Synopsys®
Technology File and Routing Rules Reference Manual

Version O-2018.06, June 2018

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

Contents

Preface

This preface includes the following sections:

- About This Manual
- Customer Support

About This Manual

The *Synopsys Technology File and Routing Rules Reference Manual* provides details about the technology file and how to implement routing design rules. Synopsys physical implementation tools (IC Compiler and IC Compiler II) use this information to perform placement and routing.

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

Customer Support

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

Accessing SolvNet

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

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

https://solvnet.synopsys.com

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

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

Contacting the Synopsys Technical Support Center

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

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

    https://www.synopsys.com/support/global-support-centers.html
- Telephone your local support center.
  - Call (800) 245-8005 from within North America.
  - Find other local support center telephone numbers at

    https://www.synopsys.com/support/global-support-centers.html

1

Technology File Syntax

A technology file is an ASCII file that provides technology-specific information, such as the names and physical and electrical characteristics of each metal layer and the routing design rules. After you create a technology file, you associate it with a library in the IC Compiler II or IC Compiler tool.

The following topics describe the technology file syntax:

- 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

- PRRule Section
- DensityRule Section
- SlotRule Section
- FringeCap Section
- CapModel and CapTable Sections
- ResModel 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 write out a technology file associated with a design library,

- In the IC Compiler II tool, use the `write_tech_file` command:

  ```
  icc2_shell> write_tech_file techfile.tf
  ```

- In the IC Compiler tool, use the `write_mw_lib_files -technology` command:

  ```
  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.
- Each row in a table specification must contain at least one non-null value.

  For example, the following table specification is invalid, because the row shown in purple contains all null values:

  ```
  fatTblDimension         = 6
  fatTblFatContactMinCuts = (  "", "", "", "3", "", "",
                               "", "1, 1", "", "1, 1", "",
                               "", "", "", "", "", "",
                               "", "", "4", "4", "", "4"  )
  ```

- 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

These are the contents of the technology file:

```
/* Specifies units and unit values */
Technology {
  name
  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 */
Color color_value {
  color_attributes
}...

/* Defines the stipple styles used to display designs */
Stipple "name" {
  stipple_attributes
}...

/* Defines the line styles used to display designs */
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" {
  layout_attributes
  display_attributes
  parasitic_attributes
  physical_attributes
  design_rule_attributes
}...

/* Defines layer data types */
LayerDataType "name" {
  layerdatatype_attributes
}...
```

```
/* Defines the vias used in designs */
ContactCode "name" {
  contactcode_attributes
}...

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

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

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

/* Defines via rules */
ViaRule "name" {
  viarule_attributes
}...

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

/* Defines capacitance information for interconnect layers */
FringeCap value {
  fringecap_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
}
```

Technology Section

Use the `Technology` section of a technology file to specify global attributes, such as units and routing rule modes:

```
Technology {
     /* define technology name */
      name = tech32hvt
     /* define units */
      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
      dielectric = 0.000000e+00

     /* define operating conditions */
     minBaselineTemperature  = 25
     nomBaselineTemperature  = 25
     maxBaselineTemperature  = 25

     /* define routing rule modes */
      cornerSpacingMode = 0
      fatTblMinEnclosedAreaMode = 0
      fatTblSpacingMode = 0
      fatWireExtensionMode = 0
      fixedColor = 0
      maxStackLevelMode = 1
      metalAboveMiMCap = 'M9'
      minAreaMode = 0
      minEdgeMode = 0
      minLengthMode = 0
      parallelLengthMode = 0
      stubMode = 0
}
```

Technology Name

The `name` attribute in the `Technology` section specifies the name of the technology for the IC Compiler II tool. For example, the `get_techs` command returns this technology name. This attribute is not used in the IC Compiler tool.

Unit Definitions

The unit size and precision of each unit (length, resistance, capacitance, and so on) are defined 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.

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

`unitLengthName`

The `unitLengthName` attribute defines the linear distance unit. The valid values are `micron` and `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 is 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, 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
`fs`	1 x 10^{-15} second
`ps`	1 x 10^{-12} second
`ns`	1 x 10^{-9} second
`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

Table 1-2 Valid Units for unitCapacitanceName (Continued)

Unit	**Value**
`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 x 10^{-3} ohm
`ohm`	ohm
`kohm`	1 x 10^{3} ohm
`Mohm`	1 x 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 x 10^{-15} henry
`pH`	1 x 10^{-12} henry
`nH`	1 x 10^{-9} henry
`uH`	1 x 10^{-6} henry
`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 is 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`	1×10^{-15} watt
`pw`	1×10^{-12} watt
`nw`	1×10^{-9} watt
`uw`	1×10^{-6} watt
`mw`	1×10^{-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

Table 1-6 Valid Units for unitVoltageName (Continued)

Unit	Value
`mV`	1 x 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.

`dielectric`

The `dielectric` attribute specifies the permittivity of SiO_2. This attribute is not used by the IC Compiler or IC Compiler II tool.

Defining Routing Rule Modes

Use the following attributes to define the routing rule modes in the `Technology` section:

`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 1-8 shows the valid values for the `cornerSpacingMode` attribute.

Table 1-8 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.

`fatTblSpacingMode`

Determines the effect of the parallel length (defined with the `fatTblParallelLength` attribute) on the indexing of the `fatTblSpacing` attribute. Table 1-9 shows the valid values for the `fatTblSpacingMode` attribute.

Table 1-9 Valid Values for fatTblSpacingMode

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

For more information, see Fat Metal Spacing Rule.

`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 Fat Metal Minimum Enclosed Area Rule.

`fatWireExtensionMode`

The `fatWireExtensionMode` attribute controls the application of the fat metal extension spacing rule. Table 1-11 shows the valid values for the `fatWireExtensionMode` attribute.

Table 1-11 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.
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.

`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.

`maxStackLevelMode`

Controls the scope of the via array maximum stack level rule. Table 1-12 shows the valid values for the `maxStackLevelMode` attribute.

Table 1-12 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.

`metalAboveMiMCap`

Specifies the regular metal layer under which metal-insulator-metal (MiM) capacitors are inserted. MiM capacitors are inserted below this regular metal layer and above the next lower regular metal layer.

For more information, see Defining MiM Layers.

`minAreaMode`

Determines whether the shapes of areas are considered in applying the minimum area rule. Table 1-13 shows the valid values for the `minAreaMode` attribute.

Table 1-13 Valid Values for minAreaMode

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

This attribute is used only by the classic router. For more information, see Two-Stage Special Minimum Area Rule.

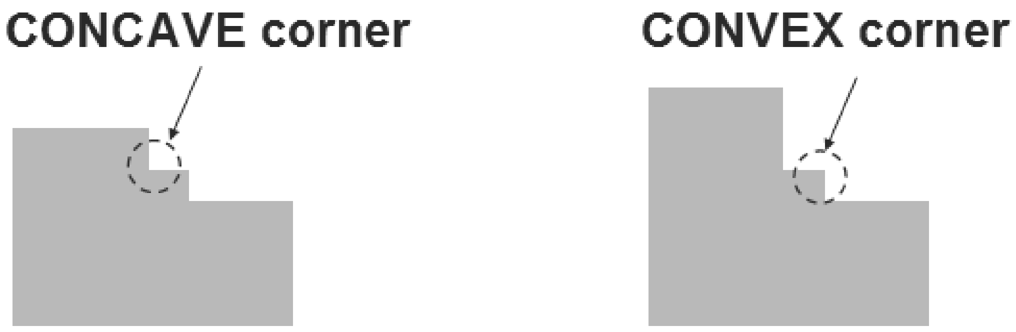
`minEdgeMode`

Controls the scope of the check for the minimum edge rules. Table 1-14 shows the valid values for the `minEdgeMode` attribute.

Table 1-14 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 the 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 `maxNumMinEdge` or if the total edge length is greater than `maxTotalMinEdgeLength`.

Figure 1-1 Concave and Convex Corners



For more information, see Minimum Edge Rules.

`minLengthMode`

Determines whether via cuts are considered in applying the minimum length rule. Table 1-15 shows the valid values for the `minLengthMode` attribute.

Table 1-15 Valid Values for minLengthMode

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

For more information, see Minimum Length Rule.

`parallelLengthMode`

The `parallelLengthMode` attribute controls the merging of wire segments based on the parallel length rule. Table 1-16 shows the valid values for the `parallelLengthMode` attribute.

Table 1-16 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 Rule.

`stubMode`

Specifies when the stub spacing rule is applied, rather than the minimum spacing rule. Table 1-17 shows the valid values for the `stubMode` attribute.

Table 1-17 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 `stubThreshold`.
1	Stub spacing is used when an end-of-line metal segment has an edge width that is less than or equal to `stubThreshold`.
2	Stub spacing is used when a metal segment with a width less than or equal to `stubThreshold` has neighboring metal shapes along two adjacent edges or any one edge, which is less than `stubThreshold` from the corner of two adjacent edges.
3	Stub spacing is used when a metal with a width less than or equal to `stubThreshold` has neighboring metals along three adjacent edges and has neighboring metal shapes along two adjacent edges within `stubThreshold`.
4	Stub spacing is used when a metal segment has a width that is less than `stubThreshold` and there is no connecting metal within `minWidth`, and if it has neighboring metal shapes along two adjacent edges or any one edge that is less than `stubThreshold` from the corner of two adjacent edges.

For more information, see End-of-Line Spacing Rule and Stub Modes.

PrimaryColor Section

The `PrimaryColor` section of the technology file defines the six basic colors that the GUI tool 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; the GUI tool 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) varies from 0 to 255, with 255 producing the brightest intensity of the color.

Colors in the GUI

The `PrimaryColor` section of a technology file defines the six primary colors: 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.

The GUI tool combines 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, the tool uses the default values shown in Example 1-1.

Example 1-1 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 1-1, 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 the tool 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 1-2 shows a `Color` section that defines color 62 (“owhite”).

Example 1-2 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 the tool 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 1-3 shows the definition for a line style named “boundary.”

Example 1-3 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 dimensions of the rectangular units used for placement of standard cells in site rows. These units are called *site definitions* in the IC Compiler II tool or *unit tiles* in the IC Compiler tool. In the IC Compiler II tool, you can define additional site definitions by using the `create_site_def` command.

You define a unit tile or site definition by specifying the following attributes in the `Tile` section:

`name`

The name of the unit tile or site definition, a string of up to 31 characters.

`width`

The width in length units of the technology.

`height`

The height in length units of the technology.

Note:
The IC Compiler II tool uses the single-height site definition for both single-height and multiple-height cells. Therefore, you only need to specify the single-height site definition for this tool. The height of each library cell must be a whole-number multiple of the site definition height (single-height, double-height, and so on).

Example 1-4 shows a typical unit tile definition.

Example 1-4 Tile Section of a Technology File

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

Layer Section

A `Layer` section defines a metal or via layer by specifying the layer attributes. In the IC Compiler II tool, you can define additional layers by using the `create_layer` command.

The layer 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.

These are some of the attributes that can be used in the `Layer` section:

```
Layer "MET1" {
      /* layout attributes */
     layerNumber = 8
     isDefaultLayer = 0
     isMarkerLayer = 0
     maskName = "metal1"
     pitch = 2.2

      /* display attributes */
     color = "blue"
     lineStyle = "solid"
     pattern = "dot"
     blink = 0
     visible = 1
     selectable = 1
     panelNumber = 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
minSpacing = 1.0
jointJointWireMinSpacing = 0
endJointWireMinSpacing = 0
jointEndWireMinSpacing = 0
... and so on ...

 /* 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
```

Layout Attributes

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

The layout attributes are

`layerNumber`

Defines the number that identifies the layer.

In the IC Compiler II tool, user-defined layer numbers are 1–32767. There are no system-defined layer numbers. The tool supports up to 31 routing layers.

In the IC Compiler tool, in the default layer mode, user-defined layer numbers are 1–187 and system-defined layer numbers are 188–255. In the extended layer mode, user-defined layer numbers are 1–4000 and system-defined layer numbers are 4001–4095.

`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.

`isMarkerLayer`

Specifies whether the layer is a marker layer. Shapes on marker layers are considered blockages and are always included in the frame view.

Valid values are 0 or 1.

`maskName`

Defines the physical layer associated with the display layer.

The following standard mask names are recognized by the physical implementation tools:

- `poly`
- `polyCont`
- `metal1`, `metal2`, ... `metal31`
- `cutMetal1`, `cutMetal2`, ... `cutMetal15`

  The cut metal layer corresponds to the metal layer with the same mask number. For example, cutMetal1 corresponds to metal1.
- `via1`, `via2`, ... `via30`
- `nwell`, `pwell`, `deepNwell`, `deepPwell`

- `mimtop`, `mimbottom`, `viaMimtop`, and `viaMimbottom` (for details, see Defining MiM Layers)
- `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.

`pitch`

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

Note:
The Milkyway Environment tool uses the specified pitch to generate wire tracks in the unit tile for the IC Compiler tool. 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

The display attributes specify how objects on the layer are displayed.

Note:
You can also set the layer display attributes in the physical implementation tools, as described in the *IC Compiler II Graphical User Interface User Guide* and the *IC Compiler Implementation User Guide*.

The display attributes are

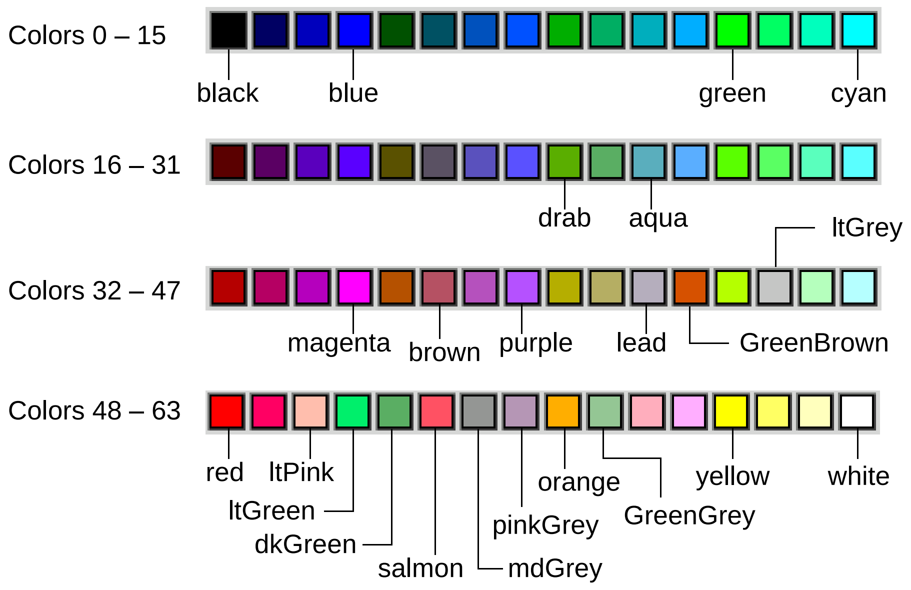
`color`

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 the GUI.) Figure 1-2 shows the standard colors and their predefined names.

Figure 1-2 Layer Display Colors



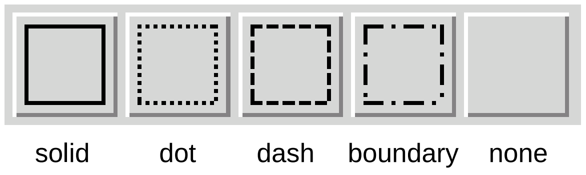
Colors that have a name can be referred to by either the name or number, such as “blue” or “3”. The rest can be referred to by number only.

`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.) Figure 1-3 shows the standard line styles and their predefined names.

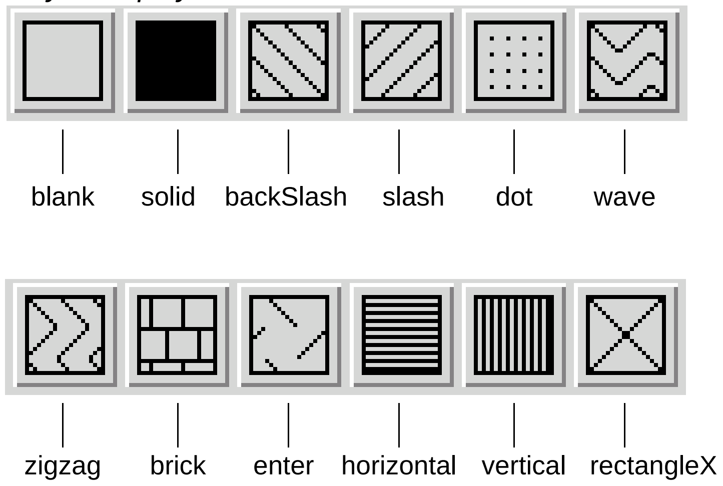
Figure 1-3 Layer Display Line Styles



`pattern`

Sets the pattern displayed inside the layer geometries. Figure 1-4 shows the standard patterns and their predefined names.

Figure 1-4 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`

For the IC Compiler tool, 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

This attribute is not used by the IC Compiler II tool.

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.) For more information about these design rules, see Routing Design Rules.

Note:

Interlayer design rules are defined in the `DesignRule` section. For more information, see DesignRule Section.

Some of the design rule attributes used in the `Layer` section are

`adjacentCutRange`

The distance for defining adjacent vias.

`checkManhattanSpacing`

The type of distance measurement 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.

`cornerMinSpacing`

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

`cornerSpacingCheckManhattan`

The type of distance measurement used for corner-to-corner spacing analysis. If this attribute is set to 0, diagonal measurement is used; the actual spacing, measured diagonally, must satisfy the minimum corner-to-corner 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 corner-to-corner spacing requirement. This attribute is supported only for metal layers.

`cutDataTypeTbl`

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

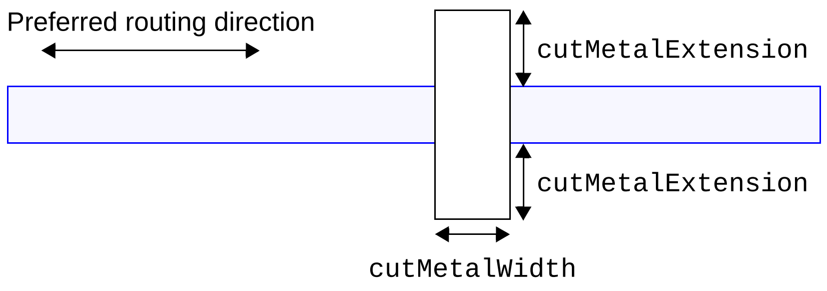
`cutHeightTbl`

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

`cutMetalExtension`

The perpendicular distance that a cut metal shape extends on either side of the metal shape. The total height of the cut metal shape is the width of the metal shape plus twice the cut metal extension value. Figure 1-5 shows the cut metal extension measurement.

Figure 1-5 cutMetalExtension

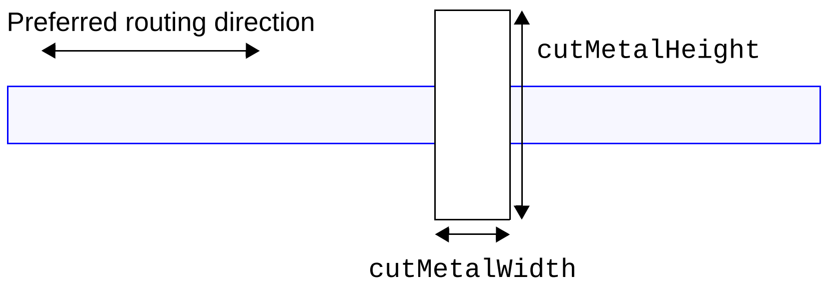


This attribute applies only to cut metal layers. This attribute and the `cutMetalHeight` attribute are mutually exclusive; you must specify one.

`cutMetalHeight`

The height of a cut metal shape.

Figure 1-6 cutMetalHeight



This attribute applies only to cut metal layers. This attribute and the `cutMetalExtension` attribute are mutually exclusive; you can specify only one.

`cutMetalWidth`

The width of a cut metal shape. This attribute applies only to cut metal layers; it is a required attribute.

`cutNameTbl`

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

`cutTblSize`

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

`cutWidthTbl`

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

`defaultWidth`

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.

`enclosedCutMinSpacing`

The minimum spacing between two enclosed vias.

`enclosedCutNeighborRange`

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

`enclosedCutNumNeighbor`

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

`enclosedCutToNeighborMinSpacing`

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

`maxLength`

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.

`maxLengthDataTypeTbl`

A list of data type numbers for the maximum horizontal length constraint. The list must contain the number of values specified in the `maxLengthTblSize` attribute.

Note:
This attribute can be specified only when the `maskName` attribute is set to `metal0`.

`maxLengthTbl`

A list of floating point distance values that correspond to the list of data types defined by the `maxLengthDataTypeTbl` attribute. The list must contain the number of values specified in the `maxLengthTblSize` attribute.

Note:
This attribute can be specified only when the `maskName` attribute is set to `metal0`.

`maxLengthTblSize`

The size of the table for the maximum horizontal length constraint data.

`maxNumAdjacentCut`

The maximum number of adjacent vias.

`maxWidth`

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:
This attribute is used only for wide wire slotting by the IC Compiler tool. The IC Compiler II tool and the IC Compiler signal router ignore the `maxWidth` attribute.

`minArea`

The minimum area for any object on the layer.

`minEdgeLength`

The threshold for short edges.

`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.

`minLength`

The minimum wire length allowed on the layer.

`minSpacing`

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

`minWidth`

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

`onWireTrack`

The routes and vias are placed on wire tracks.

`sameNetMinSpacing`

The minimum spacing for two shapes belonging to the same net.

`signalRouteMaxWidth`

The maximum width for any signal route on the layer.

For more information about design rules specified in the `Layer` section, see Routing Design Rules.

Parasitic Attributes

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

This topic 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.)

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.

Note:
The IC Compiler II tool does not use these values. The IC Compiler tool uses these resistance values only when parasitic information is not available from TLUPlus files.

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.

Note:
The IC Compiler II tool does not use these values. The IC Compiler tool uses these capacitance values only when parasitic information is not available from TLUPlus files.

Inductance

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

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.

Note:
The IC Compiler II and IC Compiler tools do not use these inductance attributes.

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 zero (0), the tool 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.)

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, the tool 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 topic 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.)

The height from substrate values are stored internally as double-precision numbers and therefore are not limited by the database unit limitations described in Unit Definitions.

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`

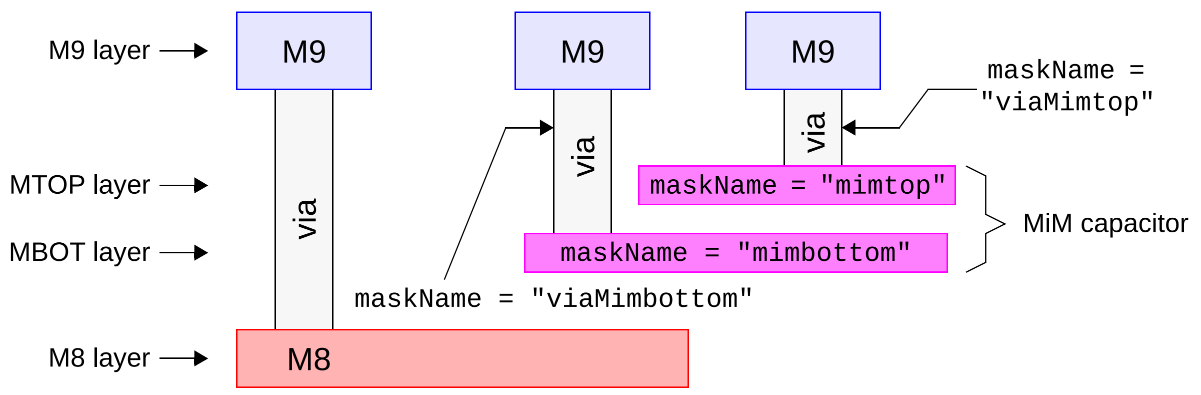
A floating-point number representing the maximum thickness.

These values are stored internally as double-precision numbers and therefore are not limited by the database unit limitations described in Unit Definitions.

Defining MiM Layers

The IC Compiler II tool supports the creation and usage of metal-insulator-metal (MiM) capacitors. A MiM capacitor is made of two special-purpose conducting layers separated by an insulator. This capacitor is inserted between two regular metal layers, as shown in Figure 1-7.

Figure 1-7 Cross-Section View of MiM Capacitor Layers



MiM capacitors are typically connected between power and ground to help maintain a constant supply voltage in the presence of electrical noise. The dimensions of a MiM capacitor are usually custom-sized to fit the power strap geometry of a specific power plan.

In the `Technology` section of the technology file, the `metalAboveMiMCap` attribute specifies the regular metal layer under which MiM capacitors are inserted. MiM capacitors are inserted below this regular metal layer and above the next lower regular metal layer.

In the `Layer` section of the technology file,

- The `mimtop` and `mimbottom` mask name attributes are reserved for the upper and lower MiM capacitor layers
- The `viaMimtop` and `viaMimbottom` mask name attributes are reserved for the via layers that connect the MiM capacitor layers to the regular layer identified by the `metalAboveMiMCap` attribute

The MiM capacitor layers are defined in the technology file as shown in the following example:

```
Technology {
  ...
  metalAboveMiMCap = "M9"
}
...
Layer "MBOT" {
  layerNumber  = 52
  maskName     = "mimbottom"
  color        = "magenta"
  lineStyle    = "solid"
  pattern      = "dot"
  blink        = 0
  visible      = 1
  selectable   = 1
}
Layer "VMBOT" {
  layerNumber  = 55
  maskName     = "viaMimbottom"
  blink        = 0
  visible      = 1
  selectable   = 1
  color        = "aqua"
  lineStyle    = "solid"
  pattern      = "rectangleX"
}
Layer "MTOP" {
  layerNumber  = 62
  maskName     = "mimtop"
  blink        = 0
  visible      = 1
  selectable   = 1
  color        = "blue"
  lineStyle    = "solid"
  pattern      = "dot"
}
Layer "VMTOP" {
  layerNumber  = 65
  maskName     = "viaMimtop"
  blink        = 0
  visible      = 1
  selectable   = 1
  color        = "brown"
  lineStyle    = "solid"
  pattern      = "rectangleX"
}
```

LayerDataType Section

The `LayerDataType` section defines a layer data object 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. In the IC Compiler II tool, you can define additional layer data objects by using the `create_purpose` command.

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 1-5 shows a typical `LayerDataType` section.

Example 1-5 LayerDataType Section

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

In the IC Compiler II tool, you can use the following command to report the layer attributes:

```
icc2_shell> report_attributes -application -class layer layer_name
```

In the IC Compiler tool, you can report the layer attributes by using the `report_attributes` or `get_layer_attribute` command.

ContactCode Section

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

- Physical Attributes

  Physical attributes define the physical attributes of the via.

- 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 vias (contact codes) used in designs in the library. You can define additional vias by using the `create_via_def` command in the IC Compiler II tool or the `create_via_master` command in the IC Compiler tool.

Example 1-6 shows a typical `ContactCode` section for a default via.

Example 1-6 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

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

Example 1-7 shows a typical `ContactCode` section for a fat via.

Example 1-7 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

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

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

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 or custom 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. They include

`name`

The name of the via definition.

`contactCodeNumber`

Identifies the via definition as 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.

`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.

`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.

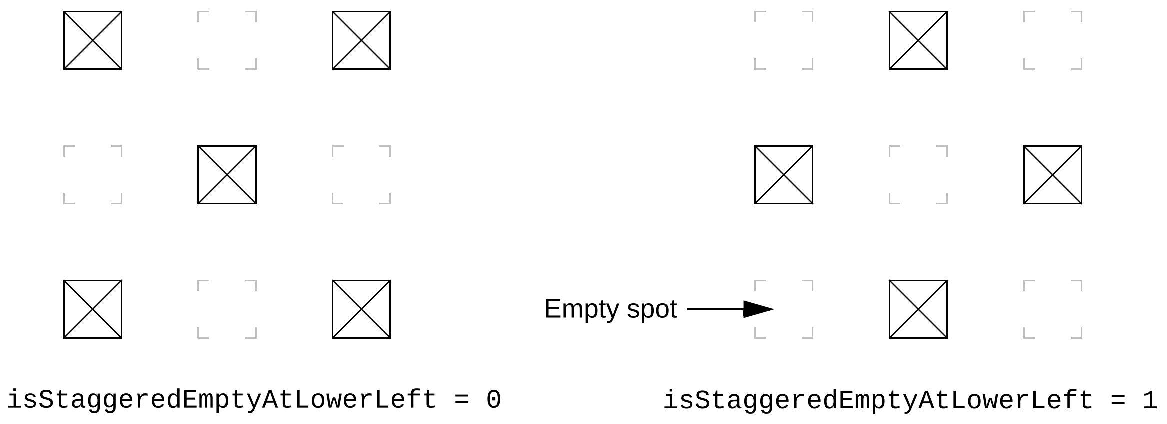
`isStaggeredPattern`

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 1-8.

`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 1-8 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 1-8 Staggered Via Arrays


Empty spot
isStaggeredEmptyAtLowerLeft = 0
isStaggeredEmptyAtLowerLeft = 1

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.

Note:
> Additional via design rules are defined in the `Layer` and `DesignRule` sections. For more information, see Design Rule Attributes and DesignRule Section.

The design rule attributes for the `ContactCode` section include

`excludedForPGRoute`

> When `excludedForPGRoute` is present and set to 1, it prevents the router from using the via for power and ground (PG) 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 for new PG routes.

`excludedForSignalRoute`

> When `excludedForSignalRoute` is present and 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.

`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.

`maxNumRows`

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

`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.

`minCutSpacing`

The minimum separation distance between the edges of the via.

`nonRotatable`

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. When the `nonRotatable` attribute is present and set to 1, it prevents the router from rotating the via.

`upperLayerEncHeight`

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

`upperLayerEncWidth`

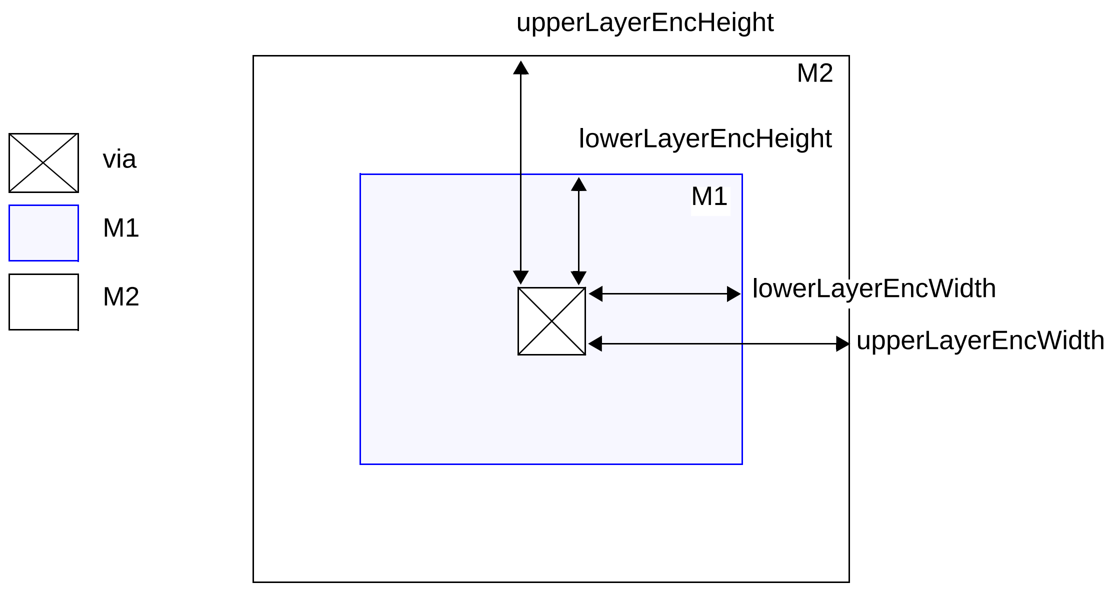
The distance at which the first of two layers encloses the via in the horizontal dimension.

`viaFarmSpacing`

The minimum separation distance between two via arrays.

Figure 1-9 shows the interpretation of the `upperLayerEncHeight`, `lowerLayerEncHeight`, `upperLayerEncWidth`, and `lowerLayerEncWidth` attributes when the lower layer is M1 and the upper layer is M2.

Figure 1-9 Via Enclosure Attributes



Parasitic Attributes

The parasitic attributes define the layer parasitics. In general, the tool 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.

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.) For more information about these design rules, see Routing Design Rules. In the IC Compiler II tool, you can define design rules by using the `create_design_rule` command.

Note:
Layer-specific design rules are defined in a `Layer` section. For more information, see Design Rule Attributes.

These are some of the attributes that can be used in the `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)
     endOfLineEnclosure = 0
     endOfLineViaJogLength = 0
     endOfLineViaJogWidth = 0
```

```
        endOfLineViaEncWidth = 0
        stackable = 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)
}
```

For more information about these types of design rules, see Routing Design Rules.

PRRule Section

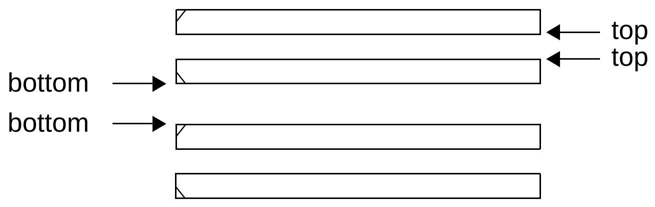
The `PRRule` section of a technology file defines cell row spacing. The cell row spacing rules differ, depending on whether you are using double-back cell rows. In the IC Compiler II tool, you can define additional cell row spacing rules by using the `create_pr_rule` command.

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.

`abuttableTopTop`

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.

`abuttableBotBot`

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 1-8 shows a typical `PRRule` section for double-back cell rows.

