aligned via cluster containing two vias. The search regions overlap at the corners, and the lengths of the segments a , b , c , and d must each be at least a minimum D_{Lmin} and no more than a maximum D_{Lmax} .

Figure 3-17 Diagonal Self-Aligned Via Cluster

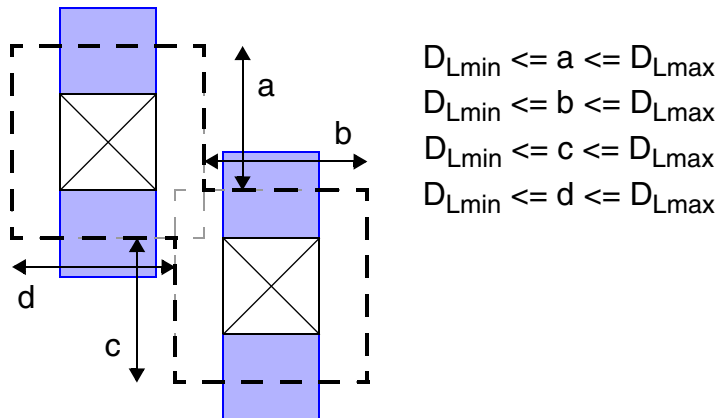
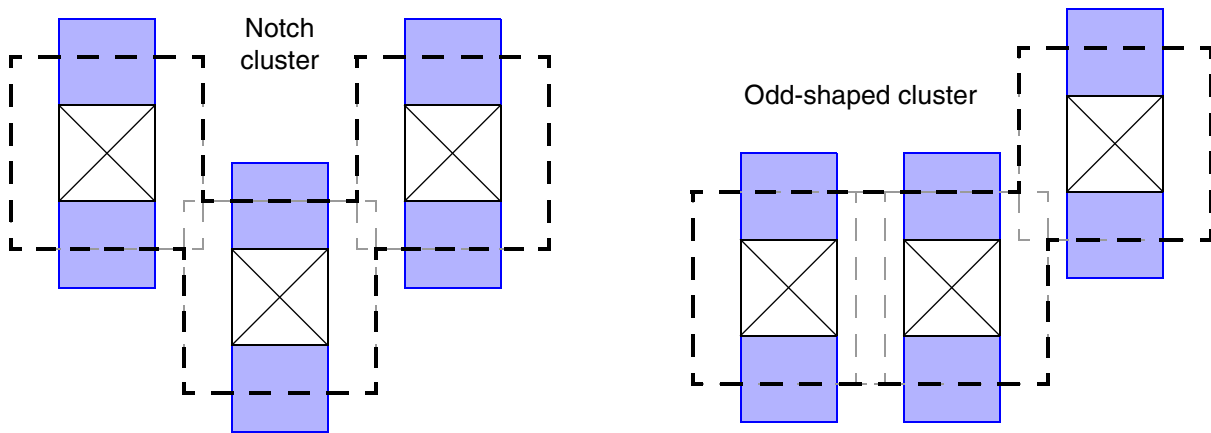
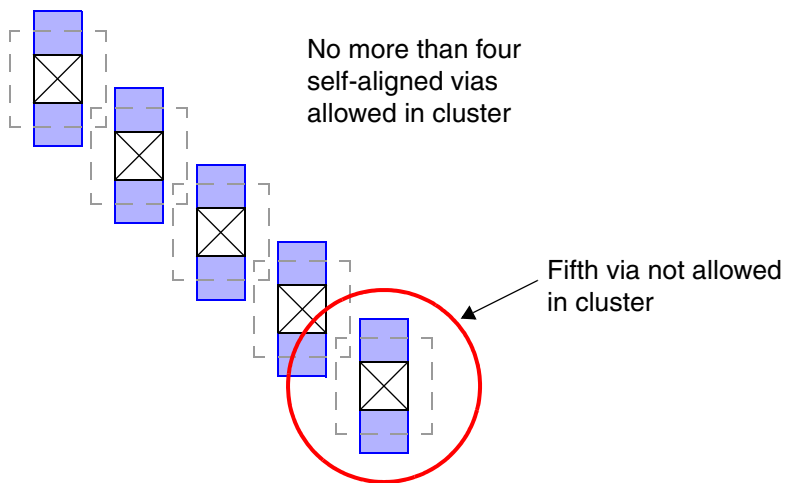


Figure 3-18 shows an example of a notch cluster and an odd-shaped cluster, which is neither a rectangular nor a diagonal cluster. These types of clusters can be forbidden by the cluster rules.

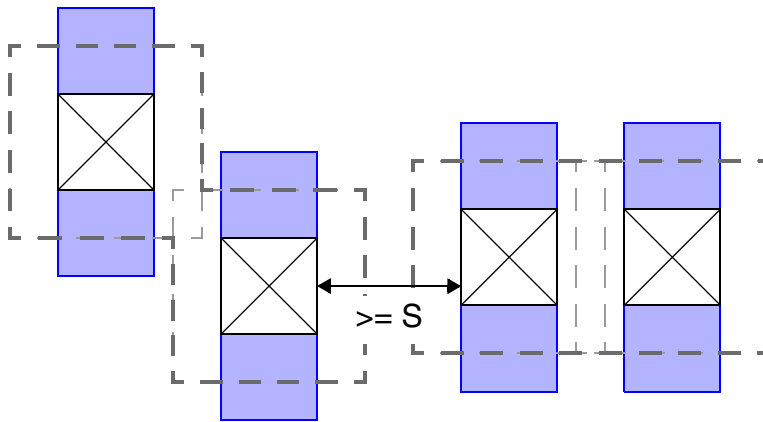
Figure 3-18 Other Types of Self-Aligned Via Clusters

The self-aligned via cluster rules can also limit the number of vias allowed in one cluster. For example, if the limit is set to 4, the 5-via cluster shown in [Figure 3-19](#) would be forbidden.

Figure 3-19 Self-Aligned Via Cluster Limit Rule

A cluster rule can specify a minimum spacing between vias that belong to different self-aligned via clusters. For example, in [Figure 3-20](#), the two vias on the left belong to a cluster and the two vias on the right belong to a different cluster, so the spacing between the vias must be at least the distance S .

Figure 3-20 Minimum Spacing Between Self-Aligned Vias in Different Clusters



This is the syntax for the self-aligned via cluster rules:

```
ViaRule SAV1 {
    selfAlignedViaClusterUpsizeTbl          = (0.025)
    selfAlignedViaClusterRectWidthTbl       = (0.097)
    selfAlignedViaClusterRectMinLengthTbl   = (0.097)
    selfAlignedViaClusterNonRectMinLengthTbl = (0.090)
    selfAlignedViaClusterNonRectMaxLengthTbl = (0.097)
    selfAlignedViaClusterUshapeAllowedTbl   = (0)
    selfAlignedViaClusterMaxNumCutsTbl      = (4)
}

DesignRule {
    layer1      = "VIA1"
    layer2      = "VIA1"
    cut1TblSize = 1
    cut2TblSize = 1
    cut1NameTbl = (Vsm)
    cut2NameTbl = (Vsm)
    selfAlignedViaClusterCenterMinSpacingTbl = (0.144)
}
```

In this example, the `ViaRule SAV1` and `DesignRule` sections specify the following requirements:

- The upsize value U used to generate the search region for each via is 0.025.
- For a rectangular merged search region having a width (shorter dimension) R_W of exactly 0.097, the length of the merged search region R_L must be at least 0.097.
- For a diagonal cluster, the lengths of the merged search region segments (a , b , c , and d in [Figure 3-17](#)) must be at least 0.090 and no more than 0.097.
- U-shaped (notched) clusters are forbidden.

- A diagonal cluster may contain no more than four self-aligned vias.
- The minimum allowed spacing between vias belonging to different clusters is 0.144.

These are table-based rules, so you can specify multiple sets of values.

By default, the `selfAlignedViaClusterCenterMinSpacingTbl` parameter in the `DesignRule` section sets the minimum spacing between vias belonging to clusters of any size, including single-via clusters.

You can optionally relax this constraint by specifying that the rule applies only when one or both of the clusters have at least a specified number of vias in the cluster. For example,

```
DesignRule {
    layer1      = "VIA1"
    layer2      = "VIA1"
    cut1TblSize = 1
    cut2TblSize = 1
    cut1NameTbl = (Vsm)
    cut2NameTbl = (Vsm)
    selfAlignedViaClusterMinNumCutsThresholdTbl = (3)
    selfAlignedViaClusterCenterMinSpacingTbl = (0.144)
}
```

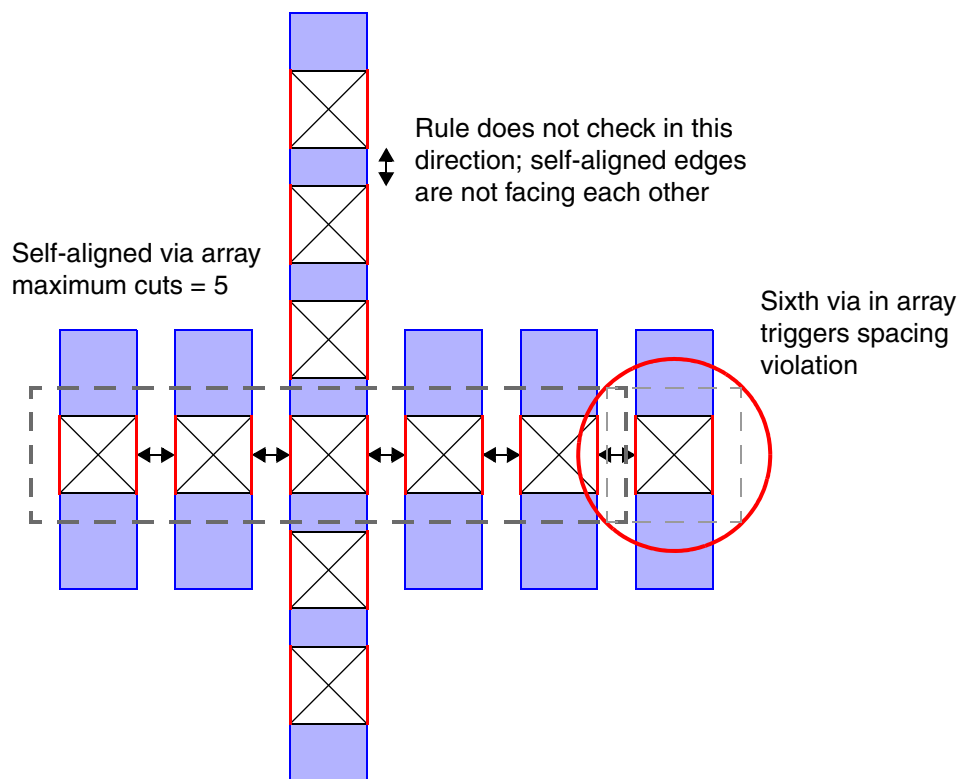
In this example, the minimum spacing requirement applies only when one or both clusters have three or more vias; it does not apply when both clusters only one or two vias each.

Self-Aligned Via Array Rule

Self-aligned vias can be placed more closely together in an array when their edges are exactly aligned. The self-aligned via array rule specifies the maximum size of such an array when the vias are closely spaced.

For example, in [Figure 3-21](#), if the rule specifies a maximum of five vias in a minimum-spacing array, the presence of a sixth via at the minimum spacing triggers an error. The error can be fixed by increasing the spacing between the sixth via and the nearby via at the end of the array. This rule checks the row of vias, but not the column because the self-aligned edges in the column are not facing each other.

Figure 3-21 Self-Aligned Via Array Rule Examples



This is the syntax for the rule:

```
ViaRule SAV2 {
  cutLayerNameTblSize           = 1
  cutLayerNameTbl               = (VIA1)
  cutNameTbl                    = (Vsm)
  selfAlignedViaUpperLayerEncTblSize = 1
  selfAlignedViaUpperLayerEncTbl   = (0.00)
  selfAlignedViaOneEdgeNotAllowed = 1
  selfAlignedViaArrayMaxSpacingThresholdTbl = (0.095)
  selfAlignedViaArrayMaxNumCutsTbl   = (5)
}
```

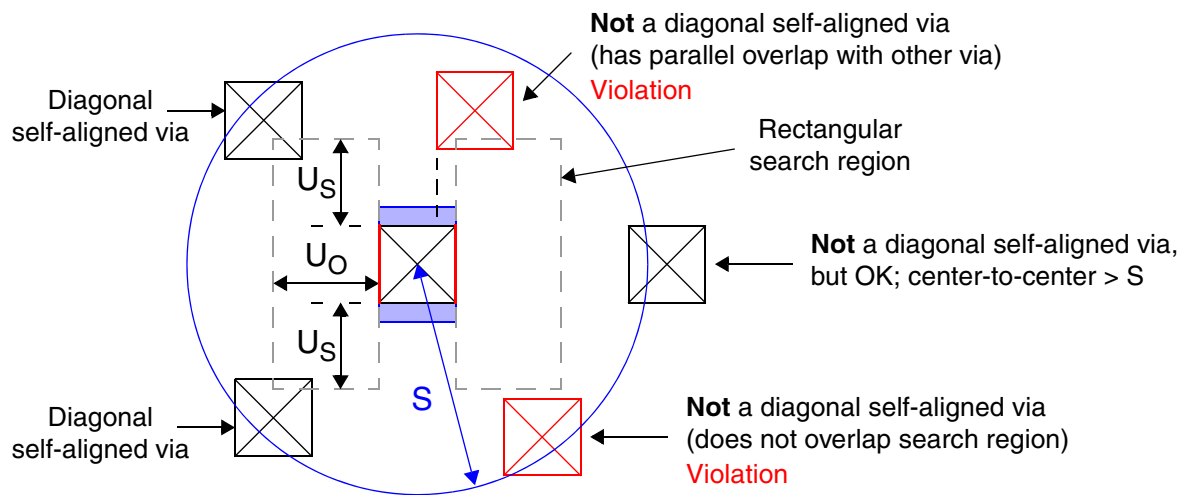
In this example, when self-aligned Vsm vias in an array have a spacing of 0.095 or less, the maximum array size is 5.

Self-Aligned Via Diagonal Spacing Rule

The self-aligned via diagonal spacing rule specifies the minimum center-to-center spacing between a self-aligned via and another self-aligned via that does not occupy a specified search region.

The rule defines two upsize lengths: U_S in the same direction as the self-aligned edge and U_O for the orthogonal direction. These two attributes establish a rectangular search region on two sides of each self-aligned via, as shown in Figure 3-22. Any nearby via that has no parallel overlap, but overlaps the search region by any amount, is considered a “diagonal” self-aligned via.

Figure 3-22 Diagonal Self-Aligned Vias



The rule specifies the minimum distance S between a self-aligned via and any nearby self-aligned via that is *not* a “diagonal” self-aligned via, measured center-to-center. A “diagonal” via is allowed to be closer than a “nondiagonal” via.

The following example shows the syntax of the rule:

```
ViaRule SAV1 {
  cutLayerNameTblSize      = 1
  cutLayerNameTbl          = (VIA1)
  cutNameTbl               = (Vsm)
  selfAlignedViaDiagSavEdgeUpsizeTbl = (0.055)
  selfAlignedViaDiagOrthoEdgeUpsizeTbl = (0.073)
}

DesignRule {
  layer1 = "VIA1"
  layer2 = "VIA1"
  selfAlignedViaDiagCenterMinSpacingTbl = (0.154)
}
```

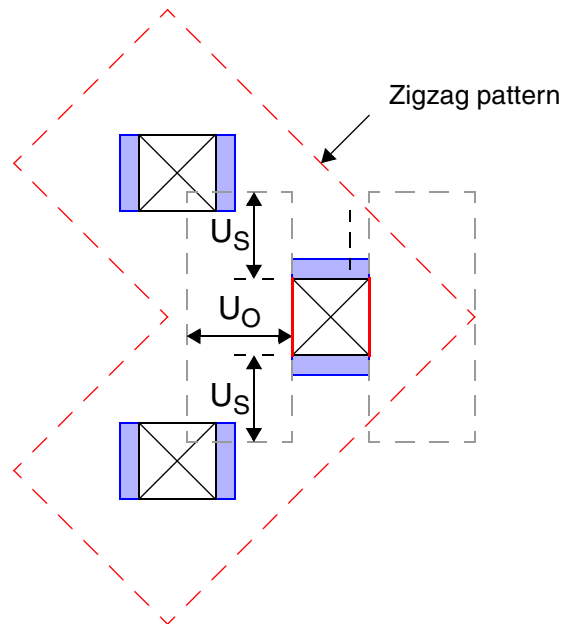
In this example, the search region extends in the direction of the self-aligned via edge by 0.055 and in orthogonal direction by 0.073. This forms a rectangular search region measuring 0.073 by $(0.055 + 0.055 + \text{via_height})$ on the two self-aligned sides of the via. Any via that is outside of the search areas or has parallel overlap with the via is a nondiagonal via and is affected by this rule; the center-to-center distance of such a via must be at least 0.154.

Self-Aligned Via Zigzag Spacing Rule

The self-aligned via zigzag spacing rule specifies the minimum center-to-center spacing between a self-aligned via that is part of a zigzag pattern and another self-aligned via that is not part of the pattern.

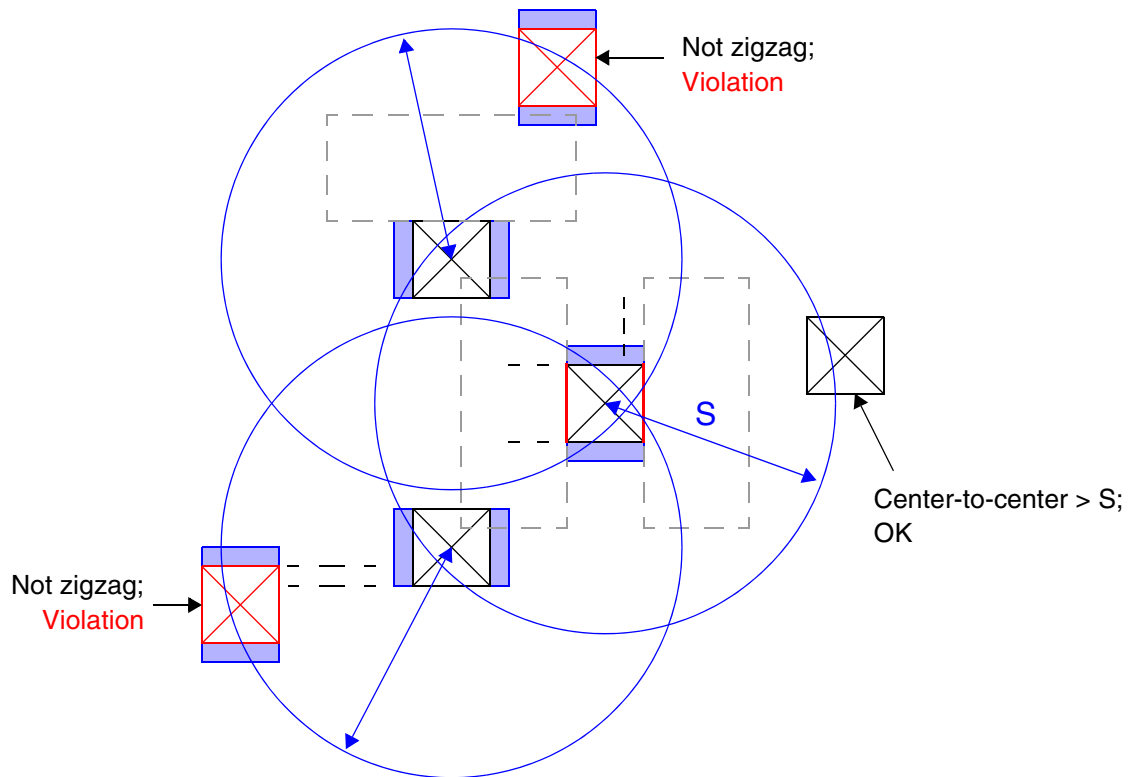
The rule defines two upsize lengths: U_S in the same direction as the self-aligned edge and U_O for the orthogonal direction. These two attributes establish a rectangular search region on two sides of each self-aligned via, as shown in [Figure 3-23](#). A zigzag via pattern exists when exactly two other self-aligned vias touch or overlap the rectangular search region on one side of the via, as shown in the figure, without any parallel overlap with the original via.

Figure 3-23 Zigzag Pattern of Self-Aligned Vias



The rule specifies the minimum distance S between a self-aligned via belonging to the zigzag pattern and any nearby self-aligned via not belonging to the pattern, measured center-to-center. Vias belonging to the zigzag pattern are allowed to be spaced more closely to each other than to vias not belonging to the pattern, as shown in [Figure 3-24](#).

Figure 3-24 Minimum Spacing Requirement in Zigzag Pattern of Self-Aligned Vias



The following example shows the syntax of the rule:

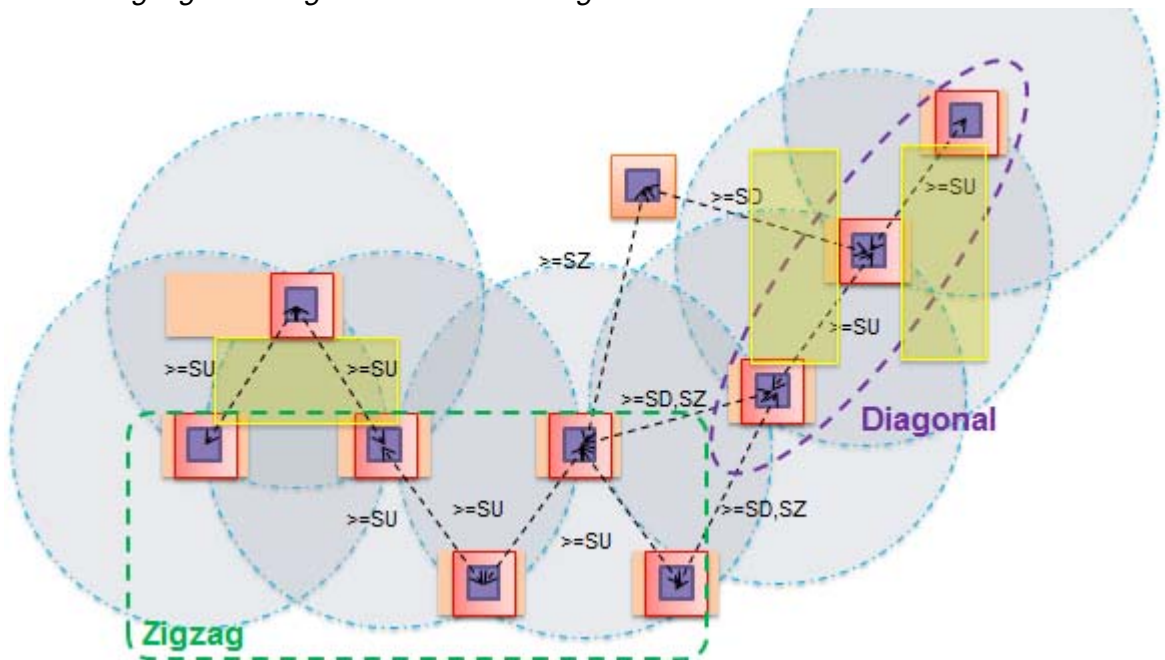
```
ViaRule SAV1 {
    cutLayerNameTblSize           = 1
    cutLayerNameTbl               = (VIA1)
    cutNameTbl                   = (Vsm)
    selfAlignedViaZigzagSavEdgeUpsizeTbl = (0.055)
    selfAlignedViaZigzagOrthoEdgeUpsizeTbl = (0.073)
}

DesignRule {
    layer1                       = "VIA1"
    layer2                       = "VIA1"
    selfAlignedViaZigzagCenterMinSpacingTbl = (0.154)
}
```

In this example, the search region extends in the direction of the self-aligned via edge by 0.055 and in the orthogonal direction by 0.073. This forms a rectangular search region measuring 0.073 by $(0.055 + 0.055 + \text{via_height})$ on the two self-aligned sides of the via. Any via that is outside of the search areas of the zigzag set or has parallel overlap with a via in the zigzag set is affected by this rule; the center-to-center distance of such a via from a via in the zigzag set must be at least 0.154.

Figure 3-25 shows an example of a zigzag set, a diagonal set, and an individual self-aligned via, along with their minimum center-to-center spacing requirements. Self-aligned vias within a set have their own, smaller (unaligned) minimum spacing requirement, whereas a larger minimum spacing requirement applies to vias belonging to different sets or between an individual via and the vias belonging to a set. In this diagram, SU, SD, and SZ are the unaligned via, diagonal, and zigzag minimum spacing requirements, respectively.

Figure 3-25 Zigzag and Diagonal Sets of Self-Aligned Via Patterns



Minimum Edge, Width, and Area Rules

Several advanced rules specify minimum metal edge lengths, widths, and areas:

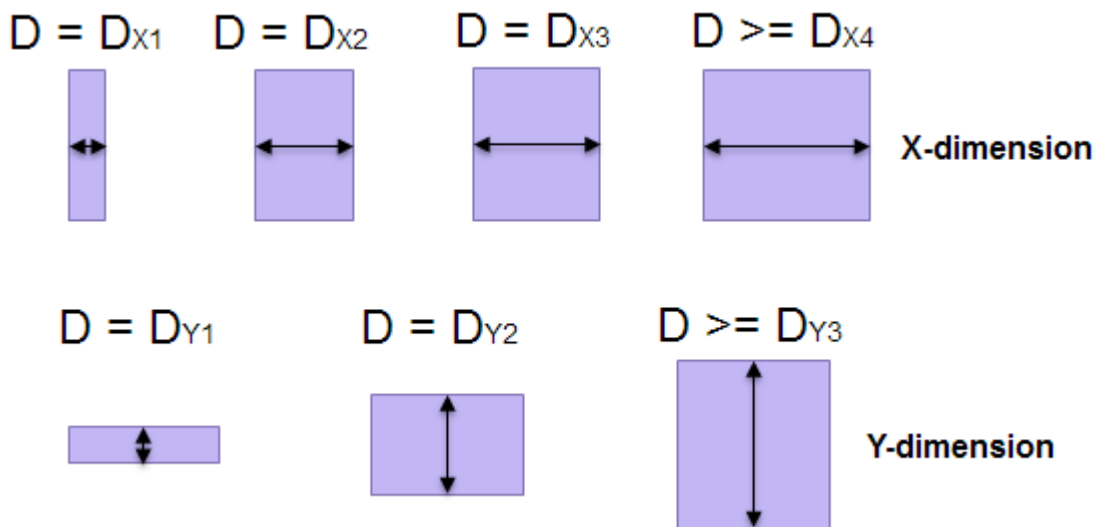
- [Discrete Metal Dimension Rule](#)
- [Two-Convex-Corner Minimum Edge Length Rule](#)
- [Line-End Concave Corner Length Rule](#)
- [Convex-Concave Minimum Edge Length Rule](#)
- [Diagonal Concave Corner Minimum Width Rule](#)
- [Minimum Metal Jog Length and Adjacent Edge Length Rules](#)
- [Fat Metal Branch Minimum Width and Length Rule](#)

- [Width-Based Minimum Area Rule](#)
- [Polygon-Edge Multistage Minimum Area Rule Enhancement](#)

Discrete Metal Dimension Rule

The discrete metal dimension rule species a list of discrete dimensions allowed for a metal layer. Only the exact dimensions are allowed for any shape in the metal layer up to the last value in the list, after which the rule has no restrictions on allowed dimensions. A separate list is specified for each dimension, X and Y, as shown in [Figure 3-26](#).

Figure 3-26 Discrete Metal Dimension Rule



This is the syntax for the rule:

```
Layer "MetalX" {
  xLegalDimTblSize = 4
  xLegalDimTbl     = (DX1, DX2, DX3, DX4)
  yLegalDimTblSize = 3
  yLegalDimTbl     = (DY1, DY2, DY3)
}
```

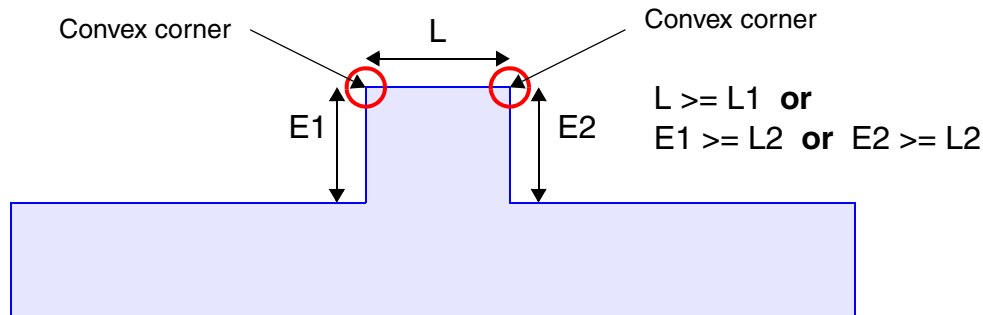
For example,

```
Layer "M2" {
  xLegalDimTblSize = 4
  xLegalDimTbl     = (0.10, 0.140, 0.18, 0.22)
  yLegalDimTblSize = 3
  yLegalDimTbl     = (0.06, 0.09, 0.11)
}
```

Two-Convex-Corner Minimum Edge Length Rule

The two-convex-corner minimum edge length rule specifies the minimum lengths of edges adjacent to two convex corners of a metal polygon. In [Figure 3-27](#), the metal polygon has two consecutive convex corners. The rule specifies that either the edge between the two corners has a length of at least L_1 , or at least one adjacent edge has a length of at least L_2 .

Figure 3-27 Line-End Edge Length Rule



This is a table-based rule that allows one set or multiple sets of L_1/L_2 values. This is the command syntax for a single-set rule:

```
Layer "MetalX" {
    convexConvexMinEdgeLengthTbl    = L1
    convexAdjacentMinEdgeLengthTbl  = L2
}
```

This is the syntax for a double-set rule:

```
Layer "MetalX" {
    convexConvexMinEdgeLengthTblSize = 2
    convexConvexMinEdgeLengthTbl     = (L11, L12)
    convexAdjacentMinEdgeLengthTbl   = (L21, L22)
}
```

For example,

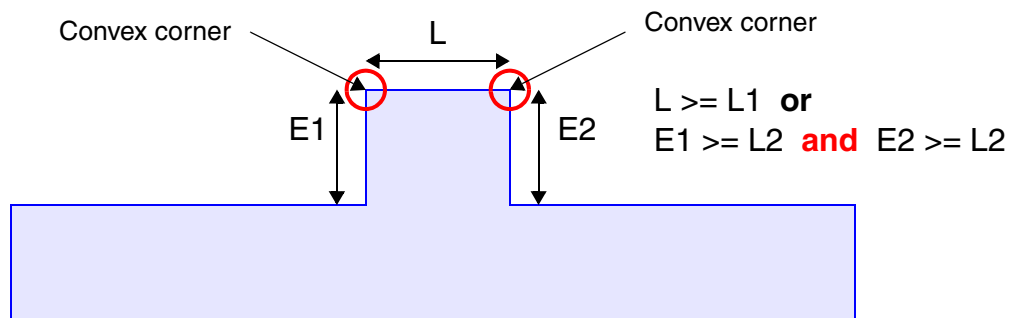
```
Layer "MetalX" {
    convexConvexMinEdgeLengthTblSize = 2
    convexConvexMinEdgeLengthTbl     = (0.120, 0.194)
    convexAdjacentMinEdgeLengthTbl   = (0.072, 0.048)
}
```

To require both adjacent edges (rather than just one) to have a length of at least L2, specify the rule with the following additional attribute:

```
Layer "MetalX" {
  convexConvexMinEdgeLengthTblSize = 2
  convexConvexMinEdgeLengthTbl     = (Q1, Q2)
  convexAdjacentMinEdgeLengthTbl   = (E1, E2)
  convexAdjacentCheckBothEdges    = 1 # default = 0
}
```

In that case, the rule is called the line-end edge length rule. See [Figure 3-28](#).

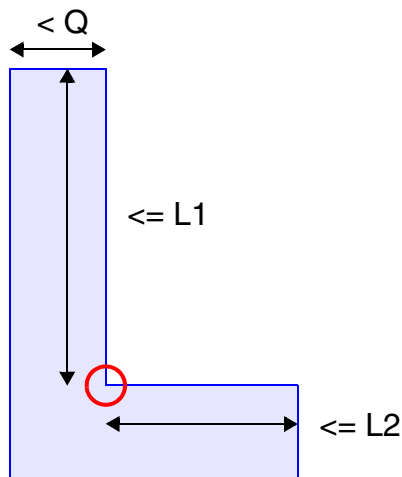
Figure 3-28 Line-End Edge Length Rule



Line-End Concave Corner Length Rule

The line-end concave corner length rule specifies a minimum length $L1$ between a line-end having a width of less than Q and a concave corner whose other edge has a length of at least $L2$, as shown in [Figure 3-29](#).