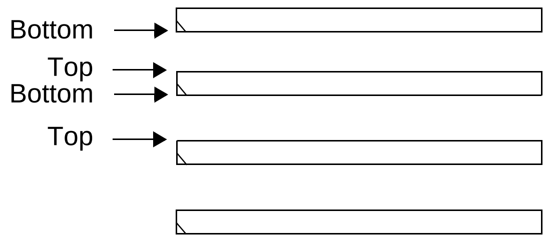
Example 1-8 PRRule Section for Double-Back Cell Rows

```
PRRule {
     rowSpacingTopTop = 2.8
     rowSpacingBotBot = 1.4
     abuttableTopTop = 1
     abuttableBotBot = 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.

`abuttableTopBot`

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 1-9 shows a `PRRule` section for non-double-back cell rows.

Example 1-9 PRRule Section for Non-Double-Back Cell Rows

```
PRRule {
     rowSpacingTopBot = 2.8
     abuttableTopBot = 0
}
```

DensityRule Section

A `DensityRule` section defines the metal density rules. In the IC Compiler II tool, these rules are used by the `signoff_create_metal_fill` and `signoff_report_metal_density` commands. In the IC Compiler tool, these rules are 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. In the IC Compiler II tool, you can define additional density rules by using the `create_density_rule` command.

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.

Example 1-10 shows an example of a `DensityRule` section.

Example 1-10 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 in the IC Compiler tool for a specific metal layer. (The IC Compiler II tool has no need for this command.) The technology file can have a `SlotRule` section for each metal layer.

This is the slot rule syntax:

```
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
}
```

These are the slot rule attributes:

`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`

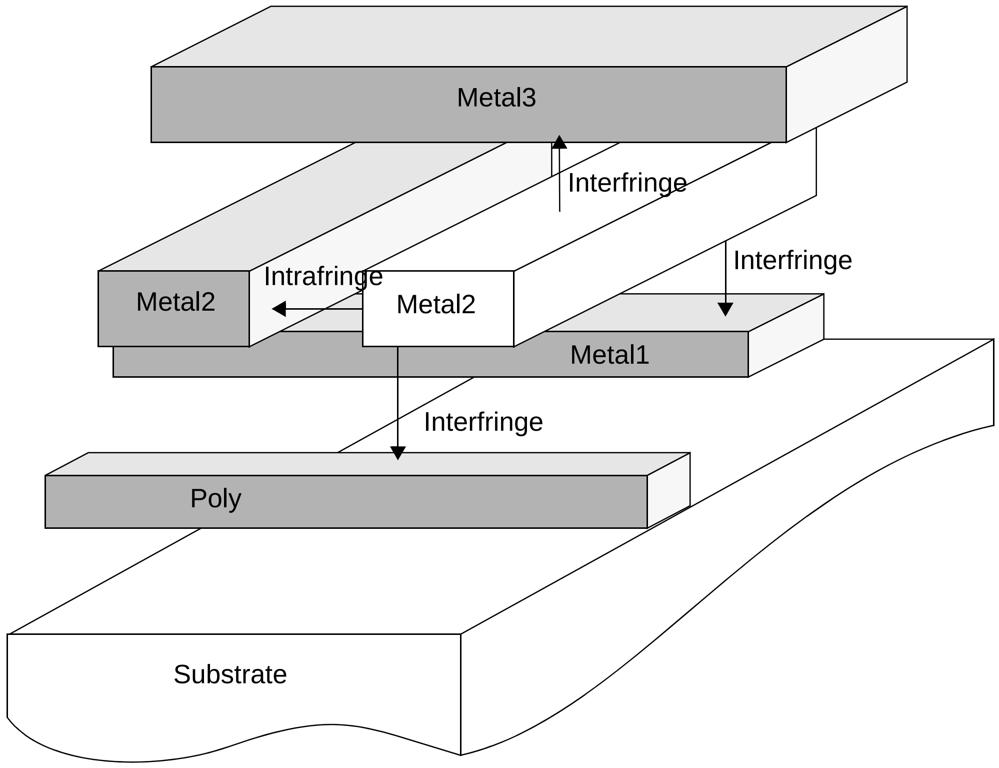
The maximum metal density allowed for a slotted wire.

FringeCap Section

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

- Capacitance per unit area when objects on different layers overlap (interfringe in Figure 1-12)
- Capacitance per unit length when objects on the same layer are separated by the minimum spacing specified in the `Layer` section (intrafringe in Figure 1-12)

Figure 1-12 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, the tool 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 1-11 shows a typical interfringe `FringeCap` section.

Example 1-11 FringeCap Section (Interfringe)

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

Example 1-12 shows a typical intrafringe `FringeCap` section.

Example 1-12 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, the tool gets the parasitic information from the TLUPlus files rather than the technology file. The `capModel`/`capTable` sections support older parasitic data methods.

For the IC Compiler tool, you can create these two sections either manually with a text editor or by using the Milkyway Environment `cmCreateCapModel` command. For details, see the *Library Data Preparation for IC Compiler User Guide*.

Example 1-13 is an example of a `CapModel` section.

Example 1-13 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 1-14 is an example of a `CapTable` section.

Example 1-14 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 1-15 is an example of a table lookup capacitance model.

Example 1-15 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, the tool 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 1-16 shows a `ResModel` section.

Example 1-16 ResModel Section

```
ResModel "metal1ResModel" {
        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)
}
```

2

Routing Design Rules

This topic describes the routing design rules supported by Zroute. Unless otherwise noted, these design rules are also supported by the classic router in the IC Compiler tool. Routing design rules that are supported only by the classic router are described in Appendix A, “Classic Router Design Rules.”

This chapter contains the following sections:

Routing Mode Rules

The routing mode rules are described in the following topics:

- No Nonpreferred Direction Routing Rule
- Color-Based Forbidden Routing Rule
- On-Grid Routing Rule
- Rectangle-Only Rule

No Nonpreferred Direction Routing Rule

By default, routing can run in the preferred or nonpreferred direction. To restrict the routing on a layer to its preferred direction, set the `nonPreferredRouteMode` attribute in the `Layer` section to 1.

The syntax for this rule is

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

See Also

- No Vias on Nonpreferred Direction Rule

Color-Based Forbidden Routing Rule

Note:
 This rule is supported only by Zroute.

The color-based forbidden routing rule disables signal routing on a specific color or all colors of a metal layer. This includes signal routing shapes, wires, via enclosures, and routing shapes connected to pin shapes.

To enable this rule for a layer, set the `forbidSignalRoute` attribute in the `Layer` section to one of the following values: `mask1`, `mask2`, or `all`. If you specify `all`, it can apply to a double-patterning or single-patterning layer.

The syntax for this rule is

```
Layer "MetalX" {
   forbidSignalRoute = color   # possible input: mask1, mask2, all
}
```

On-Grid Routing Rule

The on-grid routing rule controls whether the router must align the metal shape centerlines and via centers to the wire track grid.

To force the alignment of metal shape centerlines or via centers to the wire track grid, set the `onWireTrack` attribute in the `Layer` section to 1.

The syntax for this rule is

```
Layer "MetalX_orViaX" {
   onWireTrack = 1
}
```

When defined for a metal layer, this rule applies to both metal routes and metal enclosures associated with vias. When defined for a via layer, this rule applies only to the vias on that layer, and not to the via enclosures associated with the vias.

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 routes and not via enclosures, and the alignment characteristic specified for a via layer applies to the via 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.

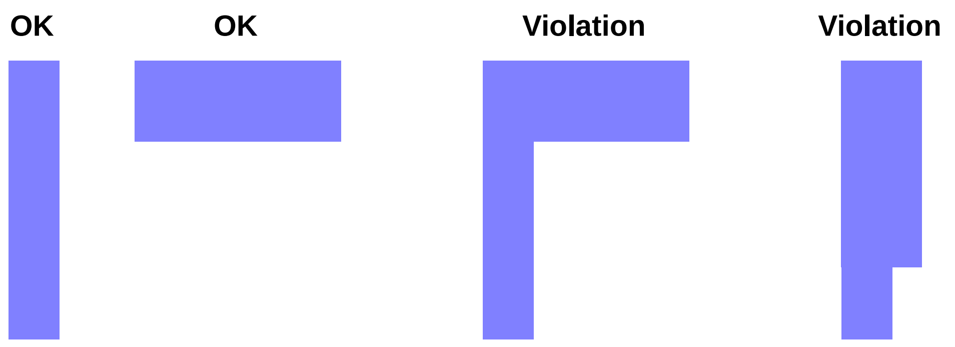
Note that the `onGrid` attribute, like `onWireTrack`, controls the alignment of routes and vias to the wire track grid, not the manufacturing database grid.

Rectangle-Only Rule

Note:
  This rule is supported only by Zroute.

The rectangle-only rule specifies that a metal layer allows only rectangular shapes. Any jog or rectilinear shape is a design rule violation. Figure 2-1 shows some examples.

Figure 2-1 Rectangle-Only Rule



To enable this rule for a layer, set the `hasRectangleOnly` attribute in the `Layer` section to 1.

The syntax for this rule is

```
Layer "MetalX" {
   hasRectangleOnly = 1
}
```

Minimum Area Rules

The minimum area rules are described in the following topics:

- Minimum Length Rule
- General Minimum Area Rule
- Special Minimum Area Rule
- Width-Based Minimum Area Rule
- Color-Based Minimum Area Rule
- Discrete Metal Dimensions Rule
- Dense Wire Minimum Dimensions Rule

Minimum Length Rule

The minimum length rule defines 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 longest dimension of the bounding box that encloses the entire geometry must meet this rule.

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

The syntax for this rule is

```
Layer "MetalX" {
   minLength = Lmin
}
```

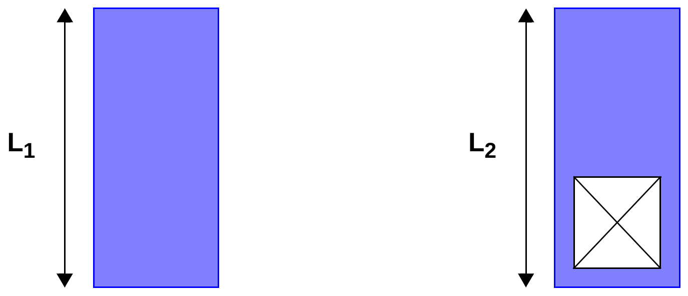
By default, the minimum length rule is enforced for all shapes, irrespective of vias. To change this behavior, set the `minLengthMode` attribute to 1 in the `Technology` section, which causes the check to be performed only on segments that fully enclose vias:

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

Note:
The minimum length is specified per layer; however, the minimum length mode applies to all metal layers.

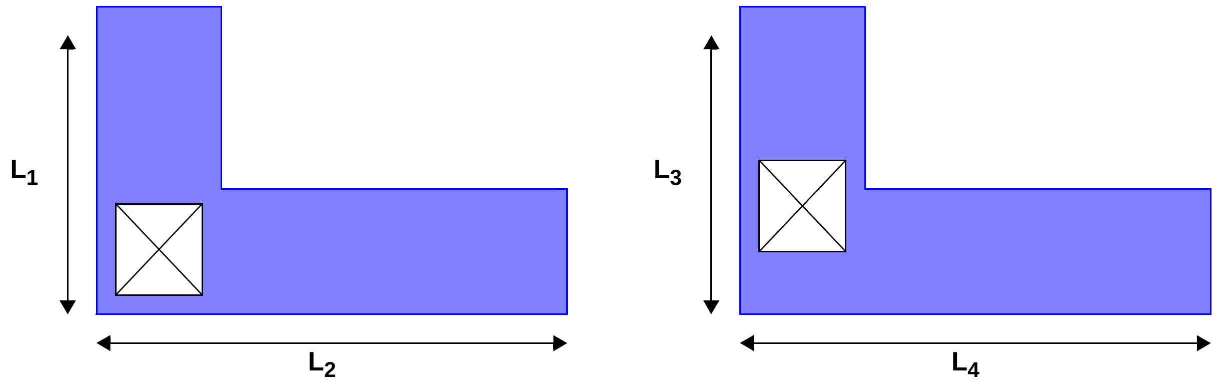
In Figure 2-2, by default, both L_1 and L_2 must be at least the minimum length, L_{min}, specified by the `minLength` attribute. However, if the `minLengthMode` attribute is set to 1, L_1 is not checked because it does not enclose a via.

Figure 2-2 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 on a given layer, the minimum length check is performed only on the longest segment. For example, in Figure 2-3, 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 2-3 Minimum Length Rule for Multiple Wire Segments Enclosing a Via



General Minimum Area Rule

The general minimum area rule defines the minimum area for all geometries.

The syntax for this rule is

```
Layer "MetalX" {
   minArea = A_min
}
```

See Also

- Special Minimum Area Rule

Special Minimum Area Rule

A special minimum area rule defines a minimum area for special shapes that is larger than the general minimum area. Defining a different minimum area 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.

The technology file supports several special minimum area rules. Typically a technology file would contain only one of these rules, depending on your technology:

- Two-Stage Special Minimum Area Rule
- Edge-Based Multistage Minimum Area Rule
- Rectangle-Based Multistage Minimum Area Rule

Zroute always honors the special minimum area rules defined in the technology file. However, for the classic router, you must enable the special minimum area rules by setting the `minAreaMode` attribute to 1 in the `Technology` section. By default, it is set to 0 and the classic router checks only the general minimum area (`minArea`) rule. With the attribute set to 1, the classic router honors the special minimum area rules for special shapes and the general minimum area rule for ordinary shapes.

See Also

- General Minimum Area Rule

Two-Stage Special Minimum Area Rule

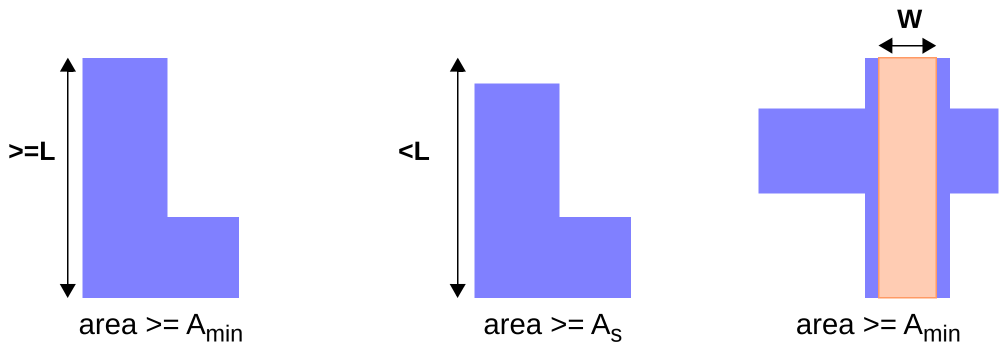
The two-stage special minimum area rule defines a special minimum area, A_s, that applies to rectangular and rectilinear polygons that meet both of the following conditions:

- All edge lengths are less than a specified threshold, L
- It cannot cover a rectangle measuring L by W

The special minimum area, A_s, must be greater than the general minimum area, A_{min}. If both of these conditions are not met, the special minimum area rule is waived and the general minimum area rule applies to the polygon.

In Figure 2-4, the leftmost polygon has at least one edge equal to L, so A_{min} applies; the middle polygon has all edges less than L, so A_s applies; and the rightmost polygon has all edges less than L but can cover an L by W rectangle, so A_{min} applies.

Figure 2-4 Two-Stage Special Minimum Area Rule



The syntax for this rule is

```
Technology {
   minAreaMode = 1 # required only for the classic router
}
```

```
Layer "MetalX"  {
   minWidth                = W
   minArea                 = $A_{min}$
   minAreaEdgeThreshold = L
   specialMinArea          = $A_s$
}
```

Note:
For Zroute, you should define the two-stage special minimum area rule using the syntax for the edge-based multistage minimum area rule, as described in Edge-Based Multistage Minimum Area Rule.

Edge-Based Multistage Minimum Area Rule

Note:
This rule is supported only by Zroute.

The edge-based multistage minimum area rule is an extension of the edge-based special minimum area rule that defines a set of minimum areas that depend on the lengths of the polygon edges.

For example, for a three-stage edge-based minimum area rule, you define a table size of two and specify two length thresholds, L_1 and L_2. This creates three ranges of polygon edge lengths: short (less than L_1), medium (L_1 to less than L_2), and long (greater than or equal to L_2). There are three minimum area values: A_{min}, A_1, and A_2. A_{min} is the smallest value, A_1 is the largest value, and A_2 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 A_{min}, the smallest of the three area values.

- Stage two

  A polygon must have an area of at least A_1, the largest of the three areas, if it meets both of the following conditions:

  - All edge lengths are less than L_1
  - It cannot cover a rectangle measuring L_1 by W_1

- Stage three

  A polygon must have an area of at least A_2, the intermediate area value, if it meets both of the following conditions:

  - All edge lengths are less than L_2
  - It cannot cover a rectangle measuring L_2 by W_2

The syntax for the edge-based three-stage minimum area rule is

```
Layer "MetalX" {
   minArea                 = $A_{min}$
   specialMinAreaTblSize   = 2
   minAreaEdgeThresholdTbl = ($L_1$, $L_2$)
   minAreaFillMinLengthTbl = ($L_1$, $L_2$)
   minAreaFillMinWidthTbl  = ($W_1$, $W_2$)
   specialMinAreaTbl       = ($A_1$, $A_2$)
}
```

To use this syntax to specify the edge-based special minimum area rule, set the table size to 1. For example,

```
Layer "MetalX" {
   minArea                 = $A_{min}$
   specialMinAreaTblSize   = 1
   minAreaEdgeThresholdTbl = (L)
   minAreaFillMinLengthTbl = (L)
   minAreaFillMinWidthTbl  = (W)
   specialMinAreaTbl       = ($A_S$)
}
```

To base the minimum area only on the edge length and not the rectangle coverage, specify an invalid value such as -1 for the length and width attributes. For example,

```
Layer "MetalX" {
   minArea                 = $A_{min}$
   specialMinAreaTblSize   = 1
   minAreaEdgeThresholdTbl = (L)
   minAreaFillMinLengthTbl = (-1)
   minAreaFillMinWidthTbl  = (-1)
   specialMinAreaTbl       = ($A_S$)
}
```

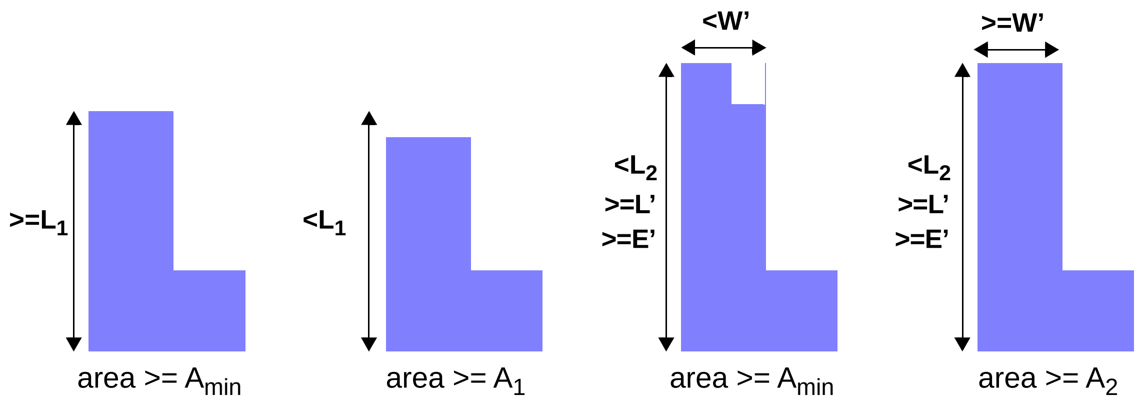
For advanced geometries, the rule includes additional attributes to modify the stage-three portion of the area check. The minimum area is A_2 for a polygon with any edge length greater than or equal to L’ and with all edge lengths less than L_2, where L’ is a length threshold between L_1 and L_2, inclusive. 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                           = Amin
  specialMinAreaTblSize             = 2
  minAreaEdgeThresholdTbl           = (L1, L2)
  minAreaMinEdgeThresholdTbl        = (0, L')
  minAreaFillMinLengthTbl           = (L1, L2)
  minAreaFillMinWidthTbl            = (W1, W2)
  specialMinAreaTbl                 = (A1, A2)
  minAreaMinEdgeLengthExcludedTbl   = (0, E')
  minAreaAdjacentEdgeLengthExcludedTbl = (0, W')
}
```

Figure 2-5 illustrates the usage of the supplemental attributes.

Figure 2-5 Edge-Based Multistage Minimum Area Rule for Advanced Geometries



Rectangle-Based Multistage Minimum Area Rule

Note:
   This rule is supported only by Zroute.

The rectangle-based multistage minimum area rule defines a set of minimum areas for rectangles. A rectangle that can completely cover another rectangle of specified dimensions has a smaller minimum area requirement.

For example, for a three-stage rectangle-based minimum area rule, you define a table size of two and specify two length thresholds, L_1 and L_2 and two width thresholds, W_1 and W_2. There are three minimum area values: A_{min}, A_1, and A_2. A_{min} is the largest value, A_1 is an intermediate value, and A_2 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 A_{min}, the largest of the three area values, unless the stage two or stage three conditions apply.

- Stage two

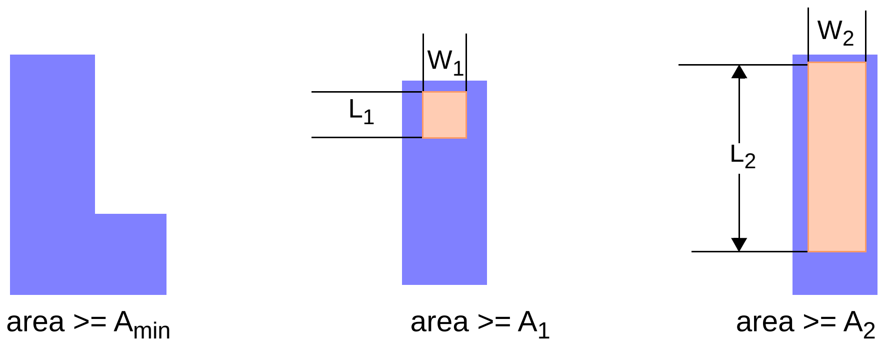
  Any rectangular geometry that can completely cover a rectangle measuring L_1 by W_1 must have an area of at least A_1, unless the stage three condition applies. A_1 must be smaller than A_{min}. To apply the smaller minimum area, A_1, to any rectangular geometry, specify L_1 and W_1 as 0.

- Stage three

  Any rectangular geometry that can completely cover a rectangle measuring L_2 by W_2 must have an area of at least A_2. A_2 must be smaller than A_1.

In Figure 2-6, the leftmost polygon is not rectangular, so A_{min} applies; the middle rectangle completely covers an L_1 by W_1 rectangle, so A_1 applies; and the rightmost rectangle completely covers an L_2 by W_2 rectangle, so A_2 applies.

Figure 2-6 Rectangle-Based Special Minimum Area Rule



The syntax for the rectangle-based three-stage minimum area rule is

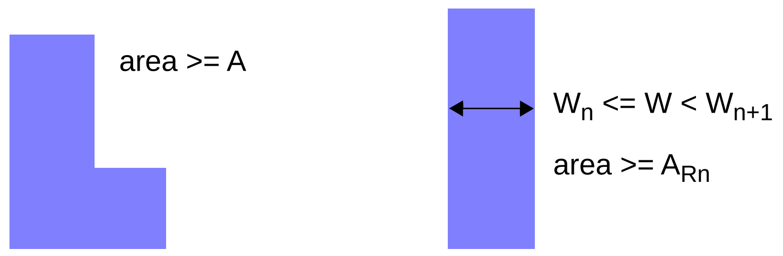
```
Layer "MetalX" {
   minArea                  = Amin
   specialMinAreaTblSize    = 2
   minAreaRectMinLengthTbl = (L1, L2)
   minAreaRectMinWidthTbl  = (W1, W2)
   specialMinAreaTbl        = (A1, A2)
}
```

Width-Based Minimum Area Rule

Note:
This rule is supported only by Zroute.

The width-based minimum area rule defines a set of minimum areas for rectangles based on the rectangle's smallest dimension (width). The rule specifies a separate minimum area value that applies to all nonrectangular areas. See Figure 2-7.

Figure 2-7 Width-Based Minimum Area Rule



In the following example, the minimum allowable area for any nonrectangular geometry is A. For rectangular metal, the minimum area is

- A_{R0} for widths greater than or equal to W_0 and less than W_1
- A_{R1} for widths greater than or equal to W_1 and less than W_2
- A_{R2} for widths greater than or equal to W_2

```
Layer "MetalX" {
   nonRectMinArea               = A
   rectMinAreaTblSize           = 3
   rectMinAreaWidthThrehsoldTbl = (W0, W1, W2)
   rectMinAreaTbl               = (AR0, AR1, AR2)
}
```

Color-Based Minimum Area Rule

Note:
  This rule is supported only by Zroute.

The color-based minimum area rule defines a minimum area required for shapes in a specific color of a metal layer.

The syntax for this rule is

```
Layer "MetalX" {
   maskMinAreaTblSize       = 2
   maskMinAreaMaskNumTbl    = (1, 2)   # mask numbers
   maskMinAreaTbl           = (A1, A2) # corresponding minimum areas
}
```

If the specified mask number is 1, the minimum area is A_1. If the mask number is 2, the minimum area is A_2.

Discrete Metal Dimensions Rule

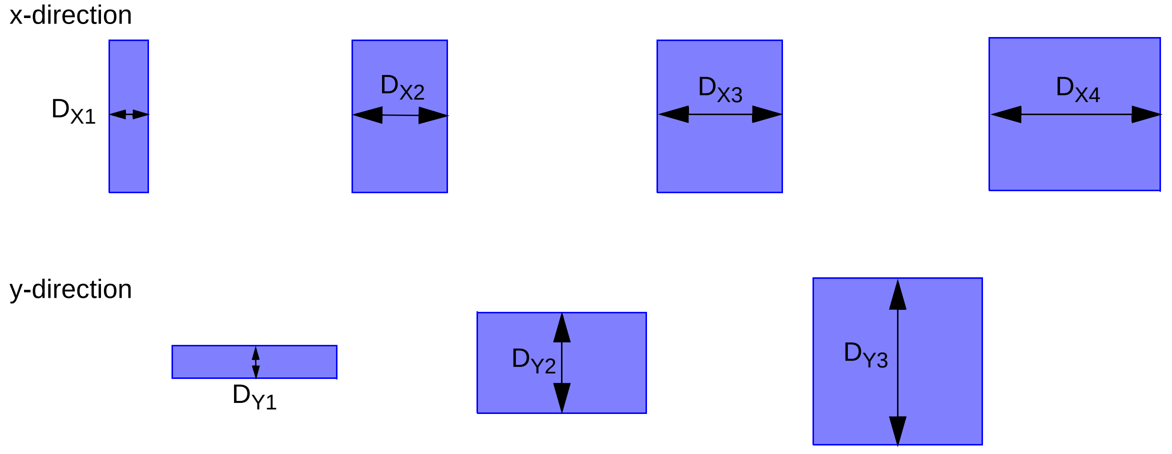
Note:
  This rule is supported only by Zroute.

The discrete metal dimensions rule defines a list of discrete dimensions allowed for a metal layer. Only the exact dimensions are allowed for any shape on the metal layer up to the last value in the list, after which the rule has no restrictions on the allowed dimensions. In the rule, you specify the number of allowed dimensions and the list of dimensions, measured in each direction. For example,

```
Layer "MetalX" {
   xLegalDimTblSize = 4
   xLegalDimTbl     = (DX1, DX2, DX3, DX4)
   yLegalDimTblSize = 3
   yLegalDimTbl     = (DY1, DY2, DY3)
}
```

Figure 2-8 shows the allowed dimensions in the x- and y-directions, as specified in the previous example.

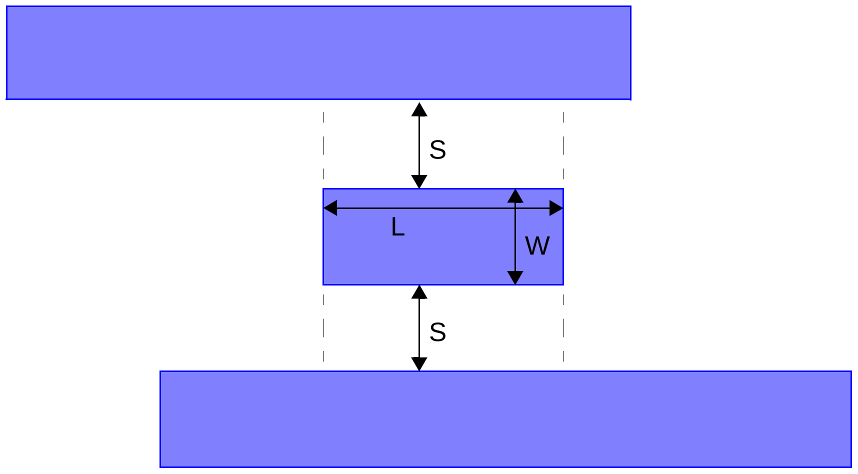
Figure 2-8 Discrete Metal Dimensions Rule



Dense Wire Minimum Dimensions Rule

The dense wire minimum dimensions rule defines the minimum length and width of a metal rectangle between closely space wires, like the rectangle shown in Figure 2-9. When the space between a rectangle and either nearby wire is less than the value S, the rectangle must have a length of at least L or a width of at least W.

Figure 2-9 Dense Wire Minimum Dimensions Rule



The syntax for this rule is

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

Minimum Enclosed Area Rules

The minimum enclosed area rules are described in the following topics:

- Minimum Enclosed Area Rule
- Minimum Enclosed Width Rule

Minimum Enclosed Area Rule

The minimum enclosed area rule defines the minimum area enclosed by ring-shaped wires or vias on a layer.

The syntax for this rule is

```
Layer "MetalX" {
   minEnclosedArea       = A
}
```

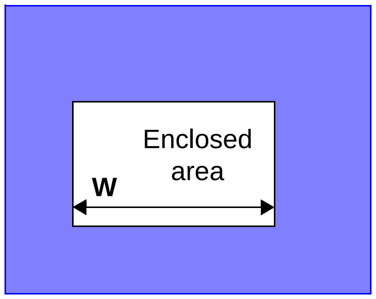
See Also

- Fat Metal Minimum Enclosed Area Rule

Minimum Enclosed Width Rule

The minimum enclosed width rule specifies the minimum width (largest dimension) of an enclosed area for a layer, as shown in Figure 2-10.

Figure 2-10 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.

The syntax for this rule is

```
Layer "MetalX" {
  minEnclosedWidth = W
}
```

Metal Width Rules

The following rules restrict the allowed widths of metal routes:

- Minimum Width Rule
- Diagonal Concave Corner Minimum Width Rule
- Default Width Rule
- Discrete Metal Widths Rule
- Signal Routing Maximum Metal Width Rule

Minimum Width Rule

The minimum width rule defines the minimum width of a metal shape.

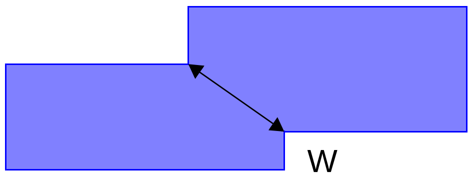
At a minimum, you must define a minimum width for each metal layer by setting the `minWidth` attribute. You can define a different minimum width for the nonpreferred direction by setting the `nonPreferredWidth` attribute in addition to the `minWidth` attribute. If used, the `nonPreferredWidth` setting overrides the `minWidth` setting for routes in the nonpreferred direction.

The syntax for this rule is

```
Layer "MetalX" {
   minWidth = Wmin
   [nonPreferredWidth = WN]
}
```

By default, for a metal jog, the minimum width is measured diagonally at the narrowest point, as shown in Figure 2-11.

Figure 2-11 Default Minimum Width Measurement

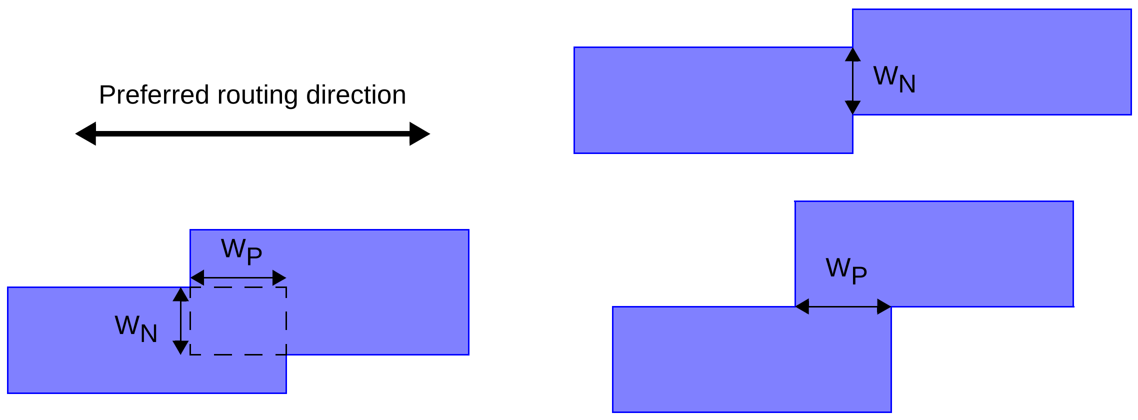


If you are using Zroute, you can use Manhattan distance instead of diagonal distance to measure the minimum width by setting the `minWidthCheckManhattan` attribute to 1. When you set this attribute to 1, you must set both the `minWidth` and `nonPreferredWidth` attributes, as shown in the following example:

```
Layer "MetalX" {
   minWidth               = WP
   nonPreferredWidth      = WN
   minWidthCheckManhattan = 1
}
```

The rule is satisfied if either one of the two width requirements is satisfied. In Figure 2-12, the width measured in the preferred direction must be at least W_P or the width measured in the nonpreferred direction must be at least W_N.

Figure 2-12 Orthogonal Minimum Width Rule



See Also

- Default Width Rule

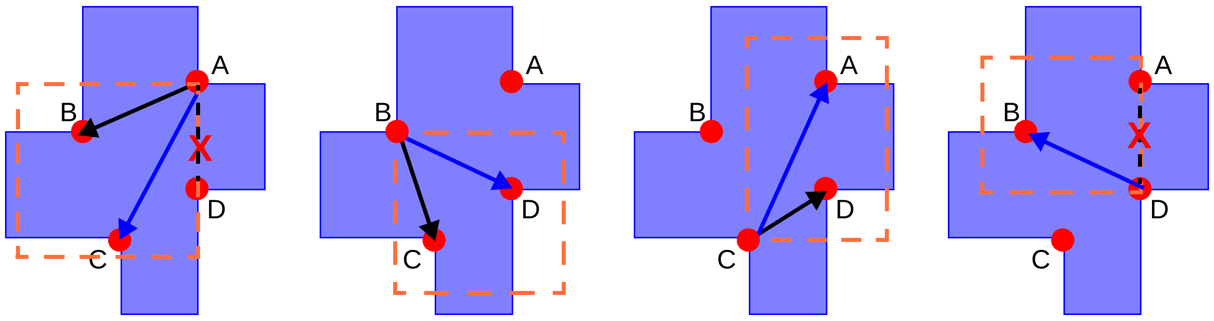
Diagonal Concave Corner Minimum Width Rule

Note:
This rule is supported only by Zroute.

The diagonal concave corner minimum width rule defines the minimum width of metal between two diagonally opposite concave corners of a metal polygon.

In Figure 2-13, 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 dashed rectangle. The rule specifies the minimum metal width measured from each point to any other point inside of 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 2-13 Diagonal Concave Corner Width Rule



The syntax for this rule is

```
Layer "MetalX" {
   minWidthForConcaveCorner = W
}
```

Default Width Rule

The default width rule defines the default width of a signal route.

At a minimum, you must define the default width for a metal layer by setting the `defaultWidth` attribute. You can define different default widths for the x- and y-directions by setting the `xDefaultWidth` and `yDefaultWidth` attributes in addition to the `defaultWidth` attribute. If used, the `xDefaultWidth` and `yDefaultWidth` settings override the `defaultWidth` setting in the relevant direction.

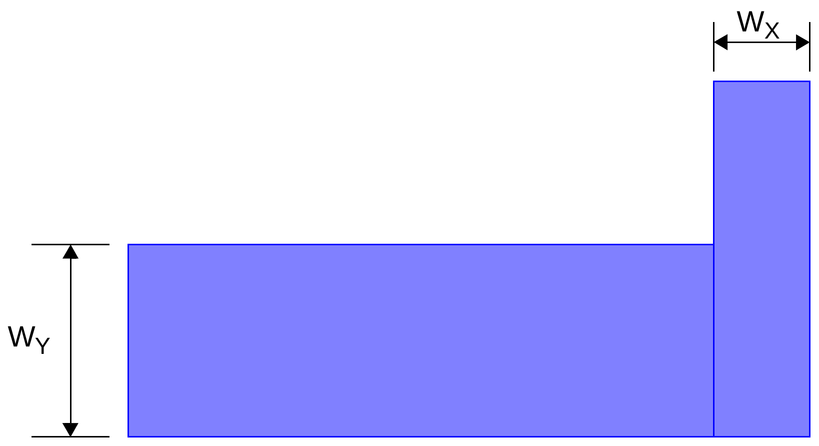
The syntax for this rule is

```
Layer "MetalX" {
   defaultWidth  = W
   [xDefaultWidth = W_X]
   [yDefaultWidth = W_Y]
}
```

Note that the `defaultWidth`, `xDefaultWidth`, and `yDefaultWidth` attributes specify the default width used 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.

When you specify the `xDefaultWidth` and `yDefaultWidth` attributes, the router uses these widths by default, as shown in Figure 2-14.

Figure 2-14 Different Default Metal Widths in the X-Direction and Y-Direction



See Also

- Minimum Width Rule

Discrete Metal Widths Rule

The discrete metal widths rule defines a limited set of allowed widths for any shape on a given layer. A separate list is specified for the x-direction and y-direction. One of the legal widths in the each list must match the default width defined for that direction.

By default, only the discrete widths specified in the lists are allowed. To allow any width beyond the last value in each list, set the `legalWidthBeyondLastValue` attribute to 1.

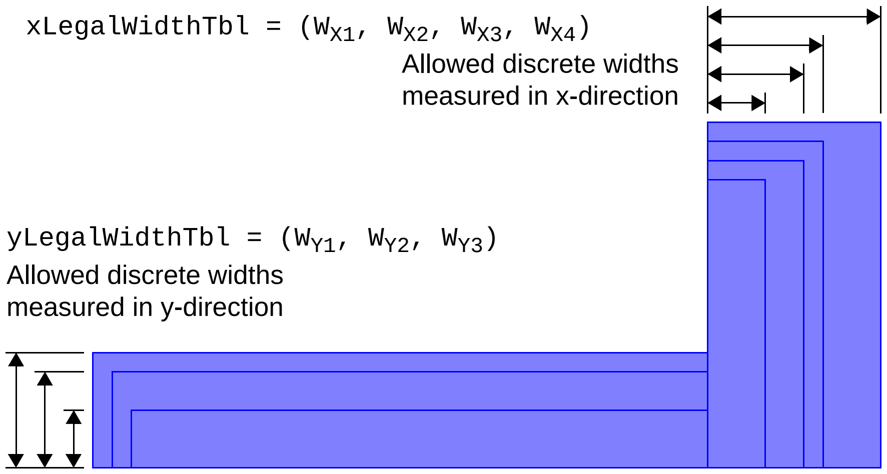
A square shape on a metal layer is treated as a wire running in the preferred direction. If the preferred direction is horizontal, the square is checked with the `xLegalWidthTbl` list. If the preferred direction is vertical, the square is checked with the `yLegalWidthTbl` list. To treat square shapes as running in the nonpreferred direction instead, set the `legalWidthCheckNonPrefForSquare` attribute to 1.

The following example shows the syntax for this rule:

```
Layer "MetalX" {
   xLegalWidthTblSize = 4
   yLegalWidthTblSize = 3
   xLegalWidthTbl     = (WX1, WX2, WX3, WX4)
   yLegalWidthTbl     = (WY1, WY2, WY3)
   [legalWidthBeyondLastValue = 1]
   [legalWidthCheckNonPrefForSquare = 1]
}
```

Figure 2-15 shows the allowed widths defined by this rule, assuming that the `legalWidthBeyondLastValue` attribute is 0.

Figure 2-15 Discrete Metal Widths Rule


xLegalWidthTbl = (W_{X1}, W_{X2}, W_{X3}, W_{X4})
Allowed discrete widths measured in x-direction
yLegalWidthTbl = (W_{Y1}, W_{Y2}, W_{Y3})
Allowed discrete widths measured in y-direction

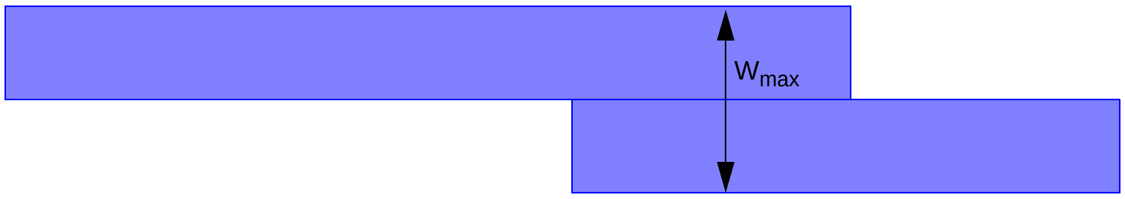
See Also

- Default Width Rule

Signal Routing Maximum Metal Width Rule

The maximum metal width rule defines the maximum width of a signal route on a given layer, as shown in Figure 2-16.

Figure 2-16 Signal Routing Maximum Metal Width Rule



The syntax for this rule is

```
Layer   "MetalX" {
   signalRouteMaxWidth = Wmax
}
```

This rule applies only to signal routing; it does not apply to power supply and ground routing.

Minimum Edge Rules

The minimum edge rules are described in the following topics:

Short Edge Rules

Short edge rules are rules that apply to edges that are less than or equal to the length specified by the `minEdgeLength` attribute for a given layer. You can specify the following short edge rules:

- Maximum total number of short edges

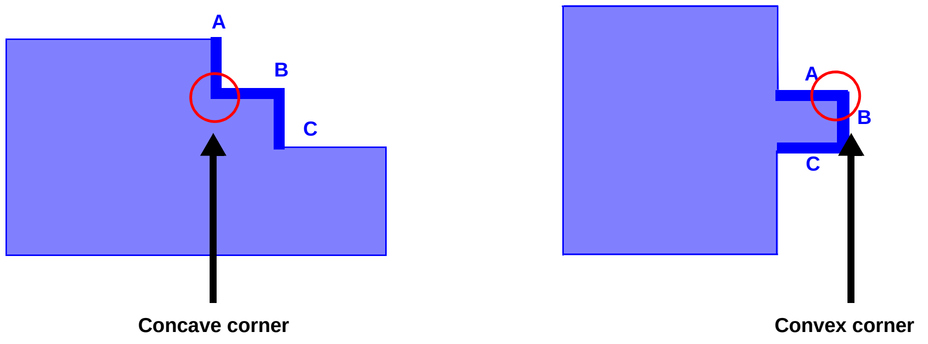
  To define the maximum number of consecutive short edges, set the `maxNumMinEdge` attribute.

- Maximum total short edge length

  To define the maximum total length of consecutive short edges, set the `maxTotalMinEdgeLength` attribute.

The scope of these rules is controlled by the `minEdgeMode` attribute in the `Technology` section. 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 2-17, assume edges A, B, and C are all short edges. For the example on the left, the rules are checked regardless of the `minEdgeMode` setting, because it contains a concave corner. For the example on the right, which does not contain a concave corner, the rules are checked only when `minEdgeMode` is 1.

Figure 2-17 Minimum Edge Examples

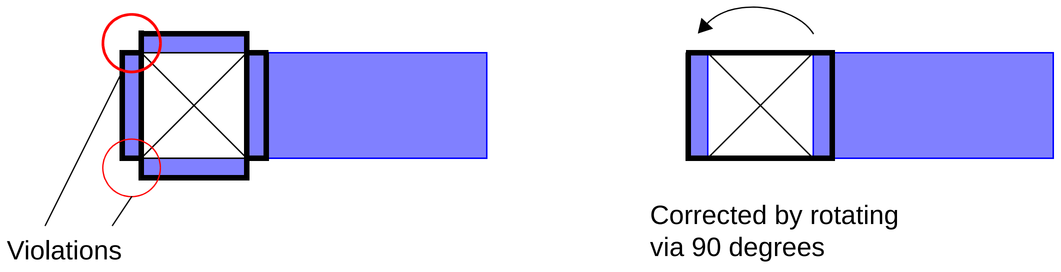


The syntax for these rules is

```
Technology {
   minEdgeMode = 0 | 1
}
Layer "MetalX" {
   minEdgeLength = L_min
   maxNumMinEdge = N
   maxTotalMinEdgeLength  = L_total
}
```

Figure 2-18 shows an example application of the maximum total short edge length 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 2-18 Total Short Edge Length Rule



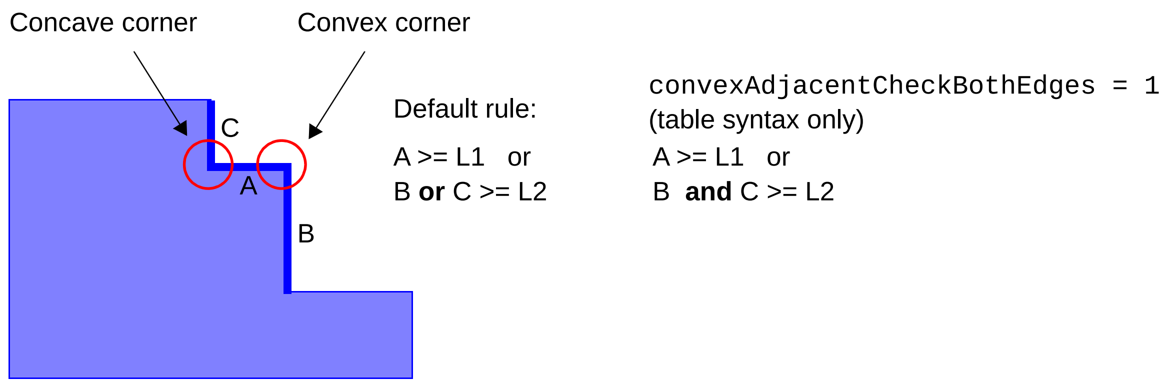
Convex-Concave Minimum Edge Length Rule

The convex-concave minimum edge length rule defines the minimum length for a metal edge that connects a concave corner to a convex corner.

To satisfy the rule, one of the following conditions, which are illustrated in Figure 2-19, must be met:

- The length of the connecting edge, A, is greater than or equal to L1
- The length of an adjacent edge, B or C, is greater than or equal to L2

Figure 2-19 Concave-Convex Minimum Length Rule



The syntax for the basic rule is

```
Layer "MetalX" {
   convexConcaveMinEdgeLength    = L1
   convexMinEdgeLength           = L2
}
```

For advanced nodes, specify this rule by using the following table syntax:

```
Layer "MetalX" {
   convexConcaveMinEdgeLengthTblSize     = 2
   convexConcaveMinEdgeLengthTbl         = (L1_1, L1_2)
   convexConcaveAdjacentMinEdgeLengthTbl = (L2_1, L2_2)
   convexAdjacentCheckBothEdges          = 0
}
```

By default, only one adjacent edge must have a length of at least L2. When you use the table syntax and set the `convexAdjacentCheckBothEdges` attribute to 1, both adjacent edges must have a length of at least L2.

Note:
   The table syntax is supported only by Zroute.

Two-Convex-Corner Minimum Edge Length Rule

Note:
This rule is supported only by Zroute.

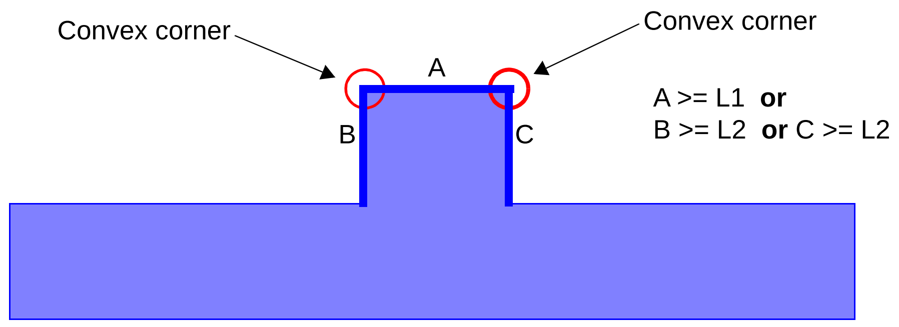
The convex-concave minimum edge length rule defines the minimum length for a metal edge that connects two convex corners.

To satisfy the rule, one of the following conditions, which are illustrated in Figure 2-20, must be met:

- The length of the connecting edge, A, is greater than or equal to L1
- The length of an adjacent edge, B or C, is greater than or equal to L2

The two-convex-corner minimum edge length rule specifies the minimum lengths of edges adjacent to two consecutive convex corners of a metal polygon.

Figure 2-20 Two-Convex-Corner Minimum Edge Length Rule



This is a table-based rule that allows one set or multiple sets of L1 and L2 values. The syntax for a single-set rule is

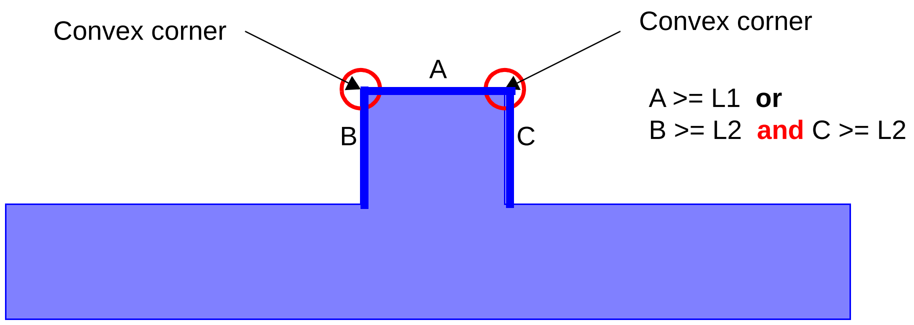
```
Layer "MetalX" {
   convexConvexMinEdgeLengthTbl   = L1
   convexAdjacentMinEdgeLengthTbl = L2
   convexAdjacentCheckBothEdges   = 0
}
```

The syntax for a double-set rule is

```
Layer "MetalX" {
   convexConvexMinEdgeLengthTblSize = 2
   convexConvexMinEdgeLengthTbl     = (L1_1, L1_2)
   convexAdjacentMinEdgeLengthTbl   = (L2_1, L2_2)
   convexAdjacentCheckBothEdges     = 0
}
```

By default, only one adjacent edge must have a length of at least L2. When you set the `convexAdjacentCheckBothEdges` attribute to 1, both adjacent edges must have a length of at least L2. In that case, the rule is called the line-end edge length rule, which is shown in Figure 2-21.

Figure 2-21 Line-End Edge Length Rule



Line-End Concave Corner Length Rule

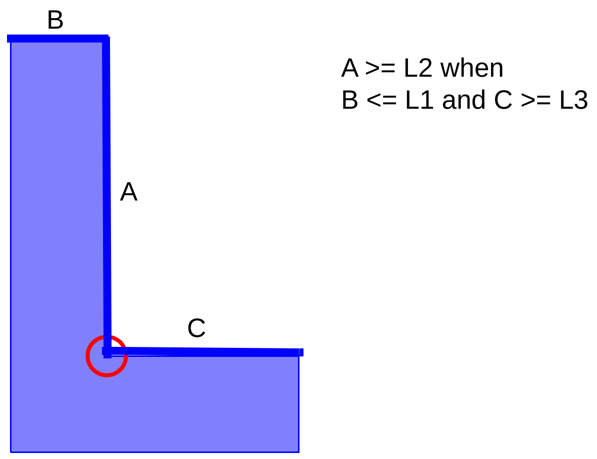
Note:

This rule is supported only by Zroute.

The line-end concave corner length rule defines the minimum length for a metal edge that connects a concave corner to a line-end edge.

To satisfy this rule, the length of the connecting edge, A, must be greater than or equal to L2 when the line-end edge, B, is less than or equal to L1 and the adjacent edge, C, is greater than or equal to L3, as shown in Figure 2-22.

Figure 2-22 Line-End Concave Corner Length Rule



The syntax for this rules is

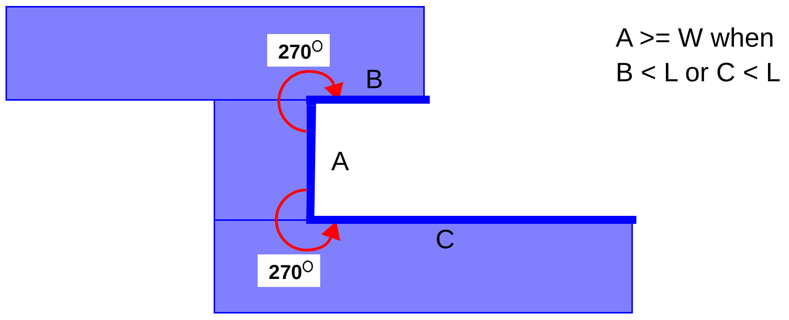
```
Layer "MetalX" {
   doubleConvexEdgeLengthTblSize               = 1
   doubleConvexMinEdgeLengthTbl                = (L1)
   doubleConvexAdjacentMinEdgeLengthTbl        = (L2)
   doubleConvexConcaveAdjacentMaxEdgeLengthTbl = (L3)
}
```

Special Notch Rule

The special notch rule defines the minimum width of a notch.

To satisfy this rule, the width of the notch, A, must be greater than or equal to W when at least one of the adjacent edges, B or C, is less than L, as shown in Figure 2-23.

Figure 2-23 Special Notch Rule



The syntax for this rule is

```
Layer   "MetalX"  {
   minEdgeLength2 = L
   minEdgeLength3 = W
}
```

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

See Also

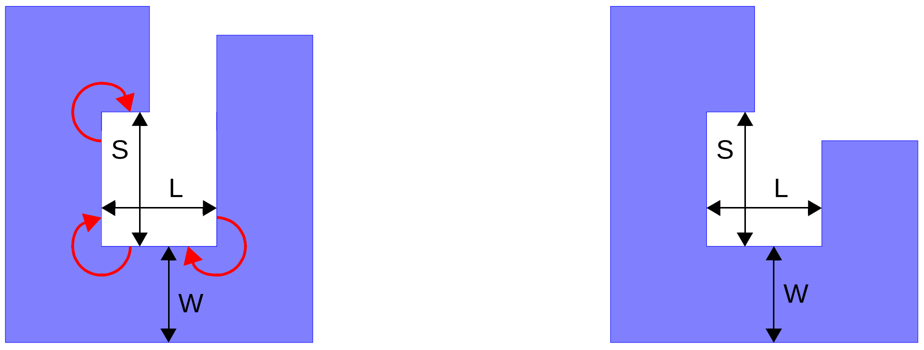
- Hook Rule

Hook Rule

The hook rule defines 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 2-24. This rule is an extension of the special notch rule.

To satisfy this rule, the inside height of the hook must be at least S 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. This rule applies to both of the cases shown in Figure 2-24.

Figure 2-24 Hook Rule Examples



The syntax for this rule is

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

See Also

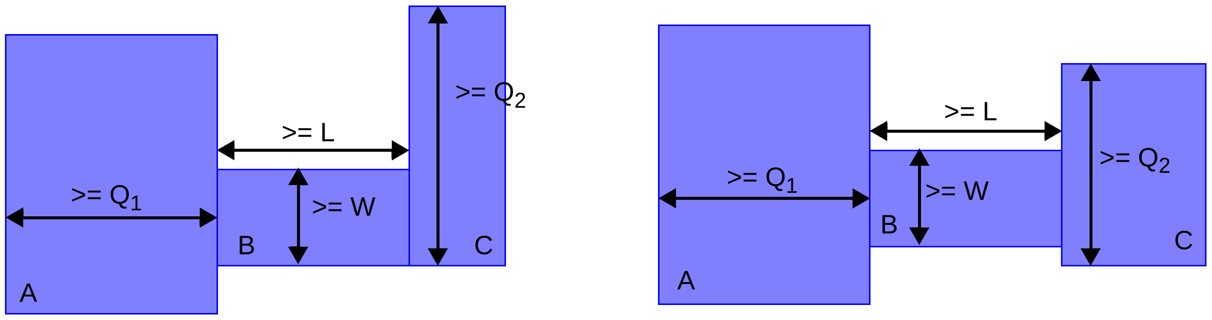
- Special Notch Rule

H-Shape Rule

The H-shape rule defines a minimum length or minimum width requirement for a metal shape that joins two other metal shapes.

In the two examples shown in Figure 2-25, shape B joins shapes A and C, forming a shape that resembles the letter “H.” To satisfy this rule, either the width of shape B must be at least W or the length of shape B must be at least L if the width of shape A is at least Q_1 and the length of shape C is at least Q_2.

Figure 2-25 H-Shape Rule



The syntax for this rule is

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

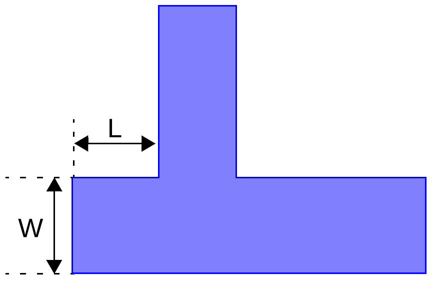
The length threshold, Q_2, must be greater than the minimum width, W. Otherwise, the shape is not considered an H-shape and the rule is not checked.

Short Edge to End-of-Line Rule

Note:
This rule is supported only by Zroute. For a similar rule supported by the classic router, see Alternative Short Edge to End-of-Line Rule.

The short edge to end-of-line rule defines the minimum length, L, of an edge adjacent to a line-end edge that is less than a specified width, W, as shown in Figure 2-26.

Figure 2-26 Short Edge to End-of-Line Rule



The syntax for this rule is

```
Layer "MetalX" {
   minEdgeLengthTblSize = 2
   minEdgeLengthTbl = (L, W)
}
```

See Also

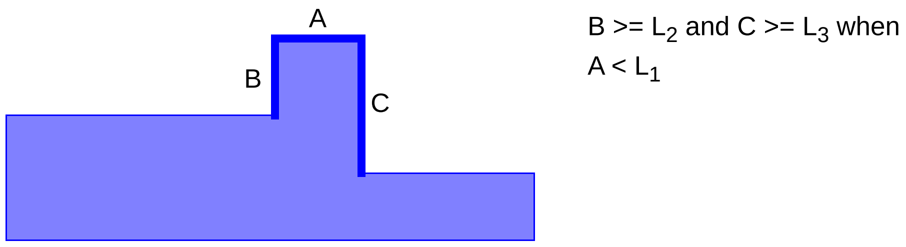
- Adjacent Minimum Edge Length Rule

Adjacent Minimum Edge Length Rule

The adjacent minimum edge length rule defines the minimum lengths of the edges adjacent to a short edge.

To satisfy this rule, the edges, B and C, adjacent to a short edge, A, must have lengths of least L_2 and L_3 if the short edge has a length less than L_1, as shown in Figure 2-27.

Figure 2-27 Adjacent Minimum Edge Length Rule Example

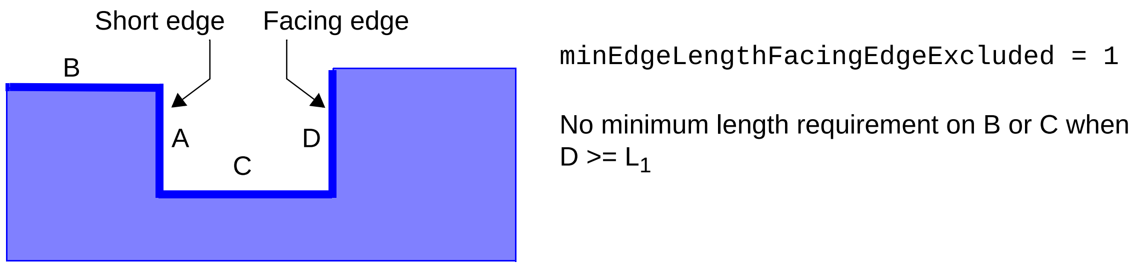


The syntax for this rule is

```
Layer "MetalX" {
  minEdgeLengthTblSize = 3
  minEdgeLengthTbl = (L1, L2, L3)
}
```

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 2-28.

Figure 2-28 Short Edge Waived With Facing Edge


Short edge
Facing edge
B
A
C
D
minEdgeLengthFacingEdgeExcluded = 1
No minimum length requirement on B or C when
D >= L_1

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

```
Layer "MetalX" {
  minEdgeLengthTblSize = 3
  minEdgeLengthTbl = (L1, L2, L3)
  minEdgeLengthFacingEdgeExcluded = 1
}
```

See Also

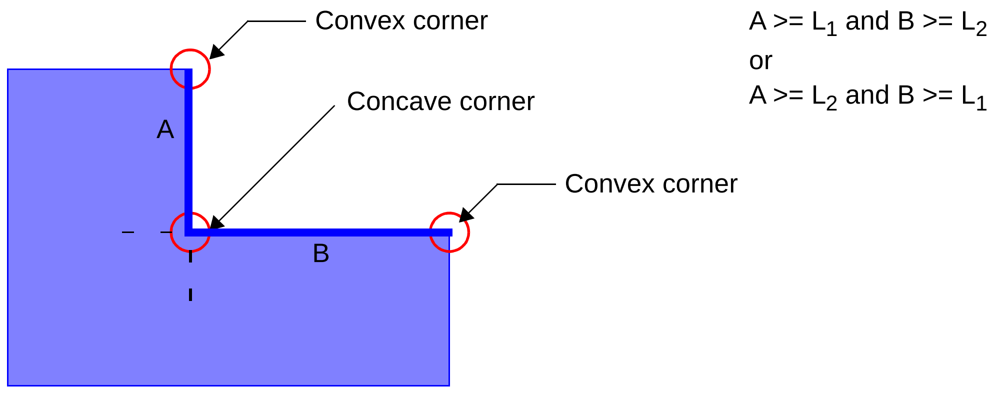
- Short Edge to End-of-Line Rule

Minimum Metal Jog Length Rule

Note:
    This rule is supported only by Zroute.

The minimum metal jog length rule defines the minimum required length for two consecutive edges of a metal jog, A and B, when the two edges meet at a concave corner and the other ends of these edges terminate at convex corners, as shown in Figure 2-29.

Figure 2-29 Metal Jog Length Rule



The syntax for this rule is

```
Layer "MetalX" {
   concaveConvexMinEdgeLength   = L1
   concaveAdjacentMinEdgeLength = L2
}
```

See Also

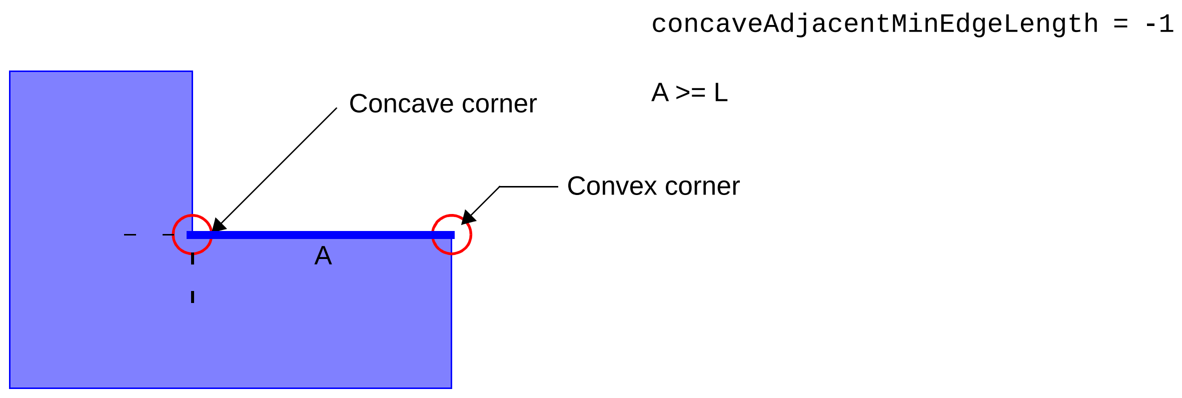
- Adjacent Edge Length Rule

Adjacent Edge Length Rule

Note:
This rule is supported only by Zroute.

The adjacent edge length rule is a variation of the minimum metal jog length that defines the minimum length for a metal edge connecting a convex corner to a concave corner, as shown in Figure 2-30.

Figure 2-30 Adjacent Edge Length Rule



The syntax for this rule is

```
Layer "MetalX" {
  concaveConvexMinEdgeLength   = L
  concaveAdjacentMinEdgeLength = -1
}
```

See Also

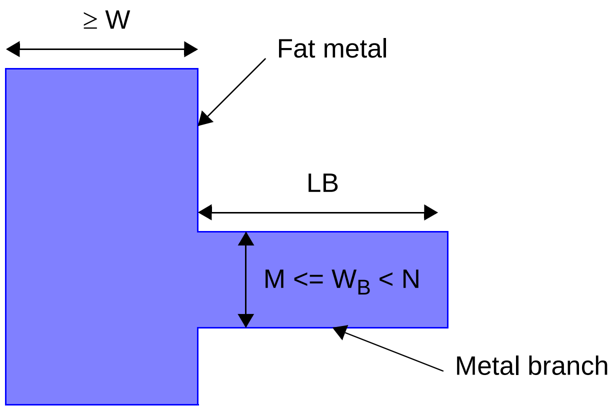
- Minimum Metal Jog Length Rule

Fat Metal Branch Minimum Width and Length Rule

Note:
This rule is supported only by Zroute.

The fat metal branch minimum width and length rule defines the minimum width, M, for a metal branch connected to a fat metal shape that has 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 L_B. See Figure 2-31.

Figure 2-31 Fat Metal Branch Minimum Width and Length Rule



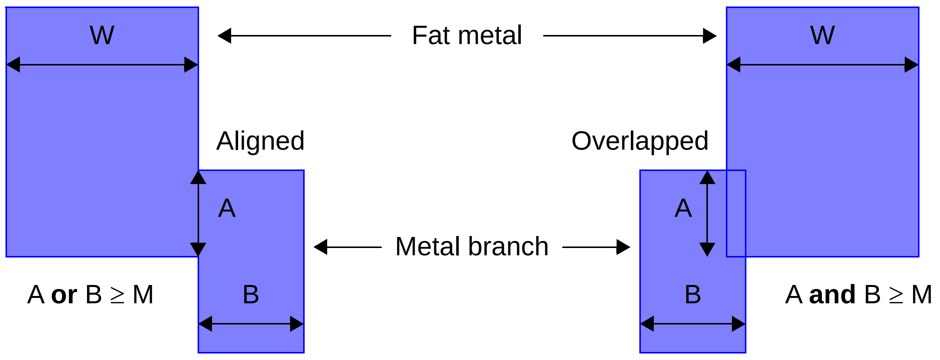
The syntax for this rule is

```
Layer "MetalX" {
   fatMetalBranchTblSize       = 3
   fatMetalBranchThresholdTbl  = (W1, W2, W3)
   fatMetalBranchMinLengthTbl  = (LB1,LB2,LB3)
   fatMetalBranchMinWidthTbl   = (M1, M2, M3)
   fatMetalBranchMaxWidthTbl   = (N1, N2, N3)
   [fatMetalBranchExcludedForFatViaExt = 1]
}
```

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, 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 to the fat metal shape and connected to the fat metal corner as shown in Figure 2-32, 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 2-32 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 2-33 shows an example of a fat metal branch area and an LB by F rectangle.

Figure 2-33 Fat Metal Parallel Branch Area Checking

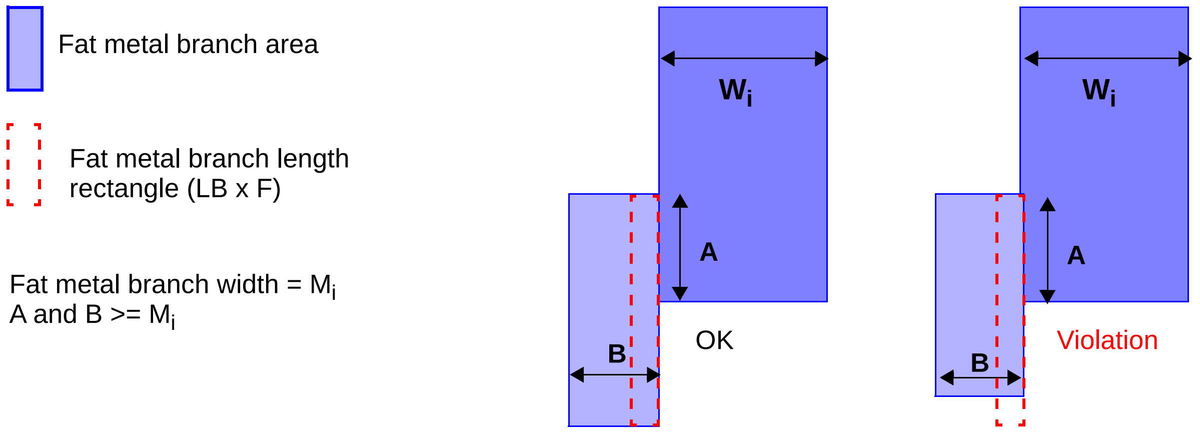
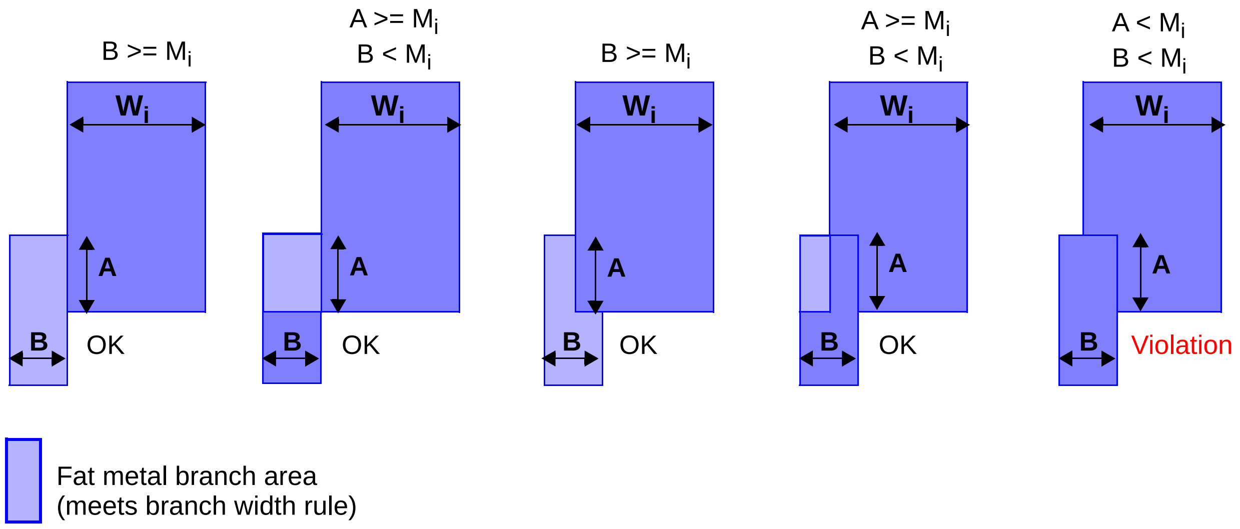

Fat metal branch area
Fat metal branch length rectangle (LB x F)
Fat metal branch width = M_i
A and B >= M_i
W_i
A
B
OK
W_i
A
B
Violation

Figure 2-34 shows some examples of fat metal branches of different widths and overlaps.

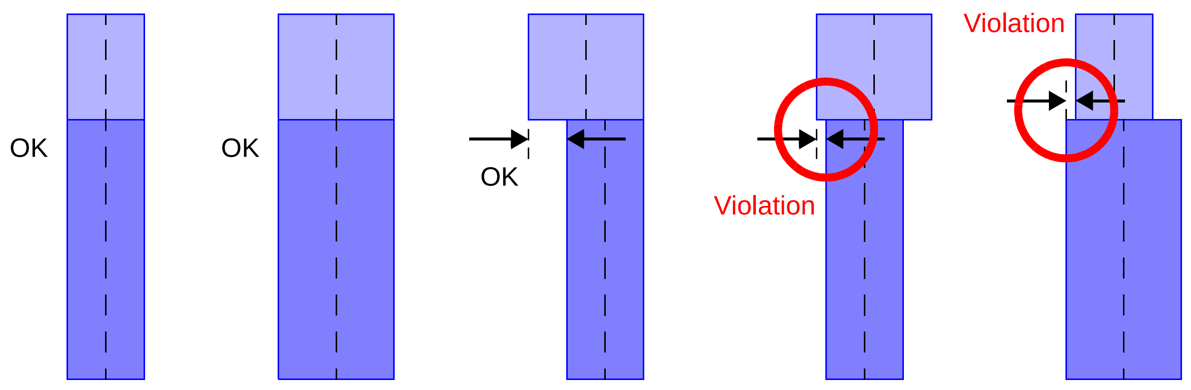
Figure 2-34 Fat Metal Parallel Branch Area Checking Examples



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 that result at the connection point must have a length of at least L. See Figure 2-35.