Figure 3-29 Line-End Concave Corner Length Rule



This is the syntax for the rule:

```

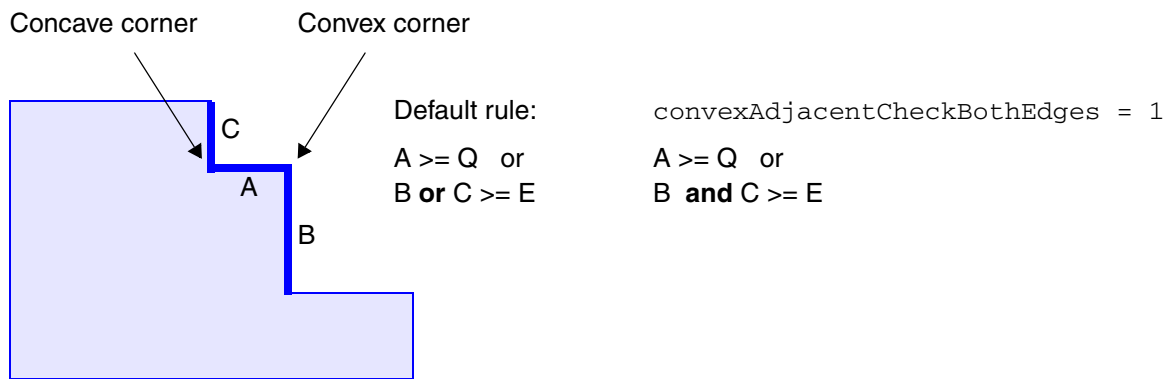
Layer "MetalX" {
  doubleConvexEdgeLengthTblSize           = 1
  doubleConvexMinEdgeLengthTbl             = (Q)
  doubleConvexAdjacentMinEdgeLengthTbl     = (L1)
  doubleConvexConcaveAdjacentMaxEdgeLengthTbl = (L2)
}

```

Convex-Concave Minimum Edge Length Rule

The advanced convex-concave edge via enclosure rule is a table-based version of the simpler rule described in [“Convex-Concave Minimum Edge Length Rule” on page 2-18](#). The rule specifies that a metal edge connecting a concave corner and a convex corner must have a length of at least Q, or if less than Q, an adjacent edge (or both adjacent edges) must have a length of at least E. See [Figure 3-30](#).

Figure 3-30 Concave-Convex Minimum Length Rule



For a table size of 2, this is the rule syntax:

```

Layer "MetalX" {
  convexConcaveMinEdgeLengthTblSize       = 2
  convexConcaveMinEdgeLengthTbl           = (Q1, Q2)
  convexConcaveAdjacentMinEdgeLengthTbl   = (E1, E2)
  convexAdjacentCheckBothEdges            = 1 # default = 0
}

```

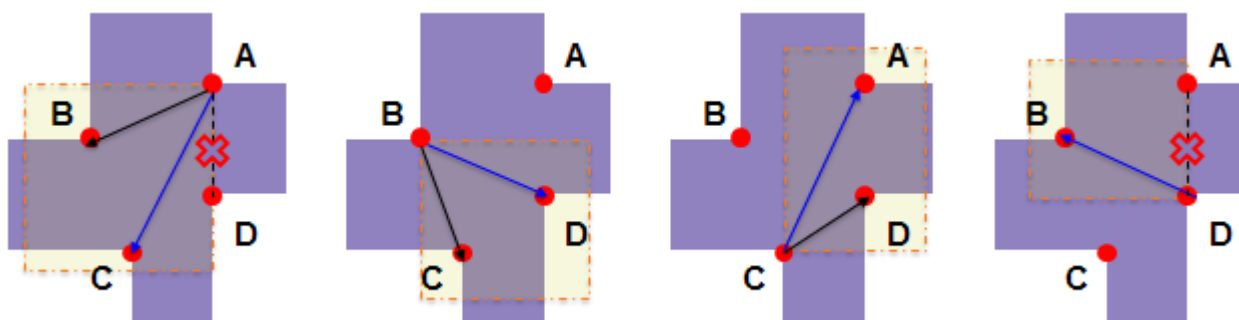
If the `convexAdjacentCheckBothEdges` attribute is set to 0 or is not present, only one adjacent edge needs to have a length of at least E. If this attribute is set to 1, then both adjacent edges must have a length of at least E.

Diagonal Concave Corner Minimum Width Rule

The diagonal concave corner minimum width rule specifies the minimum width of metal between two diagonally opposite concave corners of a metal polygon.

In [Figure 3-31](#), points A, B, C, and D are concave corners of a metal polygon. The “opposite projection” of each point toward the other points is indicated by the yellow shaded area. The rule specifies the minimum metal width measured from each point to any other point inside the opposite projection area. The distance to a point directly on the edge of the opposite projections area is not checked, such as between points A and D.

Figure 3-31 Diagonal Concave Corner Width Rule



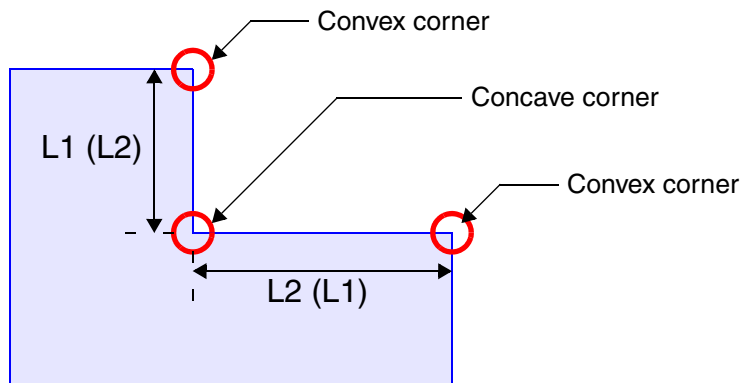
This is the syntax of the rule:

```
Layer "MetalX" {
    minWidthForConcaveCorner = W
}
```

Minimum Metal Jog Length and Adjacent Edge Length Rules

The metal jog length rule specifies the minimum required length for two consecutive edges of a metal jog, L1 and L2, when the two edges meet at a concave corner and the other ends of these edges terminate at convex corners, as shown in [Figure 3-32](#).

Figure 3-32 Metal Jog Length Rule



The labels L1 and L2 can be swapped without affecting the rule.

This is the syntax of the rule:

```
Layer "MetalX" {
  concaveConvexMinEdgeLength = 0.076 ; L1
  concaveAdjacentMinEdgeLength = 0.039 ; L2
}
```

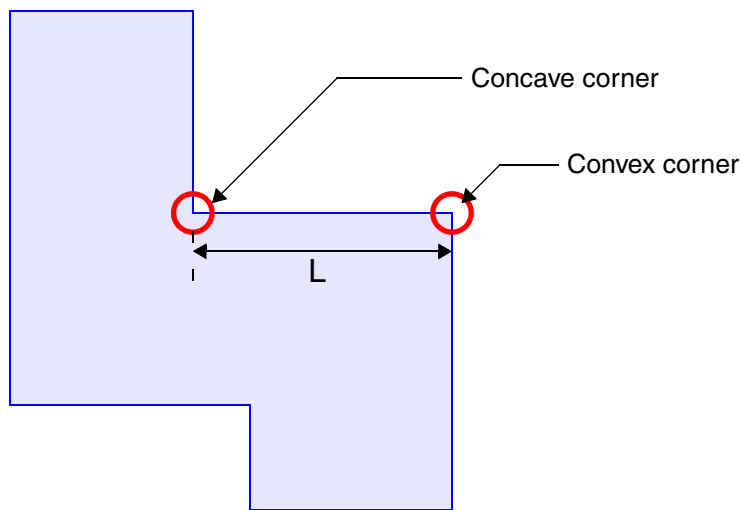
In this example, one of the two edges must have a length of at least 0.076 and the other must have a length of at least 0.039.

If the `concaveAdjacentMinEdgeLength` attribute is set to `-1`, the length of the adjacent edge connected to the concave corner is not checked. For example,

```
Layer "MetalX" {
  concaveConvexMinEdgeLength = 0.076 ; L
  concaveAdjacentMinEdgeLength = -1
}
```

In this case, the rule only enforces a minimum length for a metal edge connecting a convex corner to a concave corner, as shown in [Figure 3-33](#).

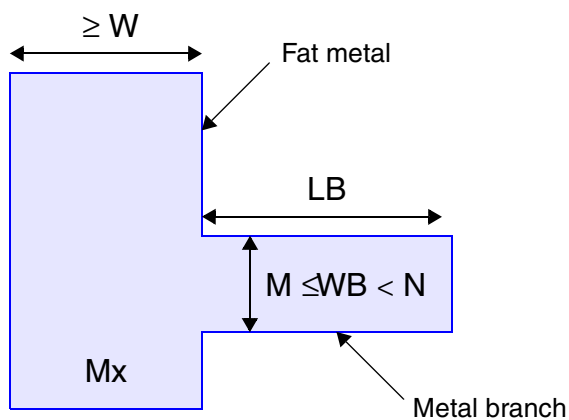
Figure 3-33 Adjacent Minimum Edge Length Rule



Fat Metal Branch Minimum Width and Length Rule

The fat metal branch minimum width and length rule specifies the minimum width M for a metal branch connected to a fat metal shape having a width of at least W . In addition, if the width of the metal branch is at least M but less than N , the length of the metal branch must be at least LB . See [Figure 3-34](#).

Figure 3-34 Fat Metal Branch Minimum Width and Length Rule



This is the syntax of the rule:

```
Layer "MetalX" {
  fatMetalBranchTblSize      = 3
  fatMetalBranchThresholdTbl = (W1, W2, W3)
  fatMetalBranchMinLengthTbl = (LB1, LB2, LB3)
  fatMetalBranchMinWidthTbl  = (M1, M2, M3)
  fatMetalBranchMaxWidthTbl  = (N1, N2, N3)
  fatMetalBranchExcludedForFatViaExt = 1 ; optional
}
```

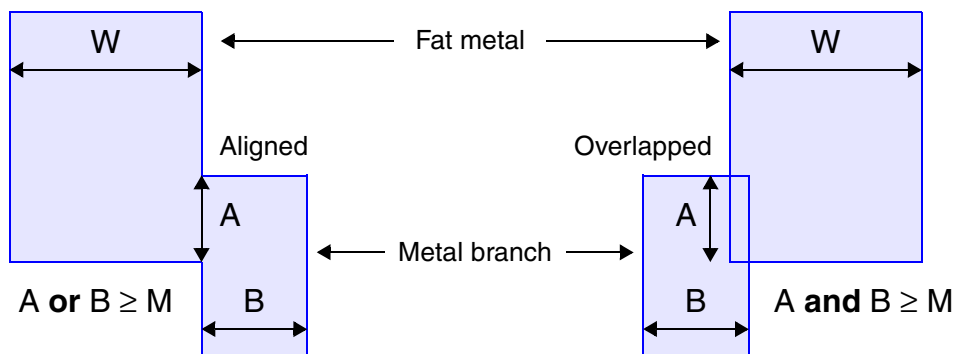
For example,

```
Layer "MetalX" {
  fatMetalBranchTblSize      = 3
  fatMetalBranchThresholdTbl = (0.32, 0.72, 1.22)
  fatMetalBranchMinLengthTbl = (0.18, 0.29, 0.60)
  fatMetalBranchMinWidthTbl  = (0.10, 0.10, 0.12)
  fatMetalBranchMaxWidthTbl  = (0.45, 0.45, 0.50)
}
```

If the `fatMetalBranchExcludedForFatViaExt` attribute is set to 1 and the branch contains vias that meet the rule described in [“Area-Based Fat Metal Contact Rule” on page 2-101](#), the metal branch length requirement is waived (but the width requirement is still enforced). If this attribute is omitted or set to 0, the width and length requirements are always enforced.

When the metal branch is parallel with the fat metal shape and connected to the fat metal corner as shown in [Figure 3-35](#), the rule applies only if the connection width A or branch width B is at least M. If the branch overlaps the fat metal, both A and B must be at least M for the rule to apply.

Figure 3-35 Fat Metal Parallel Branch Connected at Corner



The “fat metal branch area” is the region of the branch that meets the fat metal branch width rule, excluding any part of the branch that overlaps the fat metal shape. It is considered a violation if this branch area cannot completely contain a filling rectangle measuring LB by F, where F is an additional (optional) attribute:

```
fatMetalBranchFillMinWidthTbl = (F1, F2, ..., FN)
```

Figure 3-36 shows an example of a fat metal branch area and an LB-by-F rectangle.

Figure 3-36 Fat Metal Parallel Branch Area Checking

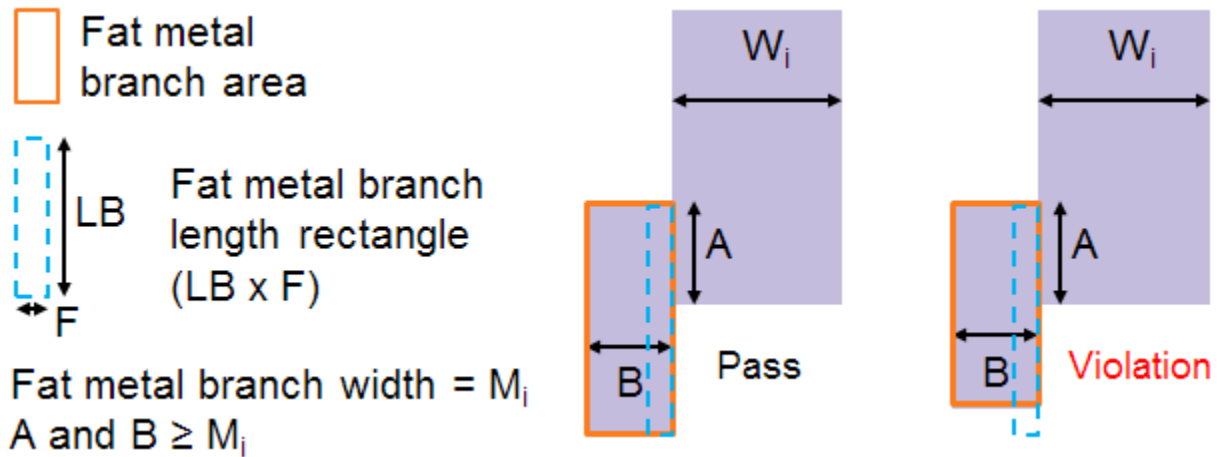
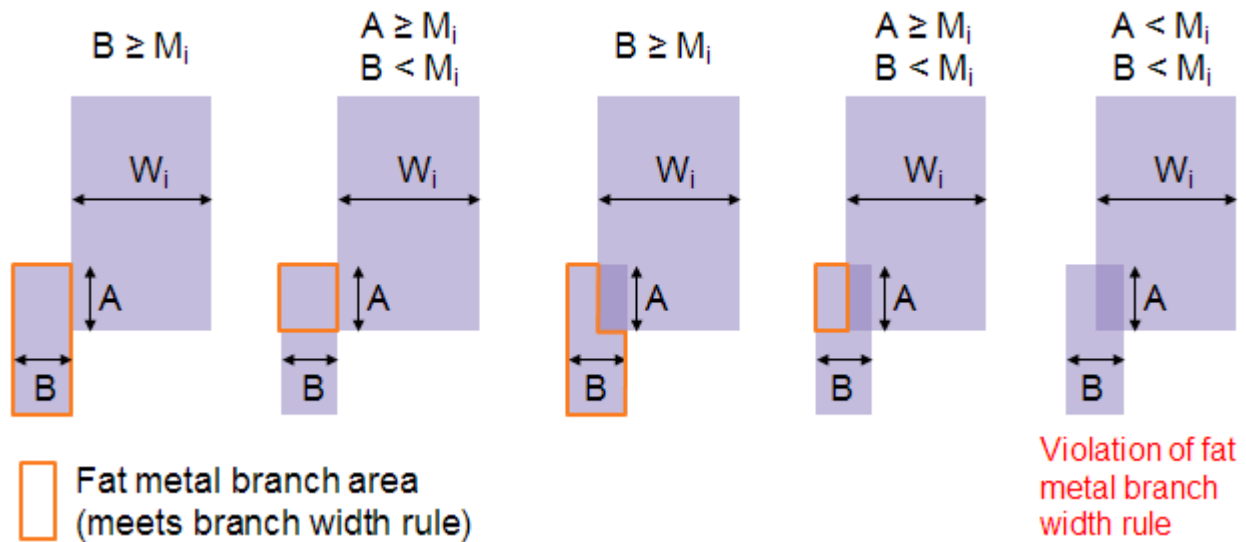


Figure 3-37 shows some examples of fat metal branches of different widths and overlaps.

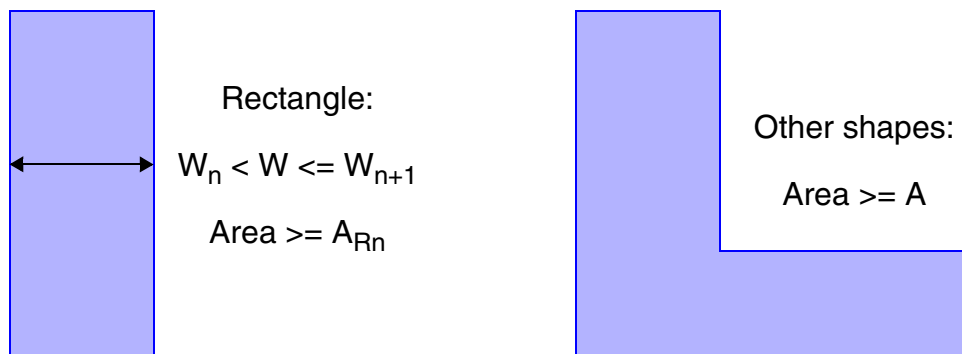
Figure 3-37 Fat Metal Parallel Branch Area Checking Examples



Width-Based Minimum Area Rule

The width-based minimum area rule specifies the minimum allowable area for geometries in a particular layer. For rectangular areas, the rule specifies a table of different minimum area values corresponding to the width (smaller dimension) of the rectangle. The rule specifies a separate minimum area value that applies to all nonrectangular areas. See [Figure 3-38](#).

Figure 3-38 Width-Based Minimum Area Rule



For example,

```

Layer "Metal2" {
    nonRectMinArea           = 0.028
    rectMinAreaTblSize       = 3
    rectMinAreaWidthThreshTbl = (0.00, 0.04, 0.06)
    rectMinAreaTbl           = (0.018, 0.022, 0.026)
}
  
```

In this example, the minimum allowable area for any nonrectangular geometry is 0.028. For rectangular metal, the minimum area is 0.018 square units for widths up to 0.04, 0.022 square units for widths greater than 0.04 but no more than 0.06, and 0.026 square units for widths more than 0.06.

Polygon-Edge Multistage Minimum Area Rule Enhancement

The polygon-edge-based multistage minimum area rule specifies a set of minimum areas that depend on the lengths of the segments of the polygon. A polygon made exclusively with shorter segments requires a larger minimum area. The segment length thresholds and the corresponding minimum areas are specified in a table. The basic rule is described in [“Polygon-Edge-Based Multistage Minimum Area Rule” on page 2-7](#).

For advanced geometries, the rule includes additional attributes that let you change the stage-three portion of the area check. The minimum area is A2 for a polygon with any edge

length greater than or equal to L' and with all edge lengths less than L_2 , where L' is a length threshold between L_1 and L_2 , inclusive. The rule is waived if one metal edge of the polygon has a length of at least E' and its adjacent edge has a length less than a minimum width W' .

The additional attributes are specified as follows:

```

Layer "MetalX" {
    minArea                      = A0
    specialMinAreaTblSize        = 2
    minAreaEdgeThresholdTbl      = (L1, L2)
    minAreaMinEdgeThresholdTbl  = (0, L')
    minAreaFillMinLengthTbl      = (L1, L2)
    minAreaFillMinWidthTbl       = (W1, W2)
    specialMinAreaTbl            = (A1, A2)
    minAreaMinEdgeLengthExcludedTbl = (0, E')
    minAreaAdjacentEdgeLengthExcludedTbl = (0, W')
}

```

For example,

```

Layer "MetalX" {
    minArea                      = 0.009
    specialMinAreaTblSize        = 2
    minAreaEdgeThresholdTbl      = (0.098, 0.228)
    minAreaMinEdgeThresholdTbl  = (0.000, 0.130)
    minAreaFillMinLengthTbl      = (0.094, 0.142)
    minAreaFillMinWidthTbl       = (0.094, 0.054)
    specialMinAreaTbl            = (0.023, 0.090)
    minAreaMinEdgeLengthExcludedTbl = (0.000, 0.054)
    minAreaAdjacentEdgeLengthExcludedTbl = (0.000, 0.048)
}

```

Figure 3-39 and Figure 3-40 illustrate the usage of the supplemental attributes.

Figure 3-39 Polygon-Edge Multistage Minimum Area Rule Example Set 1

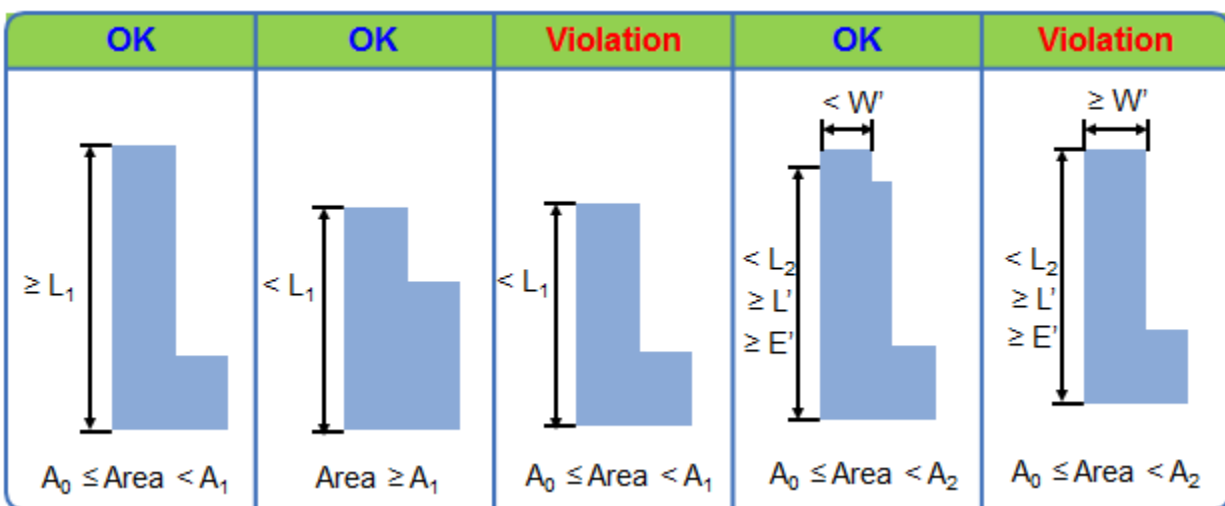
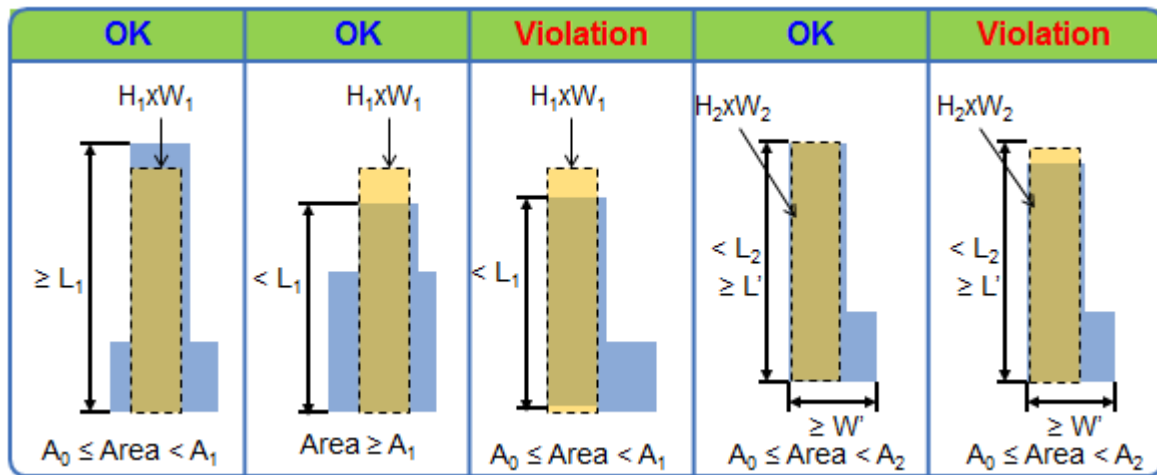


Figure 3-40 Polygon-Edge Multistage Minimum Area Rule Example Set 2



Metal Spacing Rules

Several advanced rules specify the minimum spacing between metal geometries:

- [Cut to Metal Orthogonal Spacing Rule](#)
- [Line-End to Concave Corner Spacing Rule](#)
- [Concave Corner Keepout Rule](#)
- [Fat Metal Corner Keepout Rule](#)
- [Preferred and Nonpreferred Corner-to-Corner Spacing Rule](#)
- [Preferred and Nonpreferred End-of-Line Spacing Rule](#)
- [Preferred and Nonpreferred Fat Metal Spacing Rule](#)
- [Metal Span Orthogonal Spacing Rule](#)
- [Orthogonal Minimum Width Rule](#)
- [Forbidden Pitch Rule](#)
- [Wide Metal Jog Rule](#)
- [Preferred-Direction Forbidden Spacing Rule](#)
- [Fat Metal Forbidden Spacing Rule](#)
- [Three-Wire Forbidden Space Rule](#)

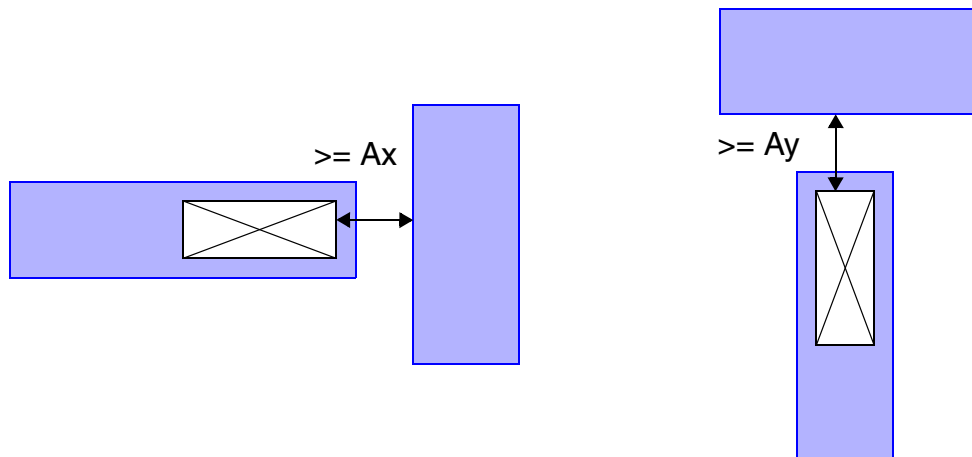
- [Via-Induced Metal Bridge Rule](#)
- [Enclosure-Dependent Metal Spacing Rule](#)
- [U-Shape Leg Spacing Rule](#)
- [Line-End to U-Shape Spacing Rule](#)
- [Two-Nighbor End-of-Line Spacing Rule Enhancements](#)

Cut to Metal Orthogonal Spacing Rule

The cut to metal orthogonal spacing rule specifies the minimum spacing between the short edges of a rectangular cut to nearby metal belonging to a different net. This rule extends the general cut spacing rule (see [“General Cut Spacing Rule” on page 2-33](#)) to allow a cut layer and a metal layer to be specified as the two layers affected by the rule.

In [Figure 3-41](#), the spacing between the short side of the horizontal bar cut and the outside metal must be at least A_x . Likewise, the short side of the vertical bar cut must be at least the distance A_y from the external metal.

Figure 3-41 Cut to Metal Orthogonal Spacing Rule



The following example shows the syntax of the rule:

```
DesignRule {
  layer1           = "VIA1"
  layer2           = "MET2"
  cutTblSize       = 2
  cutNameTbl       = (VBAR_H, VBAR_V)
  diffNetXMinSpacingTbl = (0.065, -1) # specifies Ax, no Y requirement
  diffNetYMinSpacingTbl = (-1, 0.068) # specifies Ay, no X requirement
}
```

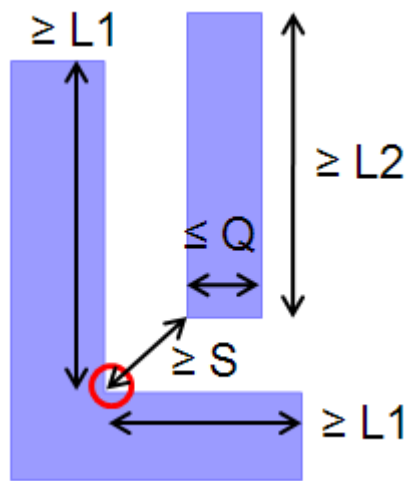
In this example, only the spacing between the short side of the cut and the external metal is specified, between VBAR_H and MET2 in the x-direction, and between VBAR_V and MET2 in the y-direction. The value -1 (or any other negative value) in the spacing table means that no extra spacing requirement applies to that combination of objects in the two layers, so there is no spacing requirement specified for the long side of the cut.

The cut metal to orthogonal spacing rule supports only the `diffNetXMinSpacingTbl` and `diffNetYMinSpacingTbl` attributes of the general cut spacing rule. It does not support other general cut spacing rules such as the same-net and same-segment rules.

Line-End to Concave Corner Spacing Rule

The line-end to concave corner spacing rule specifies the minimum distance between a line-end and the concave corner of a neighboring wire, measured diagonally. The rule applies only when the line-end wire has a width of no more than Q , and optionally, a length of at least $L2$, and optionally only when the length of the parallel edge is at least $L1$, as shown in [Figure 3-42](#).

Figure 3-42 Line-End to Concave Corner Spacing Rule



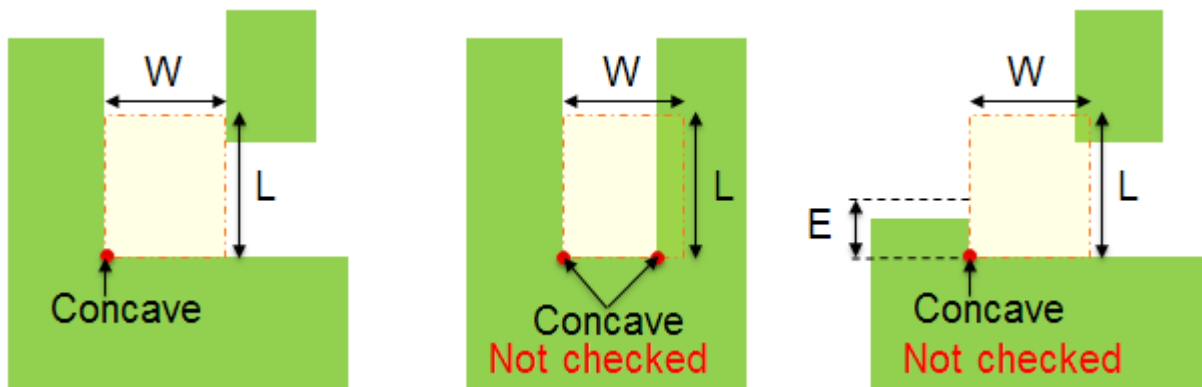
The following example shows the syntax of the rule:

```
Layer "MetalX" {
  endOfLineToConcaveCornerMaxWidthThreshold    = 0.12 # Q
  endOfLineToConcaveCornerEdgeLengthThreshold = 0.54 # L1 (optional)
  endOfLineToConcaveCornerMinLength            = 0.05 # L2 (optional)
  endOfLineToConcaveCornerMinSpacing          = 0.07 # S
}
```


Concave Corner Keepout Rule

The concave corner keepout rule specifies a rectangular region measuring L by W , extending from a concave metal corner, where no other metal shapes are allowed. The rule does not apply to an edge between two concave corners with a length less than L or W , or if either metal edge at the concave corner has a length of less than E , as shown in [Figure 3-43](#).

Figure 3-43 Concave Corner Keepout Rule



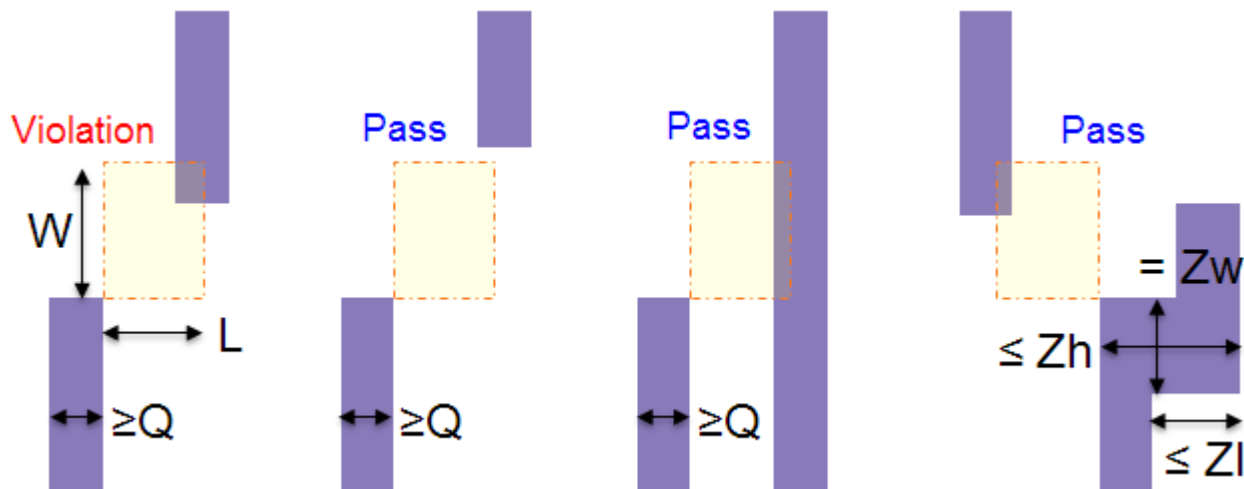
This is the syntax of the rule:

```
Layer "MetalX" {
  concaveCornerKeepoutWidth    = W
  concaveCornerKeepoutLength   = L
  concaveCornerKeepoutMinEdgeLengthThreshold = E # optional
}
```

Fat Metal Corner Keepout Rule

The fat metal corner keepout rule requires two facing corners of metal shapes, each shape having a width of at least Q , to be far enough apart to avoid a rectangular region measuring L by W , as shown in [Figure 3-44](#). The rule does not apply for a corner belonging to a Z-shape with width equal to Z_w , height no more than Z_h , and longer concave-convex edge of no more than Z_l .