Figure 2-35 Minimum Edge for Macro Pin Connection Rule



The syntax for this rule is

```
Layer "MetalX" {
   minEdgeOffsetForMacroPinConnection = L
}
```

Metal Spacing Rules

The metal spacing rules are described in the following sections:

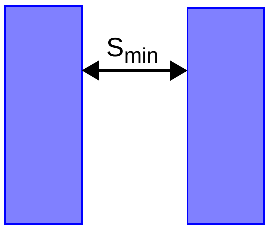
- Minimum Spacing Rule
- Same-Net Minimum Spacing Rule
- Stub Minimum Spacing Rule
- Voltage-Dependent Minimum Spacing Rule
- U-Shape Spacing Rule
- U-Shape Leg Spacing Rule
- Line-End to U-Shape Spacing Rule
- Concave Corner Keepout Rule

- Preferred and Nonpreferred Corner-to-Corner Spacing Rule
- Preferred-Direction Forbidden Spacing Rule
- Three-Wire Forbidden Spacing Rule
- Forbidden Pitch Rule

Minimum Spacing Rule

The minimum spacing rule defines the minimum spacing, S_{min}, between the edges of two objects on different nets. This is the default minimum spacing requirement for the layer.

Figure 2-36 Minimum Spacing Rule



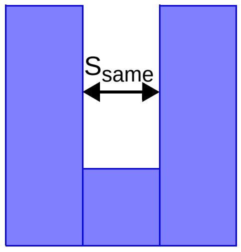
The syntax for this rule is

```
Layer "MetalX" {
  minSpacing          = Smin
}
```

Same-Net Minimum Spacing Rule

The same net minimum spacing rule defines the minimum spacing, S_{same}, between the edges of two objects on the same net. The same-net minimum spacing, S_{same}, is smaller than the minimum spacing, S_{min} specified with `minSpacing`.

Figure 2-37 Same-Net Minimum Spacing Rule



The syntax for this rule is

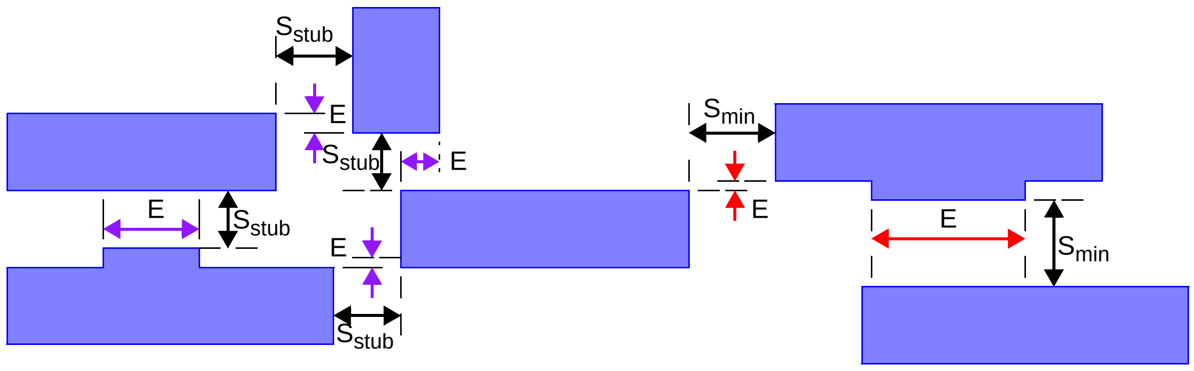
```
Layer "MetalX" {
  minSpacing          = Smin
  sameNetMinSpacing   = Ssame
}
```

Stub Minimum Spacing Rule

The stub minimum spacing rule defines the minimum spacing, S_{stub}, between metal shapes whose parallel overlap is no more than a specific threshold, P. The stub minimum spacing, S_{stub}, is smaller than the minimum spacing, S_{min} specified with `minSpacing`.

In Figure 2-38, if E is greater than 0 and less than or equal to the parallel overlap threshold, P, the stub minimum spacing, S_{stub}, is required. Otherwise, the minimum spacing, S_{min}, is required.

Figure 2-38 Less Restrictive Minimum Spacing for Small Parallel Overlaps



The syntax for this rule is

```
Layer "MetalX" {
  minSpacing                    = Smin
  stubParallelLengthMaxThreshold = P
  stubMinSpacing                = Sstub
}
```

Voltage-Dependent Minimum Spacing Rule

The voltage-dependent minimum spacing rule defines the spacing requirement between a wire or via and its neighbor when the spacing depends on the voltage associated with the objects. The required spacing depends on the higher supply voltage of the two objects.

The syntax for this rule is

```
Layer "MetalX" {
  voltageAreaTblSize       = 3
  voltageAreaThresholdTbl  = (V1, V2, V3)
  voltageAreaMinSpacingTbl = (SV1, SV2, SV3)
}
```

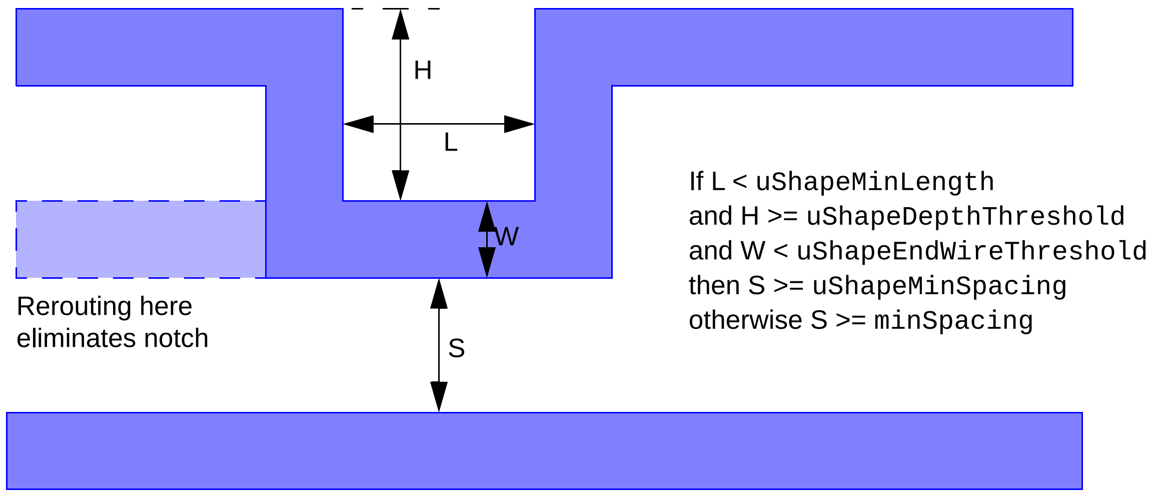
In this example, the minimum spacing requirement depends on the higher supply voltage, V, of the two nearby metal shapes:

- For $V_1 <= V < V_2$, the minimum spacing is S_{V1}
- For $V_2 <= V < V_3$, the minimum spacing is S_{V2}
- For $V_3 <= V$, the minimum spacing is S_{V3}

U-Shape Spacing Rule

The U-shape spacing rule defines the minimum spacing, S_u, between a short U-shaped notch and a neighboring wire when then notch meets the following requirements: the height is greater than or equal to H, the length is less than L, and the width is less than W. If the notch does not meet these requirements, the default minimum spacing, S_{min}, specified with `minSpacing` applies.

Figure 2-39 U-Shape Spacing Rule

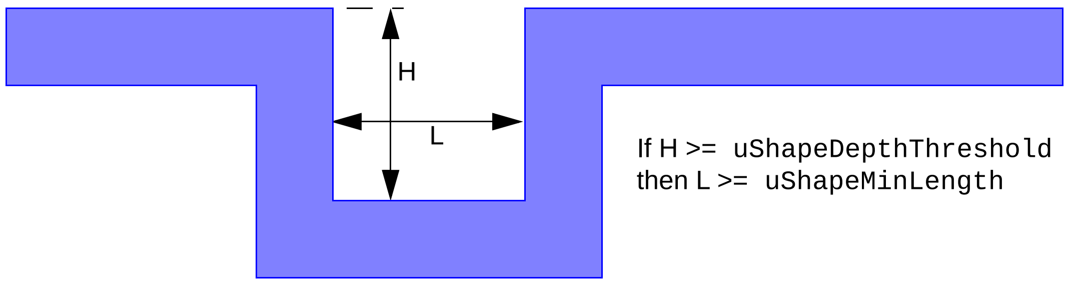

H
L
W
S
Rerouting here eliminates notch
If L < uShapeMinLength
and H >= uShapeDepthThreshold
and W < uShapeEndWireThreshold
then S >= uShapeMinSpacing
otherwise S >= minSpacing

The syntax for this rule is

```
Layer "MetalX" {
   minSpacing              = Smin
   uShapeMinSpacing        = Su
   uShapeDepthThreshold    = H
   uShapeEndWireThreshold  = W
   uShapeMinLength         = L
}
```

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 2-40.

Figure 2-40 Alternative U-Shape Spacing Rule



The syntax for this alternative rule is

```
Layer "MetalX" {
   minSpacing                = Smin
   uShapeDepthThreshold      = H
   uShapeMinLength           = L
}
```

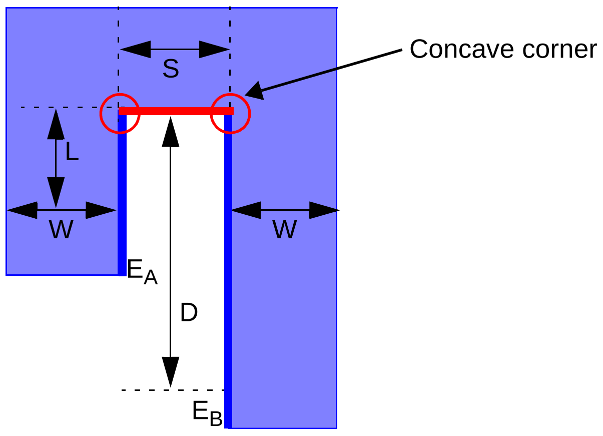
U-Shape Leg Spacing Rule

Note:
This rule is supported only by Zroute.

The U-shape leg spacing rule defines the minimum spacing, S, between the legs of a U-shaped notch for a depth, D, from the base of the U shape. The base, which is shown in red in Figure 2-41, is the edge between the two concave corners that form the notch. This rule applies when the U shape meets the following conditions:

- The two legs of the U shape have widths no greater than W for more than a length, L, from the base of the U shape.
- The two parallel inner edges of the legs, E_A and E_B, have exactly one end connected to a concave corner.

Figure 2-41 U-Shape Leg Spacing Rule



The syntax for this rule is

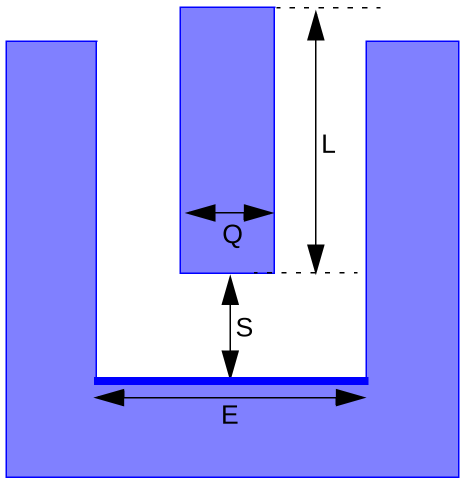
```
Layer "MetalX" {
   uShapeSideWireMaxWidthThreshold   = W
   uShapeSideWireMinLength           = L
   uShapeSideWireLengthMaxCheckRange = D
   uShapeSideToSideMinSpacing        = S
}
```

Line-End to U-Shape Spacing Rule

Note:
This rule is supported only by Zroute.

The line-end to U-shape spacing rule defines the minimum spacing, S, 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 base of the U shape has a length no more than E, as shown in Figure 2-42.

Figure 2-42 Line-End to U-Shape Spacing Rule



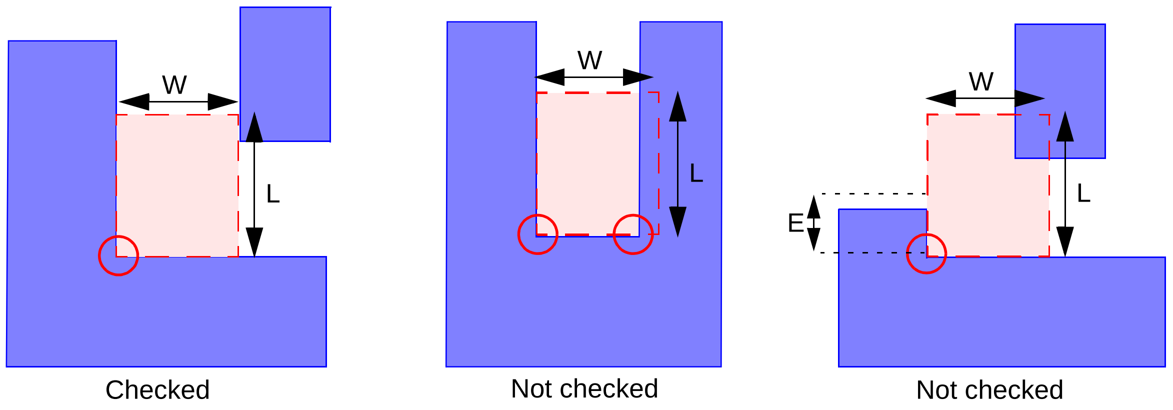
The syntax for this rule is

```
Layer "MetalX" {
   uShapeKeepoutMaxEdgeLengthThreshold     = E
   uShapeKeepoutDepth                      = S
   uShapeKeepoutEndOfLineMaxWidthThreshold = Q
   uShapeKeepoutEndOfLineMinLength         = L    # optional
}
```

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 2-43.

Figure 2-43 Concave Corner Keepout Rule



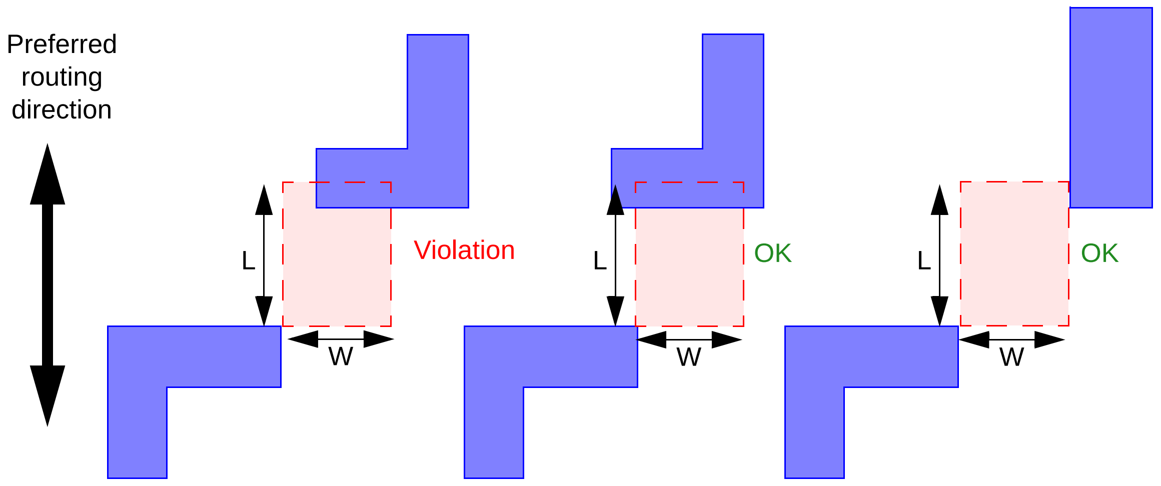
The syntax for this rule is

```
Layer "MetalX" {
  concaveCornerKeepoutWidth    = W
  concaveCornerKeepoutLength   = L
  concaveCornerKeepoutMinEdgeLengthThreshold = E  # optional
}
```

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 2-44.

Figure 2-44 Preferred and Nonpreferred Corner-to-Corner Spacing Rule



The syntax for this rule is

```
Layer "MetalX" {
  cornerKeepoutNonPrefWidth  = W
  cornerKeepoutPrefLength    = L
}
```

Preferred-Direction Forbidden Spacing Rule

Note:
    This rule is supported only by Zroute.

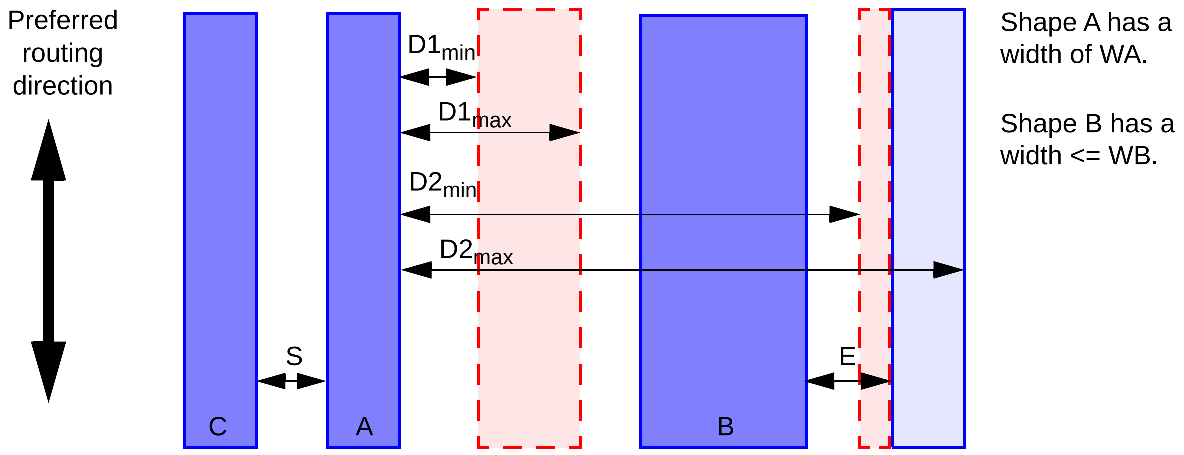
The preferred-direction forbidden spacing rule defines one or more distance ranges where metal is forbidden between or beyond two metal shapes, A and B, measured from the edge of shape A, when all of the following conditions are met:

- Shape A runs in the preferred routing direction
- The width of shape A is WA
- The width of shape B is no more than WB
- Another shape C on the other side of shape A is spaced at a distance S from shape A

An optional attribute, E, specifies a threshold for disabling the rule at a specified distance beyond the edge of shape B.

As shown in Figure 2-45, the rule defines one or more forbidden regions, with each region defined by a minimum and maximum distance from shape A. The forbidden regions are shown in red.

Figure 2-45 Preferred-Direction Forbidden Spacing Rule


Preferred routing direction
D1min
D1max
D2min
D2max
S
E
C
A
B
Shape A has a width of WA.
Shape B has a width <= WB.

The syntax for this rule is

```
Layer "MetalX" {
  forbiddenSpacePrefTblSize              = 2
  forbiddenSpacePrefWireWidthTbl         = (WA1,WA2)
  forbiddenSpacePrefWireSpacingTbl       = (S1, S2)
  forbiddenSpacePrefForbiddenWireMaxWidthTbl = (WB1,WB2)
  forbiddenSpacePrefRangeTbl             = ("D11min,D11max,D12min,D12max",
                                             "D21min,D21max, ... ")
  forbiddenSpacePrefExcludedTbl          = (E1, E2)  # optional
}
```

Three-Wire Forbidden Spacing Rule

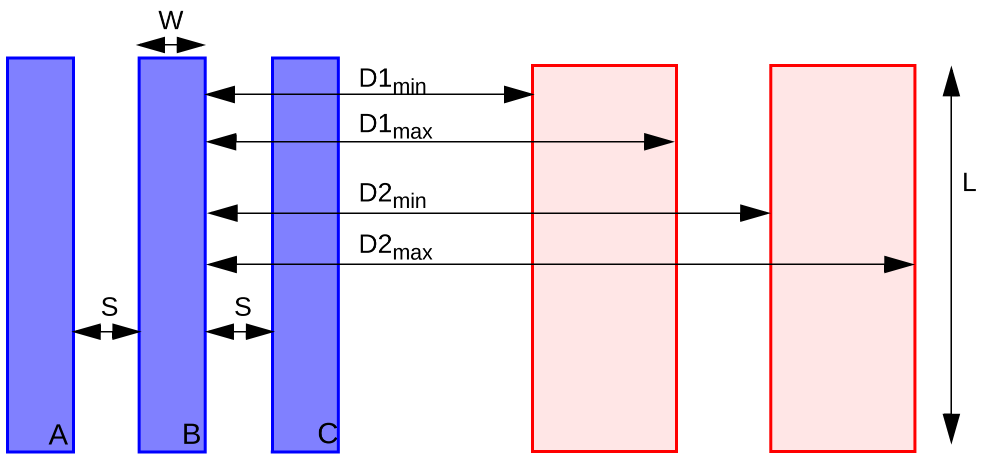
Note:
This rule is supported only by Zroute.

The three-wire forbidden spacing rule defines one or more regions relative to a metal shape, B, where metal shapes with a parallel length of at least L are forbidden when all of the following conditions are met:

- The width of shape B is no more than W
- The shape B has two neighboring shapes, A and C, each with an edge-to-edge spacing from B of no more than S

As shown in Figure 2-46, the rule defines one or more forbidden regions, with each region defined by a minimum and maximum distance from shape B. The forbidden regions are shown in red.

Figure 2-46 Three-Wire Forbidden Spacing Rule

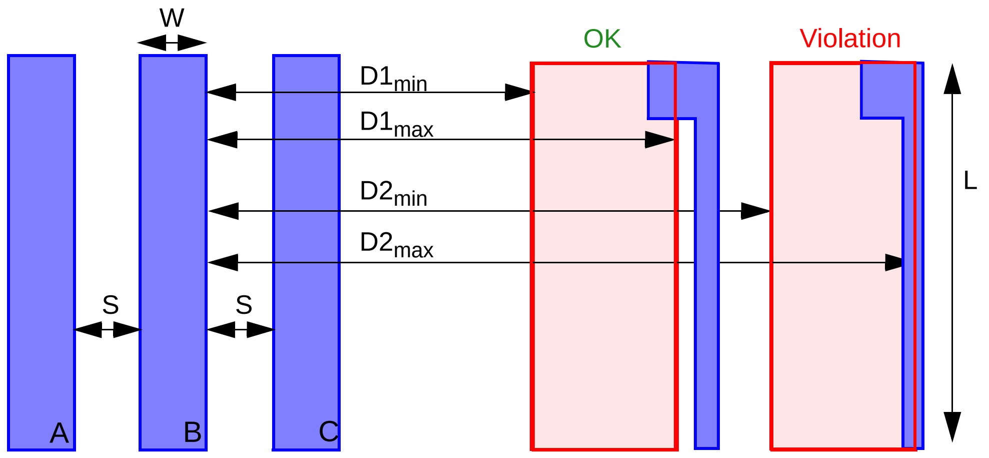


The syntax for this rule is

```
Layer "Metal1" {
   ...
  forbiddenSpaceWireMaxWidthThreshold   = W
  forbiddenSpaceWireMaxSpacingThreshold = S
  forbiddenSpaceWireParallelLength      = L
  forbiddenSpaceRangeTblSize            = 2
  forbiddenSpaceRangeTbl                = ("D1min,D1max,D2min,D2max")
}
```

Figure 2-47 illustrates usage of the rule. The shape in the forbidden region on the left is not a violation because the parallel run within the region is less than L. The shape in the forbidden region on the right is a violation because the parallel run within the region is at least L (even though a jog is present in the shape).

Figure 2-47 Three-Wire Forbidden Spacing Rule Examples



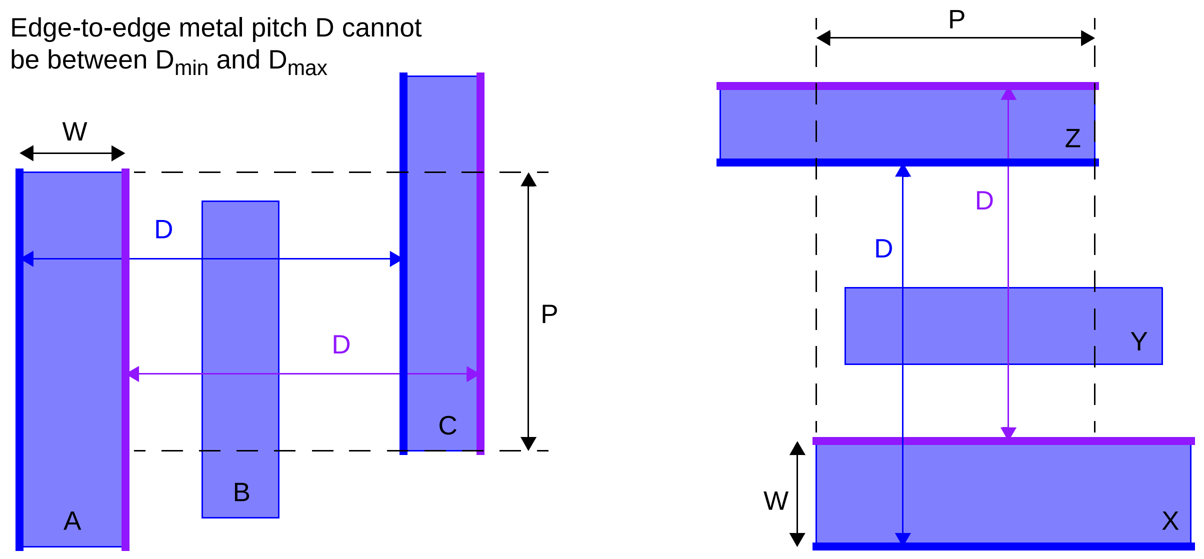
Forbidden Pitch Rule

Note:
This rule is supported only by Zroute.

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 W and there is exactly one polygon between the two metal shapes.

In Figure 2-48, 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 $D_{min} <= D <= D_{max}$.

Figure 2-48 Forbidden Pitch Rule


Edge-to-edge metal pitch D cannot be between D_{min} and D_{max}
W
D
D
P
A
B
C
P
Z
D
D
Y
W
X

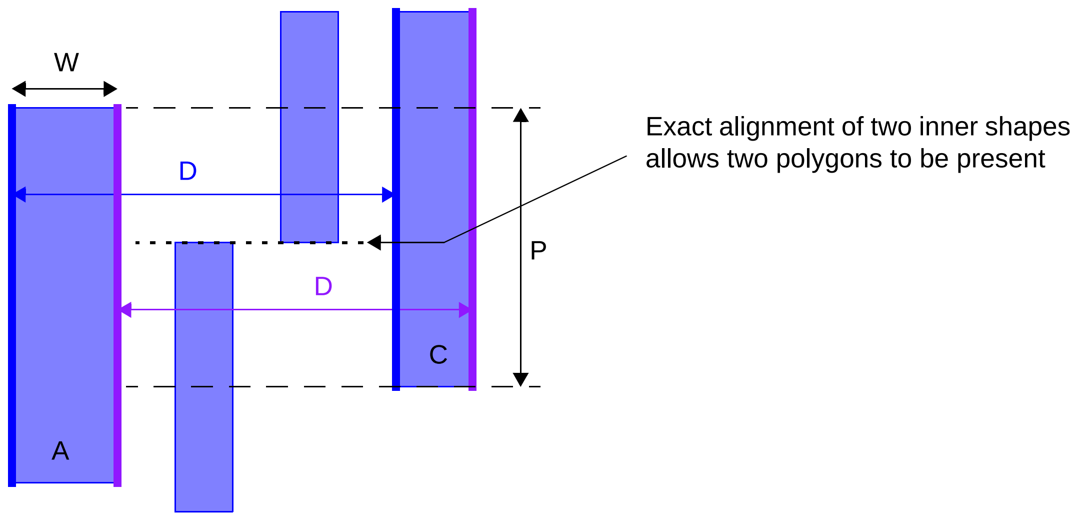
Similarly, for horizontal shapes X and Z with one shape Y between them, the pitch value D measured from top edge to top edge or bottom edge to bottom edge cannot be in the range $D_{min} <= D <= D_{max}$.

The syntax for this rule is

```
Layer "MetalX" {
  forbiddenPitchWireMaxWidthThreshold = W
  forbiddenPitchWireParallelLength    = P
  forbiddenPitchKeepoutMinDist        = Dmin
  forbiddenPitchKeepoutMaxDist        = Dmax
}
```

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 2-49.

Figure 2-49 Two Polygons in the Forbidden Pitch Rule


W
D
D
P
C
A
Exact alignment of two inner shapes allows two polygons to be present

Via Spacing Rules

The via spacing rules are described in the following sections:

- Adjacent Via Rule
- Adjacent Via Spacing Rule
- Enclosed Via Spacing Rule
- Maximum Number of Neighboring Vias Rule
- Isolated Via Rule
- Via Corner Spacing Rule
- Via Same Net Minimum Spacing Rule
- Edge-Line Via Spacing Rule
- General Cut Spacing Rule
- Misaligned Via Spacing Rule
- Maximum Number of Unaligned Vias Rule
- Non-Self-Aligned Via Edge to Metal Spacing Rule
- Constrained Via Spacing Rule
- Color-Based Constrained Via Spacing Rule

- Critical Via Spacing Rule
- Range-Enclosed 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
- Stacked Via Keepout Rule
- Double-Patterning Via Spacing Rule
- Color-Based Via Centerline Spacing Rule
- Via Center Spacing Exception Rule
- Via Ladder 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:

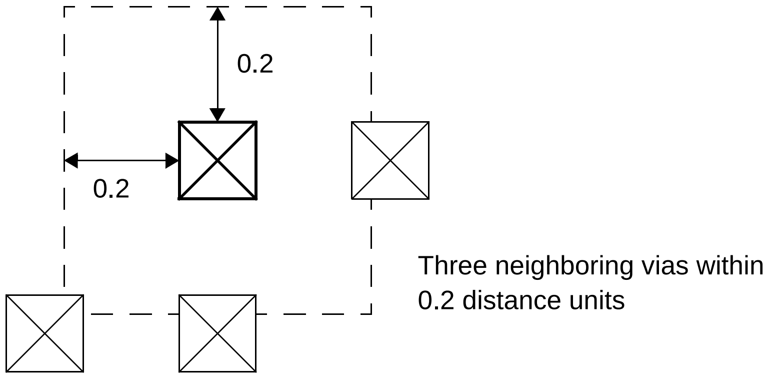
- The `maxNumAdjacentCut` attribute defines the maximum number of adjacent vias.
- The `adjacentCutRange` attribute defines the range within which to check for adjacent vias.

For example,

```
layer   "VIA1" {
  maxNumAdjacentCut      = 2
  adjacentCutRange       = 0.2
}
```

In Figure 2-50, 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 2-50 Adjacent Via Rule



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 2-51 shows examples of via placement allowed by the rule, while Figure 2-52 shows via configurations disallowed by the rule.

Figure 2-51 Adjacent Via Spacing Rule Allowed Via Placement

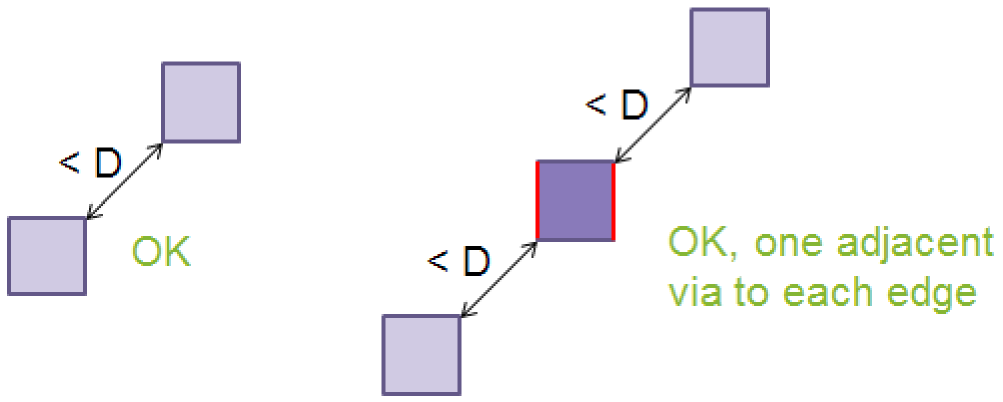
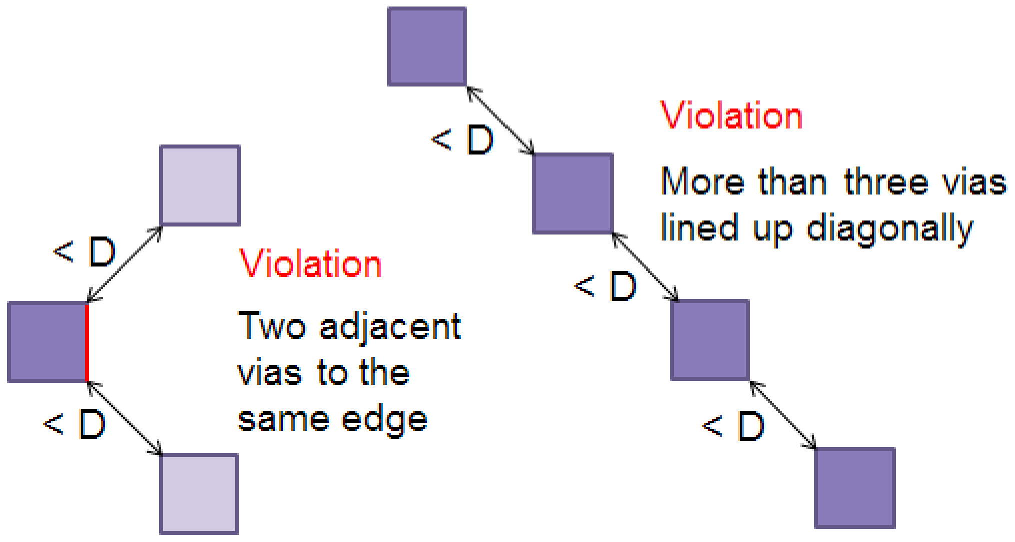


Figure 2-52 Adjacent Via Spacing Rule Disallowed Via Placement


< D
Violation
Two adjacent vias to the same edge
< D
< D
Violation
More than three vias lined up diagonally
< D
< D
< D

This is the syntax of the rule:

```
Layer Vx {
   diagAdjacentCutTblSize          = 1
   diagAdjacentCutNameTbl          = (Vs)
   diagAdjacentCutMinSpacingTbl    = (D)
}
```

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`

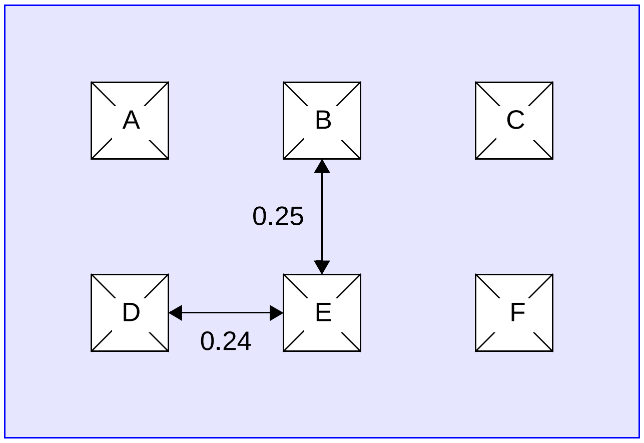
Specifies the minimum spacing allowed between an enclosed via and surrounding vias.

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 2-53, 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 2-53 Enclosed Via Spacing Rule Example



Maximum Number of Neighboring Vias Rule

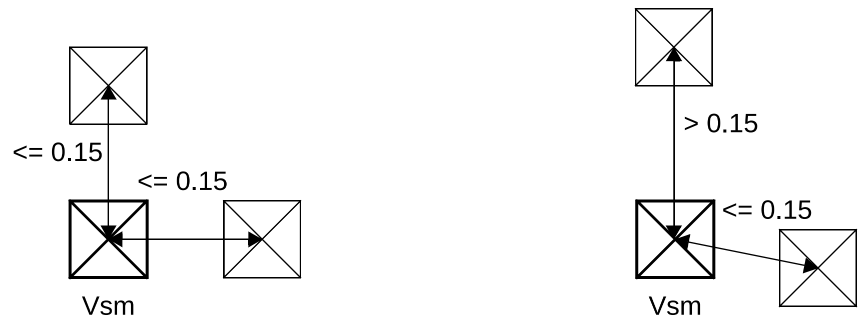
The maximum number of neighboring vias rule is similar to the enclosed via spacing rule because it specifies how many nearby vias are allowed near any given via. However, it checks center-to-center rather than edge-to-edge distances, and it can allow a larger maximum number of neighboring vias when those vias are exactly aligned vertically or horizontally.

The following example demonstrates the rule.

```
Layer "ViaX" {
  neighborCutTblSize              = 1
  neighborCutNameTbl              = (Vsm)
  neighborCutCenterRangeTbl       = (0.15)
  neighborCutMaxNumCutsTbl        = (1)
  neighborCutAlignedMaxNumCutsTbl = (2)
}
```

In this example, for Vsm type vias, if the center-to-center distance between the vias is less than or equal to 0.15, no more than two nearby vias are allowed if both vias are exactly aligned, and no more than one nearby via is allowed if that via is not exactly aligned. Vias are exactly aligned when they have the same dimensions and the same x-coordinate or y-coordinate location, as shown in the example on the left in the following figure.

Figure 2-54 Maximum Number of Neighboring Vias Rule



No more than **2** nearby vias allowed when both are **exactly aligned**

No more than **1** nearby via is allowed when the via is **not** exactly aligned

This is a table-based rule that can specify the constraint for any number of via cuts. In the following example, the rule specifies the maximum number of neighboring vias for Vsm and Vh.

```
Layer "ViaX" {
  neighborCutTblSize              = 2
  neighborCutNameTbl              = (Vsm, Vh)
  neighborCutCenterRangeTbl       = (0.15, 0.17)
  neighborCutMaxNumCutsTbl        = (1, 2)
  neighborCutAlignedMaxNumCutsTbl = (2, 3)
}
```

In this example, for a Vh type via, another Vh type via is considered “nearby” if the center-to-center distance between them is less than or equal to 0.17. No more than three nearby Vh vias are allowed if all three of them are exactly aligned, and no more than two nearby Vh vias are allowed if any of them are not exactly aligned.

Isolated Via Rule

The isolated via rule defines the maximum distance between vias in the tool, as specified by the `set_droute_options` command in the IC Compiler tool. 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 in the IC Compiler tool.

  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 in the IC Compiler tool.

  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 the IC Compiler tool.

To check for isolated via violations in your design in the IC Compiler tool, 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 in the IC Compiler tool, 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.

Via Corner Spacing Rule

Use the `cornerMinSpacing` attribute to specify the minimum corner-to-corner spacing allowed between two vias.

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

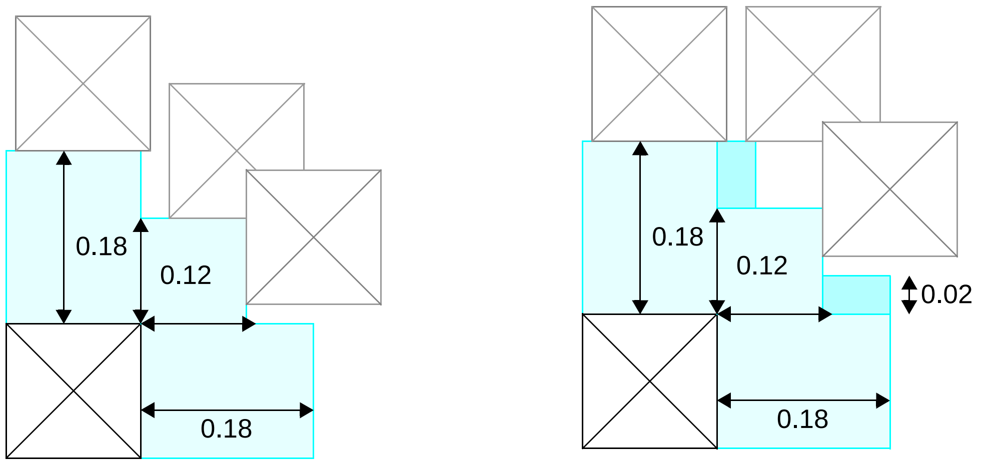
```
Layer "VIA1" {
  minSpacing       = 0.18
  cornerMinSpacing = 0.12
}
```

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, 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 2-55.

Figure 2-55 Via Corner Spacing Rule



Exclusion areas without using `minSpacingCornerKeepoutWidth`

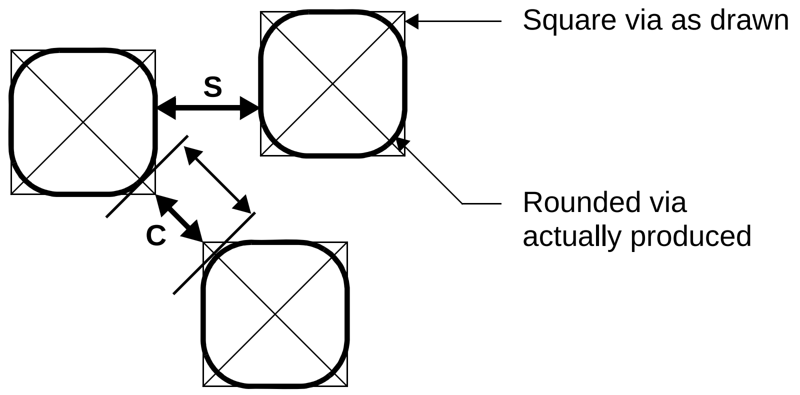
Exclusion areas with `minSpacingCornerKeepoutWidth = 0.02`

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 2-56. 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 2-56 Drawn Vias Versus Actual Physical Vias

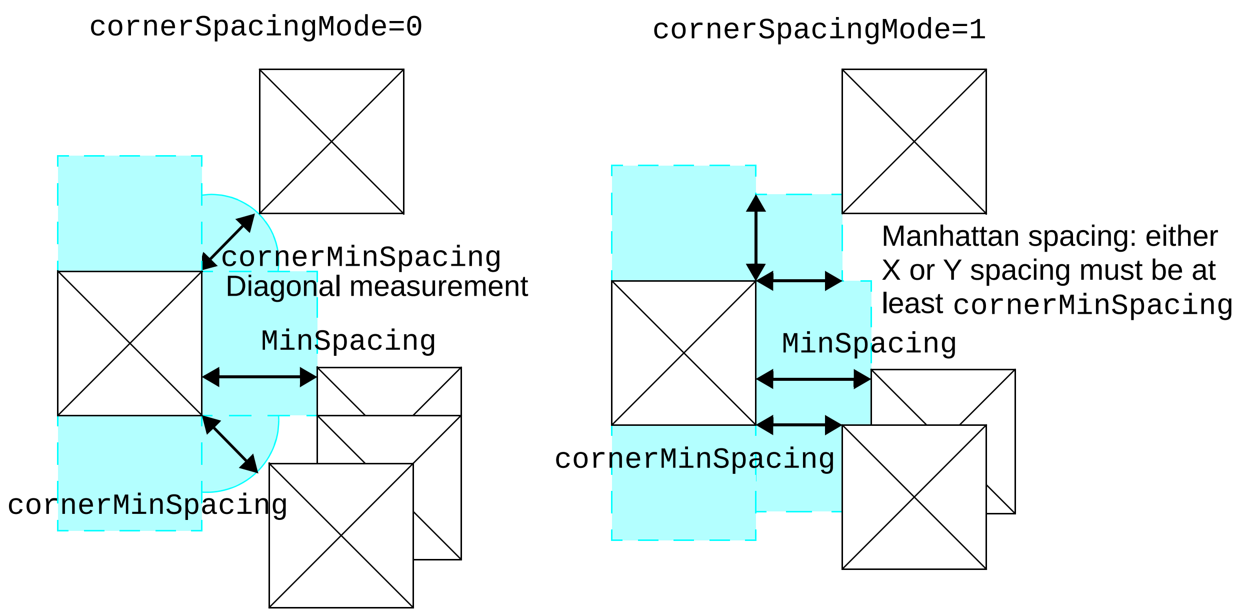


You can control the measurement method, either diagonal or Manhattan, for corner-to-corner spacing between vias by setting the `cornerSpacingMode` attribute in the `Technology` section of the technology file. By default (`cornerSpacingMode` = 0), the tool measures the diagonal distance. To use Manhattan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

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

Figure 2-57 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 2-57 Diagonal and Manhattan Distance Measurements



Via Same Net Minimum Spacing Rule

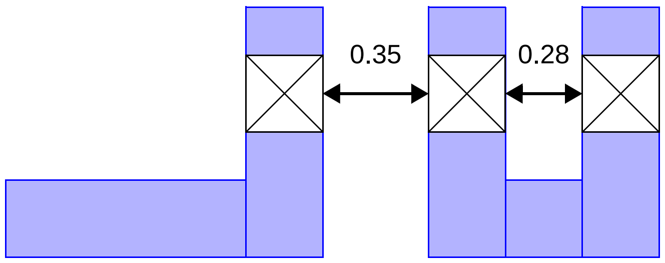
Use the `sameNetMinSpacing` attribute to override the `minSpacing` default and set a smaller value for the minimum spacing between two vias belonging to the same net.

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

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

In Figure 2-58, 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.

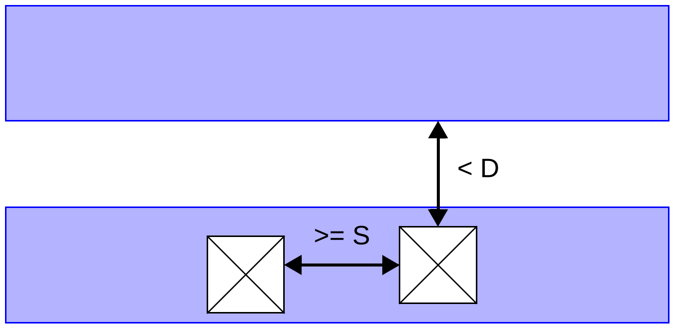
Figure 2-58 Unconnected and Connected Via Minimum Spacing Rule



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 2-59.

Figure 2-59 Edge-Line Via Minimum Spacing Rule



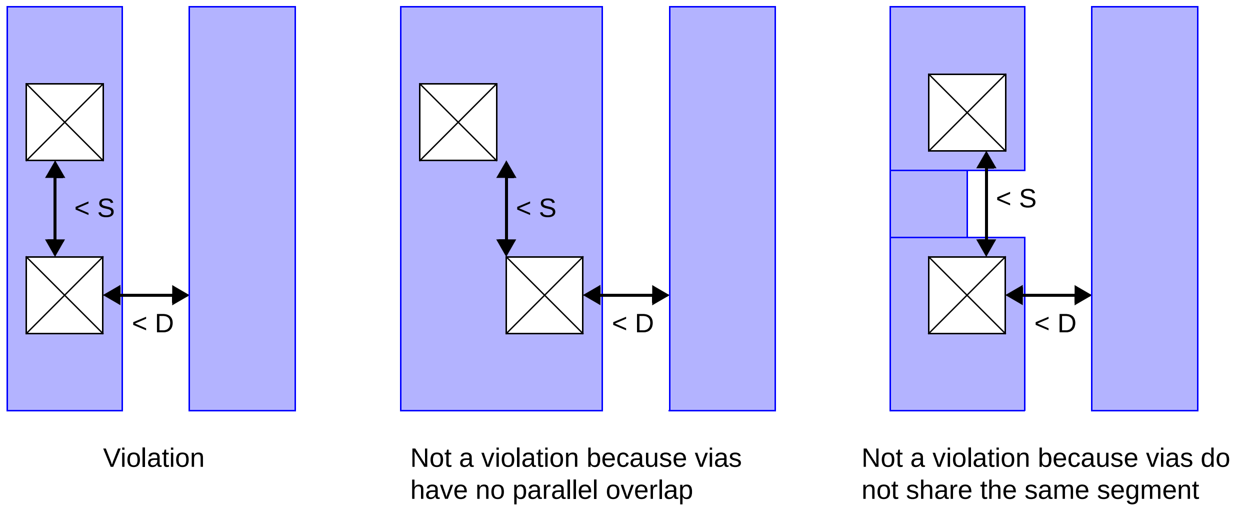
If the distance between the via and the neighboring metal is less than D, and the two vias have a parallel overlap that is greater than zero, the spacing between the two vias must be at least S. This is the syntax of the rule:

```
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 2-60 demonstrate the conditions under which the rule is applied.

Figure 2-60 Edge-Line Via Minimum Spacing Rule Examples


< S
< D
Violation
< S
< D
Not a violation because vias have no parallel overlap
< S
< D
Not a violation because vias do not share the same segment

General Cut Spacing Rule

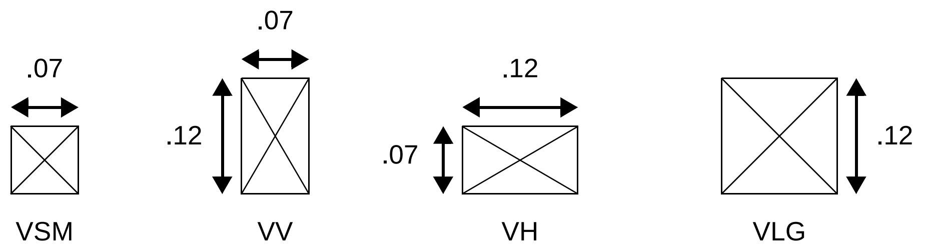
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see Via Corner Spacing Rule and Via Same Net Minimum 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.

Usage of this rule is best explained by example. Suppose that you use the four cut sizes shown in Figure 2-61 on the VIA1 layer.

Figure 2-61 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)
  }
```

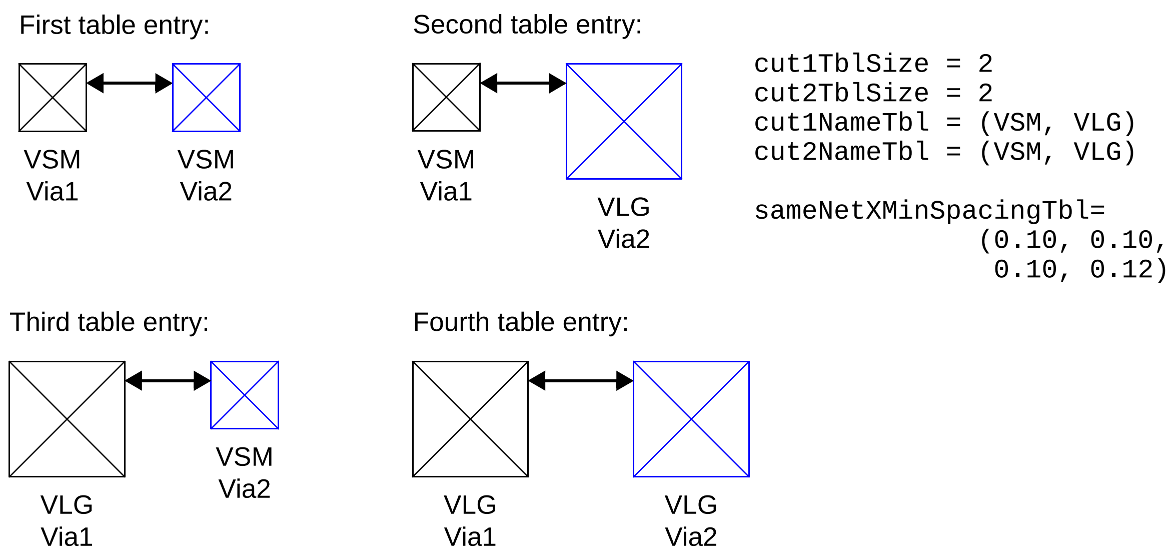
The `layer1` and `layer2` attributes specify the cut layers involved in the rule. The two layers can be different layers or the same layer. The rule specifies the minimum distance between cuts on the two specified layers or between cuts in the same layer. The `cut1TblSize` and `cut2TblSize` attributes 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 2-62.

Figure 2-62 Cut-to-Cut Spacing Specified in a 2-by-2 Table



```
cut1TblSize = 2
cut2TblSize = 2
cut1NameTbl = (VSM, VLG)
cut2NameTbl = (VSM, VLG)

sameNetXMinSpacingTbl=
              (0.10, 0.10,
               0.10, 0.12)
```

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:

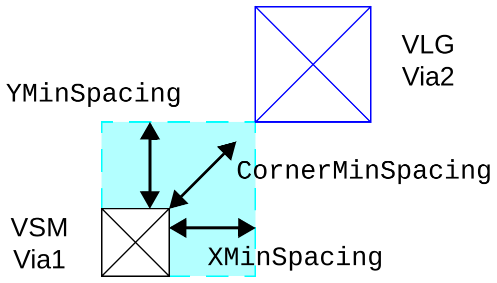
```
sameNetXMinSpacingTbl[i] >= xMinSpacing >= minSpacing
sameNetYMinSpacingTbl[i] >= yMinSpacing >= minSpacing
diffNetXMinSpacingTbl[i] >= xMinSpacing >= minSpacing
diffNetYMinSpacingTbl[i] >= yMinSpacing >= minSpacing
sameSegXMinSpacingTbl[i] >= max(minCutSpacing, xMinSpacing)
sameSegYMinSpacingTbl[i]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

where i = 0, 1, ..., 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 2-63.

Figure 2-63 XMinSpacing and YMinSpacing Enforced at Corners

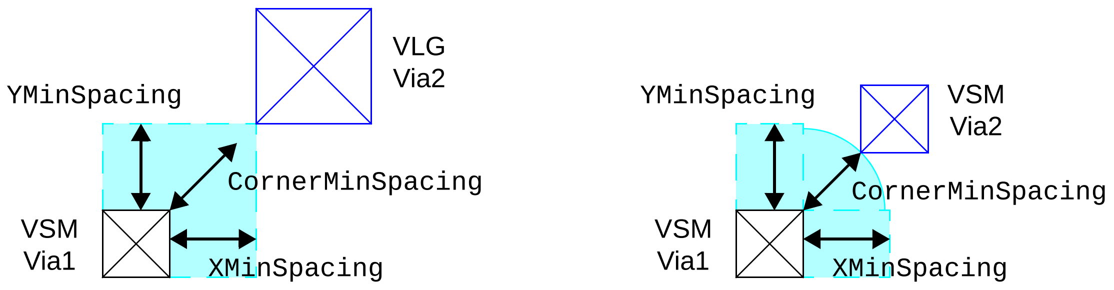


To suppress checking of `XMinSpacing`/`YMinSpacing` at corners and allow only `CornerMinSpacing` to be checked, set the `orthoSpacingExcludeCornerTbl` attribute to 1 for the corresponding check.

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 2-64.

```
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 2-64 XMinSpacing and YMinSpacing Suppressed at Corners



X and Y spacing check enforced at corners (orthoSpacingExcludeCornerTbl = 0)

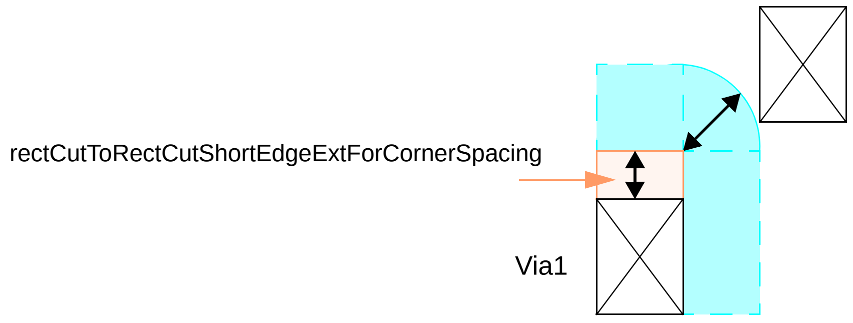
X and Y spacing check suppressed at corners (orthoSpacingExcludeCornerTbl = 1)

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)
}
```

With suppressed checking of `XMinSpacing` and `YMinSpacing` at corners, you can specify an extension value to a rectangular cut's short edge as illustrated in Figure 2-65. The extension at the short edge allows for a slightly different radial spacing that is measured from the extension corner instead of the corner of the rectangular cut.

Figure 2-65 Suppressed Corner Checking with Short Edge Extension



To specify an extension value to a rectangular cut's short edge, include at least one of the following attributes in the metal `Layer` section of the technology file:

- `rectCutToRectCutShortEdgeExtForCornerSpacing` for neighboring rectangular cuts
- `rectCutToNonRectCutShortEdgeExtForCornerSpacing` for neighboring nonrectangular cuts

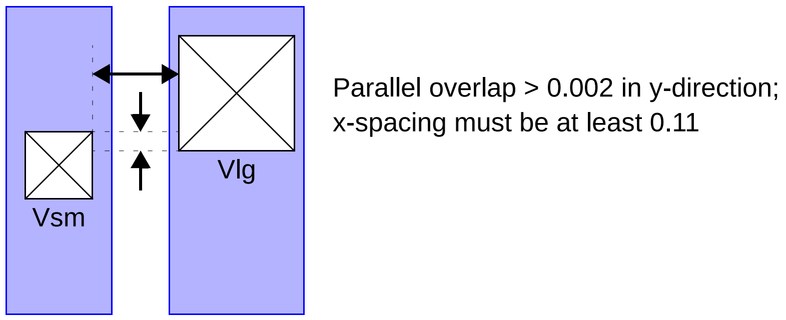
For example, a `Layer` definition for Figure 2-65 is as follows:

```
Layer "Via1" {
...
    rectCutToRectCutShortEdgeExtForCornerSpacing = 0.05
...
}
```

If you use either of these properties, the `CornerMinSpacingTbl` attribute must be defined in the `DesignRule` section.

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, 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 2-66.

Figure 2-66 Parallel Overlap Test in Y-Direction for X-Spacing Rule With a Positive Threshold


Parallel overlap > 0.002 in y-direction;
x-spacing must be at least 0.11
Vsm
Vlg

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, the edge-to-edge y-spacing rule applies between the edges of the cuts.

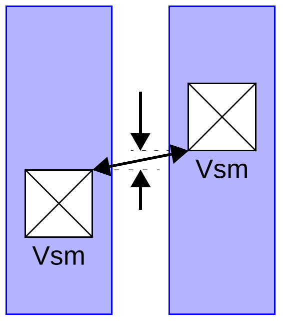
If the parallel length threshold is a negative number, the minimum spacing rule applies if the parallel overlap on the orthogonal axis is greater than the specified negative threshold but less than or equal to zero, or in other words, when the two cuts are close to overlapping but do not quite overlap.

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 2-67.

Figure 2-67 Parallel Overlap Test in Y-Direction With a Negative Threshold

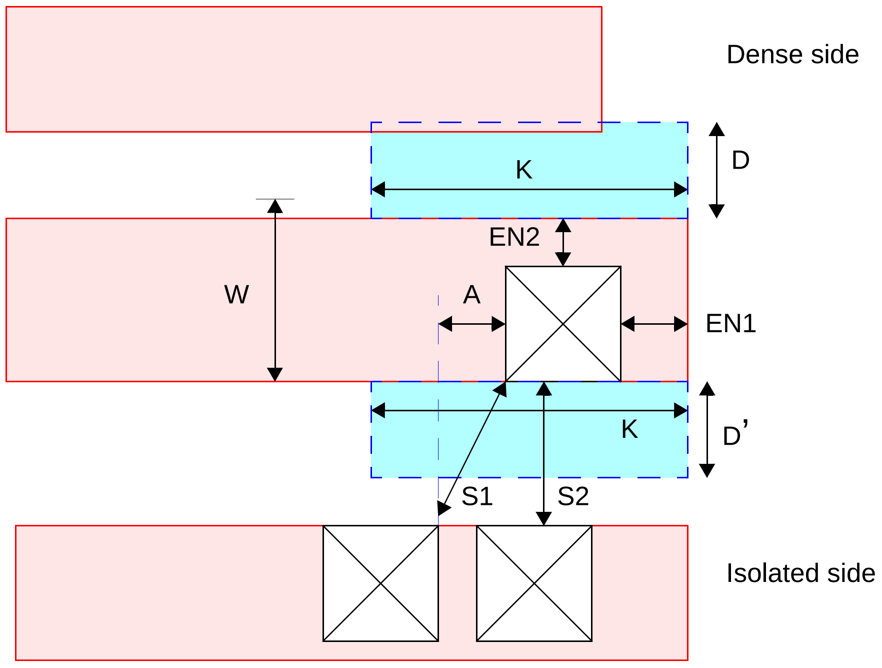


Parallel overlap > –0.007 and < 0 in y-direction;
diagonal spacing must be at least 0.12

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 2-68 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 2-68 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 2-68, 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 2-69, 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 2-70, when the table value of `misalignedViaWireMinSpacingThreshold2Tbl` is zero.

Figure 2-69 Misaligned Via Minimum Spacing Rule, Example 2

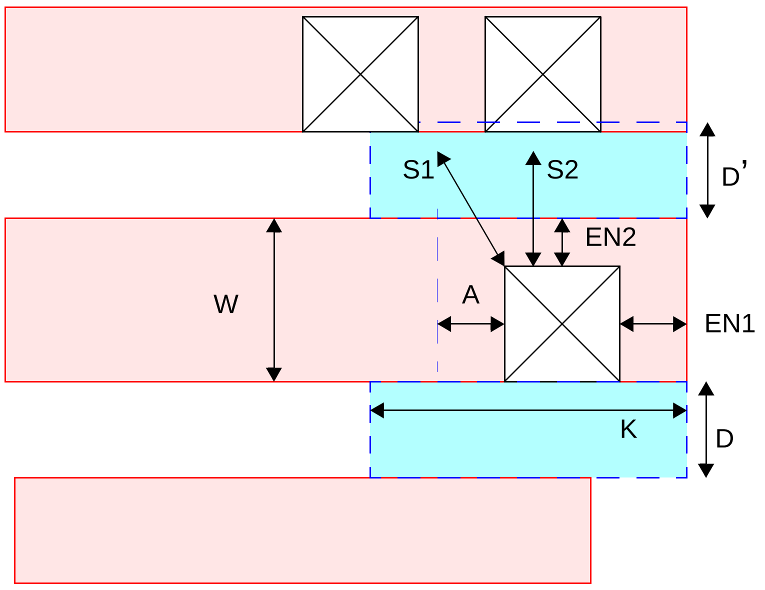
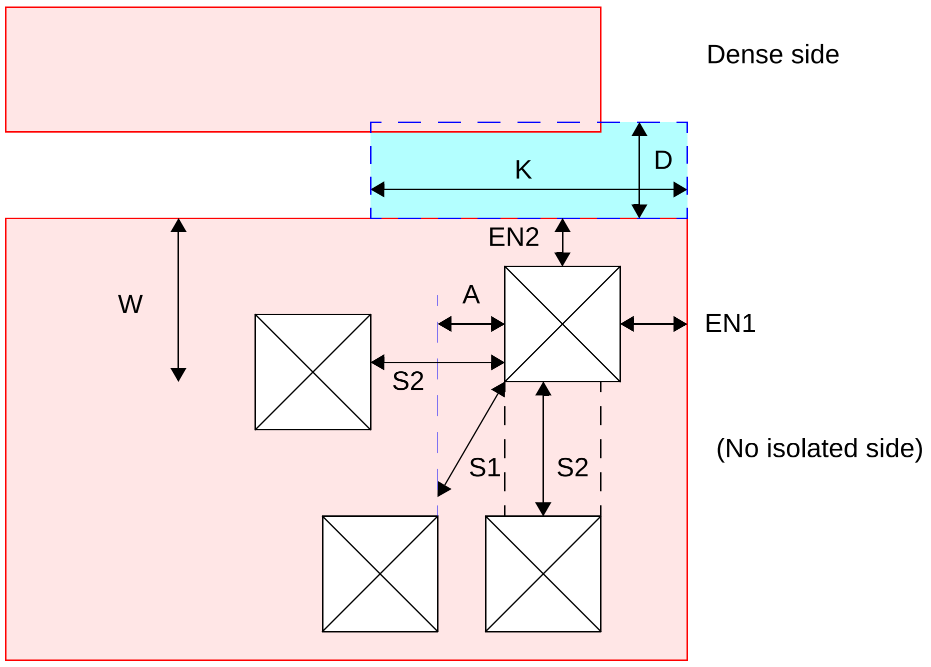


Figure 2-70 Misaligned Via Minimum Spacing Rule, Example 3



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.

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.

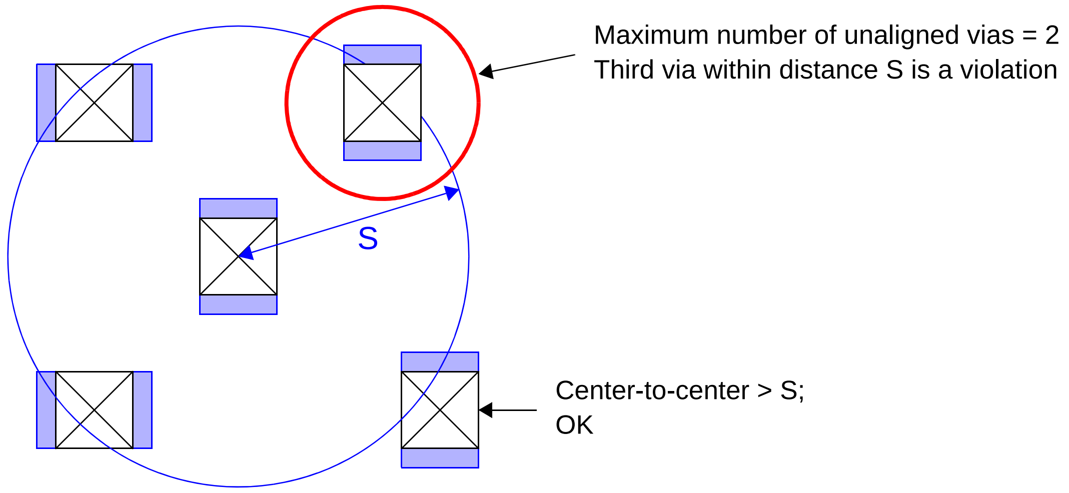
Similarly, the corner minimum spacing table attribute specifies the corner-to-corner spacing between each possible combination of vias sizes checked for adequate spacing. This is spacing S1 in the foregoing diagrams.

If you use the table-type cut1 and cut2 table size and name table attributes, 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.

Maximum Number of Unaligned Vias Rule

When self-aligned vias are unaligned (as described in Self-Aligned Via Spacing Rules), 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 2-71, no more than two self-aligned vias are allowed within a distance S, measured center-to-center.

Figure 2-71 Maximum Number of Unaligned Self-Aligned Vias Rule


Maximum number of unaligned vias = 2
Third via within distance S is a violation
S
Center-to-center > S;
OK

The following example shows the syntax of the rule:

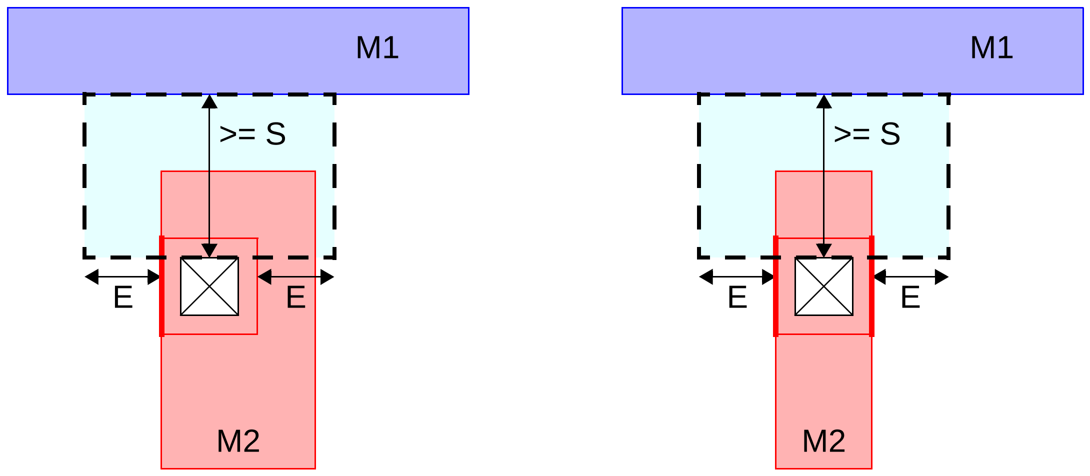
```
DesignRule {
  layer1                       = "VIA1"
  layer2                       = "VIA1"
  cut1TblSize                  = 1
  cut2TblSize                  = 1
  cut1NameTbl                  = (Vsm)
  cut2NameTbl                  = (Vsm)
  unalignedViaMaxNumCutsTbl    = (2)
  unalignedViaCenterRangeTbl = (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. 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 2-72.

Figure 2-72 Non-Self-Aligned Via Edge to Metal Spacing Rule



The following example shows the syntax of the rule:

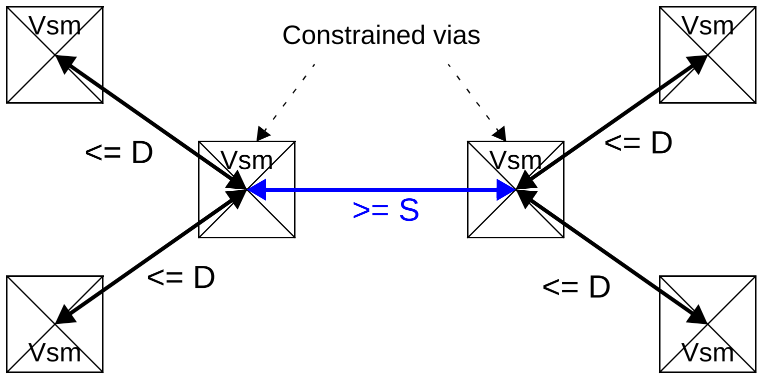
```
DesignRule {
  layer1 = "VIA1"
  layer2 = "MET1"
  metalToNonSelfAlignedEdgeExtension     = 0.42; extension E
  metalToNonSelfAlignedEdgeCutMinSpacing = 0.50; minimum spacing S
}
```

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.

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 that 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 2-73.