Figure 3-44 Fat Metal Corner Keepout Rule



This is the syntax of the rule:

```

Layer "MetalX" {
  fatMetalCornerTblSize      = 3
  fatMetalCornerWidthThresholdTbl = (Q1,Q2,Q3)
  fatMetalCornerKeepoutWidthTbl  = (W1,W2,W3)
  fatMetalCornerKeepoutLengthTbl = (L1,L2,L3)
  # EOL definition
  fatMetalCornerEndOfLineWidthThresholdTbl = (Qe1,Qe2,Qe3)
  fatMetalCornerEndOfLineMinLengthTbl      = (T1,T2,T3) # optional
  # Z-shape definition
  fatMetalCornerZWireWidthTbl              = (Zw1,Zw2,Zw3)
  fatMetalCornerZWireMaxHeightTbl          = (Zh1,Zh2,Zh3)
  fatMetalCornerZWireMaxEdgeLengthTbl      = (Zl1,Zl2,Zl3)
  # L-shape considered as Z-shape
  fatMetalCornerZWireIncludeLWire          = 1 # optional, default=0
}

```

The optional attributes Qe and T define a line-end for which the rule is waived, where the line-end width is no more than Qe and the tip length is at least T. The rule is also waived for an L-shape (as well as a Z-shape) if the `fatMetalCornerZWireIncludeLWire` attribute is set to 1 and the L-shape meets the Zw, Zh, and Zl criteria.

For example,

```

Layer "MetalX" {
  fatMetalCornerTblSize      = 3
  fatMetalCornerWidthThresholdTbl = (0.00, 0.24, 0.60)
  fatMetalCornerKeepoutWidthTbl  = (0.10, 0.18, 0.25)
  fatMetalCornerKeepoutLengthTbl = (0.10, 0.18, 0.25)
  # EOL definition
  fatMetalCornerEndOfLineWidthThresholdTbl = (0.07, -1, -1)
  # Z-shape definition

```

```

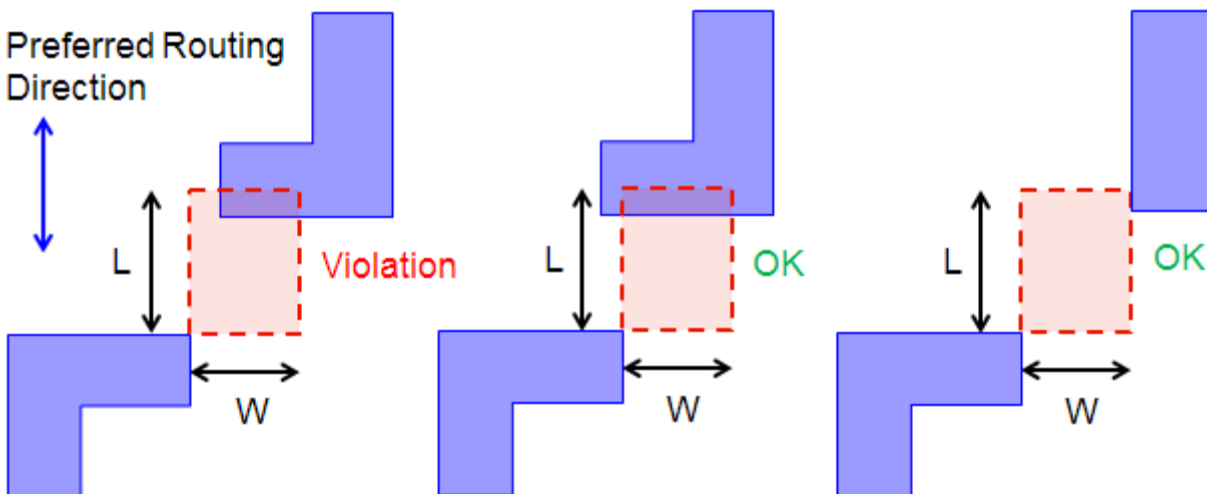
fatMetalCornerZWireWidthTbl           = (0.10, -1, -1)
fatMetalCornerZWireMaxHeightTbl       = (0.14, -1, -1)
fatMetalCornerZWireMaxEdgeLengthTbl   = (0.12, -1, -1)
}

```

Preferred and Nonpreferred Corner-to-Corner Spacing Rule

The preferred and nonpreferred corner-to-corner spacing rule specifies a rectangular keepout region where no other metal corner is allowed. The keepout region extends from a given metal corner by a length L in the preferred routing direction and W in the nonpreferred routing direction, as shown in [Figure 3-45](#).

Figure 3-45 Preferred and Nonpreferred Corner-to-Corner Spacing Rule



This is the syntax of the rule:

```

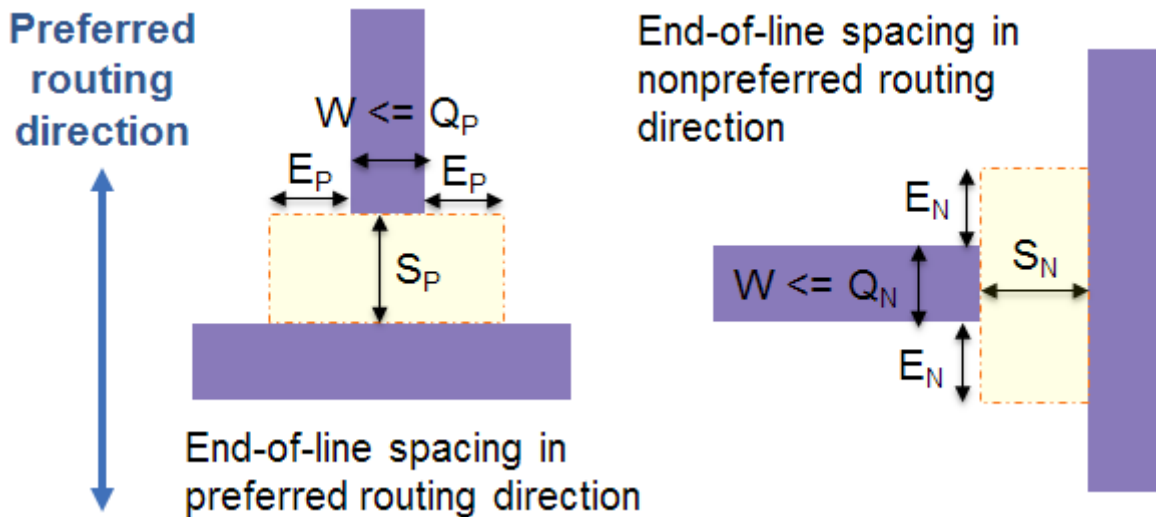
Layer "MetalX" {
  cornerKeepoutNonPrefWidth  = W
  cornerKeepoutPrefLength    = L
}

```

Preferred and Nonpreferred End-of-Line Spacing Rule

The preferred and nonpreferred end-of-line spacing rule specifies the minimum spacing between the end of a metal line and a nearby metal shape, with different width, extension, and spacing attributes in the preferred and nonpreferred routing directions. In [Figure 3-46](#), the subscript “P” denotes an attribute for a wire running in the preferred direction, and similarly the subscript “N” for the nonpreferred direction.

Figure 3-46 Preferred and Nonpreferred End-of-Line Spacing Rule



In each direction, for a wire of width no more than Q , there is an exclusion area with a minimum spacing S from the wire-end and extension E beyond the sides of the wire. An optional attribute T defines a minimum length of the line-end for which the rule applies.

This is the syntax of the rule:

```

Layer "MetalX" {
  endOfLine1NeighborPrefTblSize                = 2
  endOfLine1NeighborPrefThresholdTbl            = (QP1, QP2)
  endOfLine1NeighborPrefMinLengthTbl            = (TP1, TP2) # optional
  endOfLine1NeighborPrefCornerKeepoutWidthTbl   = (EP1, EP2)
  endOfLine1NeighborPrefMinSpacingTbl           = (SP1, SP2)
  endOfLine1NeighborNonPrefTblSize              = 1
  endOfLine1NeighborNonPrefThresholdTbl         = (QN)
  endOfLine1NeighborNonPrefMinLengthTbl         = (TN) # optional
  endOfLine1NeighborNonPrefCornerKeepoutWidthTbl = (EN)
  endOfLine1NeighborNonPrefMinSpacingTbl       = (SN)
}

```

For example,

```

Layer "MetalX" {
  endOfLine1NeighborPrefTblSize                = 2
  endOfLine1NeighborPrefThresholdTbl            = (0.05, 0.07)
  endOfLine1NeighborPrefCornerKeepoutWidthTbl   = (0.05, 0.07)
  endOfLine1NeighborPrefMinSpacingTbl           = (0.11, 0.09)
  endOfLine1NeighborNonPrefTblSize              = 1
  endOfLine1NeighborNonPrefThresholdTbl         = (0.09)
  endOfLine1NeighborNonPrefCornerKeepoutWidthTbl = (0.10)
  endOfLine1NeighborNonPrefMinSpacingTbl       = (0.12)
}

```

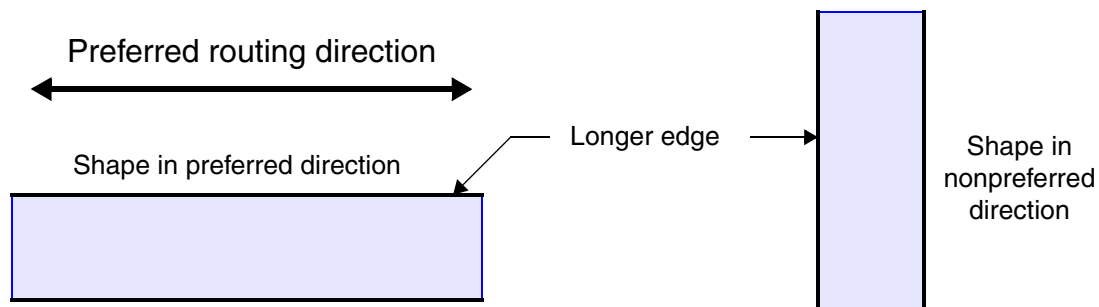
Preferred and Nonpreferred Fat Metal Spacing Rule

The fat metal spacing rule defines the minimum spacing between two parallel metal segments, based on the width of the metal segments and the length of the parallel overlap between them. The basic rule is described in [“Fat Metal Spacing Table Rule” on page 2-43](#).

For advanced geometries, the minimum spacing can depend on whether the metal is running in the preferred or nonpreferred direction. The preferred and nonpreferred fat metal spacing rule specifies different spacing values for the preferred and nonpreferred directions, and for the x-direction and y-direction.

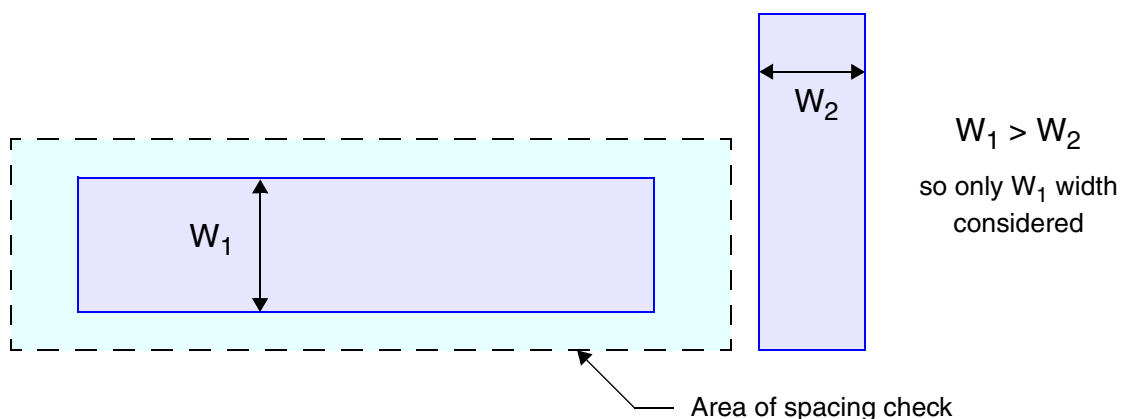
A rectangle is considered to be running in the preferred routing direction if the larger edges (sides) of the rectangle are parallel to that direction, as shown in [Figure 3-47](#). Otherwise, the rectangle is considered to be running in the nonpreferred direction.

Figure 3-47 Rectangle's Routing Direction



In checking the spacing between two metal shapes of different widths, the rule considers the only the wider shape's width. See [Figure 3-48](#).

Figure 3-48 Only Greater Width Considered

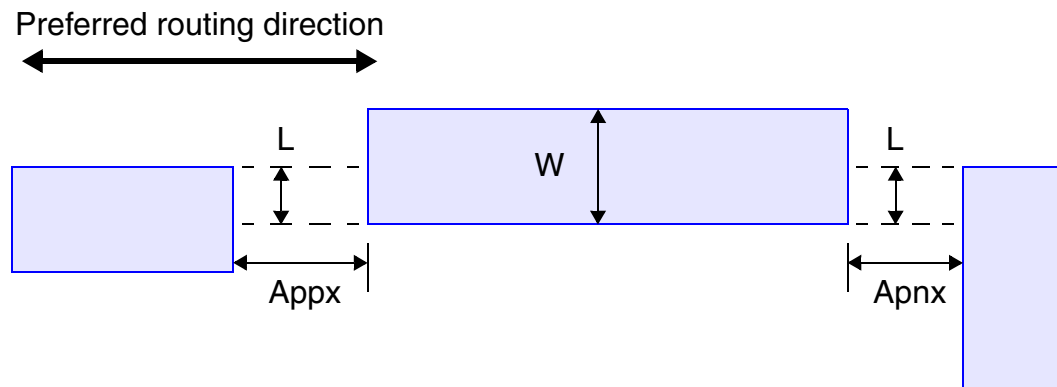


For the wider shape running in the preferred direction, the minimum spacing in the x-direction depends on the width, the length of the parallel overlap between the metal and the nearby metal, and whether the nearby metal is also running in the preferred direction.

X-Spacing From Shape in Preferred Direction

In [Figure 3-49](#), the widest metal is in the center, nearby metal to the left runs in the preferred direction, and the nearby metal to the right runs in the nonpreferred direction.

Figure 3-49 Fat Metal X Spacing Rule, Wider Metal Running in Preferred Direction



In the diagram, the notation **Appx** means the minimum spacing in the **x** direction between metal running in the **preferred** direction to nearby metal running in the **preferred** direction. Similarly, the notation **Apnx** means the minimum spacing in the **x** direction between metal running in the **preferred** direction to nearby metal running in the **nonpreferred** direction.

This is the syntax of the rule for x spacing from metal running in the preferred direction:

```
Layer "Mx" {
    fatTblPrefWidthDimension           = M
    fatTblPrefWidthThreshold           = (W1,W2,...,Wm)
    fatTblPrefYParallelLengthDimension = N
    fatTblPrefYParallelLengthThreshold = (L1,L2,...,Ln)
    fatTblPrefToPrefXMinSpacing        = (Appx1,Appx2,...)
    fatTblPrefToNonPrefXMinSpacing     = (Apnx1,Apnx2,...)
    ...
}
```

Each spacing attribute is a list of (M x N) values corresponding to each combination of fat metal width and parallel overlap. For example,

```
Layer "Mx" {
    fatTblPrefWidthDimension   = 3
    fatTblPrefWidthThreshold   = (0.09,0.14,0.18)
    fatTblPrefYParallelLengthDimension = 2
    fatTblPrefYParallelLengthThreshold = (0.0, 0.08)
}
```

```

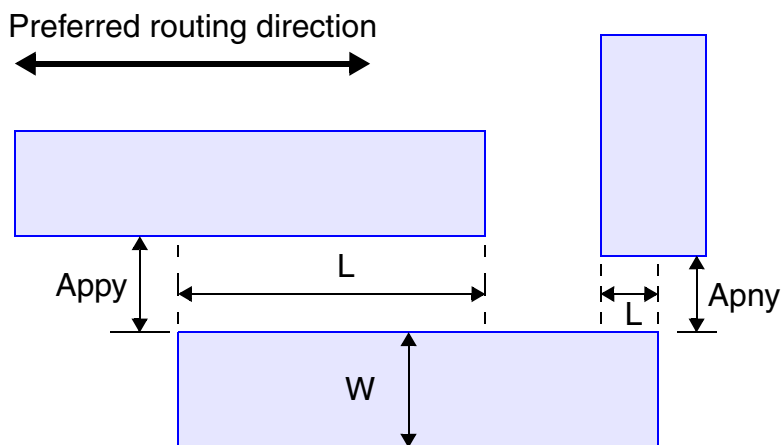
fatTblPrefToPrefXMinSpacing      = (0.06,0.08,0.11
                                     0.07,0.09,0.13)
fatTblPrefToNonPrefXMinSpacing  = (0.07,0.09,0.14
                                     0.08,0.10,0.15)
...

```

Y-Spacing From Shape in Preferred Direction

In [Figure 3-50](#), the widest metal is at the bottom, the nearby metal at the top left runs in the preferred direction, and the nearby metal to the right runs in the nonpreferred direction.

Figure 3-50 Fat Metal Y Spacing Rule, Wider Metal Running in Preferred Direction



This is the syntax of the rule for y spacing from metal running in the preferred direction:

```

Layer "Mx" {
  fatTblPrefWidthDimension      = M
  fatTblPrefWidthThreshold     = (W1,W2,...,Wm)
  fatTblPrefXParallelLengthDimension = N
  fatTblPrefXParallelLengthThreshold = (L1,L2,...,Ln)
  fatTblPrefToPrefYMinSpacing   = (Appy1,Appy2,...)
  fatTblPrefToNonPrefYMinSpacing = (Apny1,Apny2,...)
  ...
}

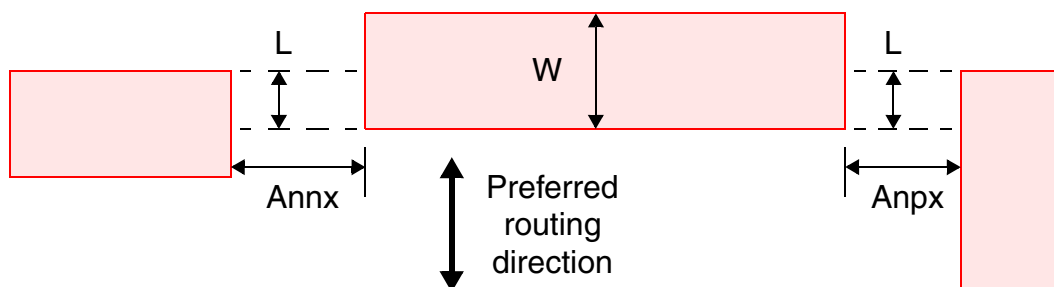
```

Each spacing attribute is a list of (M x N) values corresponding to each combination of fat metal width and parallel overlap.

X-Spacing From Shape in Nonpreferred Direction

In [Figure 3-51](#), the widest metal is in the center, nearby metal to the left runs in the nonpreferred direction, and the nearby metal to the right runs in the preferred direction.

Figure 3-51 Fat Metal X Spacing Rule, Wider Metal Running in Nonpreferred Direction



This is the syntax of the rule for x spacing from metal running in the nonpreferred direction:

```

Layer "Mx" {
    fatTblNonPrefWidthDimension           = R
    fatTblNonPrefWidthThreshold           = (W1,W2,...,Wr)
    fatTblNonPrefYParallelLengthDimension = S
    fatTblNonPrefYParallelLengthThreshold = (L1,L2,...,Ls)
    fatTblNonPrefToPrefXMinSpacing        = (Anpx1,Anpx2,...)
    fatTblNonPrefToNonPrefXMinSpacing     = (Annx1,Annx2,...)
    ...

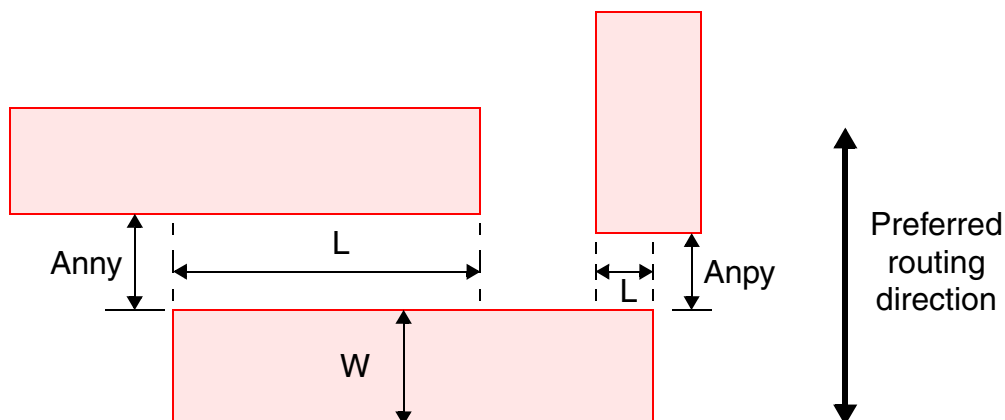
```

Each spacing attribute is a list of (R x S) values corresponding to each combination of fat metal width and parallel overlap.

Y-Spacing From Shape in Nonpreferred Direction

In [Figure 3-52](#), the widest metal is at the bottom, the nearby metal at the top left runs in the nonpreferred direction, and the nearby metal to the right runs in the preferred direction.

Figure 3-52 Fat Metal Y Spacing Rule, Wider Metal Running in Nonpreferred Direction



This is the syntax of the rule for y spacing from metal running in the nonpreferred direction:

```

Layer "Mx" {
  fatTblNonPrefWidthDimension          = R
  fatTblNonPrefWidthThreshold          = (W1,W2,...,Wr)
  fatTblNonPrefXParallelLengthDimension = S
  fatTblNonPrefXParallelLengthThreshold = (L1,L2,...,Ls)
  fatTblNonPrefToPrefYMinSpacing       = (Anpy1,Anpy2,...)
  fatTblNonPrefToNonPrefYMinSpacing    = (Anny1,Anny2,...)
  ...

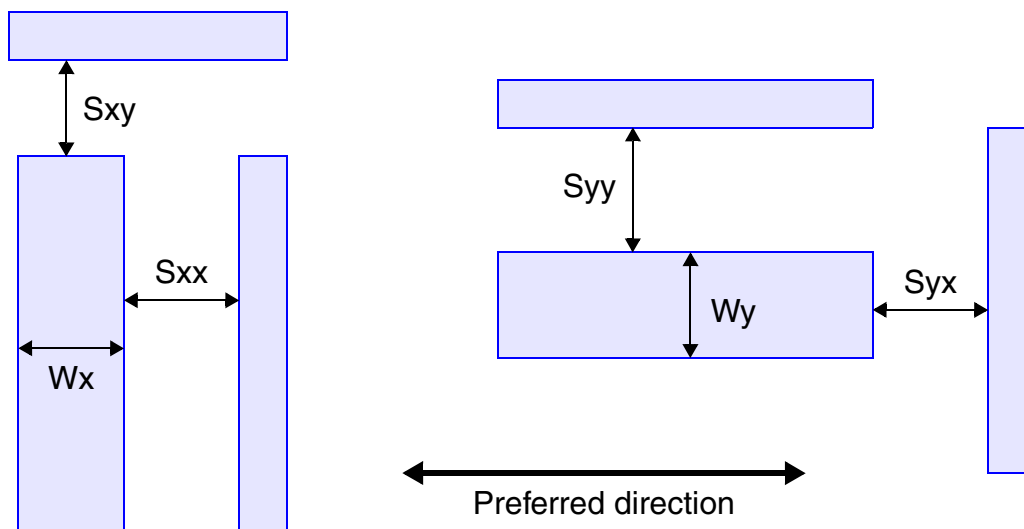
```

Each spacing attribute is a list of (R x S) values corresponding to each combination of fat metal width and parallel overlap.

Fat Metal Spacing Exclusion Range

For advanced geometries, you can have the fat metal spacing check ignored when a thin metal geometry falls within a distance range from the fat metal in the preferred direction, in the nonpreferred direction, or in both directions, as shown in [Figure 3-53](#).

Figure 3-53 Fat Metal Spacing Exclusion Ranges



To invoke the fat metal spacing exclusion range, use the following syntax in the `Layer` section:

```

Layer "MetalX" {
  fatTblPrefWidthDimension    = M
  fatTblPrefWidthThreshold    = (...)M
  fatTblPrefYParallelLengthDimension = N1
  fatTblPrefYParallelLengthThreshold = (...)N1
  fatTblPrefXMinSpacing       = (...)MxN1
  fatTblPrefXParallelLengthDimension = N2

```

```

fatTblPrefXParallelLengthThreshold = (...) N2
fatTblPrefYMinSpacing = (...) MxN2
fatTblNonPrefWidthDimension = R
fatTblNonPrefWidthThreshold = (...) R
fatTblNonPrefYParallelLengthDimension = S1
fatTblNonPrefYParallelLengthThreshold = (...) S1
fatTblNonPrefXMinSpacing = (...) RxS1
fatTblNonPrefXParallelLengthDimension = S2
fatTblNonPrefXParallelLengthThreshold = (...) S2
fatTblNonPrefYMinSpacing = (...) RxS2
fatTblPrefXExcludedSpacingRange = ("L11", U11, ..., ..., ...) MxN1
fatTblPrefYExcludedSpacingRange = (...) MxN2
fatTblNonPrefXExcludedSpacingRange = (...) RxS1
fatTblNonPrefYExcludedSpacingRange = (...) RxS2
}

```

For example,

```

Layer "MetalX"
fatTblPrefWidthDimension = 7
fatTblPrefWidthThreshold = (0.00, 0.12, 0.16, 0.23, 0.35, 0.70, 2.6)
fatTblPrefYParallelLengthDimension = 4
fatTblPrefYParallelLengthThreshold = (0, 0.18, 0.25, 0.36)
fatTblPrefToPrefXMinSpacing = (0.041, 0.041, 0.041, 0.041,
                                0.041, 0.094, 0.094, 0.094,
                                0.041, 0.136, 0.136, 0.136,
                                0.041, 0.136, 0.160, 0.160,
                                0.041, 0.136, 0.160, 0.180,
                                0.250, 0.250, 0.250, 0.250,
                                0.400, 0.390, 0.390, 0.390)
fatTblPrefToNonPrefXMinSpacing = (0.041, 0.041, 0.041, 0.041,
                                   0.041, 0.094, 0.094, 0.094,
                                   0.041, 0.136, 0.136, 0.136,
                                   0.041, 0.136, 0.160, 0.160,
                                   0.041, 0.136, 0.160, 0.180,
                                   0.250, 0.250, 0.250, 0.250,
                                   0.400, 0.390, 0.390, 0.390)
fatTblPrefXExcludedSpacingRange = (-1, -1, -1, -1,
                                    -1, "0.055, 0.074", "0.055, 0.074", "0.055, 0.074",
                                    -1, "0.055, 0.074", "0.055, 0.074", "0.055, 0.074",
                                    -1, "0.055, 0.074", -1, -1,
                                    -1, "0.055, 0.074", -1, -1,
                                    -1, "0.055, 0.074", -1, -1,
                                    -1, "0.055, 0.074", -1, -1)
fatTblPrefXParallelLengthDimension = 4
fatTblPrefXParallelLengthThreshold = (0, 0.184, 0.240, 0.378)
fatTblPrefToPrefYMinSpacing = (0.041, 0.041, 0.041, 0.041,
                                0.041, 0.094, 0.094, 0.094,
                                0.041, 0.136, 0.136, 0.136,
                                0.041, 0.136, 0.160, 0.160,
                                0.041, 0.136, 0.160, 0.180,
                                0.250, 0.250, 0.250, 0.250,
                                0.400, 0.390, 0.390, 0.390)

```

```

fatTblPrefToNonPrefYMinSpacing = (0.041, 0.041, 0.041, 0.041,
                                   0.041, 0.094, 0.094, 0.094,
                                   0.041, 0.136, 0.136, 0.136,
                                   0.041, 0.136, 0.160, 0.160,
                                   0.041, 0.136, 0.160, 0.180,
                                   0.250, 0.250, 0.250, 0.250,
                                   0.400, 0.390, 0.390, 0.390)
fatTblPrefYExcludedSpacingRange = (-1, -1, -1, -1,
                                     -1, "0.055, 0.074", "0.055, 0.074", "0.055, 0.074",
                                     -1, "0.055, 0.074", "0.055, 0.074", "0.055, 0.074",
                                     -1, "0.055, 0.074", -1, -1,
                                     -1, "0.055, 0.074", -1, -1,
                                     -1, "0.055, 0.074", -1, -1,
                                     -1, "0.055, 0.074", -1, -1)
}

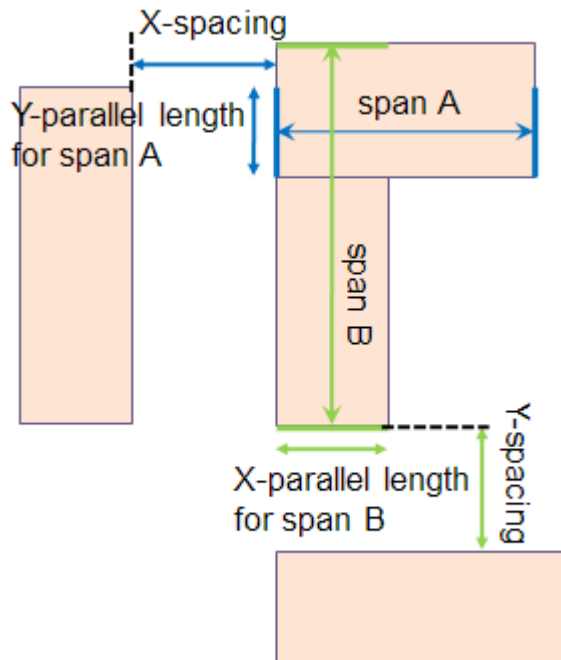
```

Metal Span Orthogonal Spacing Rule

The metal span spacing rule defines the minimum spacing between two parallel metal segments, subject to the parallel length between them, with the minimum spacing value based on the span rather than width of the metal segments. The span of a metal segment, unlike its width, is always measured perpendicular to the edges of the two metal segments whose spacing is under consideration. The basic rule is described in [“Metal Span Spacing Rule” on page 2-57](#).

For advanced geometries, you can specify different X spacing values that depend on the X spans and Y parallel overlaps, and different Y spacing values that depend on the Y spans and X parallel overlaps, as indicated in [Figure 3-54](#).

Figure 3-54 Metal Span Orthogonal Spacing Rule



This is the syntax for the advanced form of the rule:

```

Layer "MetalX" {
    spanTblXDimension           = M
    spanTblXThreshold           = (X1, ..., Xm)
    spanTblYParallelLengthDimension = K
    spanTblYParallelLength      = (P1, .. , Pk)
    spanTblXMinSpacing          = (Sx1, Sx2, ...) # M*K table

    spanTblYDimension           = N
    spanTblYThreshold           = (X1, .. , Xn)
    spanTblXParallelLengthDimension = L
    spanTblXParallelLength      = (P1, .. , Pl)
    spanTblYMinSpacing          = (Sx1, Sx2, ...) # N*L table
}

```

For example,

```

Layer "Metal5" {
    ...
    spanTblXDimension           = 5
    spanTblXThreshold           = (0, 0.09, 0.14, 0.18, 0.23)
    spanTblYParallelLengthDimension = 4
    spanTblYParallelLength      = (0.0, 0.002, 0.15, 0.24)
}

```

```

spanTblXMinSpacing          = (0.09, 0.09, 0.09, 0.09,
                               0.09, 0.09, 0.09, 0.09,
                               0.09, 0.09, 0.09, 0.09,
                               0.11, 0.11, 0.14, 0.14,
                               0.16, 0.16, 0.16, 0.18)

spanTblYDimension           = 5
spanTblYThreshold           = (... 5 values ...)
spanTblXParallelLengthDimension = 4
spanTblXParallelLength      = (... 4 values ...)
spanTblYMinSpacing         = (... 4 x 5 array ...)
...
}

```

You can optionally waive the rule based on specified line-end widths, Z-shape dimensions, and ranges of spacing values.