Figure 2-73 Constrained Via Spacing Rule



The following example shows the syntax of the rule:

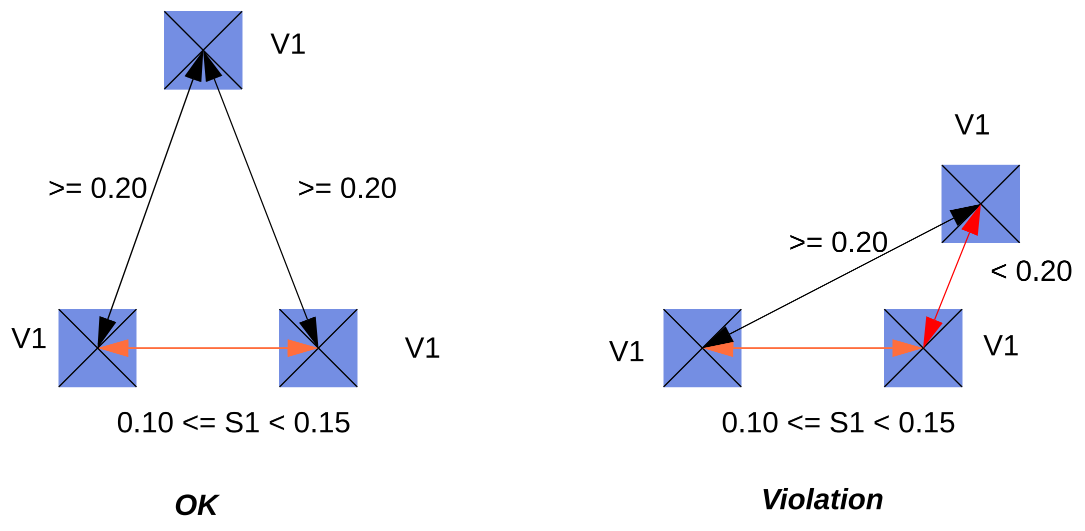
```
Layer "VX" {
  constrainedCutTblSize                  = 3
  constrainedCutNameTbl                  = (Vsm,Vh,Vv)
  constrainedCutConstrainingRangeTbl     = (0.136,0.148,0.148) # D values
  constrainedCutConstrainingNumCutsTbl = (2,2,2)
  constrainedCutCenterMinSpacingTbl      = (0.138,0.150,0.150) # 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.

Color-Based Constrained Via Spacing Rule

In the Color-Based Constrained Via Spacing rule, if two vias are on the same-color mask and their center-to-center spacing is within a certain range, then any other neighboring vias on the same mask must have a spacing greater than a specified value. Figure 2-74 shows two examples: the example on the left shows the two constrained vias within a range between 0.10 and less than 0.15. The third via is located at least 0.20 away from the two constrained vias. The example on the right shows the upper right via is located far enough away from one constrained via, but is too close to the other via (red arrow) and therefore is a violation to this rule.

Figure 2-74 Same-Color Constrained Via Spacing Rule



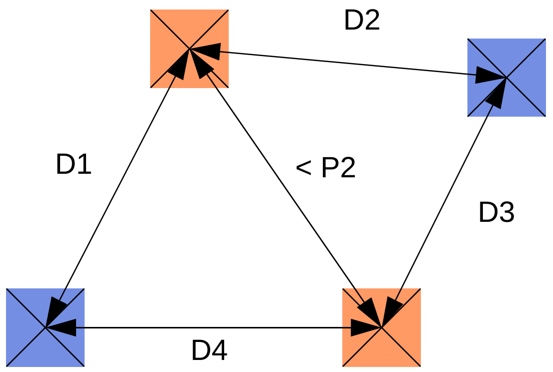
The syntax for this example is as follows:

```
Layer "Via1" {
   cutTblSize = 3
   cutNameTbl = (V1, V2, V3)
   ...
   sameColorConstrainEnclosedCutTblSize = 1
   sameColorConstrainEnclosedCutNameTbl = (V1)
   sameColorConstrainEnclosedCutRangeTbl = ("0.10, 0.15")
   sameColorConstrainEnclosedCutCenterMinSpacingTbl = (0.20)
}
```

Critical Via Spacing Rule

In the Critical Via Spacing rule, a via with a center-to-center spacing of less than P2 between itself and its N neighboring vias is a critical via. A common via is a via that has two critical via neighbors. The center-to-center spacing between a common via and a critical via must be at least a distance of P2. For example, in Figure 2-75, if distances D1, D2, D3, and D4 are at least P2, there is no violation. If distances D1, D2, D3, and D4 are all less than P2, then this is a violation of the rule.

Figure 2-75 Critical Via Spacing Rule



🟧 critical via (Vsq)

🟦 common via

The technology file syntax is as follows:

```
Layer ViaX {
   criticalCutTblSize = 1
   criticalCutNameTbl = (Vsq)
   criticalCutMinNumCutsTbl = N
   criticalCutRangeTbl  = (P2)
   critcalCutCenterMinSpacingTbl = (P2)
}
```

The entries in the `criticalCutNameTbl` are previously defined in the via's `cutNameTbl`. In Figure 2-75, N = 3.

Range-Enclosed Via Spacing Rule

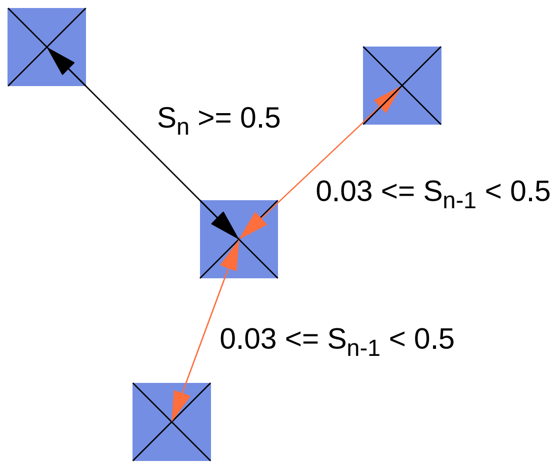
The range-enclosed via spacing rule states that if a via has N-1 neighboring vias with center-to-center spacing within a certain range, then the Nth neighboring via must be placed beyond the upper end of the range.

For example, in Figure 2-76, if

- N = 3
- Two of the neighboring vias' center-to-center spacing range is at least 0.03 but less than 0.50

Then, the third neighboring via must be at least 0.50 away.

Figure 2-76 Range-Enclosed Via Spacing



The syntax for this example is:

```
Layer ViaX {
   rangeEnclosedCutTblSize = 1
   rangeEnclosedCutNameTbl = Vsq
   rangeEnclosedCutRangeTbl = ("0.03, 0.5")
   rangeEnclosedCutMaxNumCutsTbl = (3)
}
```

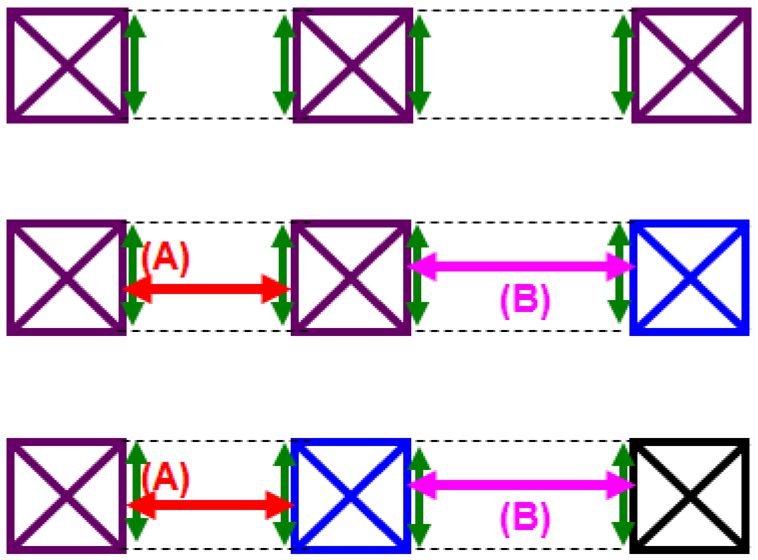
The `rangeEnclosedCutNameTbl` specifies values previously defined in the layer's `cutNameTbl`.

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 2-77 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 2-77 Line Via Spacing Rule



Three vias belong to the same net -- **rule does not apply**

Two vias belong to the same net and the remaining via belongs to a different net -- **rule applies**

Three vias belong to three different nets -- **rule applies**

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  = (b1, b2, b3)
}
```

The *n* monotonically increasing values b1 through b*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 2-1.

Table 2-1 Line Via Spacing Rule Values

Spacing to nearby via "A"	**Minimum spacing requirement "B" on other side**
b1 <= A < b2	B >= b3
b2 <= A < b3	B >= b2
b3 <= A	B >= b1

Figure 2-78 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, b1. This means that the minimum spacing requirement “B” on the other side is b3, the maximum table value, which defines the blue exclusion zone shown. This example shows a spacing violation.

Figure 2-78 Line Via Spacing Rule Example 1

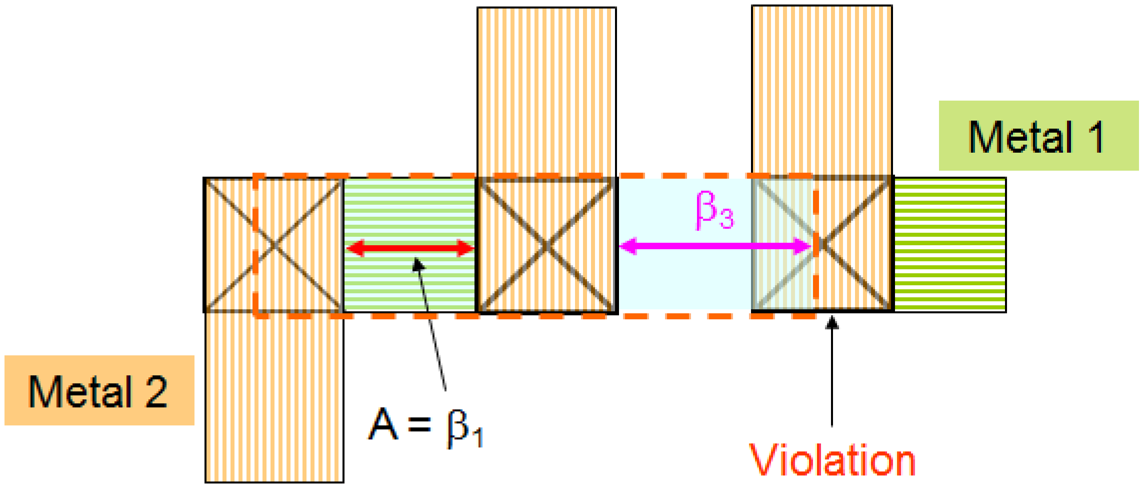
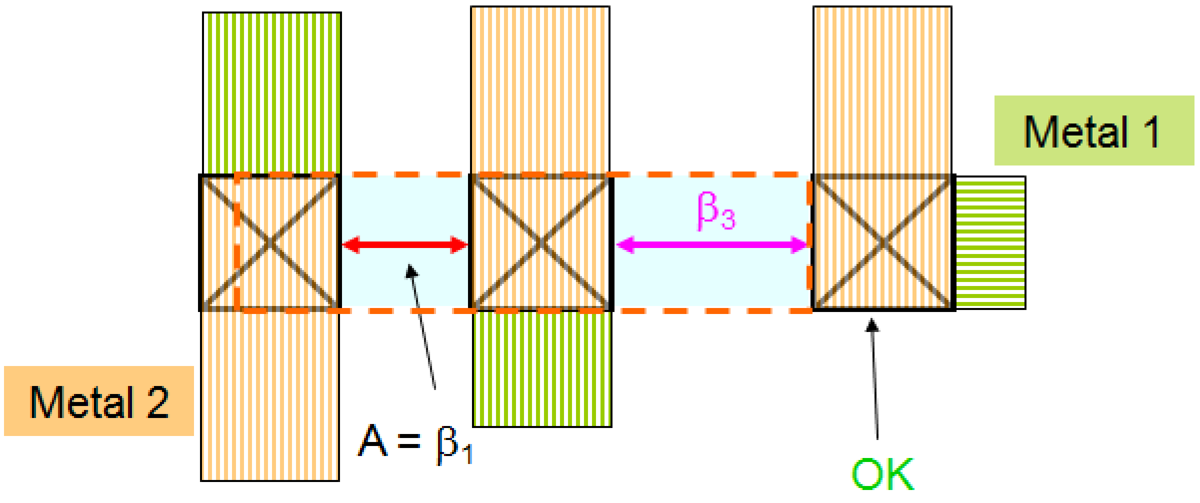


Figure 2-79 provides another example of rule usage. All three vias belong to different nets, so the rule applies. Again, the distance “A” between the left and middle vias is exactly the minimum value in the table, b1, so the minimum spacing requirement “B” on the other side is b3, the maximum table value. In this example, the spacing requirement is met.

Figure 2-79 Line Via Spacing Rule Example 2

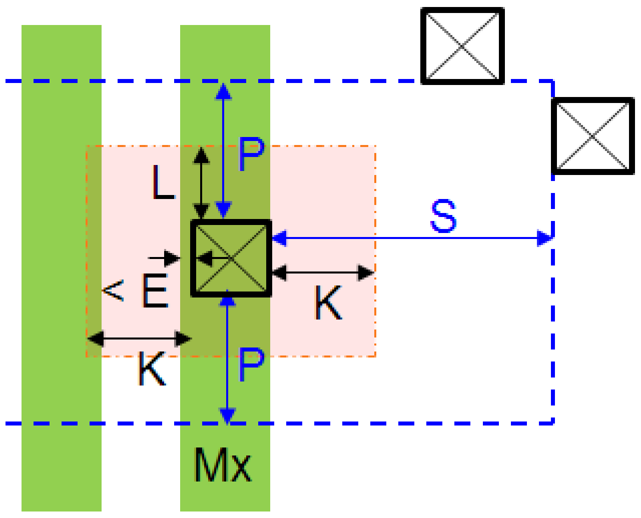


Edge Via Spacing Rule

An “edge via” is a via that meets both of the following conditions, as shown in Figure 2-80:

- 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 2-80 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:

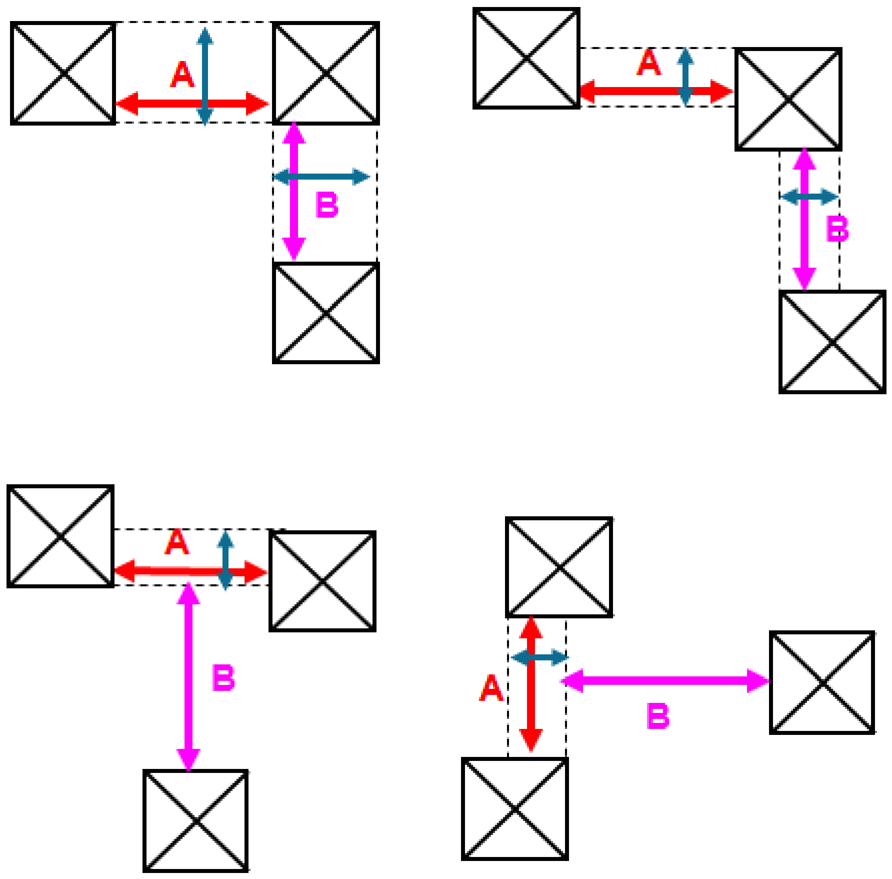
```
DesignRule {
  layer1                         = "Mx"
  layer2                         = "VIAx"
  edgeViaCutTblSize              = 3
  edgeViaCutNameTbl              = (Vs,Vrh,Vrv)
  edgeViaWireParallelLengthTbl  = (Ls, Lh, Lv)
  edgeViaWireMinSpacingTbl       = (Ks, Kh, Kv)
  edgeViaCutMinEnclosureTbl      = (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 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 2-81   Matrix Via Spacing Rule



This is the syntax of the matrix via spacing rule:

```
Layer "V2" {
  matrixCutTblSize      = 5
  matrixCutSpacingTbl  = (a1, a2, a3, a4, a5)
}
```

The *n* monotonically increasing values a1 through a*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 2-2.

Table 2-2 Matrix Via Spacing Rule Values

Spacing to nearby via “A”	**Minimum spacing requirement “B”**
a1 <= A < a2	B >= a5
a2 <= A < a3	B >= a4
a3 <= A < a4	B >= a3
a4 <= A < a5	B >= a2
a5 <= A	B >= a1

Figure 2-82 demonstrates usage of the rule. In Example 1, the distance “A” between the vias is exactly the minimum value in the table, a1. This means that the minimum spacing requirement “B” in the orthogonal direction is a5, the maximum table value, which defines the blue exclusion zone shown. Conversely, in Example 2, the distance “A” is a5, which causes the minimum spacing requirement to be a1.

Figure 2-83 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 2-82 Matrix Via Spacing Rule Examples 1 and 2

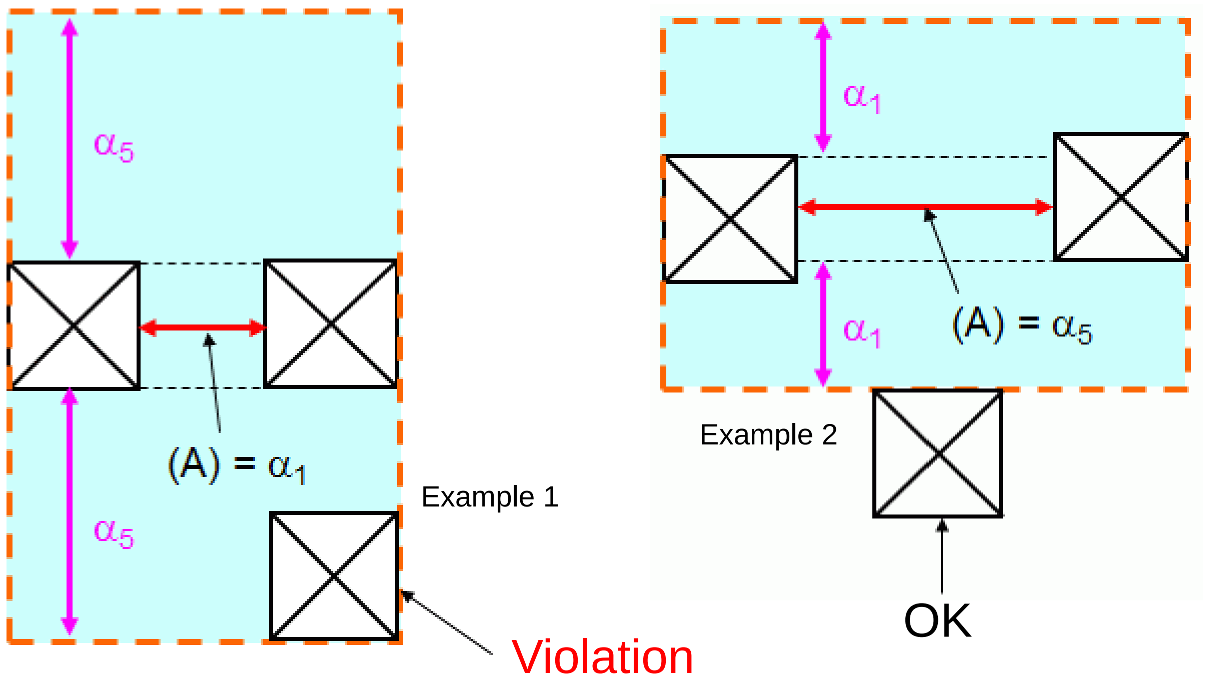
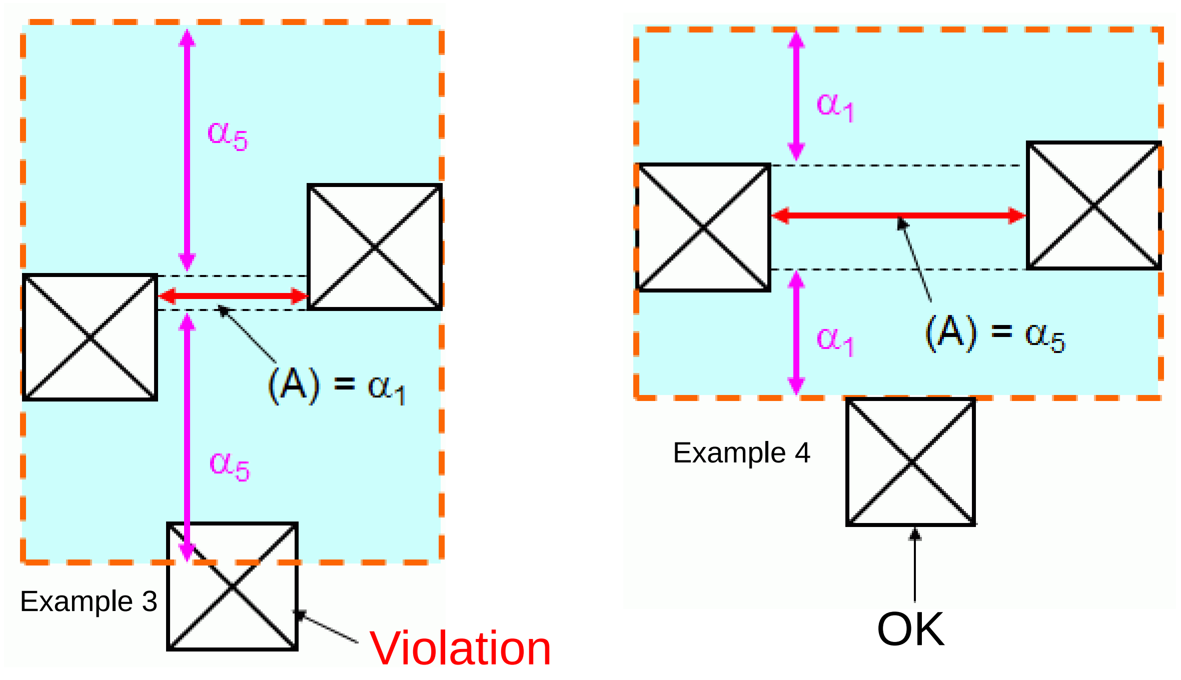


Figure 2-83 Matrix Via Spacing Rule Examples 3 and 4



Via Corner Spacing Rule Enhancement

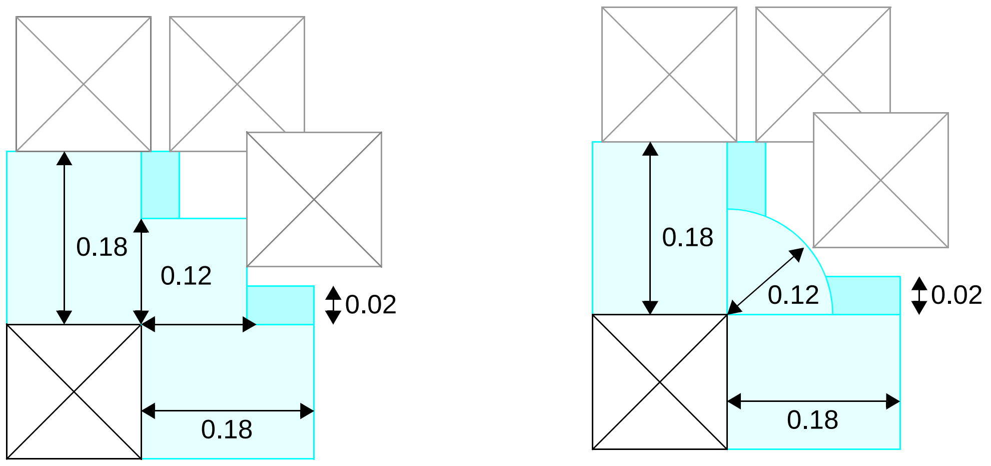
The via corner spacing rule specifies the minimum corner-to-corner spacing allowed between two vias. This rule applies only when the parallel spacing between the two vias is less than a given threshold. The basic rule is described in Via Corner Spacing Rule.

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 2-55.

Figure 2-84 Via Corner Spacing Rule



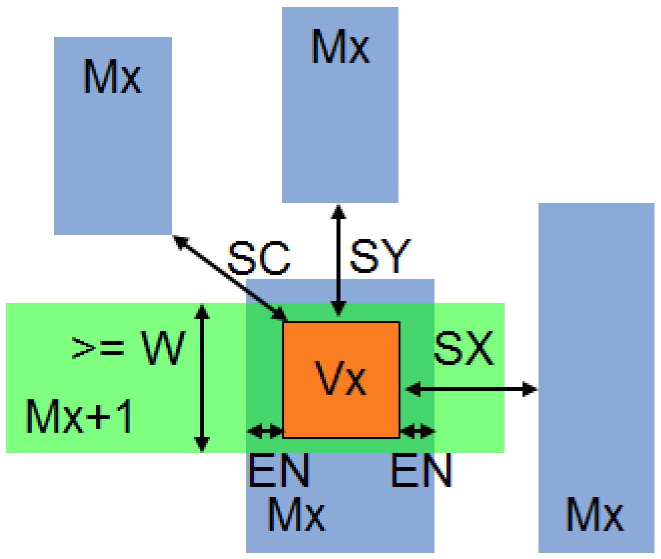
Exclusion area without using `minSpacingCornerKeepoutDiagonalCheck`

Exclusion area with `minSpacingCornerKeepoutDiagonalCheck=1`

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 2-85.

Figure 2-85 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                              = "MetalX"
  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.

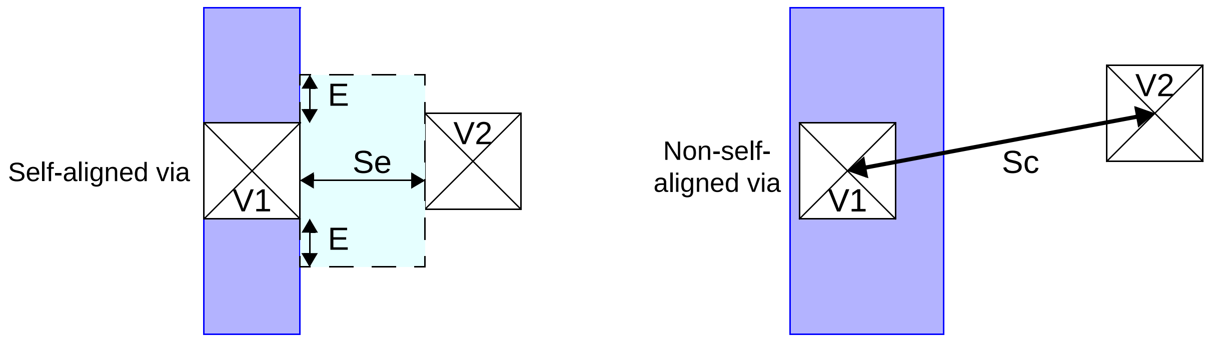
When the `minSpacingUpperEncCheckTbl` attribute is set to 1, the rule requires the upper-metal enclosure to be at least zero. Otherwise, there is no upper-metal enclosure requirement.

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.

In Figure 2-86, 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 2-86 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          = "ViaX"
  layer2          = "ViaX+1"
  cut1TblSize     = 1
  cut2TblSize     = 1
  cut1NameTbl     = (VNAME1)
  cut2NameTbl     = (VNAME2)
  minSpacingYParallelLengthThresholdTbl  = (-E)
  diffNetXMinSpacingTbl                  = (Se)
  minSpacingXParallelLengthThresholdTbl  = (-E)
  diffNetYMinSpacingTbl                  = (Se)
  diffNetNonSavCenterMinSpacingTbl       = (Sc)
 }
```

For example,

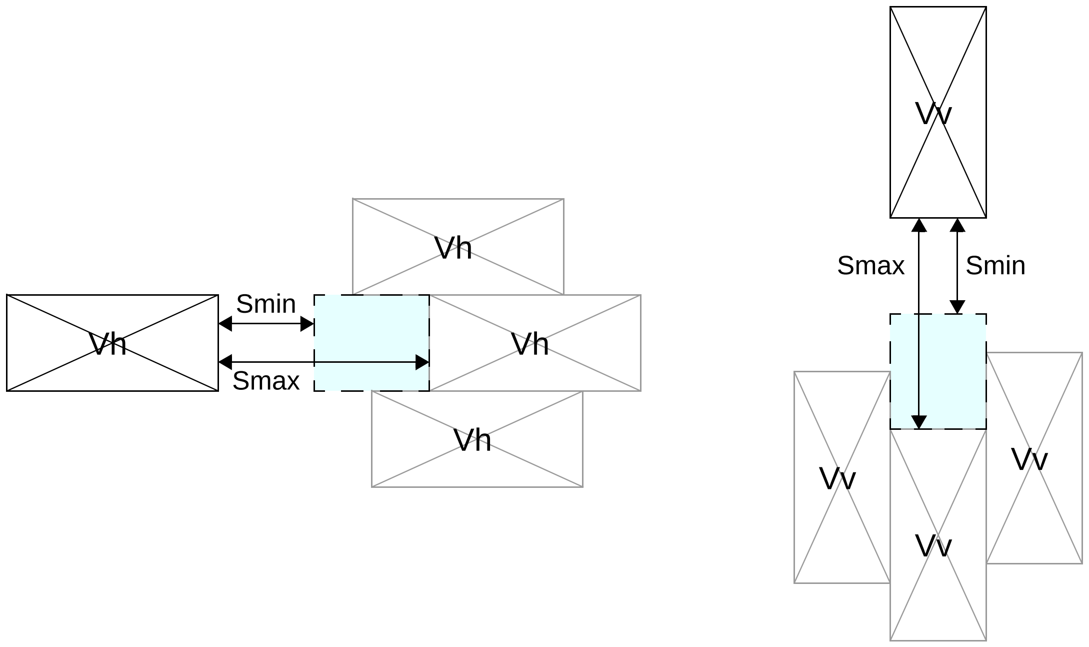
```
DesignRule {
  layer1          = "VIA1"
  layer2          = "VIA2"
  cut1TblSize     = 1
  cut2TblSize     = 1
  cut1NameTbl     = (V1LX)
  cut2NameTbl     = (V2LX)
  minSpacingYParallelLengthThresholdTbl  = (-0.022)
  diffNetXMinSpacingTbl                  = ( 0.045)
  minSpacingXParallelLengthThresholdTbl  = (-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 Smin to Smax, 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 2-87, 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 Smin to Smax from the edge of the via, as long as the parallel overlap is greater than zero. The colored rectangle shows the forbidden area.

Figure 2-87 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.

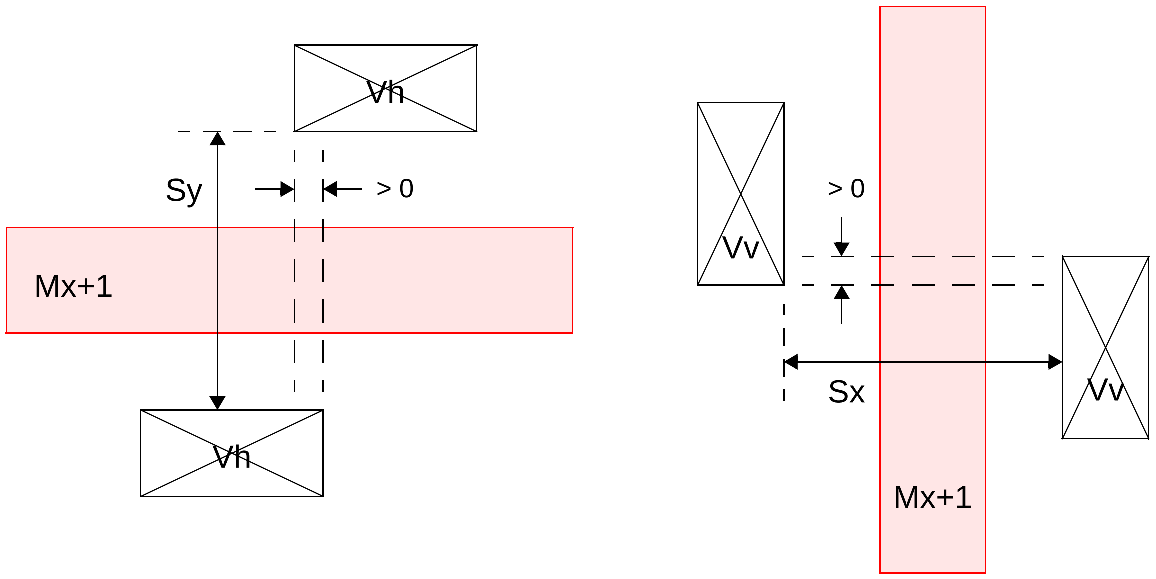
Here is an example of the rule:

```
DesignRule {
  layer1                        = "ViaX"
  layer2                        = "ViaX"
  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 2-88.