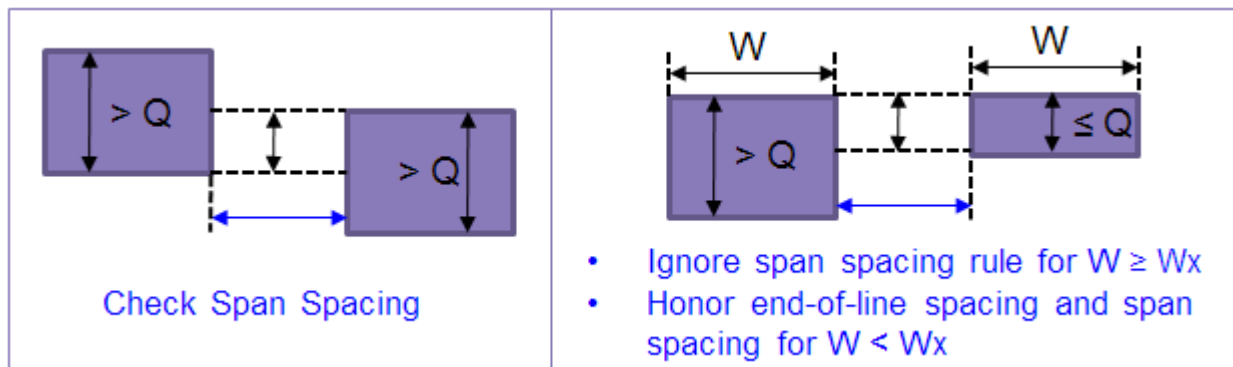
Line-End Exceptions

The span spacing rule is waived for a line-end if both of the following conditions are met:

- The line-end width is no more than the attribute Q
(`spanRuleExcludedForEndOfLineMaxWidthThreshold`)
- The line-end X or Y span is at least the attribute Wx or Wy, respectively
(`spanRuleExcludedForEndOfLineXThreshold` or
`spanRuleExcludedForEndOfLineYThreshold`)

The line-end exceptions are illustrated in [Figure 3-55](#).

Figure 3-55 Metal Span Orthogonal Spacing Line-End Exceptions



For example,

```

spanRuleExcludedForEndOfLineMaxWidthThreshold = 0.09
spanRuleExcludedForEndOfLineXThreshold       = 0.18
spanRuleExcludedForEndOfLineYThreshold       = 0.23

```

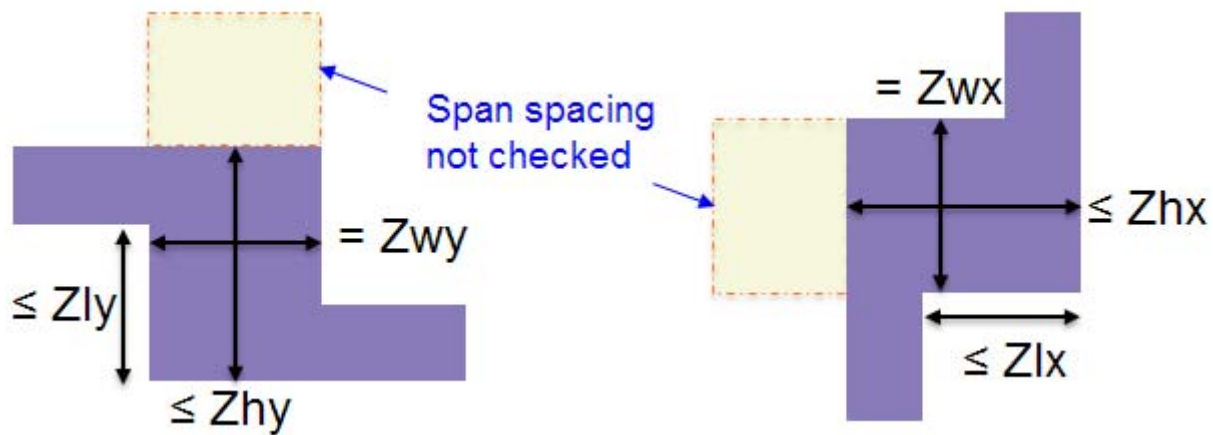
You can optionally specify a line-end minimum length for which the rule exclusion applies, for example,

```
spanRuleExcludedForEndOfLineMinLength = 0.09
```

Z-Shape Exceptions

The span spacing rule can be waived for any Z-shape with width equal to Z_w , height no more than Z_h , and edge length no more than Z_l , as shown in [Figure 3-56](#).

Figure 3-56 Metal Span Orthogonal Spacing Z-Shape Exceptions



This is the syntax for the Z-shape exception:

```
spanTblXExcludedForZWireWidth      = (Zwx, -1, ...)
spanTblXExcludedForZWireMaxHeight  = (Zhx, -1, ...)
spanTblXExcludedForZWireMaxEdgeLength = (Zlx, -1, ...)

spanTblYExcludedForZWireWidth      = (Zwy, -1, ...)
spanTblYExcludedForZWireMaxHeight  = (Zhy, -1, ...)
spanTblYExcludedForZWireMaxEdgeLength = (Zly, -1, ...)
```

For example,

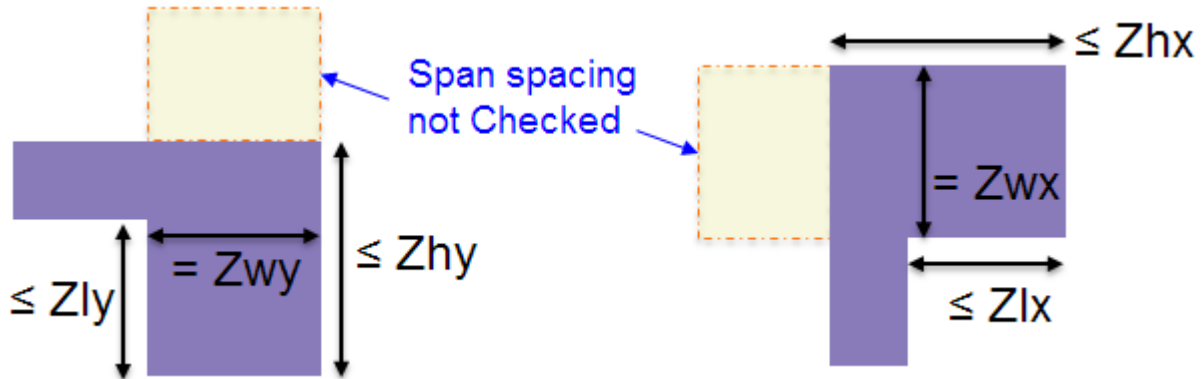
```
spanTblYDimension = 5
...
spanTblYExcludedForZWireWidth      = ( -1, -1, -1, 0.09, 0.09)
spanTblYExcludedForZWireMaxHeight  = ( -1, -1, -1, 0.14, 0.14)
spanTblYExcludedForZWireMaxEdgeLength = ( -1, -1, -1, 0.10, 0.10)
```

In this example, the rule is waived for the fourth and fifth span ranges when the Z-shape dimensions meet the Z_{wy} , Z_{hy} , and Z_{ly} requirements of the exception. The -1 values in the table indicate no waiving of the rule for the corresponding parallel overlap ranges.

L-Shape Exceptions

The span spacing rule can be waived for any L-shape with width equal to Z_w , height no more than Z_h , and edge length no more than Z_l , as shown in [Figure 3-57](#).

Figure 3-57 Metal Span Orthogonal Spacing L-Shape Exceptions



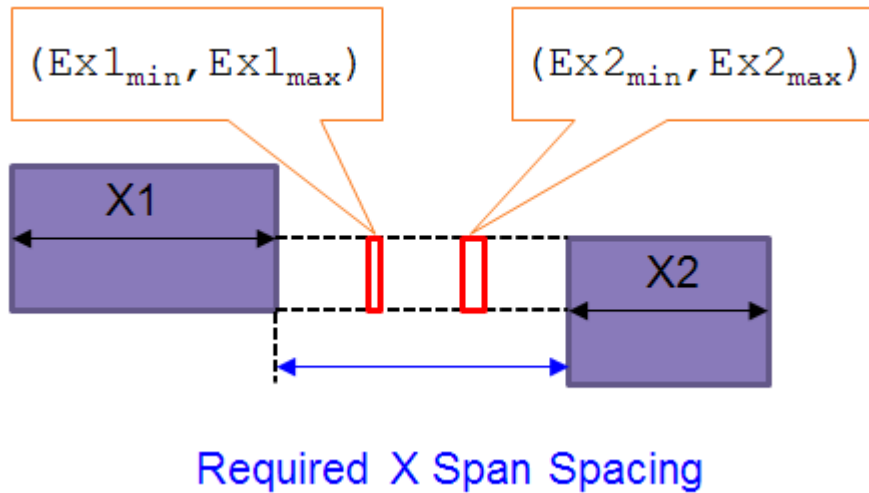
This is the syntax for the L-shape exception is made by adding a line to the syntax for the Z-shape exception:

```
spanTblXExcludedForZWireWidth           = (Zwx, -1, ...)
spanTblXExcludedForZWireMaxHeight        = (Zhx, -1, ...)
spanTblXExcludedForZWireMaxEdgeLength    = (Zlx, -1, ...)
spanTblYExcludedForZWireWidth           = (Zwy, -1, ...)
spanTblYExcludedForZWireMaxHeight        = (Zhy, -1, ...)
spanTblYExcludedForZWireMaxEdgeLength    = (Zly, -1, ...)
spanRuleZWireIncludeLWire              = 1 #default=0
```

Spacing Range Exceptions

The span spacing rule can be waived for certain specified spacing ranges in the x-direction and y-direction. A table of value pairs specifies the spacing ranges where the rule is waived, depending on the parallel overlap length ranges, as shown in [Figure 3-57](#).

Figure 3-58 Metal Span Orthogonal Spacing Range Exceptions



This is the syntax for the for the spacing range exceptions:

```
spanTblYParallelLengthDimension = K
spanTblXParallelLengthDimension = L
...
spanTblXExcludedSpacingRange = ("Ex1min,Ex1max", ...) # 1*K table
spanTblYExcludedSpacingRange = ("Ey1min,Ey1max", ...) # 1*L table
```

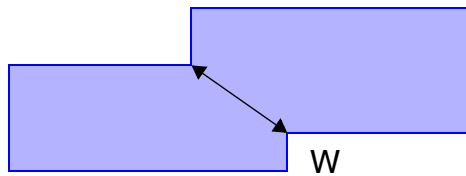
For example,

```
spanTblYExcludedSpacingRange = (-1, -1, "0.04,0.04,0.060,0.062", -1, -1)
```

In this example, for the third parallel overlap range, a spacing of exactly 0.04 or between 0.060 and 0.062 is allowed instead of prohibited. The -1 values in the table indicate no waiving of the rule for the corresponding parallel overlap range.

Orthogonal Minimum Width Rule

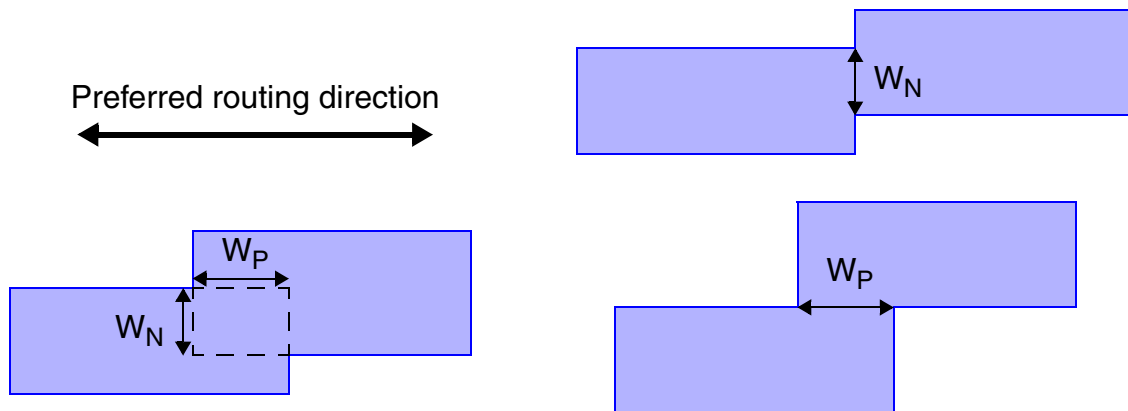
The `minWidth` attribute in the `Layer` section specifies the minimum width of any metal geometry. By default, for a metal jog, the minimum width is measured diagonally at the narrowest point, as shown in [Figure 3-60](#).

Figure 3-59 Default Minimum Width Measurement

The orthogonal minimum width rule specifies different minimum widths for a metal jog in the preferred and nonpreferred routing directions. The following example shows the rule syntax:

```
Layer "M1" {
  minWidth           = 0.48 # WP, preferred direction
  nonPreferredWidth   = 0.92 # WN, nonpreferred direction
  minWidthCheckManhattan = 1
}
```

The rule is satisfied if either one of the two width requirements is satisfied, in the preferred or nonpreferred direction. In [Figure 3-60](#), the orthogonal minimum width rule requires the width measured in the preferred direction to be at least W_P or the width measured in the nonpreferred direction to be at least W_N .

Figure 3-60 Orthogonal Minimum Width Rule

Forbidden Pitch Rule

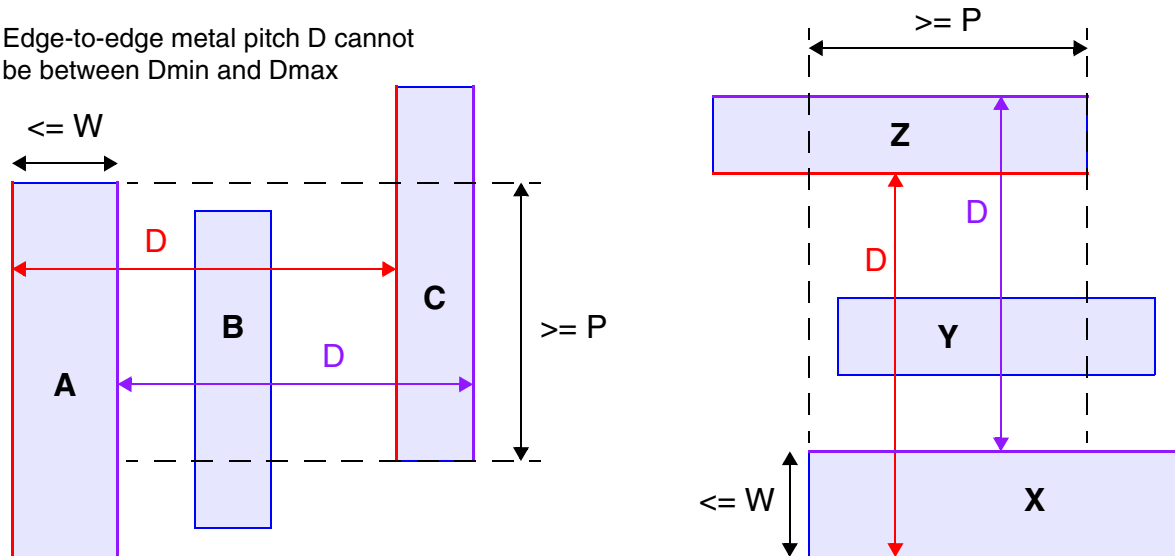
The forbidden pitch rule prevents two metal shapes from having a pitch in the range from D_{min} to D_{max} when the width of one or both shapes is no more than a threshold W , and there is exactly one polygon sandwiched between the two metal shapes.

In [Figure 3-61](#), shapes A and C have a parallel overlap of at least P , at least one of them has a width no more than W , and they have exactly one polygon between them, so the rule checks the pitch of the two shapes A and C. There are two pitch values to test: the distance

from the left edge of A to the left edge of C and the distance from the right edge of A to the right edge of C. Neither distance D can be in the range $D_{min} \leq D \leq D_{max}$.

Figure 3-61 Forbidden Pitch Rule

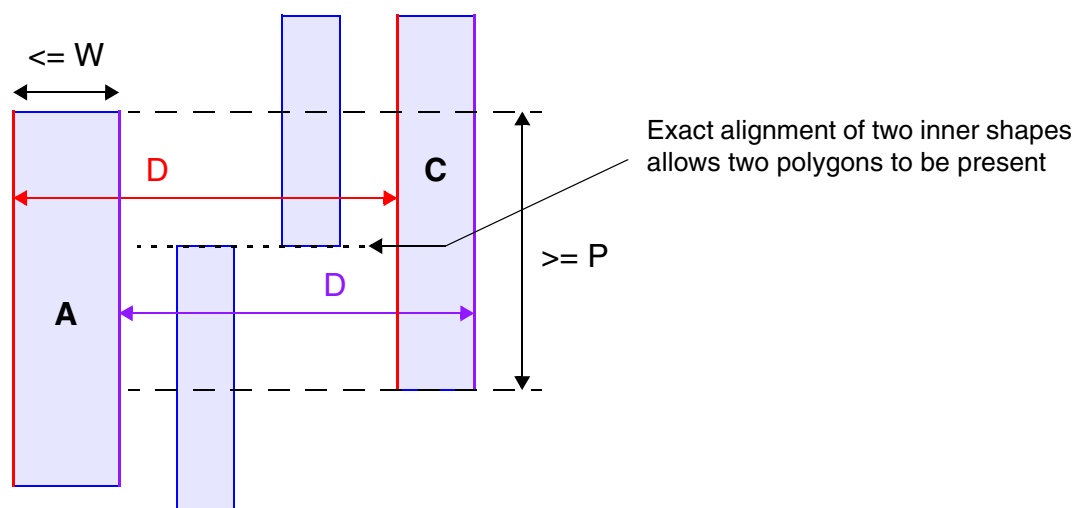
Edge-to-edge metal pitch D cannot be between D_{min} and D_{max}



Similarly, for horizontal shapes X and Z with one shape Y between them, the pitch value D measured from top edge to top edge or bottom edge to bottom edge cannot be in the range $D_{min} \leq D \leq D_{max}$.

Two polygons are allowed in the forbidden pitch range only when they do not overlap and exactly meet at the same line in the direction perpendicular to the outer shape run length, as shown in Figure 3-62.

Figure 3-62 Two Polygons in Forbidden Pitch Rule



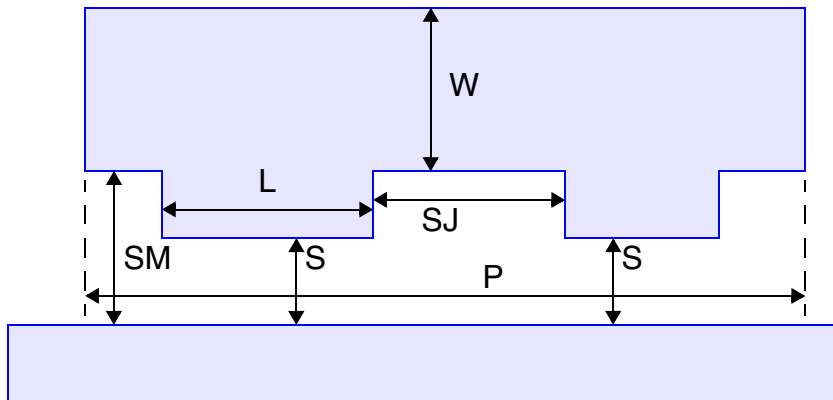
The following example shows the rule syntax:

```
Layer "MetalX" {
  forbiddenPitchWireMaxWidthThreshold = 0.420 # W
  forbiddenPitchWireParallelLength    = 0.001 # P
  forbiddenPitchKeepoutMinDist        = 0.158 # Dmin
  forbiddenPitchKeepoutMaxDist        = 0.192 # Dmax
}
```

Wide Metal Jog Rule

The wide metal jog rule specifies the minimum allowed spacing SJ between two metal jogs and between a jog and nearby metal S when one or more jogs with a length of at least L are connected to fat metal of width W , when the parallel run length of the jogs is P and the spacing from the fat metal to the nearby metal is no more than SM . See [Figure 3-63](#). This rule is waived if the spacing S between the jog and the nearby metal is in the range from SA to SB inclusive ($SA \leq S \leq SB$).

Figure 3-63 Wide Metal Jog Rule



This is a table-based rule having syntax in the following form:

```
Layer "MetalX" {
  fatMetalJogTblSize                = 4
  fatMetalJogThresholdTbl           = (W1, W2, W3, W4, W5)
  fatMetalJogParallelLengthTbl      = (P1, P2, P3, P4, P5)
  fatMetalJogMaxSpacingThresholdTbl = (SM1, SM2, SM3, SM4, SM5)
  fatMetalJogLengthTblSize          = 2
  fatMetalJogLengthTbl              = (0, L1)
  fatMetalJogMinSpacingTbl           = (S11, S12,
                                         S21, S22,
                                         S31, S32,
                                         S41, S42)
  fatMetalJogToJogMinSpacingTbl     = (SJ1, SJ2, SJ3, SJ4)
```

```

    fatMetalJogExcludedMinSpacingTbl = (SA1,SA2,-1,-1)
    fatMetalJogExcludedMaxSpacingTbl = (SB1,SB2,-1,-1)
}

```

For example,

```

Layer "MetalX" {
    fatMetalJogTblSize = 5
    fatMetalJogThresholdTbl = (0.118, 0.121, 0.161, 0.358, 0.700)
    fatMetalJogParallelLengthTbl = (0.350, 0.350, 0.350, 0.350, 0.590)
    fatMetalJogMaxSpacingThresholdTbl = (0.138, 0.170, 0.184, 0.340, 0.406)
    fatMetalJogToJogMinSpacingTbl = (0.400, 0.400, 0.400, 0.400, 0.400)
    fatMetalJogLengthTblSize = 2
    fatMetalJogLengthTbl = (0.000, 0.100)
    fatMetalJogMinSpacingTbl = (0.042, 0.098,
                                0.098, 0.136,
                                0.138, 0.166,
                                0.166, 0.194,
                                0.194, 0.269)
    fatMetalJogExcludedMaxSpacingTbl = (0.078, 0.078, -1, -1, -1, -1)
    fatMetalJogExcludedMinSpacingTbl = (0.051, 0.051, -1, -1, -1, -1)
}

```

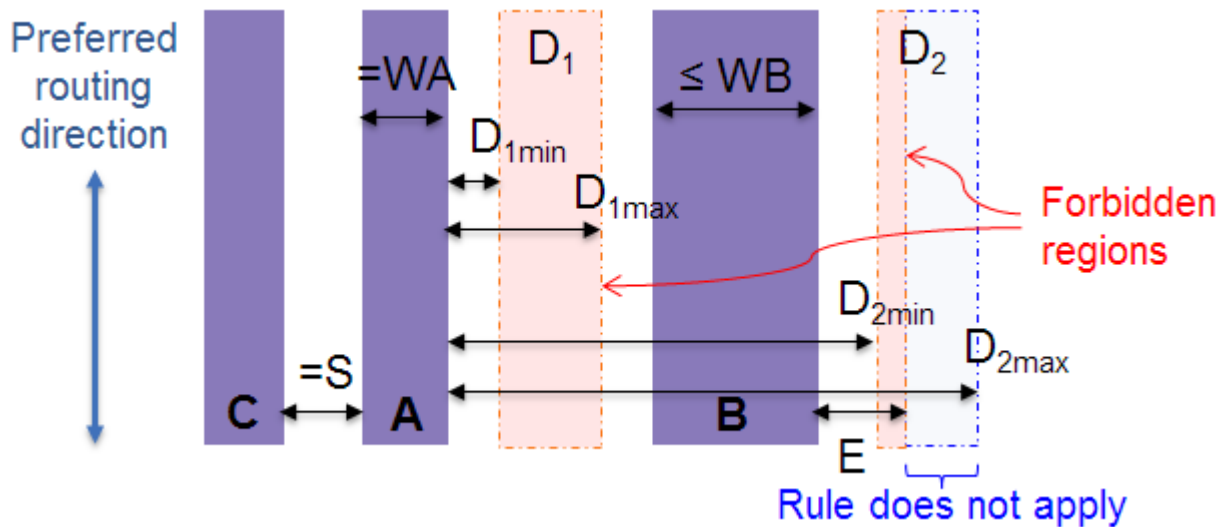
Preferred-Direction Forbidden Spacing Rule

The preferred-direction forbidden spacing rule specifies one or more distance ranges where metal is forbidden between or beyond two metal shapes, A and B, measured from the edge of shape A, under all of the following conditions:

- Shape A runs in the preferred routing direction
- The width of shape A is W_A
- The width of shape B is no more than W_B
- Another shape C on the other side of shape A is spaced at a distance S from shape A.

The rule defines one or more forbidden regions, with each region defined by a minimum and maximum distance from shape A, as shown in [Figure 3-64](#).

Figure 3-64 Preferred-Direction Forbidden Spacing Rule



In this example, there are two forbidden regions, each defined by a pair of values, D_{min} and D_{max} . An optional attribute E specifies a threshold for disabling the rule at a specified distance beyond the edge of shape B .

This is the syntax of the rule:

```

Layer "MetalX" {
  forbiddenSpacePrefTblSize           = 2
  forbiddenSpacePrefWireWidthTbl      = (WA1, WA2)
  forbiddenSpacePrefWireSpacingTbl    = (S1, S2)
  forbiddenSpacePrefForbiddenWireMaxWidthTbl = (WB1, WB2)
  forbiddenSpacePrefRangeTbl          = ("D11min, D11max, D12min, D12max",
                                         "D21min, D21max, ... ")
  forbiddenSpacePrefExcludedTbl       = (E1, E2) ; optional
}

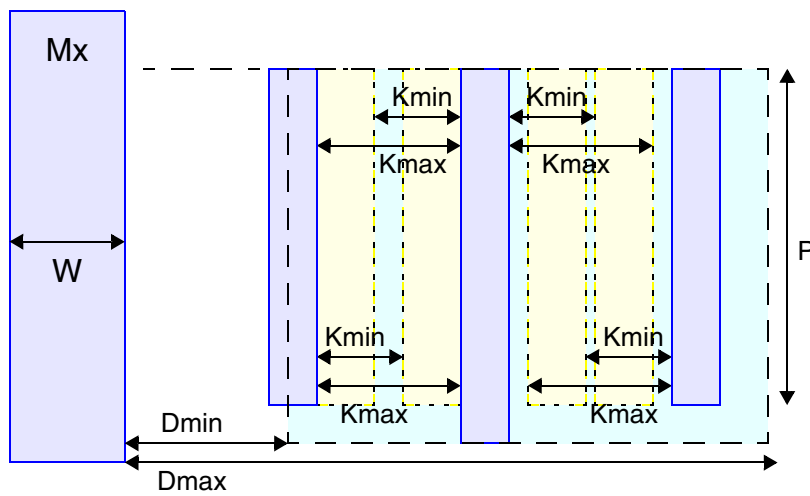
```

Fat Metal Forbidden Spacing Rule

The fat metal forbidden spacing rule specifies a range of prohibited spacing between two parallel metal shapes that are within a specified range away from a fat metal shape.

In [Figure 3-65](#), the metal shape on the left is fat metal because it has a width of at least W . When two or more metal shapes have a parallel run length P within a distance range from the fat metal between D_{min} and D_{max} , each metal shape in that area cannot occupy the space from K_{min} to K_{max} away from the other metal shapes (the yellow areas).

Figure 3-65 Fat Metal Forbidden Spacing Rule



This is the syntax for the rule:

```

Layer "MetalX" {
  forbiddenSpaceWidthThreshold      = W
  forbiddenSpaceCheckRangeLower     = Dmin
  forbiddenSpaceCheckRangeUpper     = Dmax
  forbiddenSpaceParallelLength      = P
  forbiddenSpaceKeepoutMinDist      = Kmin
  forbiddenSpaceKeepoutMaxDist      = Kmax
}

```

For example,

```

Layer "Metal1" {
  forbiddenSpaceWidthThreshold      = 0.182
  forbiddenSpaceCheckRangeLower     = 0.051
  forbiddenSpaceCheckRangeUpper     = 0.880
  forbiddenSpaceParallelLength      = 0.159
  forbiddenSpaceKeepoutMinDist      = 0.072
  forbiddenSpaceKeepoutMaxDist      = 0.078
}

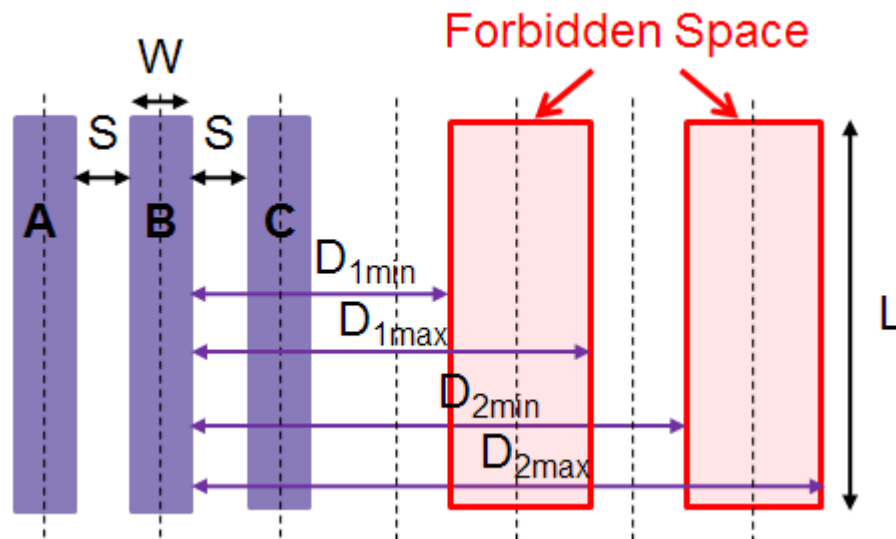
```

Three-Wire Forbidden Space Rule

The three-wire fat metal forbidden space rule specifies one or more regions surrounding a metal line and two nearby metal lines where the same metal layer is forbidden if the parallel run of that metal is at least a specified value.

In [Figure 3-66](#), the metal wire B has a width no more than W and has two neighboring wires A and C, each with an edge-to-edge spacing from B of no more than S . In this situation, a metal shape with a parallel run of at least L is forbidden in the red region from a distance D_{1min} to distance D_{1max} from the edge of wire B. The rule can optionally specify multiple regions at different distances, such as a second region from D_{2min} to D_{2max} .

Figure 3-66 Three-Wire Forbidden Space Rule

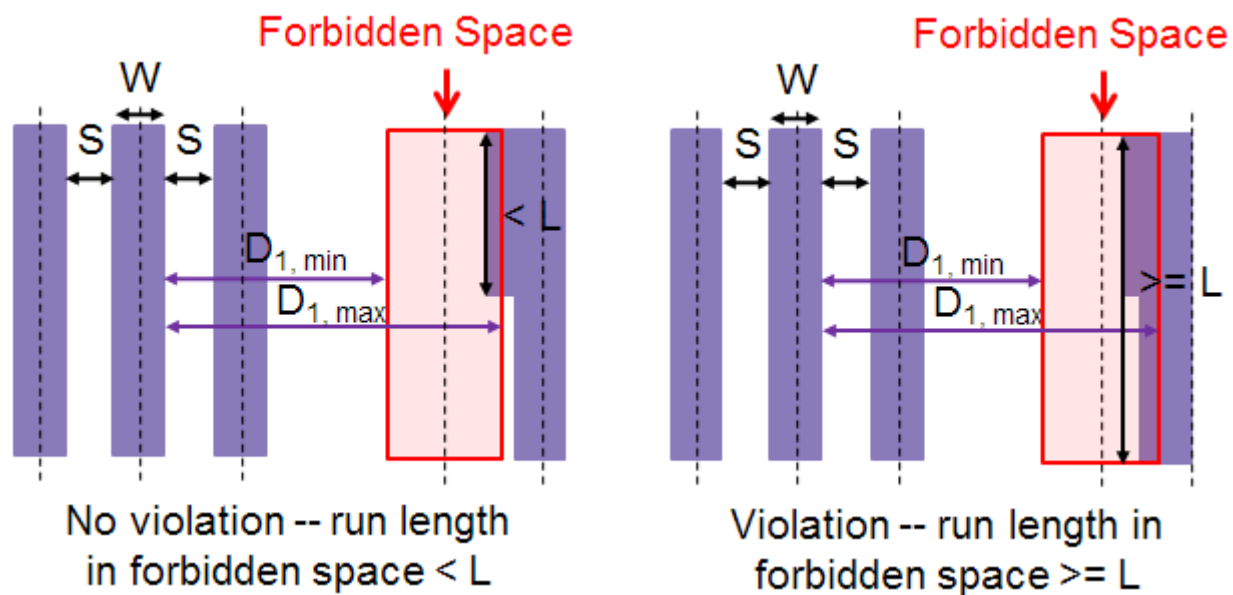


The following example demonstrates the rule syntax:

```
Layer "Metal1" {
    ...
    forbiddenSpaceWireMaxWidthThreshold    = 0.045    # W
    forbiddenSpaceWireMaxSpacingThreshold  = 0.045    # S
    forbiddenSpaceWireParallelLength       = 0.12     # L
    forbiddenSpaceRangeTblSize             = 2         # n
    forbiddenSpaceRangeTbl                 = ("0.12, 0.16, 0.18, 0.22") # D
}
```

[Figure 3-67](#) illustrate usage of the rule. The example on the left is not a violation because the parallel run within the forbidden space is less than L . On the other hand, the example on the right is a violation because the parallel run within the forbidden space is at least L (even though a jog is present in the run).

Figure 3-67 Three-Wire Forbidden Space Rule Examples

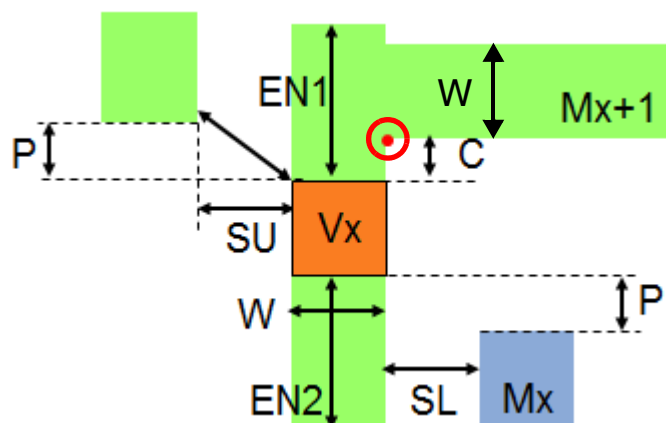


Via-Induced Metal Bridge Rule

The via-induced metal bridge rule specifies a minimum spacing between a via and a neighboring upper-metal shape under conditions that favor the formation of an unwanted metal bridge between the neighboring metal and the metal covering the via.

In [Figure 3-68](#), the rule specifies a minimum spacing SU between a via on layer Vx and a neighboring upper-metal shape not connected to the via.

Figure 3-68 Via-Induced Metal Bridge Rule



The rule applies when all of the following conditions are true:

- The parallel run length between the via and the neighboring upper-metal shape is at least P, where P is a negative number.
- The via is connected to an upper-metal shape having a width of exactly W.
- The via has zero enclosure on the two sides parallel to the neighboring upper-metal shape and an enclosure no greater than EN on the other two sides.
- The via is located no more that a distance C from a concave corner of the connected upper-metal shape.
- A neighboring lower-metal shape having a parallel run length with the via of at least P is exactly the distance SL from the via's connecting upper-metal shape.

When all of these conditions are true, the neighboring upper-metal shape must be at least the distance SU away from the via. This distance is measured diagonally from a via corner.

This is the syntax for the rule:

```
DesignRule {
    layer1                = "Vx"
    layer2                = "Mx+1"
    thinWireCutMetalExactWidth = W
    thinWireCutTblSize      = 1
    thinWireCutNameTbl      = (Vsm)
    thinWireCutSideEncTbl   = (0)
    thinWireCutOrthoEncTbl  = (EN)
    thinWireCutParallelLengthTbl = (P)
    thinWireCutToConcaveCornerMaxDistTbl = (C)
    thinWireCutToLowerMetalMinSpacingTbl = (SL)
    thinWireCutToUpperMetalMinSpacingTbl = (SU)
}
```

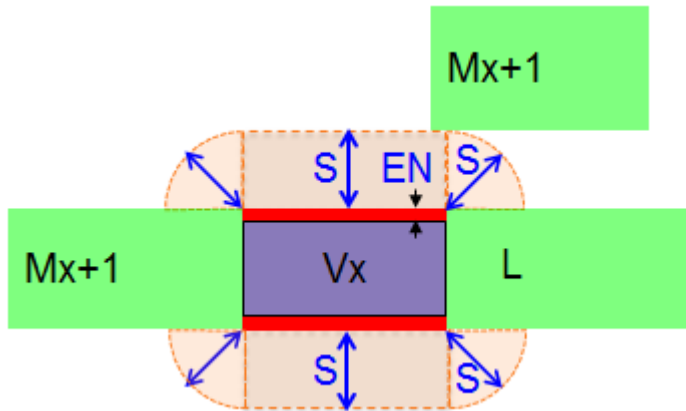
For example,

```
DesignRule {
    layer1                = "VIA2"
    layer2                = "MET2"
    thinWireCutMetalExactWidth = 0.038
    thinWireCutTblSize      = 1
    thinWireCutNameTbl      = (Vsm)
    thinWireCutSideEncTbl   = (0)
    thinWireCutOrthoEncTbl  = (0.055)
    thinWireCutParallelLengthTbl = (-0.005)
    thinWireCutToConcaveCornerMaxDistTbl = (0.055)
    thinWireCutToLowerMetalMinSpacingTbl = (0.040)
    thinWireCutToUpperMetalMinSpacingTbl = (0.040)
}
```

Enclosure-Dependent Metal Spacing Rule

The upper-metal enclosure around a rectangular via must keep a minimum spacing S to its neighboring metal shapes when the enclosure width on the longer edge of the via is less than a threshold EN . The spacing is measured diagonally from each corner of the enclosure metal, as shown in [Figure 3-69](#).

Figure 3-69 Enclosure-Dependent Metal Rule



This is a table-based rule having the following syntax:

```
DesignRule {
  layer1                = "Vx"
  layer2                = "Mx+1"
  encMetalCutTblSize    = 2
  encMetalCutNameTbl    = (Vh, Vv)
  encMetalEncThresholdTblSize = n
  encMetalEncThresholdTbl = (EN1, EN2, ...)
  # enclosure on long edge of bar via
  encMetalXMinSpacingTbl = (Sx11, Sx12, ..., Sx1n, ...)
  # 2D (2*n), metal-to-metal X spacing
  encMetalYMinSpacingTbl = (Sy11, Sy12, ..., Sy1n, ...)
  # 2D (2*n), metal-to-metal Y spacing
}
```

For example,

```
DesignRule {
  layer1                = "VIA1"
  layer2                = "MET2"
  encMetalCutTblSize    = 2
  encMetalCutNameTbl    = (Vh, Vv)
  encMetalEncThresholdTblSize = 2
```

```

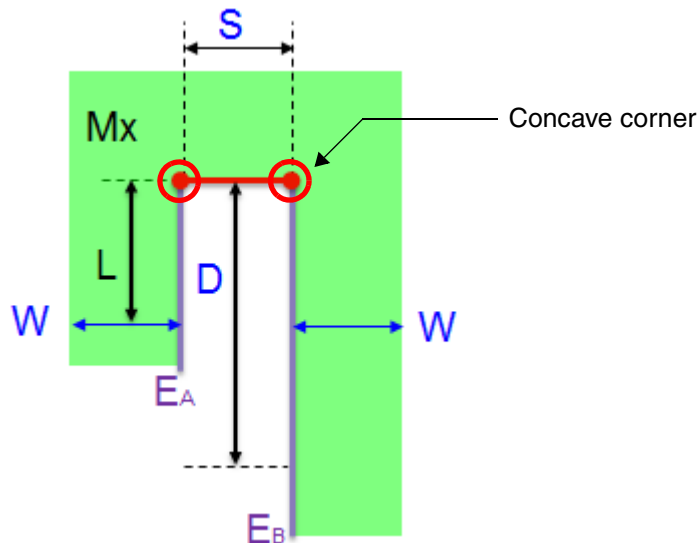
encMetalEncThresholdTbl      = (0.002, 0.006)
encMetalXMinSpacingTbl      = (  -1,   -1,
                                0.045, 0.036)
encMetalYMinSpacingTbl      = (0.045, 0.036,
                                -1,   -1)
}

```

U-Shape Leg Spacing Rule

The U-shape leg spacing rule specifies the minimum spacing between the legs of a metal “U” shape under certain conditions. In [Figure 3-70](#), the two legs of the “U” shape have widths no greater than the threshold W for more than a length L from the base of the “U.” The two parallel inner edges of the legs, labeled E_A and E_B , have exactly one end connected to a concave corner. The rule specifies a minimum spacing of at least S between these two edges for a depth D from the base of the “U” shape.

Figure 3-70 U-Shape Leg Spacing Rule



The rule uses the following syntax:

```

Layer "MetalX" {
    uShapeSideWireMaxWidthThreshold = W
    uShapeSideWireMinLength         = L
    uShapeSideWireLengthMaxCheckRange = D
    uShapeSideToSideMinSpacing      = S
}

```

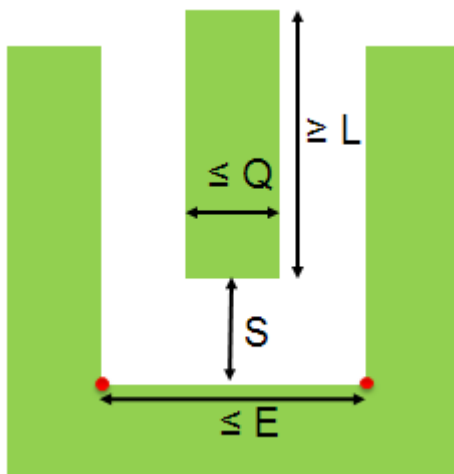
For example,

```
Layer "MET1" {
  uShapeSideWireMaxWidthThreshold = 0.044
  uShapeSideWireMinLength         = 0.096
  uShapeSideWireLengthMaxCheckRange = 0.296
  uShapeSideToSideMinSpacing       = 0.090
}
```

Line-End to U-Shape Spacing Rule

The line-end to U-shape spacing rule specifies the minimum spacing between the end of a line and the inner bottom edge of a U-shape, when the line has a width no more than Q and length of at least L , and the bottom inside edge of the U-shape has a length no more than E , as shown in [Figure 3-71](#).

Figure 3-71 Line-End to U-Shape Spacing Rule



The rule uses the following syntax:

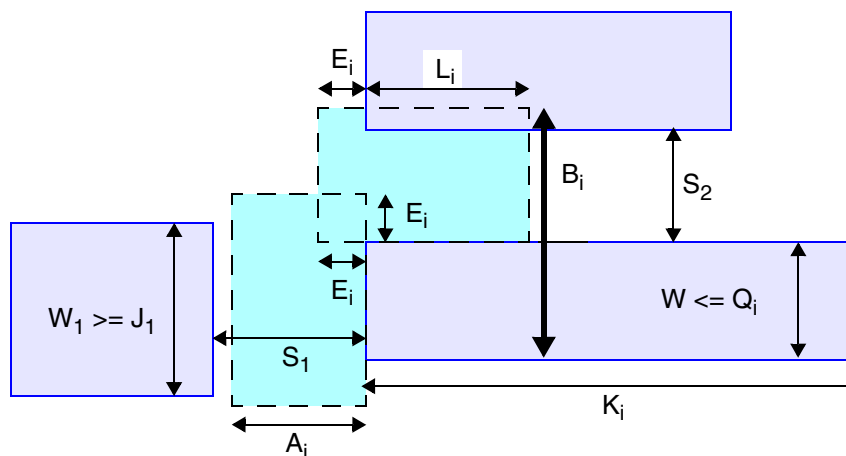
```
Layer "MetalX"
  uShapeKeepoutMaxEdgeLengthThreshold = E
  uShapeKeepoutDepth                  = S
  uShapeKeepoutEndOfLineMaxWidthThreshold = Q
  uShapeKeepoutEndOfLineMinLength       = L # optional
}
```

Two-Neighbor End-of-Line Spacing Rule Enhancements

The two-neighbor end-of-line spacing rule specifies the minimum distance between the end of a narrow metal line and a wider neighboring metal line, and also from the side of the same narrow metal line to another neighboring metal. In the “Mod1” modified version of this rule, the side spacing requirement is measured from the far edge of the metal line. The basic rule and “Mod1” modified version are described in [“Two-Neighbor End-of-Line Spacing Rule” on page 2-68](#).

For advanced geometries, the rule is enhanced to allow table-based specification of the rule attributes. [Figure 3-72](#) shows the rule measurements and following the figure is the enhanced syntax for the rule.

Figure 3-72 Enhanced Modified Two-Neighbor End-of-Line Spacing Rule



This is the enhanced, table-based syntax for the rule:

```

Layer "Mx" {
  endOfLine2NeighborMod1TblSize                = N
  endOfLine2NeighborMod1ThresholdTbl            = (Q1,...,QN)
  endOfLine2NeighborMod1WireMinThresholdTbl     = (J1,...,JN) # optional
  endOfLine2NeighborMod1MinSpacingTbl           = (A1,...,AN)
  endOfLine2NeighborMod1CornerKeepoutWidthTbl   = (E1,...,EN)
  endOfLine2NeighborMod1SideKeepoutLengthTbl    = (L1,...,LN)
  endOfLine2NeighborMod1SideKeepoutWidthTbl     = (B1,...,BN)
  endOfLine2NeighborMod1MinLengthTbl            = (K1,...,KN) # optional
}

```

If $W \leq Q_i$ and $W_1 \geq J_i$, and two neighboring metal geometries are detected within each of the two dashed boxes, then at least one of the following minimum spacing requirements must be met:

$$S_1 \geq A_i$$

$$(S_2 + W) \geq B_i$$

Here is an example of the rule:

```

Layer "Mx" {
  endOfLine2NeighborMod1TblSize                = 2
  endOfLine2NeighborMod1ThresholdTbl           = (0.049, 0.062)
  endOfLine2NeighborMod1WireMinThresholdTbl    # not used
  endOfLine2NeighborMod1MinSpacingTbl          = (0.080, 0.068)
  endOfLine2NeighborMod1CornerKeepoutWidthTbl  = (0.026, 0.026)
  endOfLine2NeighborMod1SideKeepoutLengthTbl   = (0.063, 0.063)
  endOfLine2NeighborMod1SideKeepoutWidthTbl    = (0.106, 0.106)
  endOfLine2NeighborMod1MinLengthTbl           # not used
}

```

Via Spacing Rules

Several advanced rules specify the minimum spacing between vias:

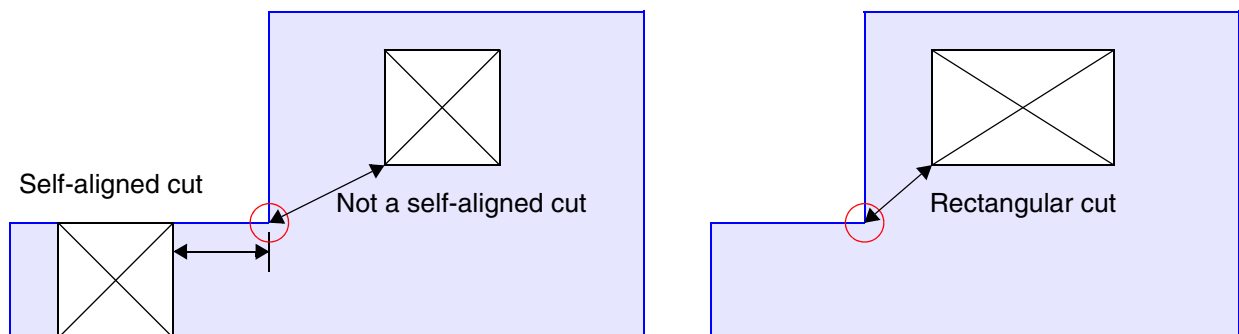
- [Via to Concave Metal Corner Spacing Rule](#)
- [Maximum Number of Unaligned Vias Rule](#)
- [Non-Self-Aligned Via Edge to Metal Spacing Rule](#)
- [Adjacent Via Spacing Rule](#)
- [Constrained Via Spacing Rule](#)
- [Line Via Spacing Rule](#)
- [Edge Via Spacing Rule](#)
- [Matrix Via Spacing Rule](#)
- [Via Corner Spacing Rule Enhancement](#)
- [Interlayer Cut-to-Metal Spacing Rule](#)
- [Non-Self-Aligned Interlayer Via Spacing Rule](#)
- [Via Forbidden Spacing Rule](#)
- [Separated Via Spacing Rule](#)

- [Via Corner Spacing at Zero Projection Rule](#)
- [Mixed Edge-to-Edge and Center-to-Center Via Spacing Rule](#)

Via to Concave Metal Corner Spacing Rule

The via to concave metal corner spacing rule specifies the minimum distance between a via and a concave corner in the upper metal covering the via. Different spacing values can apply to self-aligned vias, not-self-aligned vias, and rectangular cuts, as shown in [Figure 3-73](#).

Figure 3-73 Via to Concave Metal Corner Spacing Rule



The following example shows the syntax of the rule:

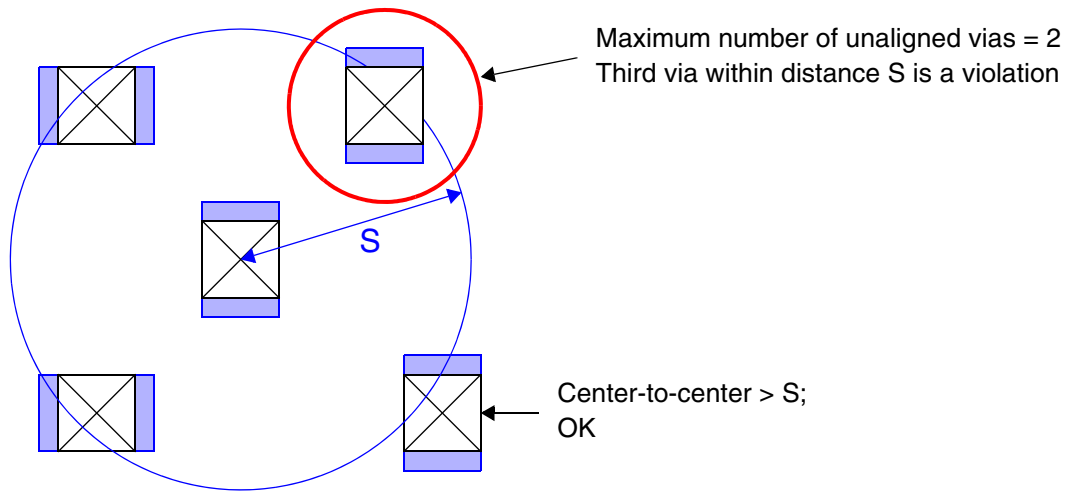
```
DesignRule {
  layer1                = "METAL2"
  layer2                = "VIA1"
  minSpacing            = 0
  concaveMetalToSelfAlignedCutMinDist = 0.036
  concaveMetalToNonSelfAlignedCutMinDist = 0.034
  concaveMetalToRectCutMinDist = 0.025
  concaveMetalToViaArrayIncluded = 1
}
```

In this example, the minimum spacing between a concave upper-metal corner and a cut is 0.036 for a self-aligned via, 0.034 for a not-self-aligned via, or 0.025 for a rectangular cut. The `concaveMetalToViaArrayIncluded` attribute is set to 1, so the rule applies to vias in a via array.

Maximum Number of Unaligned Vias Rule

When self-aligned vias are unaligned (as described in [“Self-Aligned Via Spacing Rules” on page 3-12](#)), a limit might apply to the number of nearby self-aligned vias within a given distance. This number can be specified as a rule. In [Figure 3-74](#), no more than two self-aligned vias are allowed within a distance S , measured center-to-center.

Figure 3-74 Maximum Number of Unaligned Self-Aligned Vias Rule



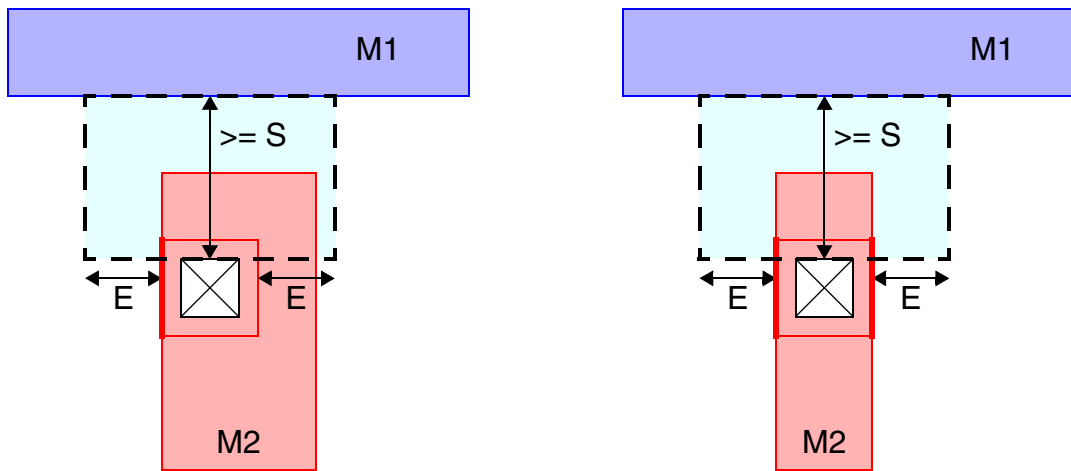
The following example shows the syntax of the rule:

```
DesignRule {
    layer1                = "VIA1"
    layer2                = "VIA1"
    cut1TblSize           = 1
    cut2TblSize           = 1
    cut1NameTbl           = (Vsm)
    cut2NameTbl           = (Vsm)
    unalignedViaMaxNumCutsTbl = (2)
    unalignedViaCenterRangeTbl = (0.154)
}
```

In this example, the table size is 1. A maximum of two unaligned self-aligned vias is allowed within a distance of 0.154, measured center-to-center. Two vias are unaligned if there is no parallel overlap between them; they are not partially or fully aligned.

Non-Self-Aligned Via Edge to Metal Spacing Rule

The non-self-aligned via edge to metal spacing rule specifies the minimum spacing S between a non-self-aligned edge of a cut and a lower or upper metal edge not belonging to the same net as the cut. The rule applies to any edge of a via that is not a self-aligned edge, as defined in [“Self-Aligned Via Rules” on page 3-9](#). The spacing check is performed for the length of the via edge plus an extension E on both sides of the edge, as indicated in [Figure 3-75](#).

Figure 3-75 Non-Self-Aligned Via Edge to Metal Rule

The following example shows the syntax of the rule:

```
DesignRule {
  layer1 = "VIA1"
  layer2 = "MET1"
  metalToNonSelfAlignedEdgeExtension      = 0.42; extension E
  metalToNonSelfAlignedEdgeCutMinSpacing = 0.50; minimum spacing S
}
```

The spacing from the non-self-aligned via edge to the unconnected lower or upper metal must be at least S for the length of that edge plus the extension E beyond the ends of the edge.

Adjacent Via Spacing Rule

The adjacent via spacing rule specifies that no more than three vias can be lined up diagonally when the spacing between them is less than a threshold D . The rule also prohibits more than one diagonally adjacent via with respect to any one via edge, at a distance below the threshold D . [Figure 3-76](#) shows examples of via placement allowed by the rule, while [Figure 3-77](#) shows via configurations disallowed by the rule.

Figure 3-76 Adjacent Via Spacing Rule Allowed Via Placement

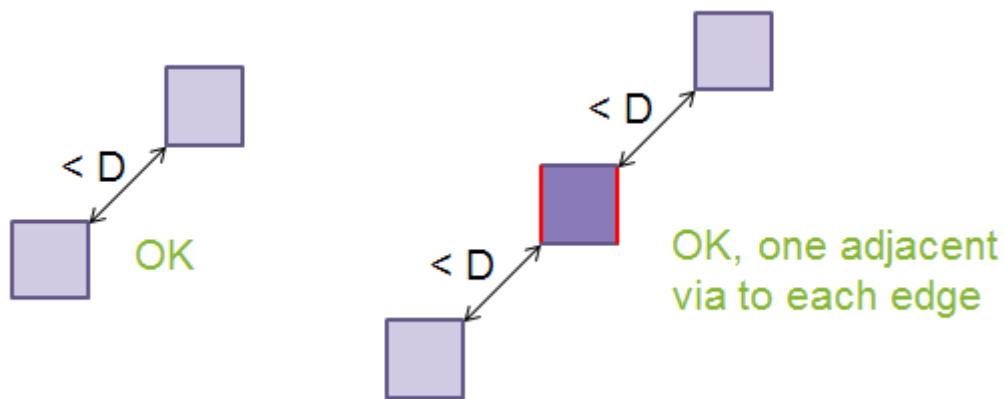
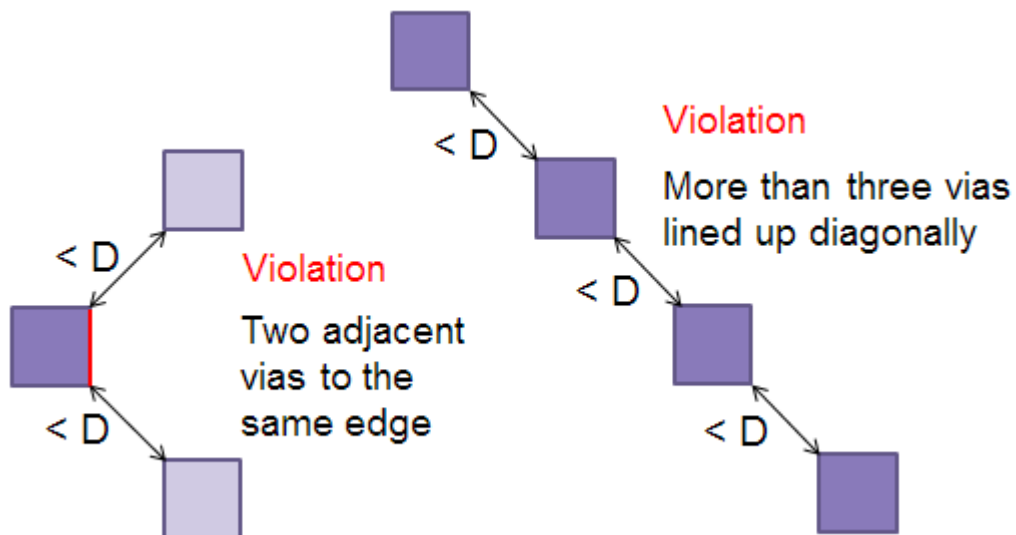


Figure 3-77 Adjacent Via Spacing Rule Disallowed Via Placement



This is the syntax of the rule:

```

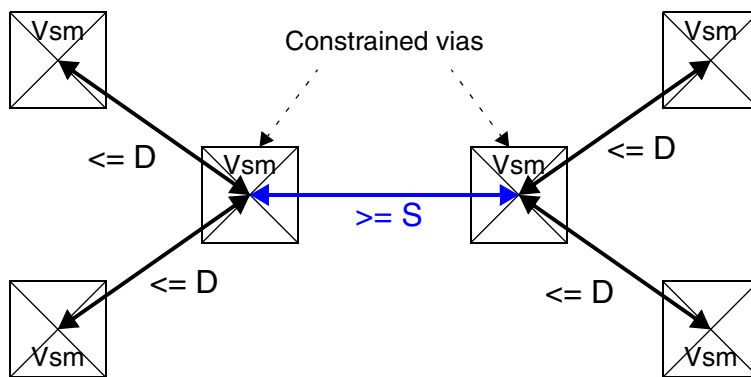
Layer Vx {
    diagAdjacentCutTblSize           = 1
    diagAdjacentCutNameTbl          = (Vs)
    diagAdjacentCutMinSpacingTbl     = (D)
}

```

Constrained Via Spacing Rule

The constrained via spacing rule specifies the minimum spacing between vias when a specified number of other vias of the same type are nearby. A *constrained via* is any via that has multiple neighboring vias of the same cut type no more than a distance D away, measured center-to-center. Any two constrained vias must be separated by a distance of at least S , also measured center-to-center, as shown in [Figure 3-78](#).

Figure 3-78 Constrained Via Spacing Rule



The following example shows the syntax of the rule:

```

Layer "VX" {
    constrainedCutTblSize           = 3
    constrainedCutNameTbl          = (Vsm,Vh,Vv)
    constrainedCutConstrainingRangeTbl = (0.136,0.148,0.148) # D values
    constrainedCutConstrainingNumCutsTbl = (2,2,2)
    constrainedCutCenterMinSpacingTbl = (0.138,0.150,0.150) # S values
}

```

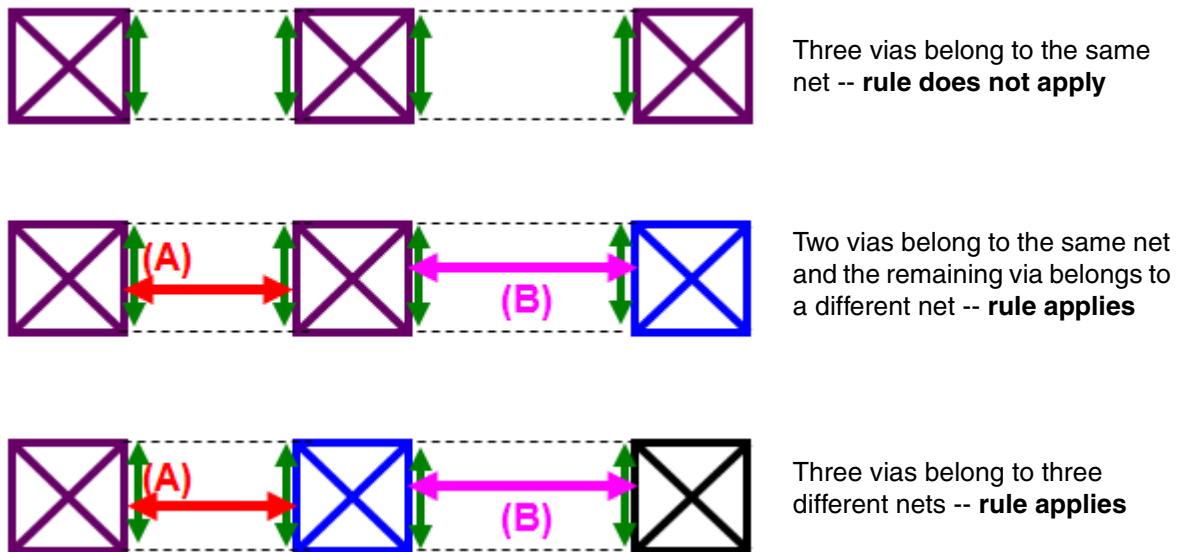
In this example, for a Vsm cut, if there are two or more nearby Vsm vias within a distance of 0.136, then the via is a constrained via, which must be a distance of at least 0.138 from any other constrained Vsm via. All via-to-via distances are measured center-to-center.

Line Via Spacing Rule

The line via spacing rule specifies the minimum spacing between vias when three or more vias are positioned in a line. In this situation, the vias are called line vias. This is a table-based rule that specifies different minimum spacing values that depend on the distances between adjacent vias on both sides of a given via. The rule is less conservative than the constrained via spacing rule because it allows the minimum spacing between two vias to be reduced when the via on the opposite side is not too close.

Figure 3-79 shows some examples of three vias positioned in a line. The vias share a parallel overlap greater than zero, as indicated by the green arrows. The minimum spacing requirement “B” between the two vias depends on the distance “A” to the via on the other side.

Figure 3-79 Line Via Spacing Rule



The rule applies to the case where any two vias belong the same net and the third via belongs to a different net, or when all three vias belong to different nets. The rule does not apply when all three vias belong to the same net.

This is the syntax of the line via spacing rule:

```
Layer "V2" {
  lineCutTblSize      = 3
  lineCutSpacingTbl   = ( $\beta_1$ ,  $\beta_2$ ,  $\beta_3$ )
}
```

The n monotonically increasing values β_1 through β_n in the table specify both the distance thresholds “A” between the vias and the minimum spacing requirement “B” on the other side, but in reverse order. For example, in the case of $n=3$, the three table values specify “A” and “B” as indicated in Table 3-1.

Table 3-1 Line Via Spacing Rule Values

Spacing to nearby via "A"	Minimum spacing requirement "B" on other side
$\beta_1 \leq A < \beta_2$	$B \geq \beta_3$
$\beta_2 \leq A < \beta_3$	$B \geq \beta_2$
$\beta_3 \leq A$	$B \geq \beta_1$

Figure 3-80 demonstrates usage of the rule. The via on the right belongs to a net different from that of the other two vias, so the rule applies. The distance "A" between the left and middle vias is exactly the minimum value in the table, β_1 . This means that the minimum spacing requirement "B" on the other side is β_3 , the maximum table value, which defines the blue exclusion zone shown. This example shows a spacing violation.

Figure 3-80 Line Via Spacing Rule Example 1

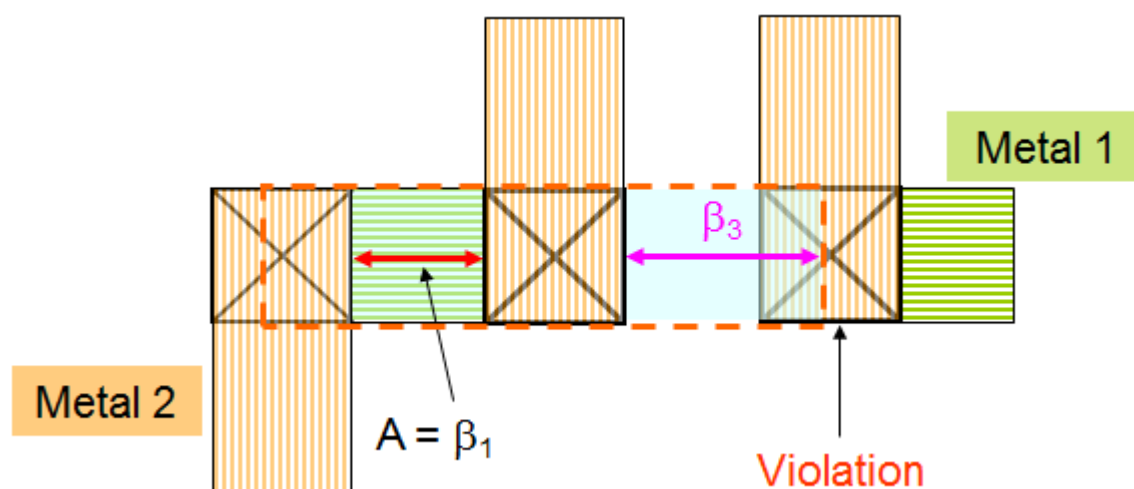
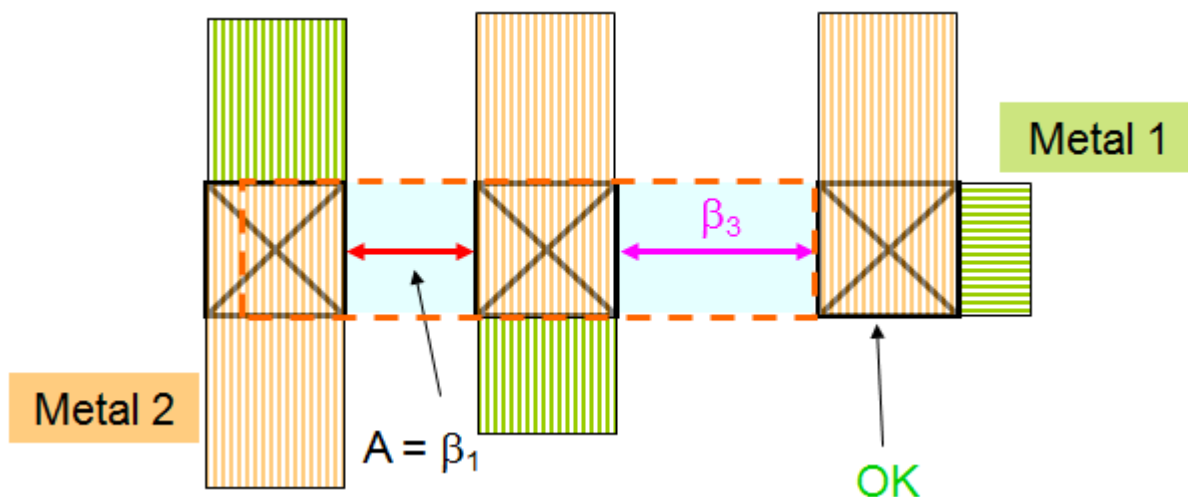


Figure 3-81 provides another example of rule usage. All three vias belong to different nets, so the rule applies. Again, the distance "A" between the left and middle vias is exactly the minimum value in the table, β_1 , so the minimum spacing requirement "B" on the other side is β_3 , the maximum table value. In this example, the spacing requirement is met.