Figure 2-88 Separated Via Spacing Rule



For example,

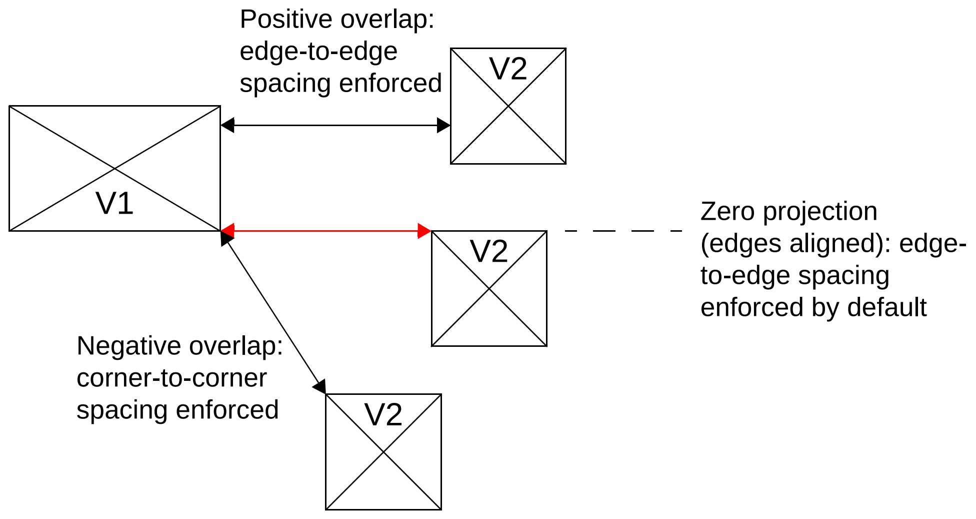
```
DesignRule {
  layer1                                      = "VIA1"
  layer2                                      = "VIA1"
  separatedCut1TblSize                        = 2
  separatedCut1NameTbl                        = (Vh, Vv)
  separatedCut2TblSize                        = 2
  separatedCut2NameTbl                        = (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 Sx = 0.136. Similarly, the Vh-to-Vh spacing in the y-direction is Sy = 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. 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 2-89.

Figure 2-89 Via Spacing Rules for Positive, Zero, and Negative Projection


Positive overlap: edge-to-edge spacing enforced
V2
V1
V2
Zero projection (edges aligned): edge-to-edge spacing enforced by default
Negative overlap: corner-to-corner spacing enforced
V2

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.

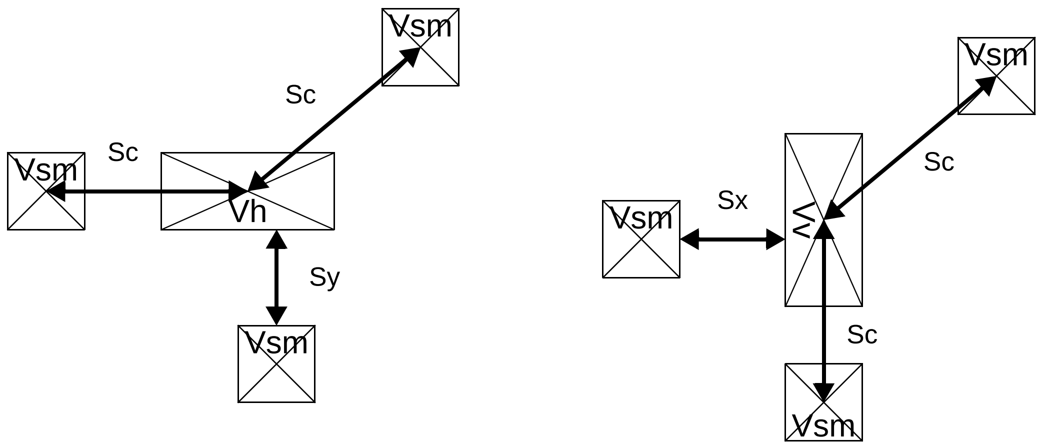
```
DesignRule {
  layer1          = "ViaX"
  layer2          = "ViaX+1"
  cut1TblSize     = 1
  cut2TblSize     = 2
  cut1NameTbl     = (V1NAME)
  cut2NameTbl     = (V2NAME1, V2NAME2)
  diffNetXMinSpacingTbl             = (Sxh,Sxv)
  diffNetYMinSpacingTbl             = (Syh,Syh)
  diffNetCornerMinSpacingTbl        = (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 2-90, 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 2-90 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                        = "ViaX"
  layer2                        = "ViaX+1"
  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)
}
```

Stacked Via Keepout Rule

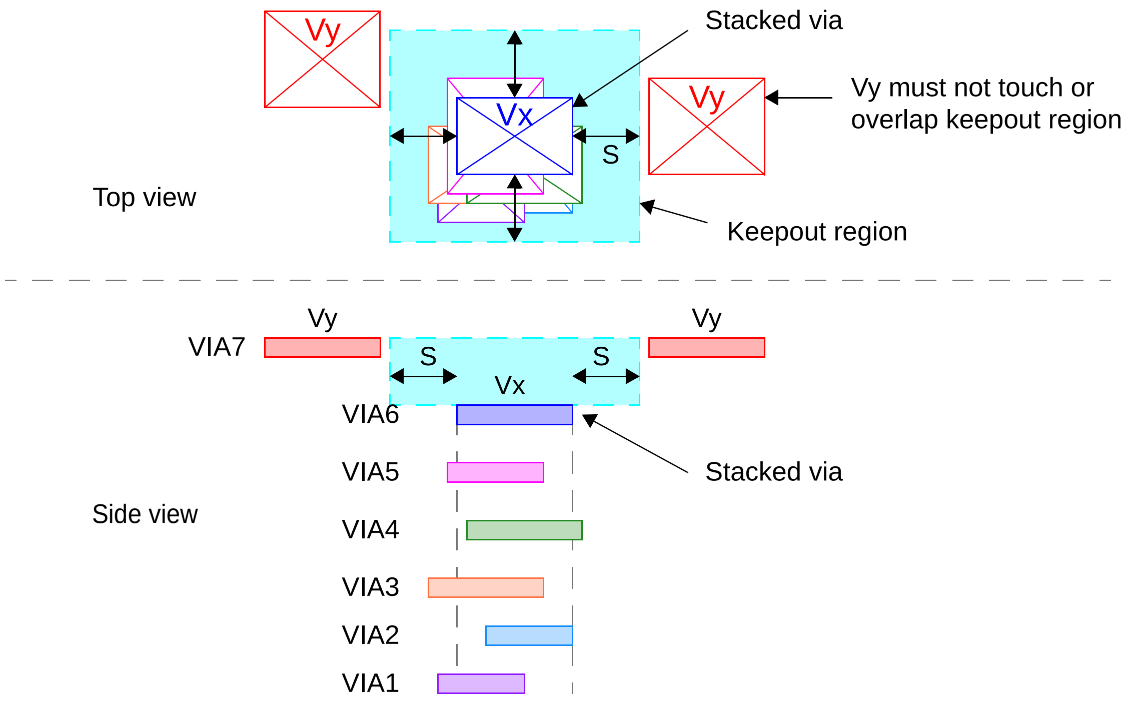
A stacked via is a via that overlaps all vias in a consecutive sequence of lower-level via layers.

The stacked via keepout rule specifies the minimum spacing between a stacked via and a via in the next-higher via layer, as shown in Figure 2-91. The Vx via on the VIAn layer is a stacked via because it overlaps vias in the lower-level via layers. The rule specifies a keepout region extending by a distance S from the edges of Vx. The Vy vias in the next-higher via layer must be placed outside of the keepout region. You can specify different keepout distances between different cuts defined for the VIAn layer and the next-higher via layer.

The rule is checked only when the stacked via has the number of layers specified in the `stackViaLayerCount` attribute in the via `Layer` section associated with the stacked via. In Figure 2-91, the `stackViaLayerCount` attribute for the VIA6 layer is 6.

You can optionally upsize the vias in the stack in the x- and y-directions for the purpose of determining whether the vias overlap by setting the `stackViaCutXUpsizeTbl` and `stackViaCutYUpsizeTbl` attributes in the associated via `Layer` sections.

Figure 2-91 Stacked Via Keepout Rule

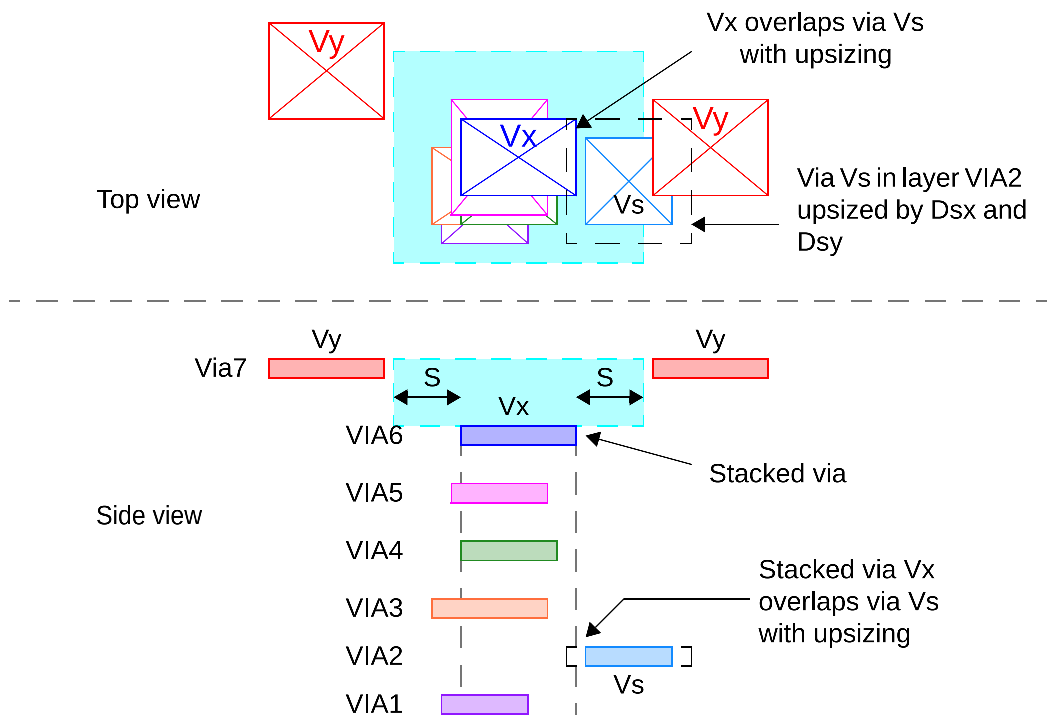


This is the basic syntax for the rule:

```
Layer "VIAn" {
   ...
   stackViaLayerCount = *count*
   ...
}
 ...
DesignRule {
  layer1                      = "VIAn"
  layer2                      = "VIAm"
  stackViaKeepoutCut1TblSize = 1
  stackViaKeepoutCut1NameTbl = (Vx)
  stackViaKeepoutCut2TblSize = 1
  stackViaKeepoutCut2NameTbl = (Vy)
  stackViaKeepoutDistTbl     = (S)
}
Layer "VIAn" {
   ...
   stackViaLayerCount = 6
   ...
}
```

Figure 2-92 shows the effect of upsizing the vias for the purpose of determining whether the vias overlap. In this figure, the Vx via does not actually overlap the Vs via on the VIA2 layer; however, when the upsizing is considered, the vias create a stacked via on the VIA1 through VIA6 layers.

Figure 2-92 Stacked Via Minimum Spacing Rule With Upsized VIA2 Vias


Vy
Vx overlaps via Vs with upsizing
Vx
Vy
Top view
Vs
Via Vs in layer VIA2 upsized by Dsx and Dsy
Vy
Vy
Via7
S
S
Vx
VIA6
Stacked via
VIA5
Side view
VIA4
Stacked via Vx overlaps via Vs with upsizing
VIA3
VIA2
Vs
VIA1

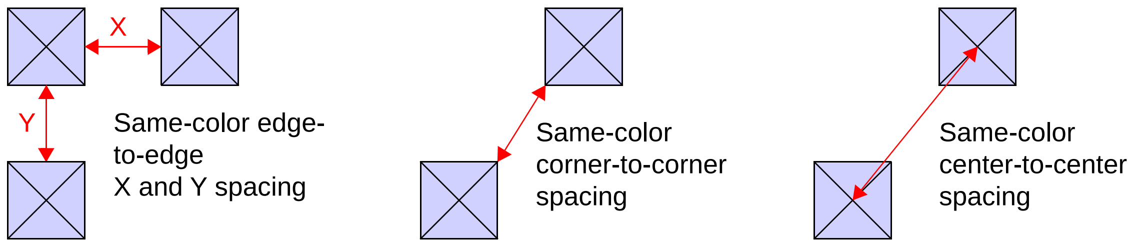
Use the following syntax to specify the upsizing attributes for a via layer:

```
Layer "VIAx" {
  cutTblSize              = 4
  cutNameTbl              = (Vs,Vh,Vv,Vb)
   ...
  stackViaCutXUpsizeTbl = (Dsx,Dhx,Dvx,Dbx)
  stackViaCutYUpsizeTbl = (Dsy,Dhy,Dvy,Dby)
   ...
}
```

Double-Patterning Via Spacing Rule

At advanced technology nodes, the lowest-level via mask layer can be implemented with double-patterning masks. A stricter minimum via spacing requirement can apply for via cuts of the same color. The minimum spacing requirement can be specified using edge-to-edge, corner-to-corner, or center-to-center measurements, as shown in the following figure.

Figure 2-93 Multiple-Patterning Same-Color Via Spacing Rule Measurements


X
Y
Same-color edge-to-edge X and Y spacing
Same-color corner-to-corner spacing
Same-color center-to-center spacing

A table-based rule specifies the same-color minimum spacing requirement for each combination of nearby vias, using the following syntax:

```
Layer "Vx" {
  sameColorCutTblSize             = 3
  sameColorCutNameTbl             = (Vs, Vh, Vv)
  sameColorCutXEdgeMinSpacingTbl  = (Vs-Vs, Vs-Vh, Vs-Vv,   ; 3X3 table
                                     Vh-Vs, Vh-Vh, Vh-Vv,
                                     Vv-Vs, Vv-Vh, Vv-Vv)

  sameColorCutYEdgeMinSpacingTbl  = ( ... ... ... )         ; 3X3 table

  sameColorCutCornerMinSpacingTbl = ( ... ... ... )         ; 3X3 table

  sameColorCutCenterMinSpacingTbl = ( ... ... ... )         ; 3X3 table

}
```

Vs-Vh, for example, represents the minimum spacing between Vs and Vh via cuts of the same color. A value of -1 in the table means that the rule does not apply for that pair.

By default, the rule is waived when the nearby same-color cuts are connected by the same metal segment, either on the upper or lower metal layer. To enforce (not waive) the rule under these conditions, add the following attributes to the `Layer` section:

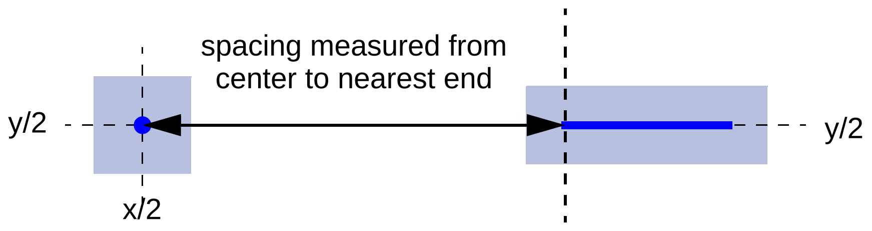
```
  sameColorCutSameSegXEdgeExcludedSpacingTbl = ("Sxmin,Sxmax", ...)
  sameColorCutSameSegYEdgeExcludedSpacingTbl = ("Symin,Symax", ...)
  sameColorCutSameSegCornerExcludedSpacingTbl= ("Scmin,Scmax", ...)
  sameColorCutSameSegCenterExcludedSpacingTbl= ("Semin,Semax", ...)
```

For each via pair, the rule is enforced (not waived) when the distance between the two vias is in the range from the specified minimum value to the specified maximum value. Each attribute takes a 3-by-3 table of min-max value pairs.

Color-Based Via Centerline Spacing Rule

The color-based via centerline spacing rule specifies the minimum spacing between same-color vias based on the centers (not the edges) of the vias. The center of a square via is a dot and the center of a bar via is a line (see Figure 2-94).

Figure 2-94 Square Via Center and Bar Via Center



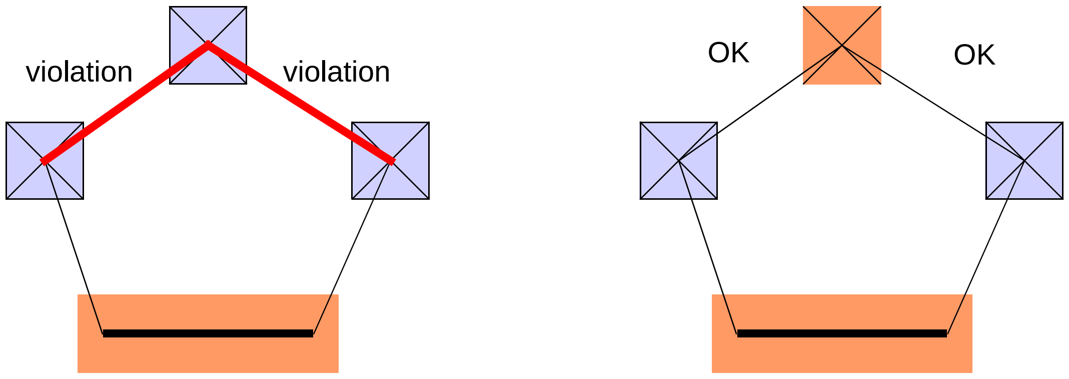
The syntax for this rule is:

```
Layer "ViaX" {
   sameColorCutTblSize = 3
   sameColorCutNameTbl = (Vsq, Vh, Vv)
   sameColorCutCenterLineMinSpacingTbl = (... ..., ...) ; 3 x 3 table
}
```

If the center-to-center spacing is less than the value in the `sameColorCutCenterLineMinSpacingTbl`, then the two vias should be placed on different masks. Otherwise, it is a violation this spacing rule.

For the example in Figure 2-95, assume that the centerline spacing between vias is less than the value in the `sameColorCutCenterLineMinSpacingTbl`. The figure shows a violation to the rule and the subsequent color change needed to fix the violation.

Figure 2-95 Violation to the Color-Based Via Centerline Rule



In an exception to this rule, you can have a smaller via center-to-center spacing when the via falls into a specified parallel run length range. This is the definition of the rule:

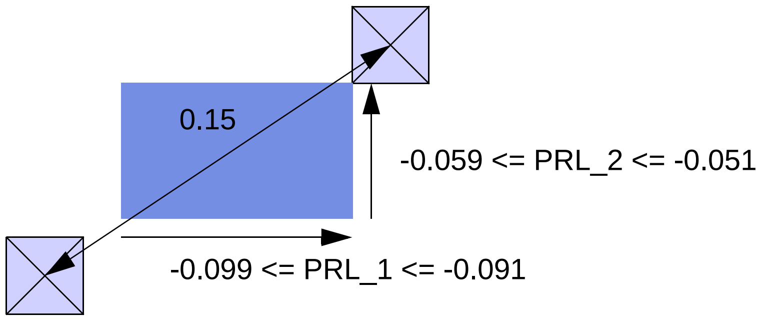
```
Layer ViaX {
   sameColorCutTblSize = 3
   sameColorCutNameTbl = (Vsq, Vh, Vv)
   sameColorCutCenterLineMinSpacingTbl = (... ..., ...) ; 3X3 table
   sameColorCutCenterLineExcludedMinSpacingTbl = ( S1, -1, ...)
   sameColorCutCenterLineExcludedParallelLength1RangeTbl =
                                          ("-PRL_a1, -PRL_a2", -1, ...)
   sameColorCutCenterLineExcludedParallelLength2RangeTbl =
                                          ("-PRL_b1, -PRL_b2", -1, ...)
}
```

For example, if the technology file contains the following entries:

```
sameColorCutNameTbl = 3
sameColorCutCenterLineMinSpacingTbl = (0.22, 0.22, 0.22,
                                       0.22, 0.22, 0.22,
                                       0.22, 0.22, 0.22)
sameColorCutCenterLineExcludeMinSpacingTbl = (0.15, -1, -1,
                                                 -1. -1, -1,
                                                 -1, -1, -1)
sameColorCutCenterLineExcludeParallelLength1RangeTbl =
                                   ("-0.059, -0.051", -1, -1,
                                                 -1, -1, -1,
                                                 -1, -1, -1)

sameColorCutCenterLineExcludeParallelLength2RangeTbl =
                                   ("-0.099, -0.091", -1, -1,
                                                 -1, -1, -1,
                                                 -1, -1, -1)
```

If the PRL_1 falls within the range of -0.059 and -0.051 and PRL_2 falls within range -0.099 and -0.091, then the excluded area minimum spacing is 0.15 (see Figure 2-96).

Figure 2-96 Exception to Color-Based Via Centerline Spacing Rule



A violation of this exception occurs when:

- The parallel run lengths of the excluded area do not fall within the ranges specified by the exclusion tables
- The center-to-center spacing is less than the specified minimum spacing

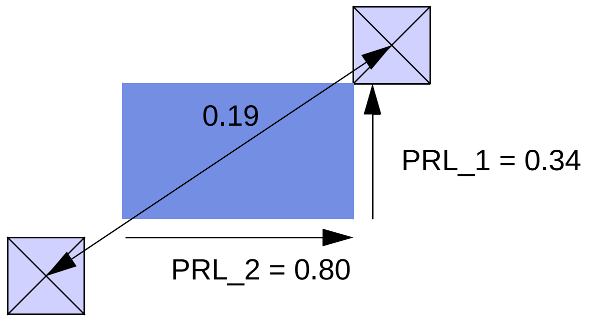
Via Center Spacing Exception Rule

The via center spacing exception rule allows a smaller via center-to-center spacing when the parallel run lengths in the excluded area between the two vias fall into a specific range. In the example in Figure 2-97, if the parallel run lengths, PRL_1 and PRL_2, fall into a range as specified by the parallel length tables, then the spacing value in the minimum spacing table corresponds to this value. In these tables, the values are a range pair. For example, if the table contains the following definitions:

```
sameNetCenterMinSpacingTbl = (0.20, 0.20, 0.20,
                              0.20, 0.20, 0.20,
                              0.20, 0.20, 0.20)
sameNetCenterExcludedParallelLength1RangeTbl =
                             ("0.25, 0.23, 0.35, 0.30", -1, -1, ...)
sameNetCenterExcludedParallelLength2RangeTbl =
                             ("0.75, 0.65, 0.85, 0.76", -1, -1, ...)
sameNetCenterExcludedMultiMinSpacingTbl = ("0.15, 0.19", -1, -1, ...)
```

If PRL_1 is 0.34 and PRL _2 is 0.80, the parallel lengths fall into the second range pair for the range tables and therefore, the minimum spacing is 0.19 (the second spacing value).

Figure 2-97 Violation to Via Center Spacing Exception Rule



Define this rule in the `DesignRule` section as follows:

```
DesignRule {
   layer1                          = "Vx"
   layer2                          = "Vy"
   cut1TblSize                     = 3
   cut2TblSize                     = 3
   cut1NameTbl                     = (Vsm, Vh, Vv)
   cut2NameTbl                     = (Vsm, Vh, Vv)
   sameNetCenterMinSpacingTbl      = (S2, ...)   # existing
   diffNetCenterMinSpacingTbl      = (S2, ...)   # existing
   sameNetCenterExcludedMultiMinSpacingTbl = ("S11, S12", -1, ...)
   sameNetCenterExcludedParallelLength1RangeTbl =
                                     ("-a21, -a11, -a22, -a12", -1, ...)
   sameNetCenterExcludedParallelLength2RangeTbl =
                                     ("-b21, -b11, -b22, -b12", -1, ...)
   diffNetCenterExcludedMultiMinSpacingTbl = ("S11, S12", -1, ...)
   diffNetCenterExcludedParallelLength1RangeTbl =
                                     ("-a21, -a11, -a22, -a12", -1, ...)
   diffNetCenterExcludedParallelLength2RangeTbl =
                                     ("-b21, -b11, -b22, -b12", -1, ...)
```

Via Ladder Rule

A via ladder is a structure that consists of metal geometry and cut geometry. Use this rule to guide the router on specifying the via layers connecting metal layers. A via ladder contains a user-specified number of cuts on each cut layer. You specify the number of cuts in a row and the number of rows. A row of cuts can be in the vertical or horizontal direction, depending on the preferred direction of the metal layer.

By default, a via ladder is not inserted if there are fixed shapes on higher layers that block the insertion. To increase the insertion rate, you can allow staggering of the vias on each level of the via ladder to avoid the fixed shapes. To specify the maximum number of tracks allowed for staggering on each level, set the `maxNumStaggerTracksTbl` attribute. You can specify a value between 0 and 9 for each level; the default is 0, which disables staggering.

The syntax for this rule is

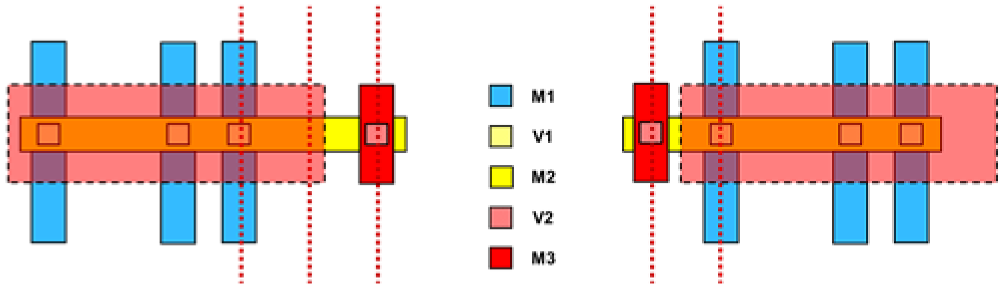
```
ViaRule "name" {
   cutLayerNameTblSize     = 3
   cutLayerNameTbl         = (VIA1, VIA2, VIA3) # via layer names
   cutNameTbl              = (CUT1, CUT2, CUT3) # cut names
   numCutRowsTbl           = (R1, R2, R3)       # rows in via ladder
   numCutsPerRowTbl        = (C1, C2, C3)       # columns in via ladder
   upperMetalMinLengthTbl  = (L1, L2, L3)       # minimum length rule
   cutXMinSpacingTbl       = (X1, X2, X3)       # X-direction spacing
   cutYMinSpacingTbl       = (Y1, Y2, Y3)       # Y-direction spacing
   maxNumStaggerTracksTbl  = (S1, S2, S3)       # maximum staggering
   forHighPerformance      = 0 | 1              # performance usage
   forElectromigration     = 0 | 1              # electromigration usage
}
```

The following example allows staggering of up to one track on the first level, up to three tracks on the second level, and no staggering on the third level:

```
maxNumStaggerTracksTbl = (1, 3, 0)
```

Figure 2-98 shows the possible solutions to avoid a blockage on the M3 layer when you allow staggering of up to three tracks.

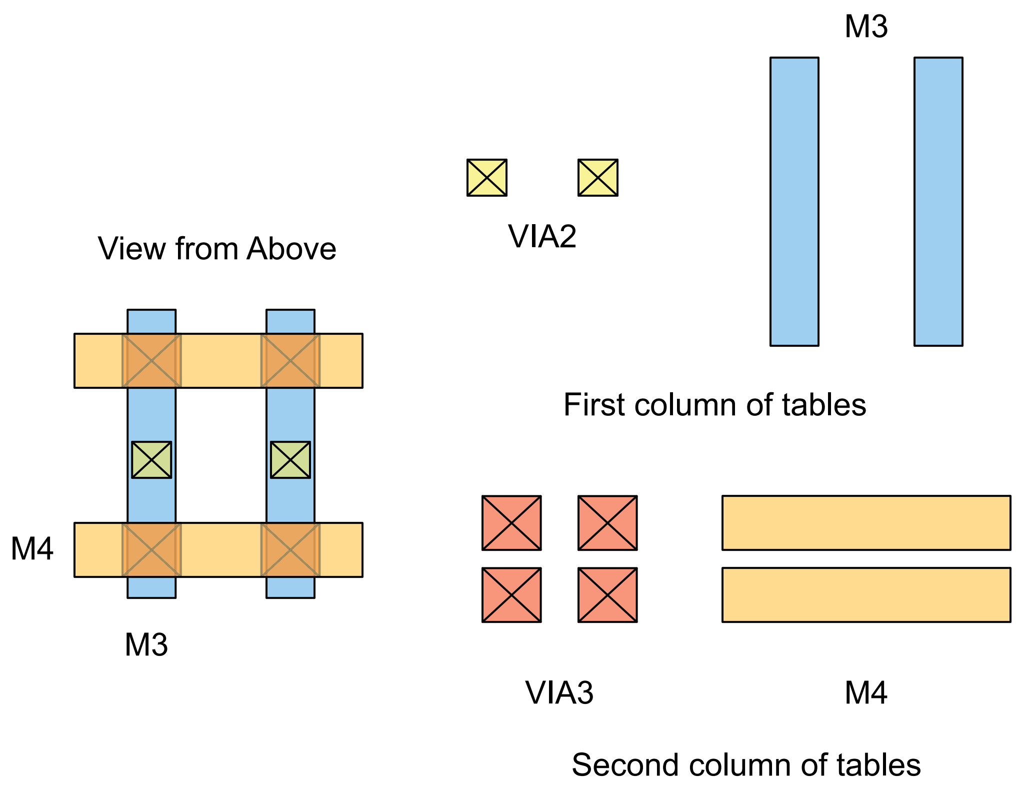
Figure 2-98 Via Ladder With Staggering



An example of a via ladder rule, shown in Figure 2-99, is defined as follows:

```
ViaRule "VL2" {
   cutLayerNameTblSize  = 2
   cutLayerNameTbl      = (VIA2, VIA3)
   cutNameTbl           = (Vsq, Vsq)
   numCutRowsTbl        = (2, 2)          # number of rows
   numCutsPerRowTbl     = (1, 2)          # number of cuts in a row
   ndrReservedTrackMode = off             # off, top, all
}
```

Figure 2-99 Example of Via Ladder Rule VL2



The `ndrReservedTrackMode` attribute only applies to nondefault routing rule (NDR) nets. Table 2-3 lists the possible values for the attribute.

Table 2-3 Values for the ndrReservedTrackMode Attribute

Mode	Description
`off`	Use only default and nonreserved tracks (default mode).
`top`	Use appropriate reserved tracks on the top via ladder layer. Use nonreserved and default tracks on all other layers. The top layer can also use nonreserved tracks where the width on the track matches the NDR width. Default tracks are not allowed on the top layer.
`all`	Use appropriate reserved tracks on all layers. Use nonreserved tracks on all layers where the width of the track matches the NDR width.

You can define multiple via ladder rules in the technology file. For example, an additional ladder rule might be specified as follows:

```
ViaRule "VL3" {
   cutLayerNameTblSize   = 2
   cutLayerNameTbl       = (VIA2, VIA3)
   cutNameTbl            = (Vsq, Vsq)
   numCutRowsTbl         = (1, 1)          # number of rows
   numCutsPerRowTbl      = (4, 2)          # number of cuts in a row
   ndrReservedTrackMode  = off             # off, top, all
}
```

The technology file must contain at least one ContactCode via specification with a cut size that matches the cut size specified for each of the relevant cut layers in a specified via ladder. When constructing a via ladder, the tool is free to choose any ContactCode with a cut size matching the specified dimensions in the via ladder. If no cut size matches the ContactCode, then it is an error and the via ladder specification is invalid.

Via-to-Metal Spacing Rules

The via-to-metal spacing rules are described in the following sections:

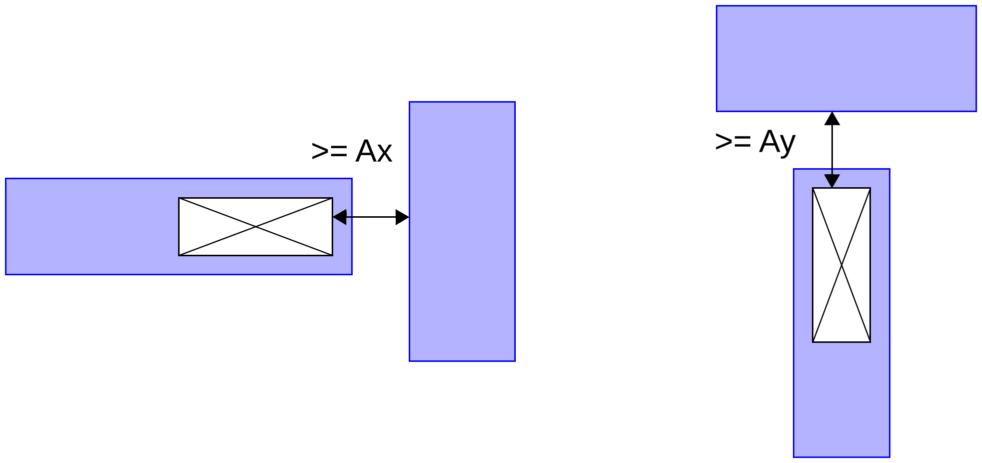
- Cut to Metal Orthogonal Spacing Rule
- Asymmetric Via-to-Metal Spacing Rule
- Via to Concave Metal Corner Spacing Rule

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) to allow a cut layer and a metal layer to be specified as the two layers affected by the rule.

In Figure 2-100, the spacing between the short side of the horizontal bar cut and the outside metal must be at least Ax. Likewise, the short side of the vertical bar cut must be at least the distance Ay from the external metal.

Figure 2-100 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.

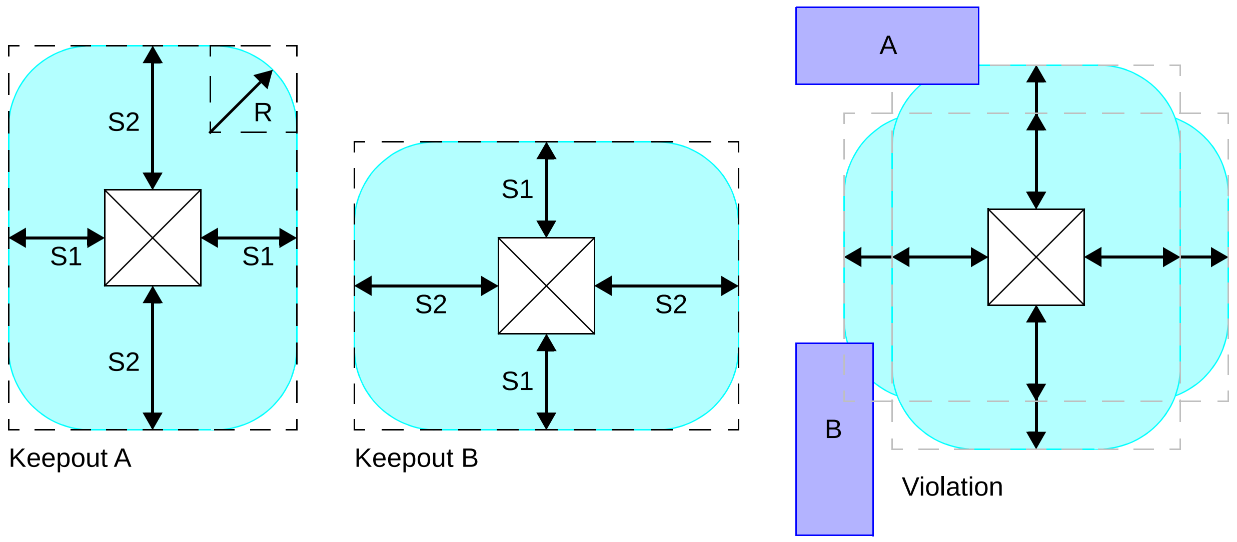
Asymmetric Via-to-Metal Spacing Rule

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 2-101. 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 2-101 Asymmetric Via-to-Metal Spacing Rule

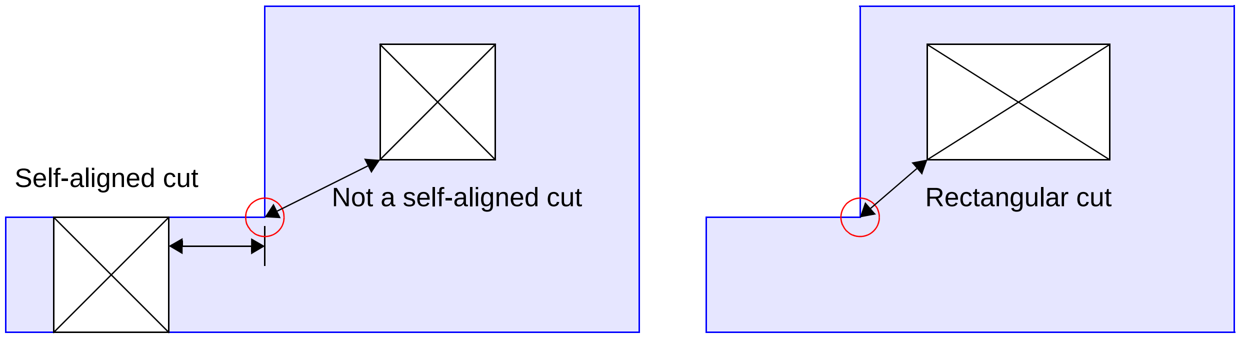


Each keepout region extends away from the edges of the via by the length and width attributes, taken separately in the x-direction and y-direction. The shorter extension S1 applies to the x-direction for Keepout A and applies to the y-direction for Keepout B. Each rectangular keepout region is reduced 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.

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 2-102.

Figure 2-102 Via to Concave Metal Corner Spacing Rule



The following example shows the syntax of the rule:

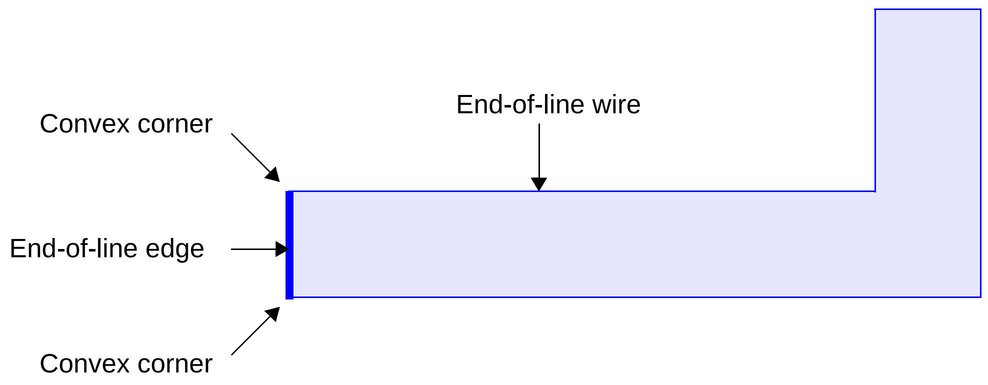
```
DesignRule {
  layer1                                  = "METAL2"
  layer2                                  = "VIA1"
  minSpacing                              = 0
  concaveMetalToSelfAlignedCutMinDist     = 0.036
  concaveMetalToNonSelfAlignedCutMinDist  = 0.034
  concaveMetalToRectCutMinDist            = 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.