Figure 3-81 Line Via Spacing Rule Example 2

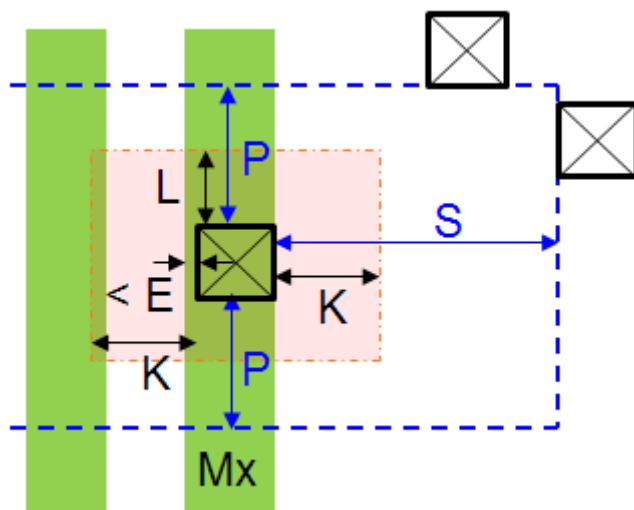


Edge Via Spacing Rule

An “edge via” is a via that meets both of the following conditions, as shown in [Figure 3-82](#):

- The upper or lower metal enclosure is less than E on two opposite sides of the via.
- Within a region extending from the edges of the via by a distance K in the direction of the measurement E and by a distance L in the orthogonal direction, one or more metal shapes exist on one side of the via, and no metal shapes exist on the other side.

Figure 3-82 Edge Via Spacing Rule



The via in the green metal route is an edge via because the upper metal enclosure around it is less than E on the left and right sides of the via, and within the pink search area, metal is present on the left side but not the right side.

The rule specifies that no neighboring via may exist in a region extending from the edges of the edge via for a distance S in the direction of measurement E and a distance P in the orthogonal direction.

This is the syntax of the rule:

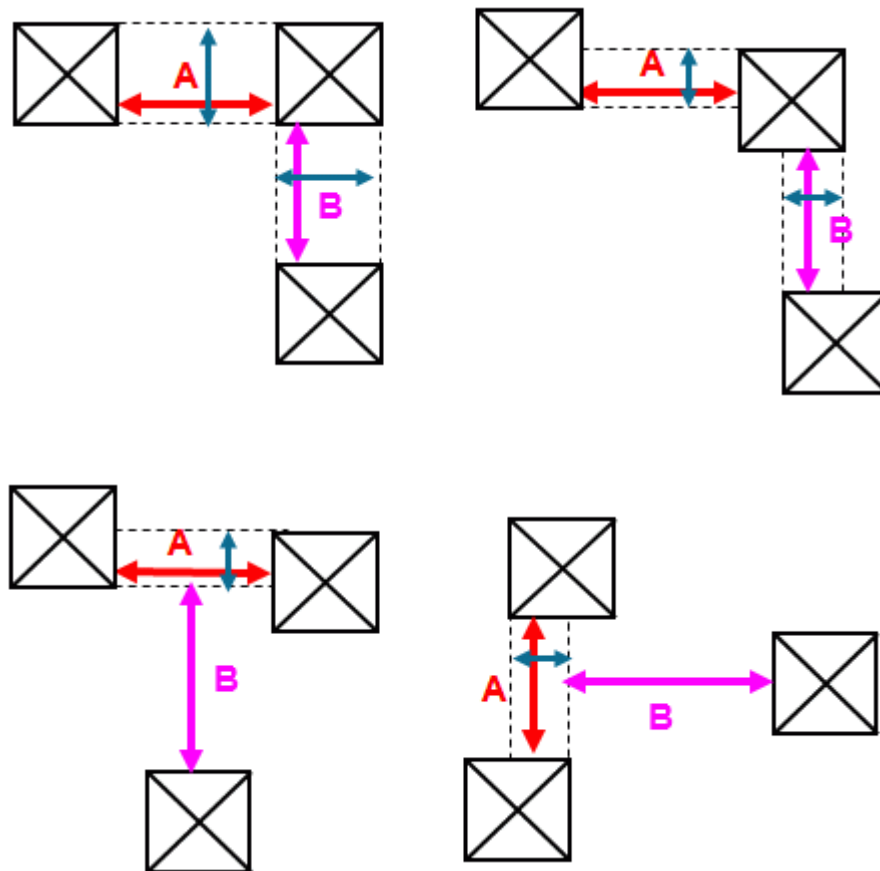
```
DesignRule {
    layer1           = "Mx"
    layer2           = "VIAx"
    edgeViaCutTblSize = 3
    edgeViaCutNameTbl = (Vs, Vrh, Vrv)
    edgeViaWireParallelLengthTbl = (Ls, Lh, Lv)
    edgeViaWireMinSpacingTbl = (Ks, Kh, Kv)
    edgeViaCutMinEnclosureTbl = (Es, Eh, Ev)
    edgeViaCutParallelLengthTbl = (Ps, Ph, Pv)
    edgeViaCutMinSpacingTbl = (Ss, Sh, Sv)
}
```

Matrix Via Spacing Rule

The matrix via spacing rule specifies the minimum spacing between two vias when a third via is located nearby in a direction orthogonal to a line joining the two vias. In this situation, the two vias are called matrix vias. This is a table-based rule that specifies different minimum spacing values that depend on the distance to the third nearby via. The rule is less conservative than the constrained via spacing rule because it allows the minimum spacing to be reduced when the third nearby via is not too close.

[Figure 3-83](#) shows several examples of three vias in a matrix configuration. In each case, two vias share a parallel overlap greater than zero, as indicated by the blue arrows. The minimum spacing “A” between the two matrix vias depends on the distance “B” to the third nearby via.

Figure 3-83 Matrix Via Spacing Rule



This is the syntax of the matrix via spacing rule:

```
Layer "V2" {
  matrixCutTblSize      = 5
  matrixCutSpacingTbl   = ( $\alpha_1$ ,  $\alpha_2$ ,  $\alpha_3$ ,  $\alpha_4$ ,  $\alpha_5$ )
}
```

The n monotonically increasing values α_1 through α_n in the table specify both the distance thresholds “A” and the minimum spacing values “B”, but in reverse order. For example, in the case of $n=5$, the five table values specify “A” and “B” as indicated in [Table 3-2](#).

Table 3-2 Matrix Via Spacing Rule Values

Spacing to nearby via “A”	Minimum spacing requirement “B”
$\alpha1 \leq A < \alpha2$	$B \geq \alpha5$
$\alpha2 \leq A < \alpha3$	$B \geq \alpha4$
$\alpha3 \leq A < \alpha4$	$B \geq \alpha3$
$\alpha4 \leq A < \alpha5$	$B \geq \alpha2$
$\alpha5 \leq A$	$B \geq \alpha1$

[Figure 3-84](#) demonstrates usage of the rule. In Example 1, the distance “A” between the vias is exactly the minimum value in the table, $\alpha1$. This means that the minimum spacing requirement “B” in the orthogonal direction is $\alpha5$, the maximum table value, which defines the blue exclusion zone shown. Conversely, in Example 2, the distance “A” is $\alpha5$, which causes the minimum spacing requirement to be $\alpha1$.

[Figure 3-85](#) provides more examples of rule usage, where the two vias are not aligned, but share a nonzero parallel overlap. The exclusion zone is measured from the edges of the parallel-overlap region.

Figure 3-84 Matrix Via Spacing Rule Examples 1 and 2

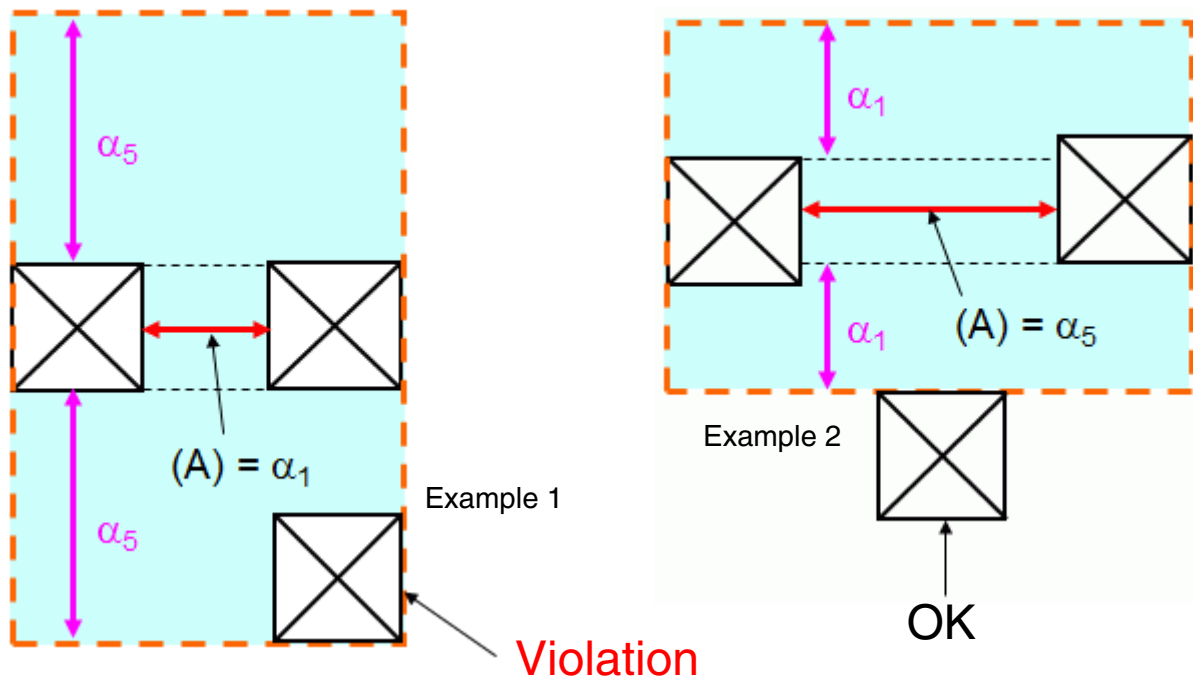
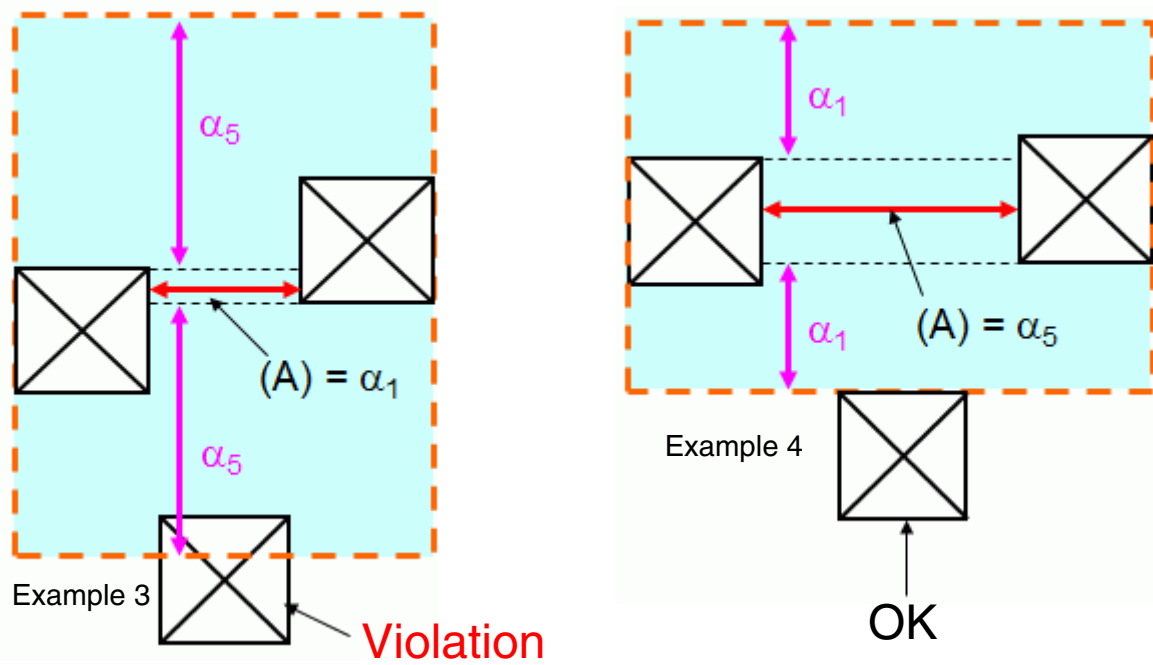


Figure 3-85 Matrix Via Spacing Rule Examples 3 and 4



Via Corner Spacing Rule Enhancement

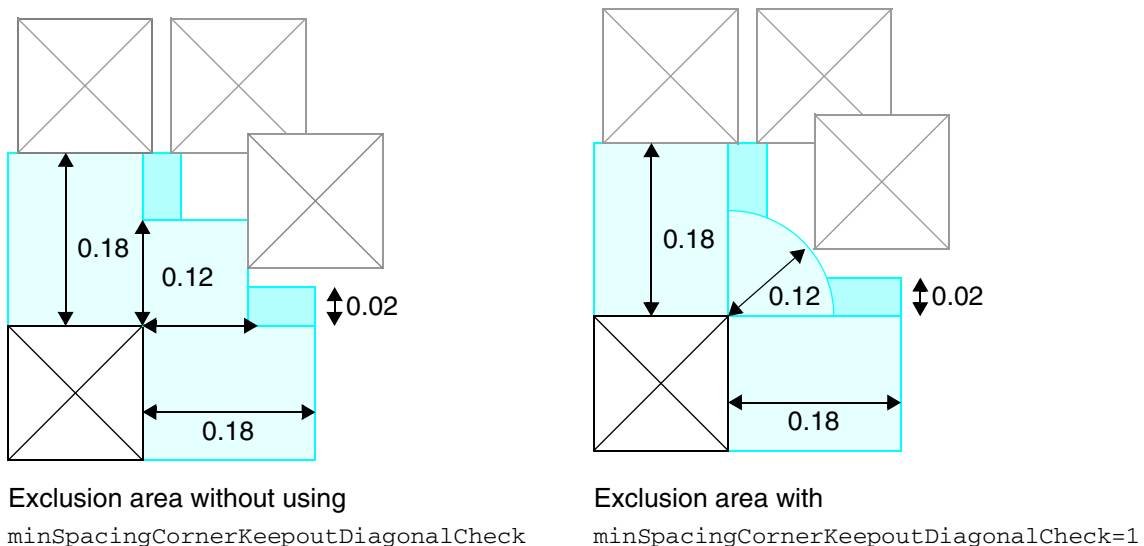
The via corner spacing rule specifies the minimum corner-to-corner spacing allowed between two vias. This rule applies only when the parallel spacing between the two vias is less than a given threshold. The basic rule is described in [“Via Corner Spacing Rule” on page 2-29](#).

By default, the minimum spacing requirement is a Manhattan check. For advanced geometries, the rule is enhanced with an additional attribute that lets you specify the minimum spacing measured diagonally instead. For example,

```
Layer "VIA1" {
    minSpacing                = 0.18
    cornerMinSpacing          = 0.12
    minSpacingCornerKeepoutWidth = -0.02
    minSpacingCornerKeepoutDiagonalCheck = 1
}
```

If the `minSpacingCornerKeepoutDiagonalCheck` attribute is omitted or set to 0, the rule is a Manhattan spacing check; or if it is set to 1, the rule is a diagonal check, as shown in [Figure 3-86](#).

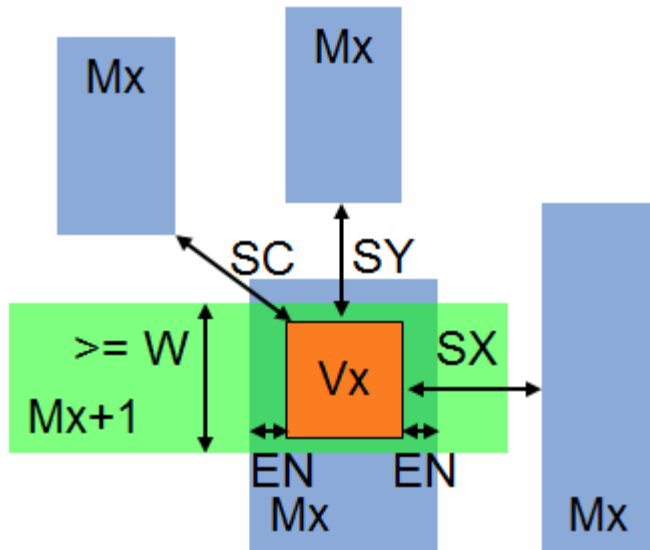
Figure 3-86 Via Corner Spacing Rule



Interlayer Cut-to-Metal Spacing Rule

The interlayer cut-to-metal spacing rule specifies the minimum required orthogonal spacing SX and SY and corner spacing SC between via Vx and lower-metal Mx of different nets, when the lower-layer metal (Mx) enclosure of Vx is no more than a threshold EN , and the upper-layer metal ($Mx+1$) has a width no less than a threshold W . See [Figure 3-87](#).

Figure 3-87 Interlayer Cut-to-Metal Spacing Rule



This is the syntax of the rule:

```
DesignRule {
    layer1                = "Mx"
    layer2                = "Vx"
    cutTblSize            = 1
    cutNameTbl            = (Vsm)
    diffNetXMinSpacingTbl = (SX)
    diffNetYMinSpacingTbl = (SY)
    diffNetCornerMinSpacingTbl = (SC)
    minSpacingUpperMetalWidthThresholdTbl = (W)
    minSpacingLowerEncMaxThresholdTbl = (EN)
    minSpacingUpperEncCheckTbl = (0)
}
```

For example,

```
DesignRule {
    layer1                = "M1"
    layer2                = "VIA1"
    cutTblSize            = 3
    cutNameTbl            = (Vs, Vrh, Vrv)
    diffNetXMinSpacingTbl = (0.044, 0.044, 0.044)
```

```

diffNetYMinSpacingTbl           = (0.044, 0.044, 0.044)
diffNetCornerMinSpacingTbl      = (0.044, 0.044, 0.044)
minSpacingUpperMetalWidthThresholdTbl = (0.047, 0.047, 0.047)
minSpacingLowerEncMaxThresholdTbl = (0.003, 0.003, 0.003)
minSpacingUpperEncCheckTbl      = (0, 0, 0)
}

```

In this example, the table size is set to 3, so the three values for each attribute apply to the three respective single cuts named Vs, Vrh, and Vrv.

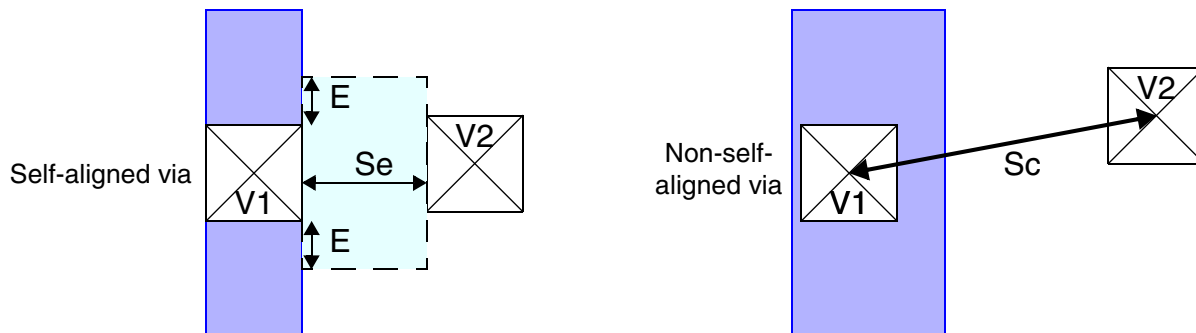
When the `minSpacingUpperEncCheckTbl` attribute is set to 1, the rule requires the upper-metal enclosure to be at least zero. Otherwise, there is no upper-metal enclosure requirement.

Non-Self-Aligned Interlayer Via Spacing Rule

The non-self-aligned interlayer spacing rule specifies the minimum center-to-center spacing between named vias on adjacent layers belonging to different nets, for example, between vias in the V1 and V2 layers. The applicable rule depends on whether the lower-level via is a self-aligned via, as defined in [“Self-Aligned Via Rules” on page 3-9](#).

In [Figure 3-88](#), vias V1 and V2 are on adjacent via layers and belong to different nets. If V1 is self-aligned as shown on the left, a minimum edge-to-edge minimum spacing Se applies within an extension range E . On the other hand, if V1 is not a self-aligned via as shown on the right, a more restrictive center-to-center minimum spacing Sc applies.

Figure 3-88 Non-Self-Aligned Interlayer Via Spacing Rule



The rule is specified in the `DesignRule` section of the technology file using the following syntax:

```
DesignRule {
  layer1          = "V1 "
  layer2          = "V2 "
  cut1TblSize     = 1
  cut2TblSize     = 1
  cut1NameTbl     = (VNAME1)
  cut2NameTbl     = (VNAME2)
  minSpacingYParallelLengthThresholdTbl = (-E)
  diffNetXMinSpacingTbl = (Se)
  minSpacingXParallelLengthThresholdTbl = (-E)
  diffNetYMinSpacingTbl = (Se)
  diffNetNonSavCenterMinSpacingTbl      = (Sc)
}
```

For example,

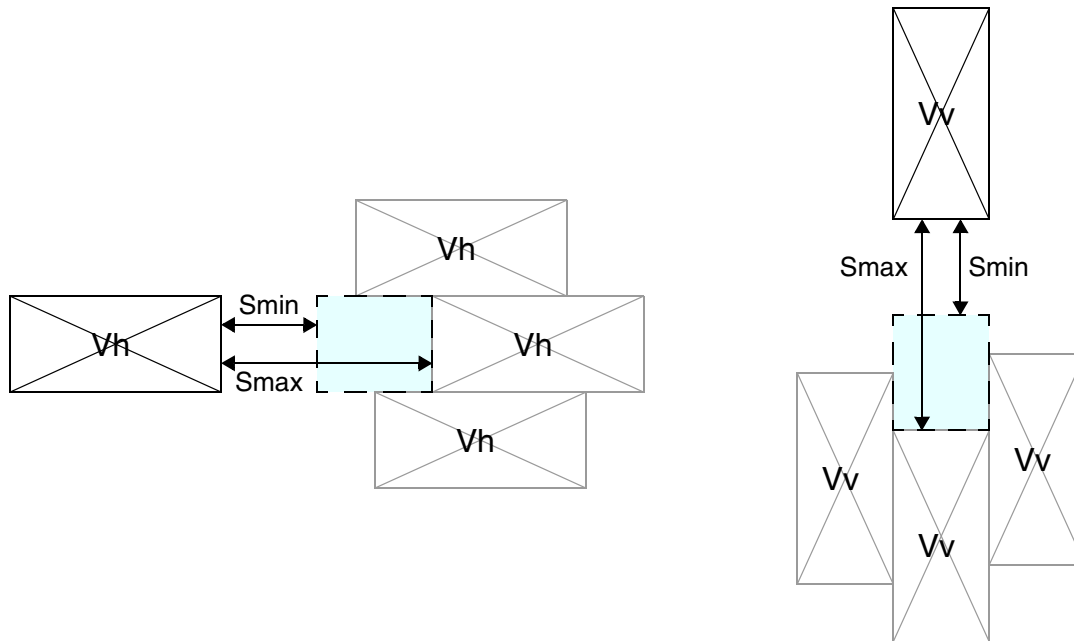
```
DesignRule {
  layer1          = "VIA1 "
  layer2          = "VIA2 "
  cut1TblSize     = 1
  cut2TblSize     = 1
  cut1NameTbl     = (V1LX)
  cut2NameTbl     = (V2LX)
  minSpacingYParallelLengthThresholdTbl = (-0.022)
  diffNetXMinSpacingTbl = ( 0.045)
  minSpacingXParallelLengthThresholdTbl = (-0.022)
  diffNetYMinSpacingTbl = ( 0.045)
  diffNetNonSavCenterMinSpacingTbl      = ( 0.103)
}
```

Via Forbidden Spacing Rule

The via forbidden spacing rule specifies that the spacing between two vias of specified types in a particular direction cannot be within a specified range, from S_{min} to S_{max} , whenever the parallel overlap between the vias is greater than zero. A table of values specifies the minimum and maximum spacing between vias of specified names in the x-direction and y-direction.

In the example shown in [Figure 3-89](#), the rule specifies the spacing between Vh vias and other Vh vias in the x-direction. Relative to the Vh via on the left, no other Vh via can occupy the space from S_{min} to S_{max} from the edge of the via, as long as the parallel overlap is greater than zero. The colored rectangle shows the forbidden area.

Figure 3-89 Via Forbidden Spacing Rule



Similarly, the rule specifies the spacing between Vv vias and other Vv vias in the y-direction. Relative to the Vv via at the top, no other Vv via can occupy the space from Smin to Smax from the edge of the via, as long as the parallel overlap is greater than zero.

This is the syntax for the rule:

```
DesignRule {
    layer1                = "Vx"
    layer2                = "Vx"
    forbiddenCut1TblSize  = 2
    forbiddenCut1NameTbl  = (Vh, Vv)
    forbiddenCut2TblSize  = 2
    forbiddenCut2NameTbl  = (Vh, Vv)
    forbiddenCutXSpacingRangeTbl = ("SminX, SmaxX", -1, -1, -1)
    forbiddenCutYSpacingRangeTbl = (-1, -1, -1, "SminY, SmaxY")
}
```

This syntax example specifies the forbidden spacing ranges between Vh and Vv vias. For Vh-to-Vh spacing in the x-direction, spacing between SminX and SmaxX is not allowed; and for Vv-to-Vv spacing in the y-direction, spacing between SminY and SmaxY is not allowed. The value -1 means that the rule does not apply to a particular combination of via types. For example, the first -1 above indicates that there is no x-spacing requirement between Vh and Vv vias.

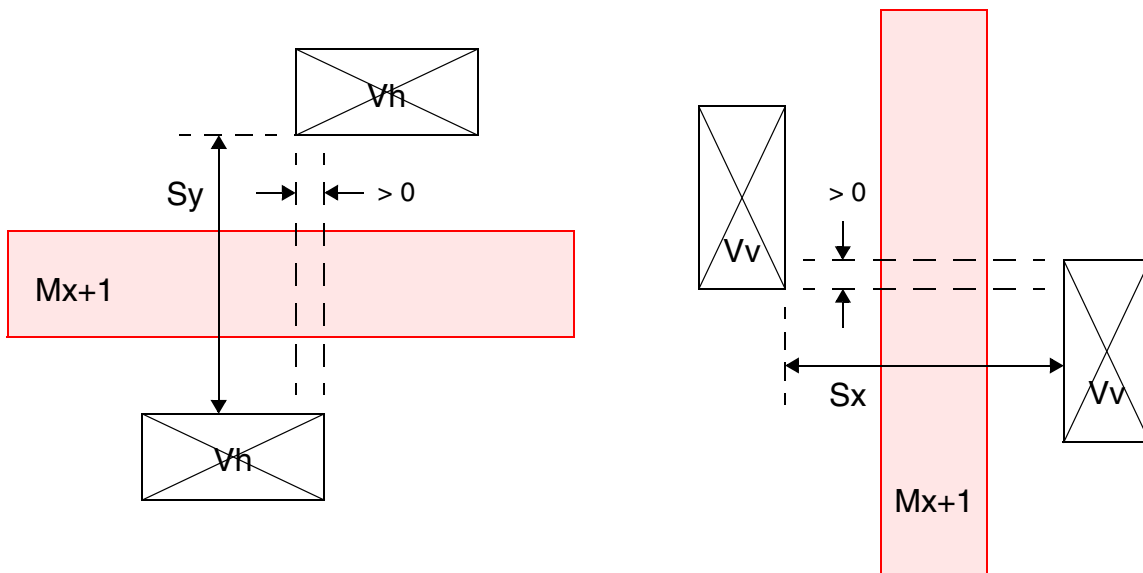
Here is an example of the rule:

```
DesignRule {
  layer1                = "V1"
  layer2                = "V1"
  forbiddenCut1TblSize  = 2
  forbiddenCut1NameTbl  = (Vh, Vv)
  forbiddenCut2TblSize  = 2
  forbiddenCut2NameTbl  = (Vh, Vv)
  forbiddenCutXSpacingRangeTbl = ("0.095, 0.112", -1, -1, -1)
  forbiddenCutYSpacingRangeTbl = (-1, -1, -1, "0.095, 0.112")
}
```

Separated Via Spacing Rule

The separated via spacing rule specifies a minimum required spacing S in a given direction between two specified types of via cuts that have a parallel overlap of more than zero and are separated by an unconnected upper-metal shape. The rule applies to the spacing between via cuts of specified names in specified directions. For example, see [Figure 3-90](#).

Figure 3-90 Separated Via Spacing Rule



For example,

```
DesignRule {
  layer1                = "VIA1"
  layer2                = "VIA1"
  separatedCut1TblSize  = 2
  separatedCut1NameTbl  = (Vh, Vv)
  separatedCut2TblSize  = 2
```



```

separatedCut2NameTbl           = (Vh, Vv)
separatedCut1ToUpperMetalExcludedSpacingTbl = (0, 0)
separatedCut2ToUpperMetalExcludedSpacingTbl = (0, 0)
separatedCutXMinSpacingTbl     = (-1, -1, -1, 0.136)
separatedCutYMinSpacingTbl     = (0.132, -1, -1, -1)
}

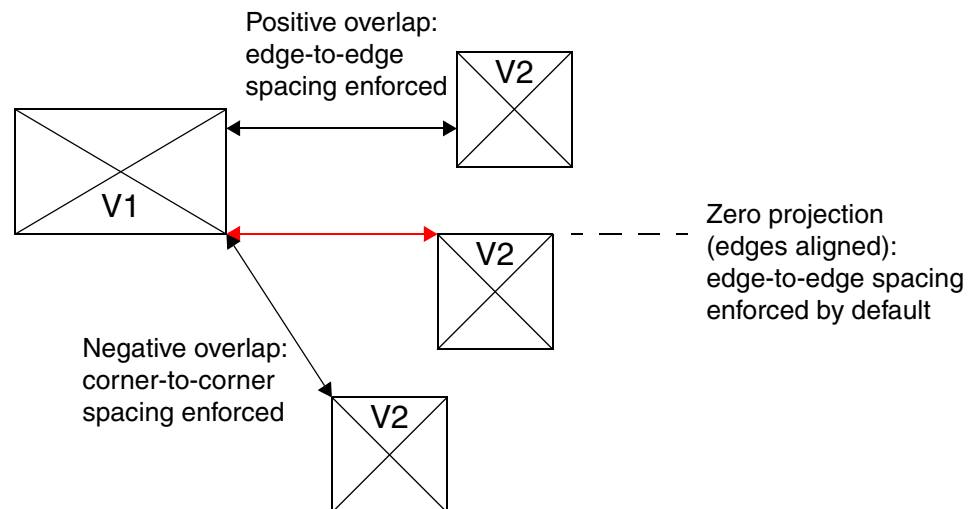
```

This syntax example specifies the minimum spacing between Vh and Vv vias separated by upper-metal. The minimum Vv-to-Vv spacing in the x-direction is $S_x = 0.136$. Similarly, the Vh-to-Vh spacing in the y-direction is $S_y = 0.132$. The value -1 means that the rule does not apply to a particular combination of via types. For example, the first -1 above indicates that there is no x-spacing requirement between Vh and Vh vias.

Via Corner Spacing at Zero Projection Rule

The general cut spacing rule specifies the minimum spacing between vias, as described in [“General Cut Spacing Rule” on page 2-33](#). When the parallel overlap or projection between two vias is positive, the router enforces the orthogonal, edge-to-edge spacing requirement. On the other hand when the parallel overlap or projection is negative, the router enforces the diagonal, corner-to-corner spacing requirement, as shown in [Figure 3-91](#).

Figure 3-91 Via Spacing Rules for Positive, Zero, and Negative Projection



When the parallel overlap or projection is exactly zero, the router enforces the edge-to-edge spacing requirement by default. To modify the default behavior and enforce the corner-to-corner requirement for this situation, use the syntax shown in the following example:

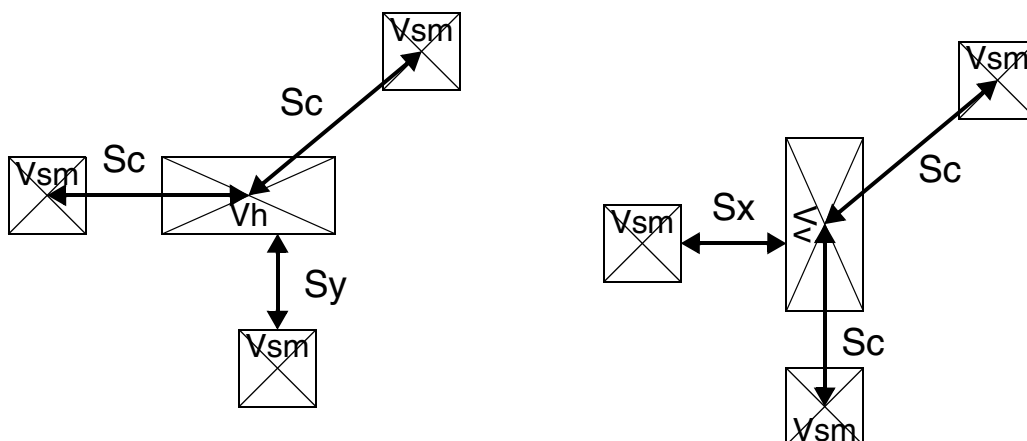
```
DesignRule {
  layer1          = "V1"
  layer2          = "V2"
  cut1TblSize     = 1
  cut2TblSize     = 2
  cut1NameTbl     = (V1NAME)
  cut2NameTbl     = (V2NAME1, V2NAME2)
  diffNetXMinSpacingTbl = (Sxh, Sxv)
  diffNetYMinSpacingTbl = (Syh, Syv)
  diffNetCornerMinSpacingTbl = (Sch, Scv)
  cornerSpacingCheckZeroProjectionTbl = (1, 1)
}
```

A “1” in the `cornerSpacingCheckZeroProjectionTbl` attribute vector causes the corner-to-corner spacing requirement to be enforced for the zero projection case, whereas a “0” causes the edge-to-edge (default) spacing requirement to be enforced for the zero projection case.

Mixed Edge-to-Edge and Center-to-Center Via Spacing Rule

In some cases, the technology might require via spacing checks to be performed center-to-center in one direction and edge-to-edge in another direction. For example, in [Figure 3-92](#), the via spacing check must be performed center-to-center in the diagonal direction and through the shorter side of the rectangular via, but edge-to-edge from the longer side of the rectangular via.

Figure 3-92 Mixed Edge-to-Edge and Center-to-Center Via Spacing Requirements



To specify a via spacing rule as shown in the figure, use the following syntax:

```
DesignRule {
  layer1                = "V0"
  layer2                = "V1"
  cut1TblSize           = 1
  cut2TblSize           = 2
  cut1NameTbl           = (Vsm)
  cut2NameTbl           = (Vh, Vv)
  diffNetXMinSpacingTbl = (Sc, Sx)
  diffNetYMinSpacingTbl = (Sy, Sc)
  diffNetCornerMinSpacingTbl = (Sc, Sc)
  # Center spacing check in X-direction
  xMinSpacingCenterCheckTbl = (1, 0)
  # Center spacing check in Y-direction
  yMinSpacingCenterCheckTbl = (0, 1)
  # Center spacing check corner area
  cornerMinSpacingCenterCheckTbl = (1, 1)
}
```

Via Enclosure Rules

Several advanced rules specify the minimum enclosure of metal around vias:

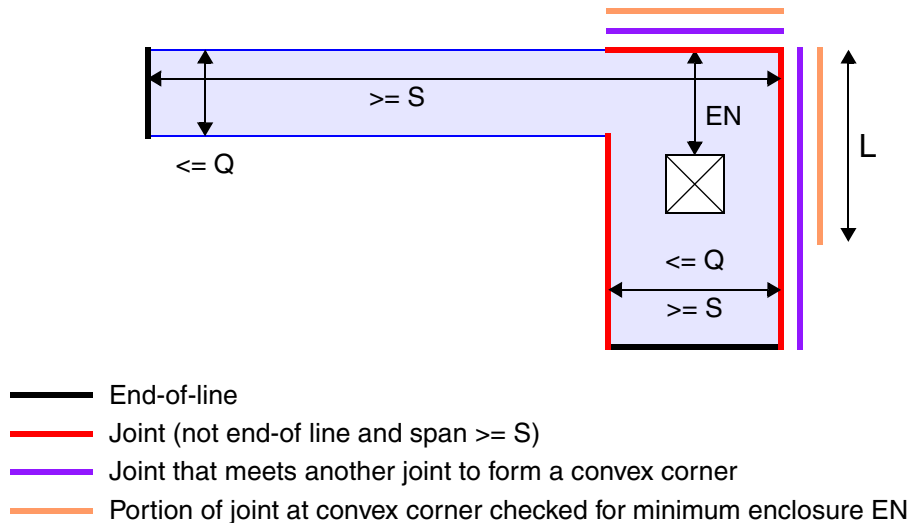
- [Via to Joint Distance Rule](#)
- [Line-End Cut Enclosure Rule](#)
- [Convex Metal Via Keepout Rule](#)
- [Concave-Convex Edge Via Enclosure Rule](#)
- [Fat Wire Via Enclosure Rule Enhancements](#)
- [Fat Metal Contact Asymmetric Enclosure Rule Enhancements](#)
- [Sparse-Dense Via Enclosure Rule](#)
- [Via-to-Jog Metal Enclosure Rule](#)
- [Via to Concave Corners Maximum Limit Rule](#)

Via to Joint Distance Rule

When a via is located near a joint corner, at least one of the enclosures of the metal joint edges around the via must meet a minimum enclosure requirement, as long as the joint meets the maximum wire width, minimum span, and maximum edge length requirements of the rule.

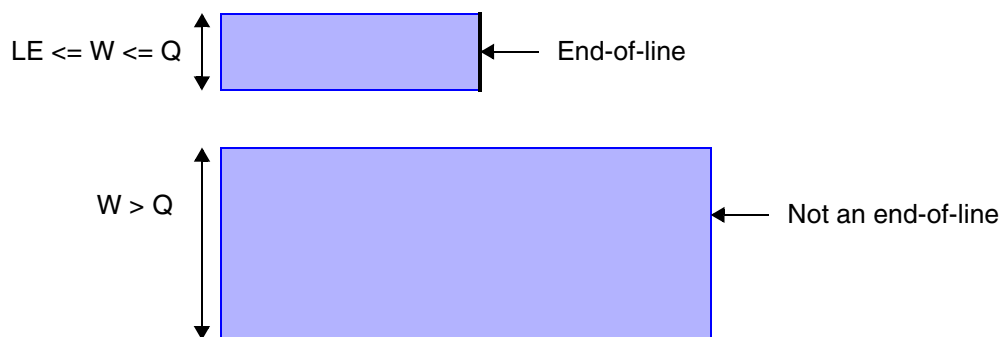
Application of the rule depends on the presence of a nearby metal joint corner and a checking length limit L , as shown in Figure 3-93.

Figure 3-93 Via to Joint Distance Rule



For the via to joint distance rule, an end-of-line is any metal edge connecting two convex corners and having a length no more than a threshold Q . You can optionally specify a minimum length for the edge as well, LE . In Figure 3-94, the narrower line has an end-of-line and the wider wire does not.

Figure 3-94 Definition of an End-of-Line for Joint Distance Rule

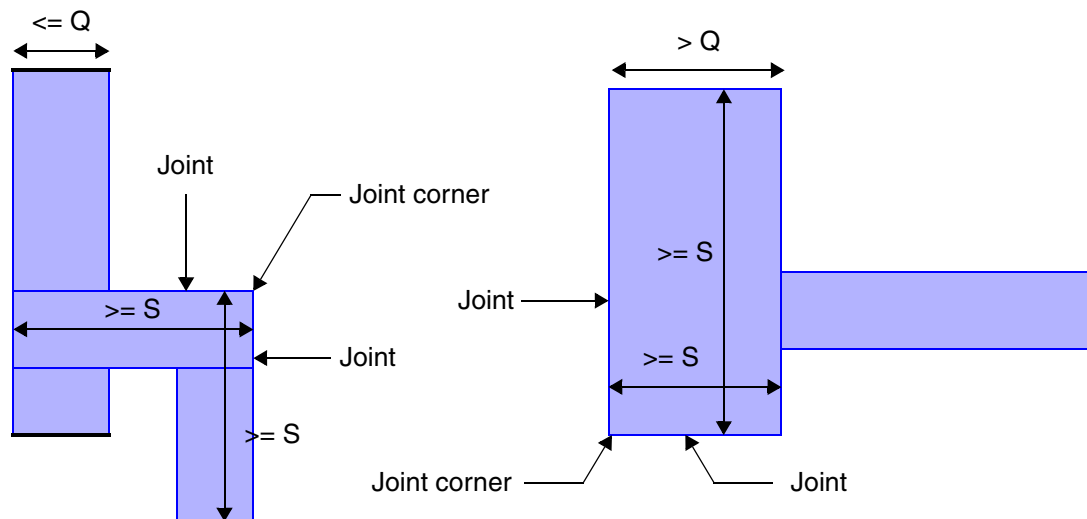


This is the syntax for defining an end-of-line for the via to joint distance rule:

```
DesignRule {
  layer1           = "METAL2"
  layer2           = "VIA1"
  ...
  jointWireViaEndWidthMaxThreshold = Q
  jointWireViaEndMinLength         = LE # optional
  ...
}
```

A joint is any metal edge that is not an end-of-line and has a span of at least S . A joint corner is a convex corner formed by two joints, as shown by the examples in [Figure 3-95](#).

Figure 3-95 Joint Corner Examples

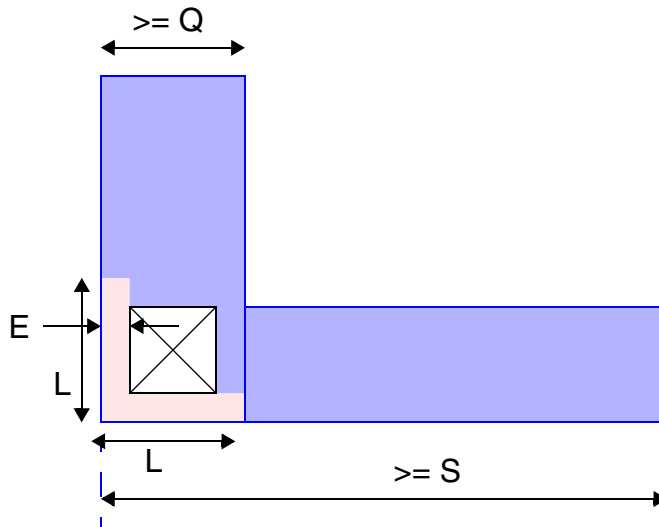


This is the syntax for defining the span S for the definition of a joint:

```
DesignRule {
    layer1                = "METAL2"
    layer2                = "VIA1"
    ...
    jointWireViaMinSpanThreshold = S
    ...
}
```

The minimum enclosure of metal around a via must be at least E for the edge length L from the joint corner along at least one of the two joint edges. Beyond the length L from the corner, the rule does not apply. In [Figure 3-96](#), the rule checks for the minimum enclosure in the pink area.

Figure 3-96 Via to Joint Distance Minimum Enclosure Checking Region



This is the syntax of the rule:

```
DesignRule {
  layer1                      = "METAL2"
  layer2                      = "VIA1"
  jointWireViaCutTblSize     = 1
  jointWireViaCutNameTbl     = (Vsm)
  jointWireViaEndWidthMaxThreshold = Q
  jointWireViaEndMinLength   = LE # optional
  jointWireViaMinSpanThreshold = S
  jointWireViaMaxEdgeLengthThreshold = L
  jointWireViaMinEnclosure    = E
  ...
}
```

The cuts listed in the cut name table, such as V_{sm} in this example, must be defined in the **Layer** section in for the via layer earlier in the technology file.

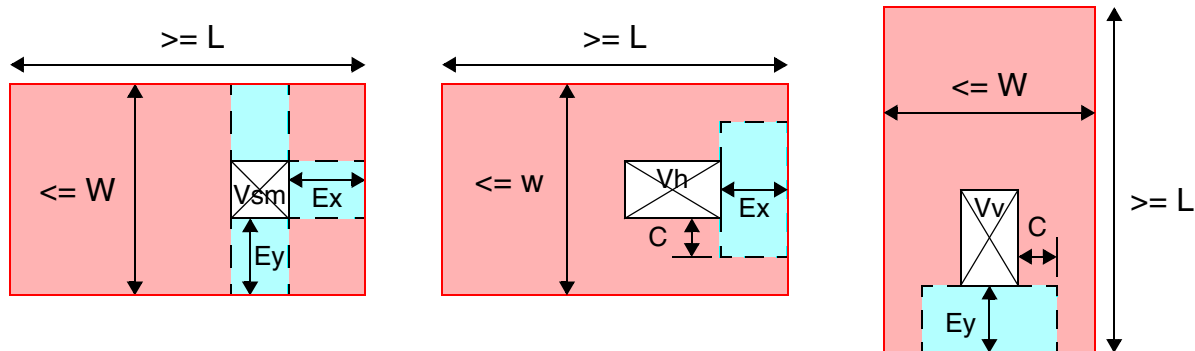
For example,

```
DesignRule {
  ...
  jointWireViaEndWidthMaxThreshold = 0.24
  jointWireViaEndMinLength         = 0.05
  jointWireViaMinSpanThreshold     = 0.08
  jointWireViaMaxEdgeLengthThreshold = 0.15
  jointWireViaMinEnclosure         = 0.04
  ...
}
```

Line-End Cut Enclosure Rule

The line-end cut enclosure rule specifies the minimum amount of upper-metal enclosure around a via located at the end of line of a wire when the wire width is no more than a threshold W , and optionally, the length of the metal wire is at least L . The enclosure must extend beyond the corners of the via by at least an amount C , as shown in Figure 3-97.

Figure 3-97 Line-End Cut Enclosure Rule



This is the syntax of the rule:

```
DesignRule {
  layer1                = "VIA1"
  layer2                = "MET2"
  endOfLineCutTblSize   = 3
  endOfLineCutNameTbl   = (Vn, ...)
  endOfLineCutWireMaxWidthThreshold = W
  endOfLineCutWireMinLength      = L # optional
  endOfLineCutCornerExtensionTbl  = (C,...)
  endOfLineCutXMinEnclosureTbl   = (Ex, ...)
  endOfLineCutYMinEnclosureTbl   = (Ey, ...)
}
```

For example,

```
DesignRule {
  layer1                = "VIA1"
  layer2                = "MET2"
  endOfLineCutTblSize   = 3
  endOfLineCutNameTbl   = (Vsm, Vv, Vh)
  endOfLineCutWireMaxWidthThreshold = 0.13
  endOfLineCutWireMinLength      = 0.20 # optional
  endOfLineCutCornerExtensionTbl  = (0, 0.024, 0.022)
  endOfLineCutXMinEnclosureTbl   = (0.021, -1, 0.033)
  endOfLineCutYMinEnclosureTbl   = (0.021, 0.031, -1)
}
```

In this example, when the upper metal width is no more than 0.13 and length is at least 0.20, the minimum enclosure around a v_{sm} via is 0.021 in the x-direction and y-direction. For a v_v via, the minimum enclosure is 0.031 in the y direction, which applies to an extension of 0.024 beyond the edges of the via; there is no enclosure requirement in the x-direction. Similarly, for a v_h via, the minimum enclosure is 0.033 in the x-direction, which applies to an extension of 0.022 beyond the edges of the via; there is no enclosure requirement in the y-direction.

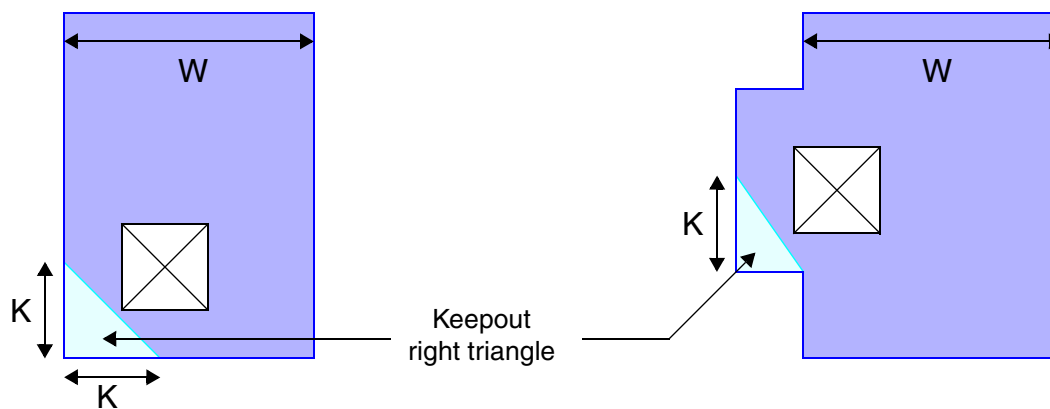
Convex Metal Via Keepout Rule

The convex metal via keepout rule defines a triangular keepout region in each convex metal corner, where a named via is not allowed to overlap. A right triangle is formed by two metal edges, each of which is not a end-of-line edge, meeting at a convex corner.

An end-of-line edge is any edge having a length no more than a maximum width attribute W . You can optionally specify a minimum length L as well.

The legs of the keepout right triangle have a length K , as shown in [Figure 3-98](#). If the triangle leg meets a nearby metal corner within the length K , the leg is shortened to the length of that metal edge, as shown by the example on the right side of the figure.

Figure 3-98 Convex Metal Via Keepout Rule



This is the syntax of the rule:

```
DesignRule {
  layer1                                = "metal1"
  layer2                                = "via1"
  convexMetalCutTblSize                  = N
  convexMetalCutNameTbl                  = (Vsm, ...)
  convexMetalEndOfLineMaxWidthThreshold = W
  convexMetalEndOfLineMinLength          = L # optional
  convexMetalCutKeepoutLengthTbl         = (K1, ...)
}
```

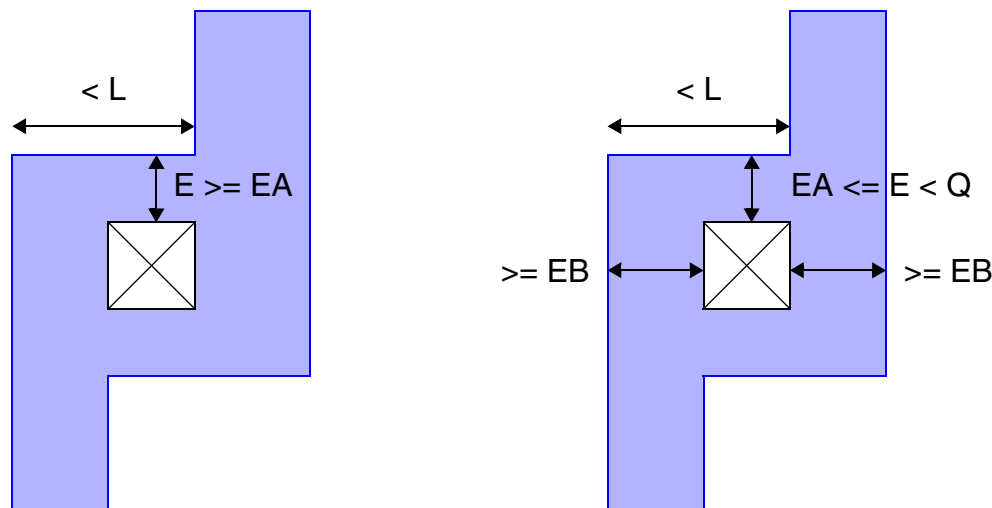

For example, to specify this rule for three named vias, use the following syntax:

```
DesignRule {
  layer1                = "metal1"
  layer2                = "via1"
  convexMetalCutTblSize = 3
  convexMetalCutNameTbl = (Vsm,Vh,Vv)
  convexMetalEndOfLineMaxWidthThreshold = 0.11
  convexMetalEndOfLineMinLength        = 0.03
  convexMetalCutKeepoutLengthTbl       = (0.30,0.32,0.32)
}
```

Concave-Convex Edge Via Enclosure Rule

The concave-convex edge via enclosure rule specifies the minimum enclosure of metal EA around a via when the metal edge has one end at a convex corner and the other edge at a concave corner, and the edge length is less than L, as shown on the left in [Figure 3-99](#).

Figure 3-99 Concave-Convex Edge Via Enclosure Rule



If the enclosure facing that edge is at least EA, but less than a higher threshold Q, then there is an additional requirement that the two enclosures in the orthogonal direction must be at least EB, as shown on the right in [Figure 3-99](#).

This is the syntax of the rule:

```
DesignRule {
    layer1                = Mx+1
    layer2                = Vx
    concaveConvexEdgeCutTblSize      = 3
    concaveConvexEdgeCutNameTbl     = (Vs, Vh, Vv)
    concaveConvexEdgeLengthThresholdTbl = (Ls, Lh, Lv)
    concaveConvexEdgeMinEncTbl      = (EAs, EAh, EAv)
    concaveConvexEdgeMaxEncThresholdTbl = (Qs, Qh, Qv)
    concaveConvexEdgeOrthoMinEncTbl = (EBs, EBh, EBv)
}
```

For example,

```
DesignRule {
    layer1                = "M5"
    layer2                = "VIA4"
    concaveConvexEdgeCutTblSize      = 3
    concaveConvexEdgeCutNameTbl     = (VS, VH, VV)
    concaveConvexEdgeLengthThresholdTbl = (0.20, 0.21, 0.21)
    concaveConvexEdgeMinEncTbl      = (0.04, 0.04, 0.04)
    concaveConvexEdgeMaxEncThresholdTbl = (0.05, 0.05, 0.05)
    concaveConvexEdgeOrthoMinEncTbl = (0.05, 0.06, 0.06)
}
```

Fat Wire Via Enclosure Rule Enhancements

The fat wire via enclosure rule specifies the minimum amount of fat metal overlap over a via for the metal layer either above or below the via. The minimum overlap requirement depends on the width of the fat metal enclosing the via, the spacing to any nearby metal, and the parallel length of that nearby metal, as specified in a table. The basic rule is described in [“Fat Wire Via Enclosure Rule” on page 2-129](#).

For advanced geometries, the rule includes additional attributes that let you do the following:

- Measure the width of the metal containing the via in the direction perpendicular to the parallel run, whether or not this direction is the smaller dimension.
- Specify overlapped width threshold ranges, using one list for lower bounds and another list for upper bounds, rather than consecutive width ranges.
- Specify minimum as well as maximum spacing thresholds.

Measure Metal Width Orthogonal to Parallel Run

By default, in the fat wire via enclosure rule, the width of the metal containing the via is measured in smaller dimension, which could be perpendicular or parallel to the parallel run length. If you want the width to be always measured in the direction perpendicular to the parallel run length, use an additional table attribute as follows:

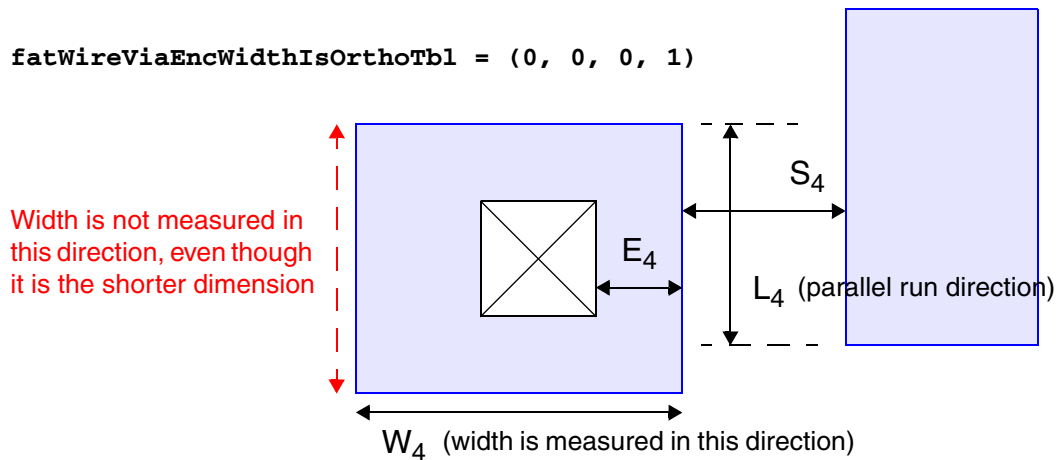
```
DesignRule {
    layer1                = "MetalX"
    layer2                = "ViaX"
    fatWireViaEncTblSize  = 4
    fatWireViaEncWidthThresholdTbl = (W1, W2, W3, W4)
    fatWireViaEncParallelLengthThresholdTbl = (L1, L2, L3, L4)
    fatWireViaEncMaxSpacingThresholdTbl = (S1, S2, S3, S4)
    fatWireViaEnclosureTbl = (E1, E2, E3, E4)
    fatWireViaArrayExcludedTbl = (0, 0, 1, 1)
    fatWireViaEncWidthIsOrthoTbl      = (0, 0, 1, 1)
}
```

For example,

```
DesignRule {
    layer1                = "M2"
    layer2                = "VIA2"
    minSpacing            = 0.000
    fatWireViaEncTblSize  = 4
    fatWireViaEncWidthThresholdTbl = (0.052,0.063,0.135,0.222)
    fatWireViaEncParallelLengthThresholdTbl = (0.098,0.098,0.098,0.098)
    fatWireViaEncMaxSpacingThresholdTbl = (0.082,0.105,0.105,0.105)
    fatWireViaEncMinSpacingThresholdTbl = (0.069,0.069,0.069,0.069)
    fatWireViaEnclosureTbl = (0.007,0.007,0.007,0.019)
    fatWireViaArrayExcludedTbl = (0, 0, 1, 1)
    fatWireViaEncWidthIsOrthoTbl      = (0, 0, 1, 1)
}
```

The `fatWireViaEncWidthIsOrthoTbl` attribute is a Boolean table. A value of 1 in the table causes the width to be measured in the direction orthogonal (perpendicular) to the parallel run of the nearby metal, as shown in [Figure 3-100](#).

Figure 3-100 Fat Wire Orthogonal Width Measurement



Overlapped Width Threshold Ranges

By default, in the fat wire via enclosure rule, the width thresholds for both the lower bound and upper bound are defined by a single table in which the threshold ranges are consecutive. Each successive value in the table is the upper bound of one range and the lower bound for the next range.

To specify overlapped threshold ranges, you use two threshold tables: one for the lower bounds and the other for the upper bounds, as shown in the following syntax:

```
DesignRule {
    layer1                = "MetalX"
    layer2                = "ViaX"
    fatWireViaEncTblSize  = 4
    fatWireViaEncMaxWidthThresholdTbl = (W1, W2, W3, W4)
    fatWireViaEncMinWidthThresholdTbl = (W1', W2', W3', W4')
    fatWireViaEncParallelLengthThresholdTbl = (L1, L2, L3, L4)
    fatWireViaEncMaxSpacingThresholdTbl = (S1, S2, S3, S4)
    fatWireViaEnclosureTbl = (E1, E2, E3, E4)
    fatWireViaArrayExcludedTbl = (0, 0, 0, 1)
}
```

The “MaxWidth” and “MinWidth” attributes replace the single “Width” attribute.

For example,

```
DesignRule {
    layer1                = "M1"
    layer2                = "VIA1"
    fatWireViaEncTblSize  = 4
    fatWireViaEncMinWidthThresholdTbl = (0.046, 0.056, 0.067, 0.122)
    fatWireViaEncMaxWidthThresholdTbl = (0.056, 0.122, 0.101, 0.221)
```

```

fatWireViaEncParallelLengthThresholdTbl = (0.184, 0.148, 0.148, 0.090)
fatWireViaEncMaxSpacingThresholdTbl    = (0.075, 0.106, 0.090, 0.106)
fatWireViaEnclosureTbl                  = (0.006, 0.006, 0.010, 0.006)
fatWireViaArrayExcludedTbl              = (0, 0, 0, 1)
}

```

When you use the single “Width” attribute, the width ranges are:

0.0–W1, W1–W2, W2–W3, W3–W4

When you use the “MaxWidth” and “MinWidth” attributes, the width ranges are:

W1–W1’, W2–W2’, W3–W3’, W4–W4’

The list of values in the “MinWidth” attribute must be monotonically increasing; each value in the list must be larger than one before it. This restriction does not apply to the “MaxWidth” list.

Minimum Spacing Requirements

By default, in the fat wire via enclosure rule specifies the maximum spacing thresholds for the distance between the fat metal and the nearby metal. To specify minimum as well as maximum thresholds, use the additional attribute shown in the following syntax:

```

DesignRule {
    layer1                = "MetalX"
    layer2                = "ViaX"
    fatWireViaEncTblSize   = 4
    fatWireViaEncWidthThresholdTbl = (W1, W2, W3, W4)
    fatWireViaEncParallelLengthThresholdTbl = (L1, L2, L3, L4)
    fatWireViaEncMaxSpacingThresholdTbl = (S1, S2, S3, S4)
    fatWireViaEncMinSpacingThresholdTbl = (S1', S2', S3', S4')
    fatWireViaEnclosureTbl = (E1, E2, E3, E4)
    fatWireViaArrayExcludedTbl = (0, 0, 0, 1)
}

```

For example,

```

DesignRule {
    layer1                = "M1"
    layer2                = "VIA1"
    fatWireViaEncTblSize   = 4
    fatWireViaEncMinWidthThresholdTbl = (0.046, 0.056, 0.067, 0.122)
    fatWireViaEncMaxWidthThresholdTbl = (0.056, 0.122, 0.101, 0.221)
    fatWireViaEncParallelLengthThresholdTbl = (0.184, 0.148, 0.148, 0.090)
    fatWireViaEncMaxSpacingThresholdTbl = (0.075, 0.106, 0.090, 0.106)
    fatWireViaEncMinSpacingThresholdTbl = (0.063, 0.063, 0.063, 0.063)
    fatWireViaEnclosureTbl = (0.006, 0.006, 0.010, 0.006)
    fatWireViaArrayExcludedTbl = (0, 0, 0, 1)
}

```

Fat Metal Contact Asymmetric Enclosure Rule Enhancements

The definition of a via in the `ContactCode` section of the technology file specifies the minimum amount of lower-metal and upper-metal enclosure around the via. An additional amount of metal enclosure might be required on certain sides of the via, depending on the metal width, which can result in asymmetric metal coverage of the via.

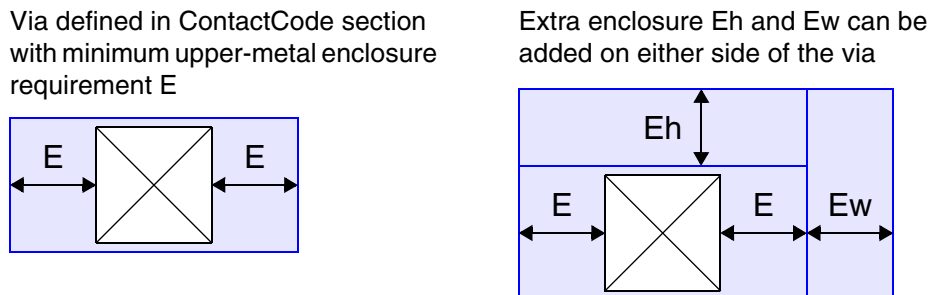
For advanced geometries, you can specify the additional upper-metal enclosure requirement either as an additional distance beyond the minimum enclosure specified in the `ContactCode` section in the height and width directions, or as a total span of metal covering the via in the height and width directions. The former is called the “extra enclosure” rule and the latter is called the “total enclosure” rule.

If both the extra enclosure and total enclosure rules are specified for the same contact number, the total enclosure rule applies and the extra enclosure rule is ignored for that contact number.

Fat Metal Contact Asymmetric Extra Enclosure Rule

In the fat metal asymmetric extra enclosure rule, you specify the additional amount of upper-metal enclosure in the height and width directions beyond the minimum specified in the `ContactCode` section. The additional metal can be added on either side of the via, as shown in [Figure 3-101](#), or it can be added on both sides as long as the total metal added is at least the additional enclosure specified in the rule.

Figure 3-101 Fat Metal Contact Asymmetric Extra Enclosure Rule



To specify this rule, use the attributes shown in the following syntax:

```
Layer "ViaX" {
  fatTblDimension      = 5
  fatTblThreshold      = (0, W1, W2, W3, W4)
  fatTblDimension2     = 4
  fatTblThreshold2     = (0, V1, V2, V3)
  fatTblFatContactNumber = (C11, C12, C13, C14,...)
  fatTblFatContactMinCuts = (C11, C12, C13, C14,...)
  fatTblExtraEncContactTblSize    = 3
  fatTblExtraEncContactNumber    = (1, 5, 7)
  fatTblExtraEncUpperLayerWidth  = (Ew1, Ew2, Ew3)
  fatTblExtraEncUpperLayerHeight = (Eh1, Eh2, Eh3)
}
```

For example,

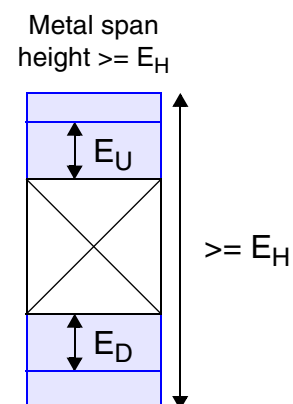
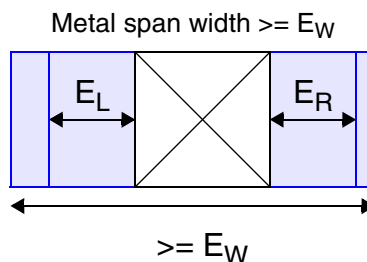
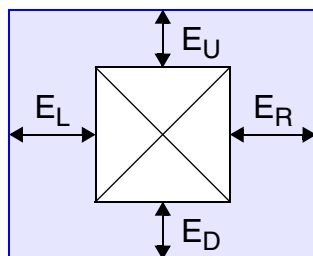
```
...
fatTblExtraEncContactTblSize    = 3
fatTblExtraEncContactNumber    = (1, 5, 7)
fatTblExtraEncUpperLayerWidth  = (0.000, 0.000, 0.062)
fatTblExtraEncUpperLayerHeight = (0.021, 0.021, 0.043)
...
```

Fat Metal Contact Asymmetric Total Enclosure Rule

In the fat metal asymmetric total enclosure rule, you specify the total span of upper-layer metal covering the via in the height and width directions. The total metal span can be distributed asymmetrically over the via, as long as the minimum enclosure requirement specified for the via in the `ContactCode` section is met, as shown in [Figure 3-102](#).

Figure 3-102 Fat Metal Contact Asymmetric Total Enclosure Rule

Via defined in `ContactCode` section with upper-metal enclosure requirements



To specify this rule, use the attributes shown in the following syntax example:

```
Layer "ViaX" {
    fatTblDimension                = 5
    fatTblThreshold                = (0, W1, W2, W3, W4)
    fatTblDimension2              = 4
    fatTblThreshold2              = (0, V1, V2, V3)
    fatTblFatContactNumber        = (C11, C12, C13, C14,...)
    fatTblFatContactMinCuts       = (N11, N12, N13, N14,...)
    fatTblTotalEncContactTblSize   = 2
    fatTblTotalEncContactNumber    = (4, 6)
    fatTblTotalEncUpperLayerWidth = (EW, 0)
    fatTblTotalEncUpperLayerHeight = (0, EH)
}
```

Alternative Syntax for Fat Metal Contact Asymmetric Enclosure

You can optionally use an alternative form of syntax to specify the fat metal contact asymmetric extra enclosure rule, similar to the method for the fat metal via symmetric enclosure rule (see [“Alternative Syntax for Fat Metal Contact and Extension Rules” on page 2-107](#)). Using this form of syntax, you specify the overlap as an arbitrary amount on each of the four sides of the contact. In many cases, using this alternative syntax can be easier than using the standard syntax.

The alternative syntax applies to upper-metal enclosure around a via. This is the general form of the syntax:

```
Layer "VIAx" {
    asymUpperEnclosureTblSize = 1 # number of asymUpperEnclosureTbl rows
    asymUpperEnclosureTbl    = (cutName, xleftEncWidth, xrightEncWidth,
                               xbotEncHeight, xtopEncHeight, topMetalWidth)

    totalUpperEnclosureTblSize = 1 # number of totalUpperEnclosureTbl rows
    totalUpperEnclosureTbl    = (cutName, baseEncWidth, baseEncHeight,
                               totalEncWidth, totalEncHeight, topMetalWidth)
}
```

The attributes *xleftEncWidth*, *xrightEncWidth*, *xbotEncWidth*, and *xtopEncWidth* are the enclosure values around the four sides (left, right, bottom, and top) of the via *cutName*, for a given upper-metal width, *topMetalWidth*. For an asymmetric via, the value of *xleftEncWidth* is different from *xrightEncWidth*, and *xbotEncWidth* is different from *xtopEncWidth*.

The attributes *baseEncWidth* and *baseEncHeight* specify the “base” minimum enclosures on each side of the via in the x-direction and y-direction, whereas the attributes *totalEncWidth* and *totalEncHeight* specify the minimum total enclosure (sum of two enclosures) in the x-direction and y-direction, respectively, for the via *cutName* and a given upper-metal width, *topMetalWidth*.

The following example demonstrates the alternative syntax.

```
Layer "ViaX" {
  ...
  asymUpperEnclosureTblSize = 6
  asymUpperEnclosureTbl    = (Vs,0.045,0.053,0.010,0.010,0.038,
                              Vs,0.010,0.010,0.045,0.053,0.038,
                              Vs,0.038,0.058,0.004,0.004,0.039,
                              Vs,0.004,0.004,0.038,0.058,0.039,
                              Vs,0.009,0.073,0.030,0.073,0.110,
                              Vs,0.030,0.073,0.009,0.073,0.110)

  totalUpperEnclosureTblSize = 2
  totalUpperEnclosureTbl    = (Vs,0.010,0.045,0.010,0.135,0.038,
                              Vs,0.045,0.010,0.135,0.010,0.038)
  ...
}
```

In this example, the `asymUpperEnclosureTbl` table contains six entries, one line per entry. Each line specifies the cut name, a combination of possible overlap values for each of the four sides, and an associated top metal width.

The `totalEnclosureTbl` table contains two entries, one line per entry. Each line specifies the cut name, the minimum (base) enclosure width in the x-direction and y-direction, the total enclosure width in the x-direction and y-direction, and an associated top metal width.

When you specify a rule using this alternative syntax, the IC Compiler tool internally converts the rule to standard syntax. To verify the accuracy of the converted rules, you write them out by setting the `write_converted_tf_syntax` variable to `true` before you write out the technology file:

```
icc_shell> set_app_var write_converted_tf_syntax true
...
icc_shell> write_mw_lib_files -technology -output my_tech.tf my_mw_lib.mw
...
```

By default, the control variable is set to `false`, which causes the technology file to be written out using the same syntax used to read it in.

Sparse-Dense Via Enclosure Rule

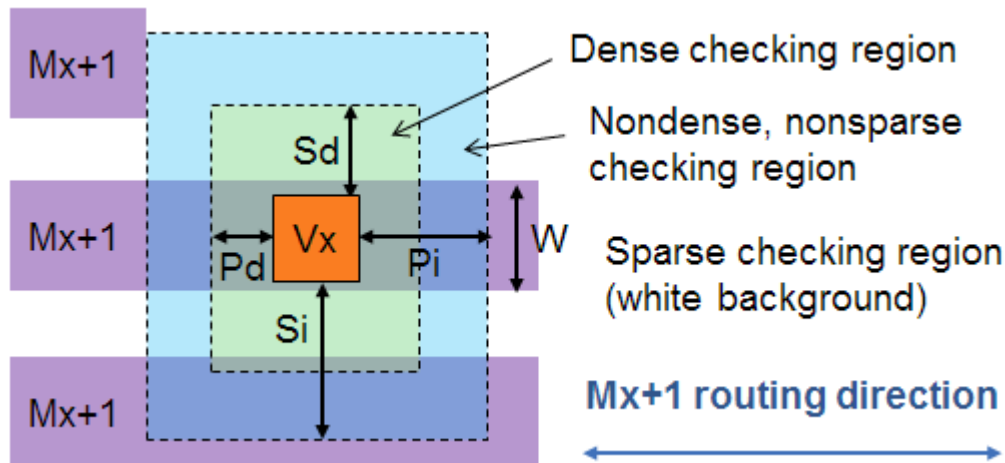
The fat wire via enclosure rule specifies the minimum amount of fat metal overlap over a via for the metal layer either above or below the via. The minimum overlap requirement depends on the width of the fat metal enclosing the via, the spacing to any nearby metal, and the parallel length of that nearby metal, as specified in a table. The basic rule is described in [“Fat Wire Via Enclosure Rule” on page 2-129](#).

For advanced geometries, the rule includes additional attributes that let you specify the minimum upper-level and lower-level metal overlap requirement based on the presence of

nearby upper-level or lower-level metal on one side of the wide metal wire and the absence of nearby metal on the opposite side.

This rule considers the case where the width of the upper-level or lower-level metal enclosing the via falls into certain range (W_{min} , W_{max}) and its neighboring metal shapes form a sparse-dense arrangement on two sides, as demonstrated in [Figure 3-103](#).

Figure 3-103 Sparse and Dense Via Upper and Lower Metal Enclosure Rule Regions

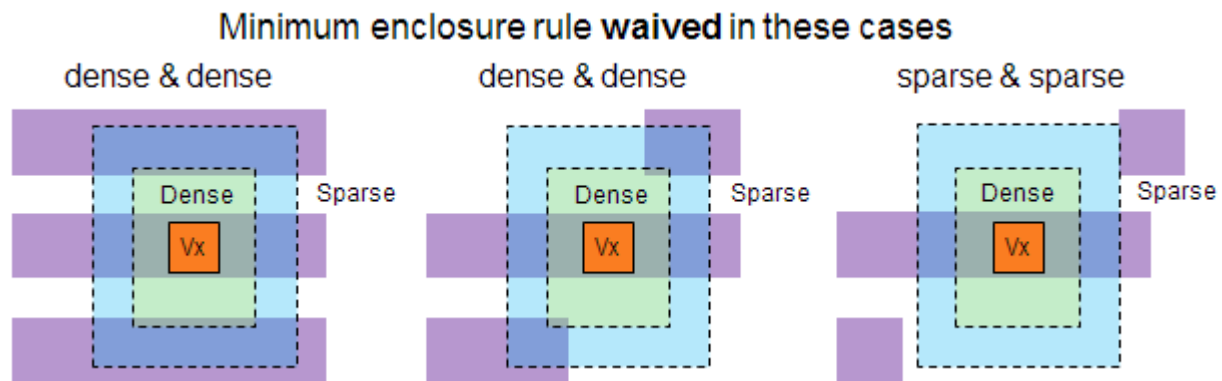


The rule defines the following regions for checking the presence of nearby metal:

- The dense region is a rectangular area expanded from edges of the via by a distance P_d in the upper metal's routing direction and S_d in the orthogonal direction (light green rectangle in the figure).
- The sparse region is the area outside of a rectangular area expanded from the edges of the via by a distance P_i in the upper metal's routing direction and S_i in the orthogonal direction (white background area in the figure).
- The ring-shaped region outside of the dense region and inside of the sparse region is the nondense, nonsparse region (light blue in the figure).

The minimum enclosure rule is waived when the nearby metal occupies the dense region on both sides of the upper metal connected to the via, or occupies only the sparse region on both sides, as shown by the examples in [Figure 3-104](#).

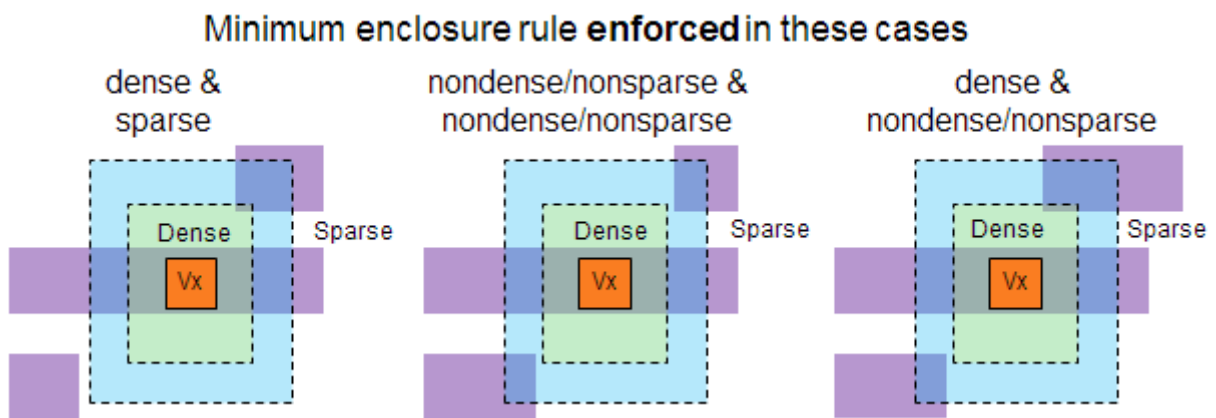
Figure 3-104 Rule Waived for Dense-Dense or Sparse-Sparse Nearby Metal on Two Sides

**Note:**

A metal shape merely touching the border of a region is not considered to be occupying that region. In the far-right example, the upper-right metal shape touching the border of the blue region is considered to occupy only the sparse (white) region.

Conversely, the rule is enforced when the nearby metal occupies the checking regions in any other configuration, as shown by the examples in [Figure 3-105](#).

Figure 3-105 Rule Enforced for Other Types of Nearby Metal Occupancy on Two Sides



This is the syntax of the rule:

```
DesignRule {
  layer1                                = "Mx"
  layer2                                = "Vx"
  fatWireViaEncCutTblSize                = 1
  fatWireViaEncCutNameTbl                = (Vs)
  fatWireViaEncTblSize                   = 1
  fatWireViaEncMinWidthThresholdTbl      = (Wmin)
  fatWireViaEncMaxWidthThresholdTbl      = (Wmax)
  fatWireViaEncParallelLengthThresholdTbl = (Pd)
  fatWireViaEncMaxSpacingThresholdTbl    = (Sd)
```

```

fatWireViaEncSparseParallelLengthThresholdTbl = (Pi)
fatWireViaEncSparseMinSpacingThresholdTbl    = (Si)
fatWireViaEnclosureTbl                       = (ESE)
fatWireViaEnclosureOrthoTbl                  = (ESO)
fatWireViaEnclosureOtherLayerTbl             = (EOO)
}

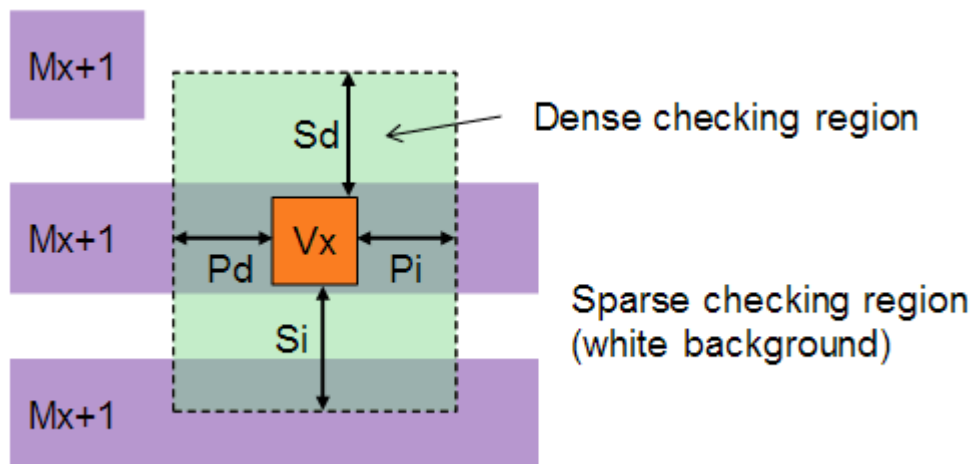
```

The last three attributes specify the minimum spacing requirements of the rule:

- ESE – Minimum enclosure in the *same* metal layer as the layer being checked (either upper-level or lower-level metal), in the *same* direction as the width measurement *W*
- ESO – Minimum enclosure in the *same* metal layer as the layer being checked, in the direction *orthogonal* to the width measurement *W*
- EOO – Minimum enclosure in the *other* metal layer (either lower-level or upper-level metal), in the direction *orthogonal* to the width measurement *W*

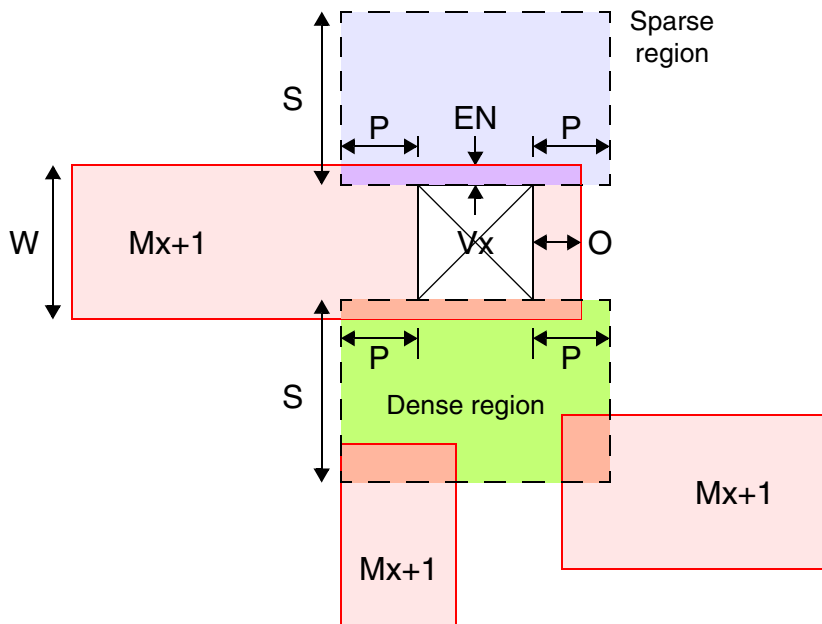
In the case where *Sd* and *Si* are the same, and *Pd* and *Pi* are the same, the dense region (green) and sparse region (white) are exactly complementary, and there is no blue region between them, as shown in [Figure 3-106](#).

Figure 3-106 Complementary Sparse and Dense Via Regions



In that case, the sparse and dense checking regions have the same dimensions, and the P_i attribute need not be specified because it is assumed to be the same as P_d . The rule checks the regions shown in [Figure 3-107](#).

Figure 3-107 Sparse-Dense Via Enclosure Rule



Here is an example of the rule:

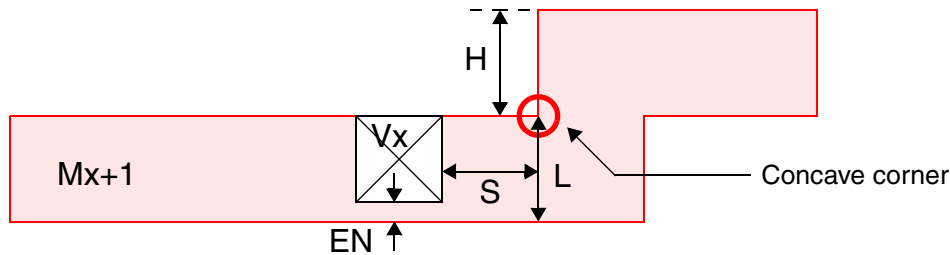
```
DesignRule {
    layer1                = "M4"
    layer2                = "VIA3"
    fatWireViaEncCutTblSize    = 3
    fatWireViaEncCutNameTbl   = (Vsm, Vh, Vv)
    fatWireViaEncTblSize      = 1
    fatWireViaEncMinWidthThresholdTbl = (0.048)
    fatWireViaEncMaxWidthThresholdTbl = (0.072)
    fatWireViaEncParallelLengthThresholdTbl = (-0.127)
    fatWireViaEncWidthIsOrthoTbl = (1)
    fatWireViaEncMaxSpacingThresholdTbl = (0.094)
    fatWireViaEncSparseMinSpacingThresholdTbl = (0.094)
    fatWireViaEnclosureTbl    = (0.018, 0.018, 0.018)
    fatWireViaEnclosureOrthoTbl = (0.059, 0.059, 0.059)
}
```

Via-to-Jog Metal Enclosure Rule

The via-to-jog metal enclosure rule specifies that either a minimum spacing or a minimum enclosure must be maintained for a via of a specified type located near a jog in the upper-level metal.

In [Figure 3-108](#), the via in layer V_x is near a jog in the upper-metal layer. The rule applies when the jog height is at least H and the width of the metal shape covering the via, measured at the concave corner of the jog, is at least L .

Figure 3-108 Metal Critical Via Enclosure Rule



When these conditions are true, the spacing from the via to the concave corner of the jog must be at least S , or the upper-metal enclosure over the via must be at least EN on at least one side of the via in the direction perpendicular to the run of metal. Either the minimum spacing or the minimum enclosure requirement satisfies the rule, or both can be true.

This is the syntax for the rule:

```
DesignRule {
  layer1                = "Vx"
  layer2                = "Mx+1"
  concaveMetalToCutTblSize      = 1
  concaveMetalToCutNameTbl      = (Vsm)
  concaveMetalToCutJogMinHeightThresholdTbl = (H)
  concaveMetalToCutWireMaxWidthThresholdTbl = (L)
  concaveMetalToCutMinEncTbl    = (EN)
  concaveMetalToCutMinDistTbl   = (S)
}
```

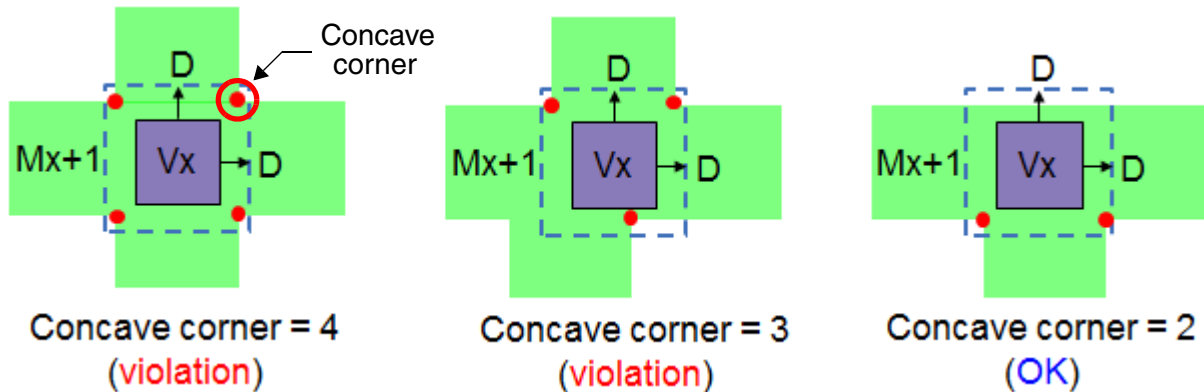
For example,

```
DesignRule {
  layer1                = "VIA3"
  layer2                = "MET4"
  concaveMetalToCutTblSize      = 1
  concaveMetalToCutNameTbl      = (Vsm)
  concaveMetalToCutJogMinHeightThresholdTbl = (0.005)
  concaveMetalToCutWireMaxWidthThresholdTbl = (0.041)
  concaveMetalToCutMinEncTbl    = (0.004)
  concaveMetalToCutMinDistTbl   = (0.036)
}
```

Via to Concave Corners Maximum Limit Rule

The via to concave corners maximum limit rule specifies the maximum number of concave corners allowed in the upper metal enclosing specified types of vias within a specified distance D. For example, if the limit is set to 2, then no more than two vias are allowed within the distance D in the upper metal enclosing the via, as shown in [Figure 3-109](#).

Figure 3-109 Via Enclosure Concave Corner Limit Rule



The rule uses the following syntax:

```
DesignRule {
  layer1           = "Mx+1"
  layer2           = "Vx"
  concaveCornerCutTblSize      = 3
  concaveCornerCutNameTbl     = (Vs, Vh, Vv)
  concaveCornerCutRangeTbl    = (0.013, 0.013, 0.013) # D
  concaveCornerMaxNumTbl      = (2, 2, 2)
}
```

In this example, for the vias named Vs, Vh, and Vv, no more than two concave corners are allowed in the enclosing upper metal within a distance of 0.013 from the via edges.

Contact Code Rules

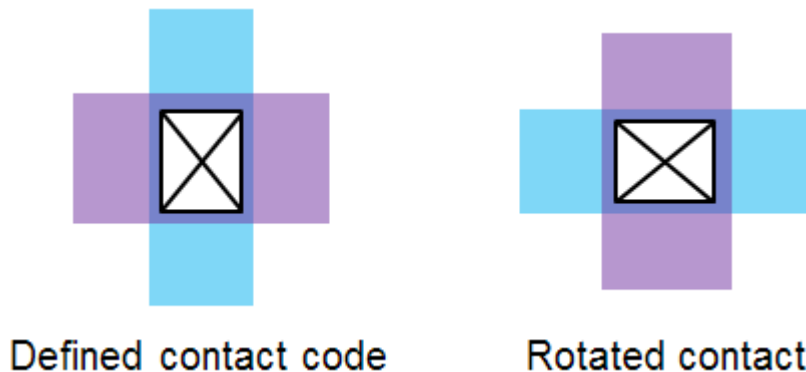
The following advanced rules apply to the usage of vias defined in the `ContactCode` section of the technology file:

- [Contact Code Rotation Control](#)
- [Fat Metal Contact Rule](#)

Contact Code Rotation Control

By default, the router can rotate a via, along with its defined upper and lower metal, to assist in making metal connections to the via, as shown in [Figure 3-110](#).

Figure 3-110 Rotated Contact



To disallow rotation of a via, use the `nonRotatable` attribute in its `ContactCode` section:

```
ContactCode "VIAx" {
  contactCodeNumber = 5
  nonRotatable      = 1    # default = 0
  ...
}
```

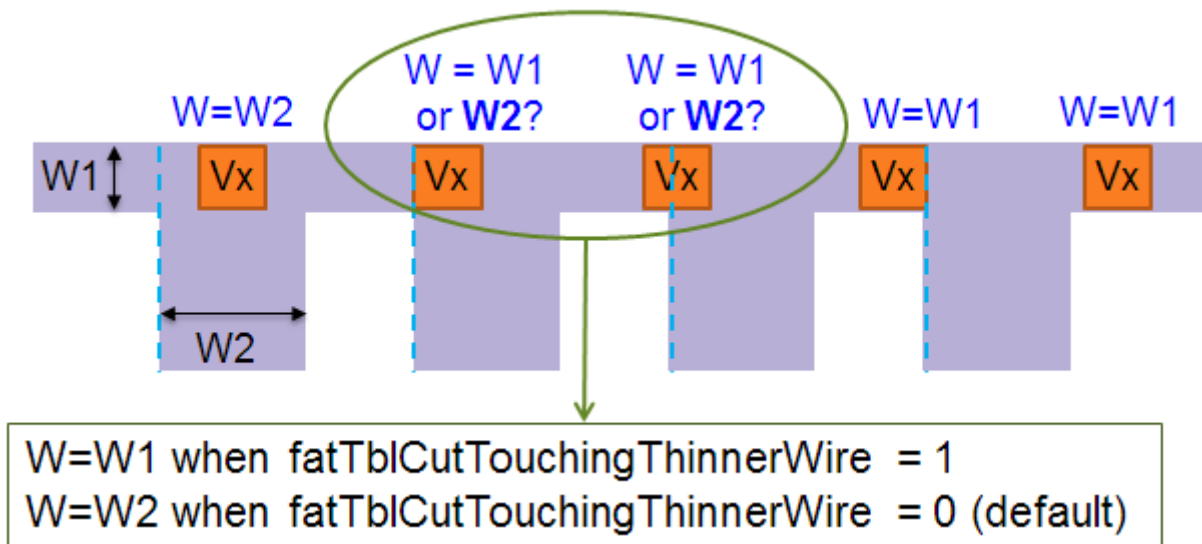
In that case, any instance of a rotated via using that contact code number is flagged as a DRC error.

Fat Metal Contact Rule

The area-based fat metal contact rule specifies the contact code numbers of the allowed vias and the minimum number of vias required to connect between two metal layers when either of the metal segments is a fat wire and the number of vias depends on the metal areas, calculated from the metal widths. The basic rule is described in [“Area-Based Fat Metal Contact Rule” on page 2-101](#).

An enhanced form of this rule determines the width of the upper or lower metal when the via touches or crosses a boundary between two metal shapes of different widths. For example, in [Figure 3-99](#), the second via from the left touches a metal shape boundary, and the third via crosses over such a boundary.

Figure 3-111 Ambiguous Metal Widths Enclosing Vias



In this situation, by default, the tool uses the larger width value. To have tool use the smaller value instead, add the following attributes to the fat metal contact rule and set them to 1 for the lower and upper metal layers, respectively:

```
fatTblCutTouchingThinnerWire = 1 # Use smaller value W1 for lower met
fatTblCutTouchingThinnerWire2 = 1 # Use smaller value W1 for upper met
```

To use the larger width value W2, set the attribute to 0 or omit it from the technology file.

IC Compiler Zroute Error Messages

The following table lists some of the DRC errors that can be reported by Zroute in IC Compiler and the corresponding rule names in this chapter.

Table 3-3 IC Compiler Zroute Error Messages and Design Rule Titles

Zroute DRC error message	Design rule title
Adjacent via spacing	Adjacent Via Spacing Rule
Concave convex edge enclosure	Concave-Convex Edge Via Enclosure Rule
Concave corner keepout	Concave Corner Keepout Rule
Constrained via spacing	Constrained Via Spacing Rule

Table 3-3 IC Compiler Zroute Error Messages and Design Rule Titles (Continued)

Zroute DRC error message	Design rule title
Diff net fat extension spacing	Fat Metal Forbidden Spacing Rule
Diff net spacing	Fat Metal Forbidden Spacing Rule
Diff net spacing	Metal Span Orthogonal Spacing Rule
Diff net spacing	Preferred and Nonpreferred Fat Metal Spacing Rule
Diff net via-cut spacing	Adjacent Via Spacing Rule
Diff net via-cut spacing	General Cut Spacing Rule
Diff net via-cut spacing	Mixed Edge-to-Edge and Center-to-Center Via Spacing Rule
Diff net via-cut spacing	Non-Self-Aligned Interlayer Via Spacing Rule
Diff net via-cut spacing	Via Corner Spacing Rule Enhancement
Diff net via-cut spacing	Via Corner Spacing at Zero Projection Rule
Diff net via-cut to metal spacing	Interlayer Cut-to-Metal Spacing Rule
Edge via spacing	Edge Via Spacing Rule
End of line spacing	Two-Neighbor End-of-Line Spacing Rule Enhancements
End of line spacing	Preferred and Nonpreferred End-of-Line Spacing Rule
End of line to concave corner distance	Line-End Concave Corner Length Rule
End of line to concave corner distance	Line-End to Concave Corner Spacing Rule
Fat metal branch	Fat Metal Branch Minimum Width and Length Rule
Fat metal branch minWidth rule	Fat Metal Branch Minimum Width and Length Rule
Fat metal via asymmetric enclosure	Alternative Syntax for Fat Metal Contact Asymmetric Enclosure
Fat wire via asymmetric enclosure	Fat Metal Contact Asymmetric Extra Enclosure Rule
Fat wire via keepout enclosure	Fat Wire Via Enclosure Rule Enhancements

Table 3-3 IC Compiler Zroute Error Messages and Design Rule Titles (Continued)

Zroute DRC error message	Design rule title
Fat wire via keepout enclosure	Sparse-Dense Via Enclosure Rule
Forbidden pitch	Forbidden Pitch Rule
Forbidden pitch	Fat Metal Forbidden Spacing Rule
Forbidden pitch	Preferred-Direction Forbidden Spacing Rule
Forbidden via-cut spacing	Via Forbidden Spacing Rule
Illegal dimension route	Discrete Metal Dimension Rule
Less than minimum area	Polygon-Edge Multistage Minimum Area Rule Enhancement
Less than minimum edge length	Minimum Metal Jog Length and Adjacent Edge Length Rules
Less than minimum edge length	Adjacent Minimum Edge Length Rule
Less than minimum edge length	Convex-Concave Minimum Edge Length Rule
Less than minimum edge length	Two-Convex-Corner Minimum Edge Length Rule
Less than minimum width	Diagonal Concave Corner Minimum Width Rule
Max number of via-metal concave corners	Via to Concave Corners Maximum Limit Rule
Metal corner keepout	Fat Metal Corner Keepout Rule
Metal corner preferred-direction keepout	Preferred and Nonpreferred Corner-to-Corner Spacing Rule
Need fat contact	Alternative Syntax for Fat Metal Contact and Extension Rules
Need fat contact	Fat Metal Contact Rule
Need fat contact on extension	Alternative Syntax for Fat Metal Contact and Extension Rules
Same net spacing	Fat Metal Forbidden Spacing Rule
Self aligned via cluster	Self-Aligned Via Cluster Rules

Table 3-3 IC Compiler Zroute Error Messages and Design Rule Titles (Continued)

Zroute DRC error message	Design rule title
Separated cut spacing	Separated Via Spacing Rule
U shape keepout	Line-End to U-Shape Spacing Rule
U shape leg spacing	U-Shape Leg Spacing Rule
U shape spacing	U-Shape Spacing Rule
Via bridge rule	Via-Induced Metal Bridge Rule
Via enclosure to metal spacing	Enclosure-Dependent Metal Spacing Rule
Via metal concave corner	Via-to-Jog Metal Enclosure Rule
Wide metal jog	Wide Metal Jog Rule