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 2-103.

Figure 2-103 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
- Preferred and Nonpreferred 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
- Single-Side Isolated Neighbor End-of-Line Spacing 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
- L-Shaped End-of-Line Spacing Rule

- End-of-Line to Concave Corner Spacing Rule

End-of-Line to End-of-Line Spacing Rule

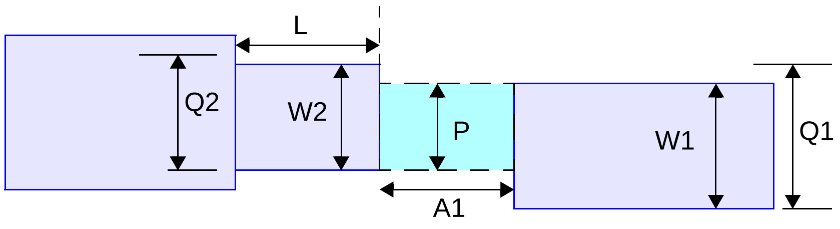
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see Alternative 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 2-104, 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, 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 2-104 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:

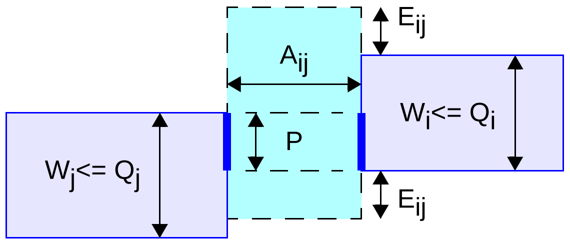
```
Layer "MetalX" {
  endOfLine1NeighborEndToEndThreshold      = 0.288
  endOfLine1NeighborEndToEndThreshold2     = 0.270
  endOfLine1NeighborEndToEndMinLength      = 0.300
  endOfLine1NeighborEndToEndParallelWidth = 0.144
  endOfLine1NeighborEndToEndMinSpacing     = 0.104
}
```

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.

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 2-105.

Figure 2-105 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}.

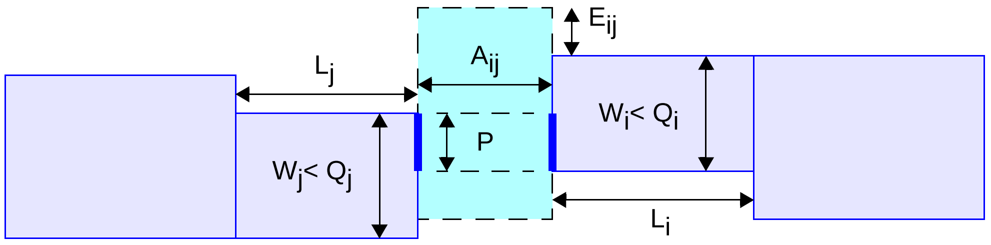
Here is an example of a rule based on a two-by-two table.

```
Layer "MetalX" {
  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, 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 2-106, use `endOfLine1NeighborEndToEndTblMinLength` to specify the length thresholds.

Figure 2-106 One-Neighbor 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 "MetalX" {
  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)
}
```

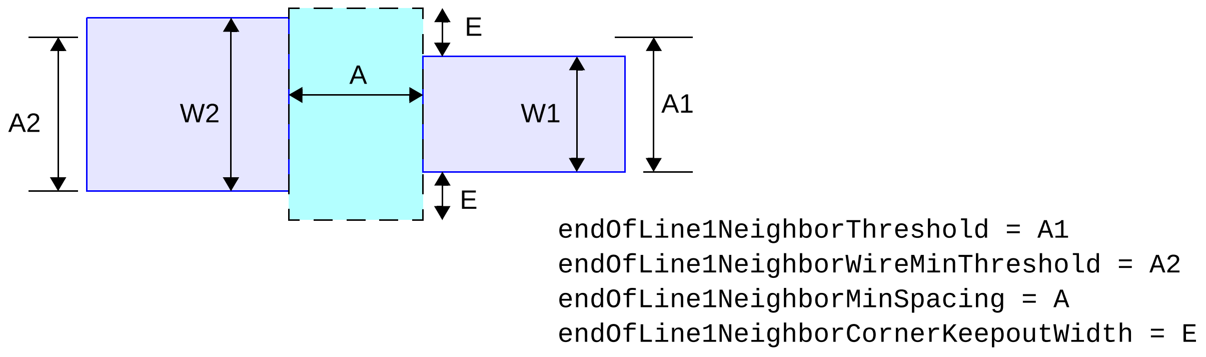
One-Neighbor End-of-Line Spacing Rule

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.

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.

In Figure 2-107, 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, 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 2-107 One-Neighbor End-of-Line Spacing Rule, Two Widths



```
endOfLine1NeighborThreshold = A1
endOfLine1NeighborWireMinThreshold = A2
endOfLine1NeighborMinSpacing = A
endOfLine1NeighborCornerKeepoutWidth = E
```

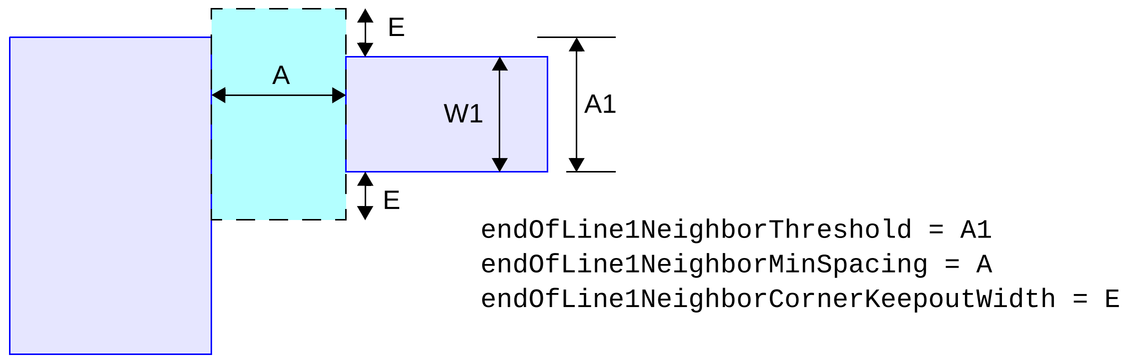
Here is an example of the syntax used for this rule:

```
Layer "MetalX" {
  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 2-108, if the metal width W1 is less than or equal to the width threshold A1, 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 2-108 One-Neighbor End-of-Line Spacing Rule, One Width



```
endOfLine1NeighborThreshold = A1
endOfLine1NeighborMinSpacing = A
endOfLine1NeighborCornerKeepoutWidth = E
```

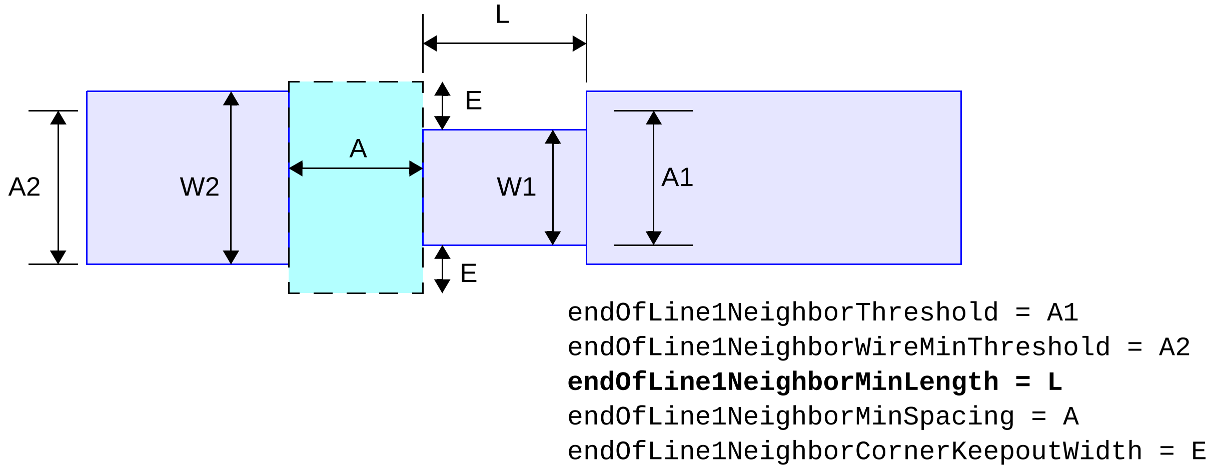
Here is an example of the syntax used for this rule:

```
Layer "MetalX" {
  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 2-109.

Figure 2-109 One-Neighbor End-of-Line Spacing Rule, Variable Width



```
endOfLine1NeighborThreshold = A1
endOfLine1NeighborWireMinThreshold = A2
endOfLine1NeighborMinLength = L
endOfLine1NeighborMinSpacing = A
endOfLine1NeighborCornerKeepoutWidth = E
```

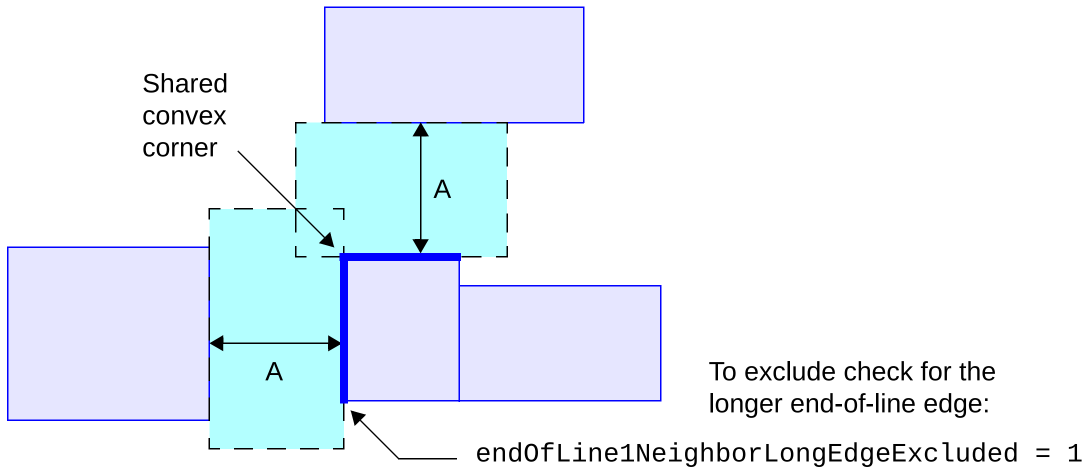
Here is an example of the syntax used for this rule:

```
Layer "MetalX" {
  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, 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 2-110, by default, the one-neighbor end-of-line spacing rule applies to both edges.

Figure 2-110 Two Adjacent End-of-Line Edges

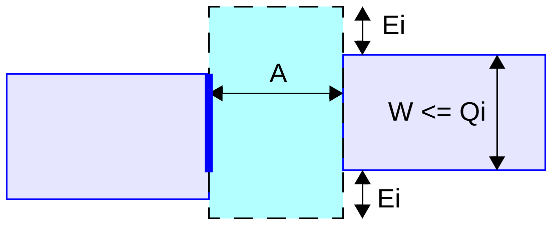

Shared convex corner
A
A
To exclude check for the longer end-of-line edge:
endOfLine1NeighborLongEdgeExcluded = 1

To apply the rule only to the shorter of the two adjacent edges, define an additional attribute, `endOfLine1NeighborLongEdgeExcluded`, and set it to 1:

```
Layer "MetalX" {
  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, ... *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 2-111.

Figure 2-111 One-Neighbor End-of-Line Spacing Rule, Table-Based



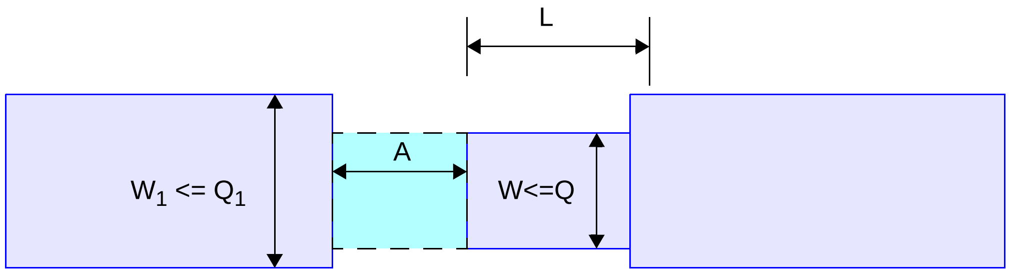
```
endOfLine1NeighborTblSize = N
endOfLine1NeighborWireThresholdTbl = (Q1, Q2, ... QN)
endOfLine1NeighborMinSpacingTbl = (A1, A2, ... AN)
endOfLine1NeighborCornerKeepoutWidthTbl = (E1, ... EN)
```

For example,

```
Layer "MetalX" {
  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 2-112.

Figure 2-112 Modified One-Neighbor End-of-Line Spacing Rule, Variable Width

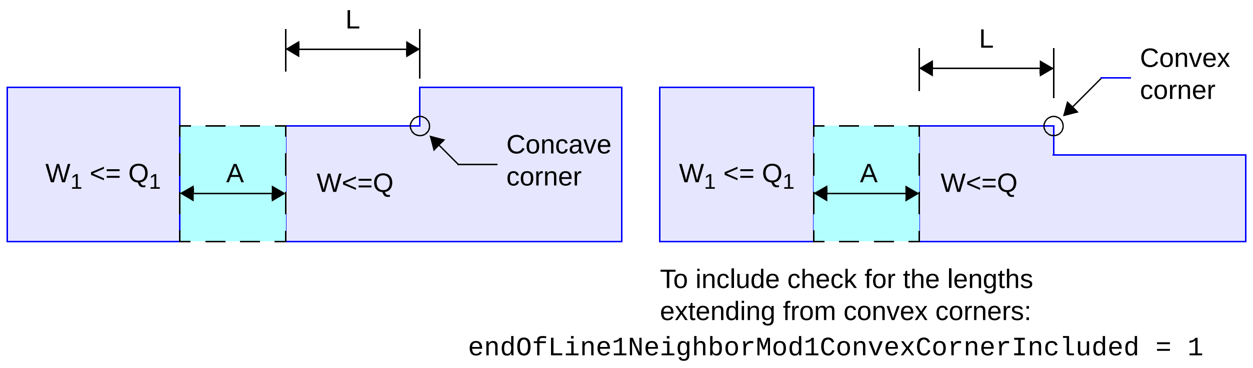


```
endOfLine1NeighborMod1MaxThreshold = Q
endOfLine1NeighborMod1MaxThreshold2 = Q1
endOfLine1NeighborMod1MaxLength = L
endOfLine1NeighborMod1MinSpacing = A
```

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 2-113.

Figure 2-113 Modified One-Neighbor End-of-Line Spacing Rule, Variable Width


L
W_1 <= Q_1
A
W<=Q
Concave corner
L
Convex corner
W_1 <= Q_1
A
W<=Q
To include check for the lengths extending from convex corners:
endOfLine1NeighborMod1ConvexCornerIncluded = 1

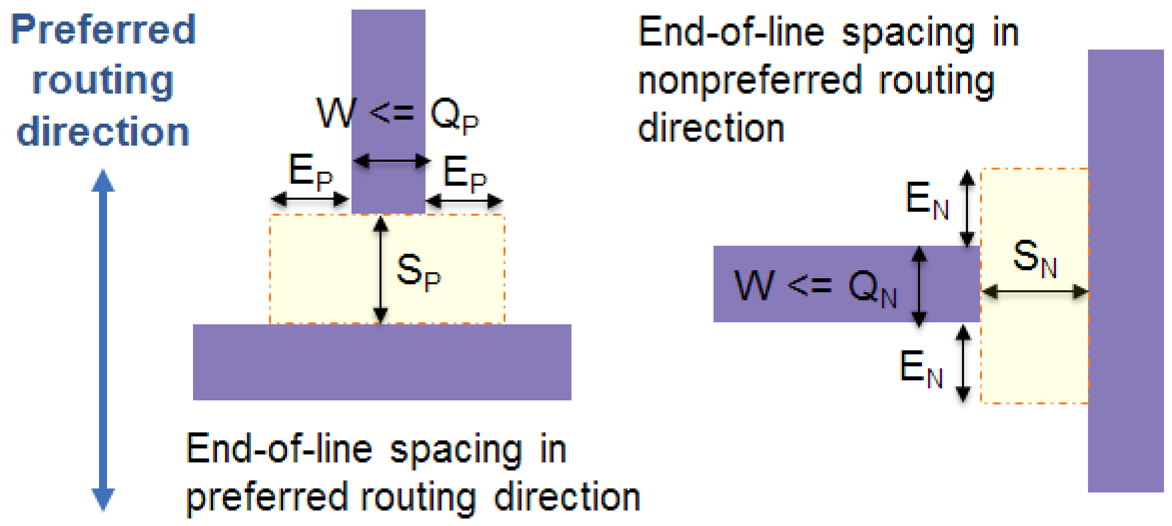
To have the rule apply to lengths extending from convex as well as concave corners, add a statement that sets `endOfLine1NeighborMod1ConvexCornerIncluded` to 1. For example,

```
Layer "MetalX" {
  endOfLine1NeighborMod1MaxThreshold = 0.288
  endOfLine1NeighborMod1MaxThreshold2 = 0.290
  endOfLine1NeighborMod1MaxLength = 0.380
  endOfLine1NeighborMod1MinSpacing = 0.140
  endOfLine1NeighborMod1ConvexCornerIncluded = 1
}
```

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 2-114, the subscript “P” denotes an attribute for a wire running in the preferred direction, and similarly the subscript “N” for the nonpreferred direction.

Figure 2-114 Preferred and Nonpreferred End-of-Line Spacing Rule


Preferred routing direction
W <= QP
EP
EP
SP
End-of-line spacing in preferred routing direction
End-of-line spacing in nonpreferred routing direction
EN
SN
W <= QN
EN

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)
}
```

Two-Neighbor End-of-Line Spacing Rule

Note:

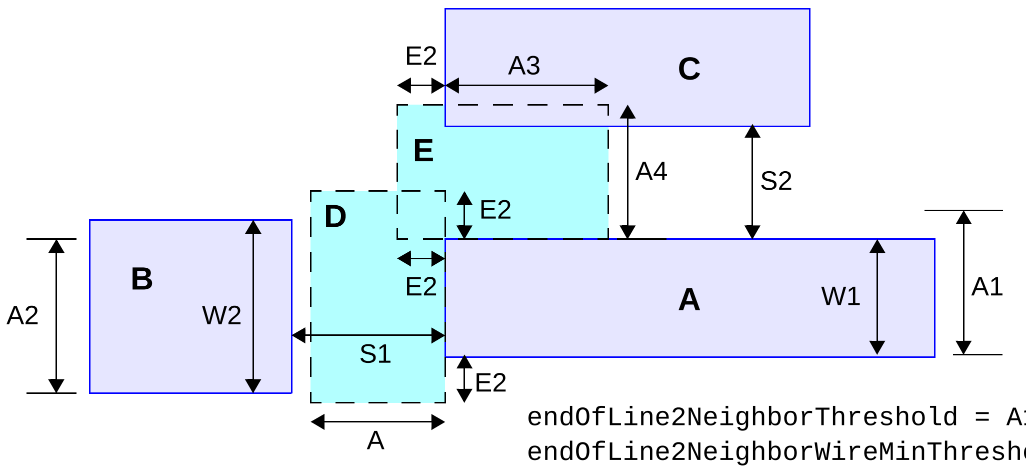
This rule is supported only by Zroute. For a similar rule supported by the classic router, see Stub Mode 4: Two-Edge Spacing With Connected Metal Exclusion.

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 Figure 2-115, if the metal width W1 is less than or equal to the width threshold A1 and the metal width W2 is at least the width threshold A2, at least one of the following conditions must be true:

- The distance between the line end and the first other metal S1 is at least the minimum spacing value A2.
- The distance between the side of the narrow metal line and the second other metal S2 is at least the minimum side spacing value A4.

Figure 2-115 Two-Neighbor End-of-Line Spacing Rule



```
endOfLine2NeighborThreshold = A1
endOfLine2NeighborWireMinThreshold = A2
endOfLine2NeighborMinSpacing = A
endOfLine2NeighborSideMinSpacing = A4
endOfLine2NeighborCornerKeepoutWidth = E2
endOfLine2NeighborSideKeepoutLength = A3
```

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, the end-of-line minimum spacing value A is used for both the end-of-line and side spacing checks.

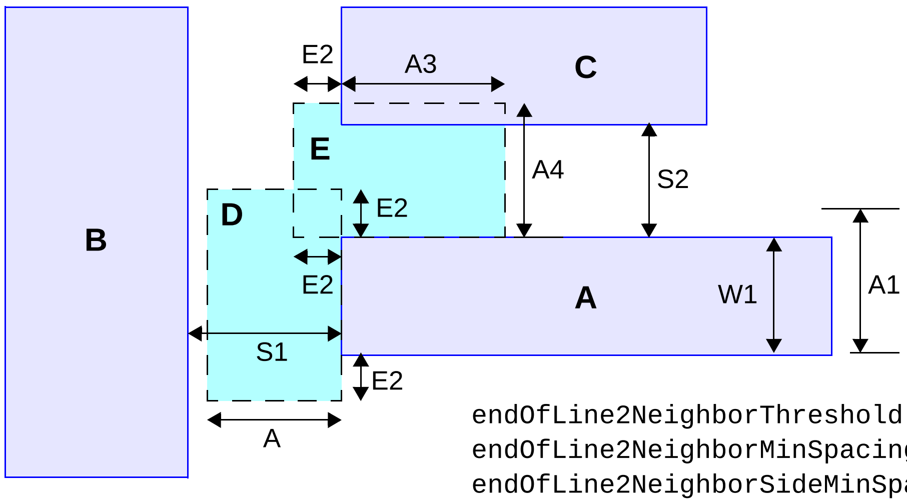
Here is an example of the syntax used for this rule:

```
Layer "MetalX" {
  endOfLine2NeighborThreshold = 0.070
  endOfLine2NeighborWireMinThreshold = 0.072
  endOfLine2NeighborMinSpacing = 0.075
  endOfLine2NeighborSideMinSpacing = 0.080
  endOfLine2NeighborCornerKeepoutWidth = 0.030
  endOfLine2NeighborSideKeepoutLength = 0.080
}
```

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 2-116, if the metal width W1 is less than or equal to the width threshold A1, 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 2-116 Two-Neighbor End-of-Line Spacing Rule



```
endOfLine2NeighborThreshold = A1
endOfLine2NeighborMinSpacing = A2
endOfLine2NeighborSideMinSpacing = A4
endOfLine2NeighborCornerKeepoutWidth = E2
endOfLine2NeighborSideKeepoutLength = A3
```

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.

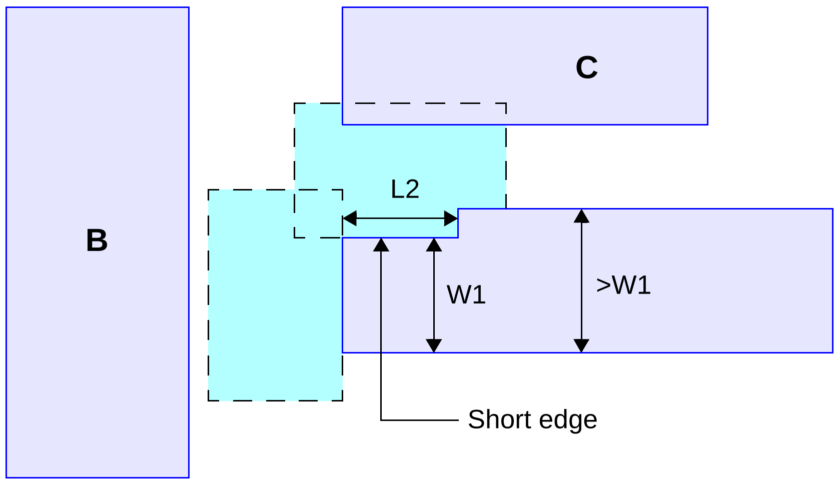
Here is an example of the syntax used for this rule:

```
Layer "MetalX" {
  endOfLine2NeighborThreshold = 0.070
  endOfLine2NeighborMinSpacing = 0.075
  endOfLine2NeighborSideMinSpacing = 0.080
  endOfLine2NeighborCornerKeepoutWidth = 0.030
  endOfLine2NeighborSideKeepoutLength = 0.080
}
```

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 2-117.

Figure 2-117 Two-Neighbor End-of-Line With Adjacent Short Edge



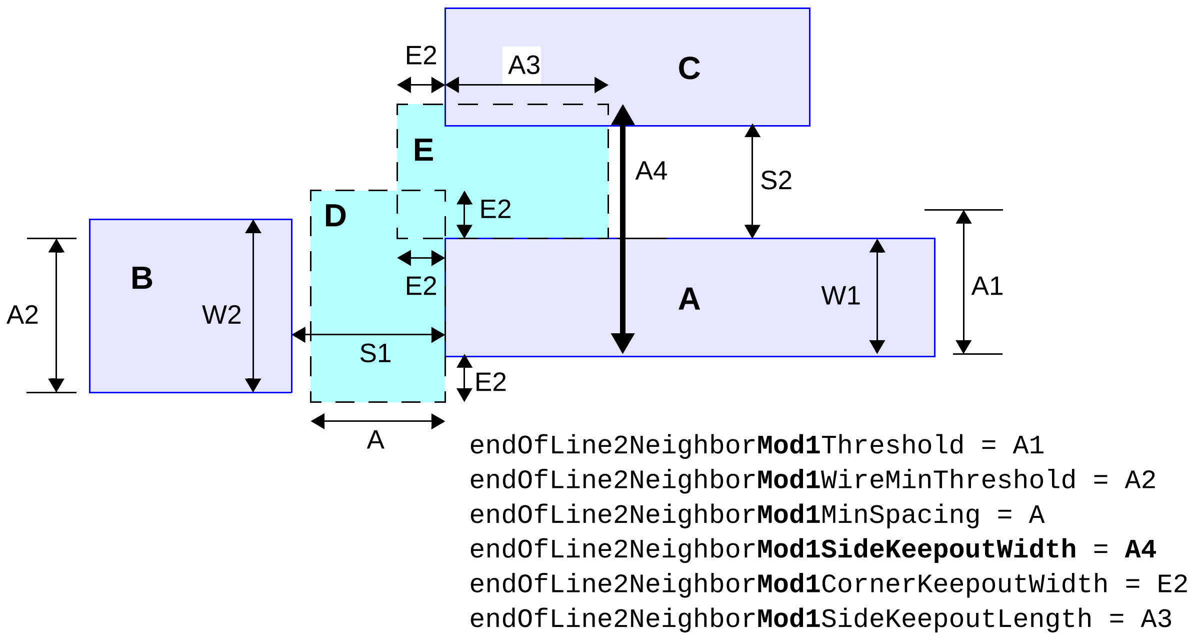
By default, the adjacent edge L2 is considered short if it is less than or equal to the `minWidth` value defined for the metal. To define a different L2 threshold for the two-neighbor end-of-line spacing rule, use the following attribute:

```
endOfLine2NeighborMinLength = L2
```

If the adjacent edge is less than or equal to the specified value, the rule does not apply.

Figure 2-118 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 2-118 Modified Two-Neighbor End-of-Line Spacing Rule

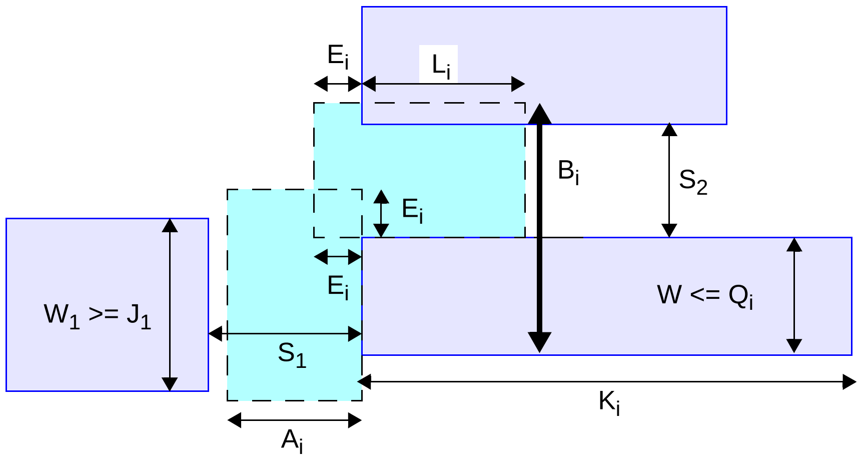

E2
A3
C
E
A4
S2
D
E2
B
E2
A
W1
A1
A2
W2
S1
E2
A
endOfLine2NeighborMod1Threshold = A1
endOfLine2NeighborMod1WireMinThreshold = A2
endOfLine2NeighborMod1MinSpacing = A
endOfLine2NeighborMod1SideKeepoutWidth = A4
endOfLine2NeighborMod1CornerKeepoutWidth = E2
endOfLine2NeighborMod1SideKeepoutLength = A3

Here is an example of the modified syntax used for this rule:

```
Layer "MetalX" {
  endOfLine2NeighborMod1Threshold = 0.070
  endOfLine2NeighborMod1WireMinThreshold = 0.072
  endOfLine2NeighborMod1MinSpacing = 0.075
  endOfLine2NeighborMod1SideKeepoutWidth = 0.150
  endOfLine2NeighborMod1CornerKeepoutWidth = 0.030
  endOfLine2NeighborMod1SideKeepoutLength = 0.075
}
```

For advanced geometries, the rule is enhanced to allow table-based specification of the rule attributes. Figure 2-118 shows the rule measurements and following the figure is the enhanced syntax for the rule.

Figure 2-119 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 $W <= Q_i$ and $W_1 >= J_i$, 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:

$S_1 >= A_i$

$(S_2 + W) >= B_i$

Here is an example of the rule:

```
Layer "Mx" {
  endOfLine2NeighborMod1TblSize                   = 2
  endOfLine2NeighborMod1ThresholdTbl              = (0.049, 0.062)
  endOfLine2NeighborMod1MinSpacingTbl             = (0.080, 0.068)
  endOfLine2NeighborMod1CornerKeepoutWidthTbl     = (0.026, 0.026)
  endOfLine2NeighborMod1SideKeepoutLengthTbl      = (0.063, 0.063)
  endOfLine2NeighborMod1SideKeepoutWidthTbl       = (0.106, 0.106)
}
```

Three-Neighbor End-of-Line Spacing Rule

Note:

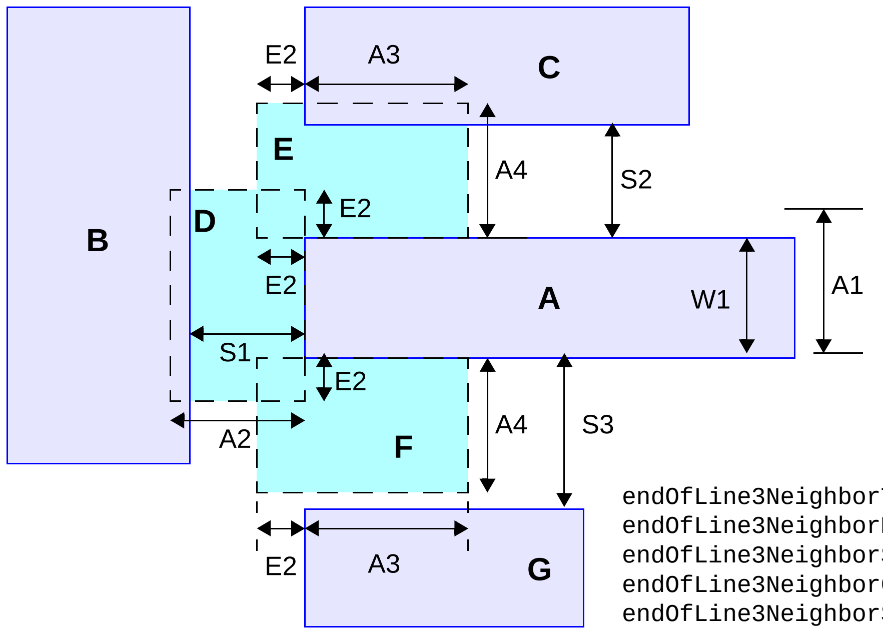
This rule is supported only by Zroute. For a similar rule supported by the classic router, see Stub Mode 3: Three-Edge Spacing at Metal End.

The three-neighbor end-of-line spacing rule specifies the minimum distance between the end of a narrow metal shape and a neighboring metal shape (end-of-line spacing check) and from the two sides of the same narrow metal shape to two other neighboring metal shapes (side spacing check). In Figure 2-120, if the metal width W1 is less than or equal to the width threshold A1, the distance S1 between the line end and the metal shape must be at least the minimum spacing value A2. If not, the distance S2 or S3 between at least one of the two sides of the narrow metal shape and the neighboring metals must be at least the minimum side spacing value A4.

By default, `endOfLine3NeighborMinSpacing` applies between a line-end and all of its neighbors. To apply this check only between a line-end and its non-line-end neighbors, set the `endOfLine3NeighborMinSpacingForLineSideOnly` attribute to 1.

If `endOfLine3NeighborSideMinSpacing` is omitted, `endOfLine3NeighborMinSpacing` applies to both the end-of-line spacing check and the side spacing check.

Figure 2-120 Three-Neighbor End-of-Line Spacing Rule



```
endOfLine3NeighborThreshold = A1
endOfLine3NeighborMinSpacing = A2
endOfLine3NeighborSideMinSpacing = A4
endOfLine3NeighborCornerKeepoutWidth = E2
endOfLine3NeighborSideKeepoutLength = A3
```

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 "MetalX" {
  endOfLine3NeighborThreshold = 0.288
  endOfLine3NeighborMinSpacing = 0.14
  endOfLine3NeighborSideMinSpacing = 0.16
  endOfLine3NeighborCornerKeepoutWidth = 0.02
  endOfLine3NeighborSideKeepoutLength = 0.27
}
```

In this example, for layer M2, when a metal shape having a width of less than 0.288 has neighboring metal shapes 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 0.27 from the corner along the sides.

Three-Neighbor End-of-Line Cap Rule

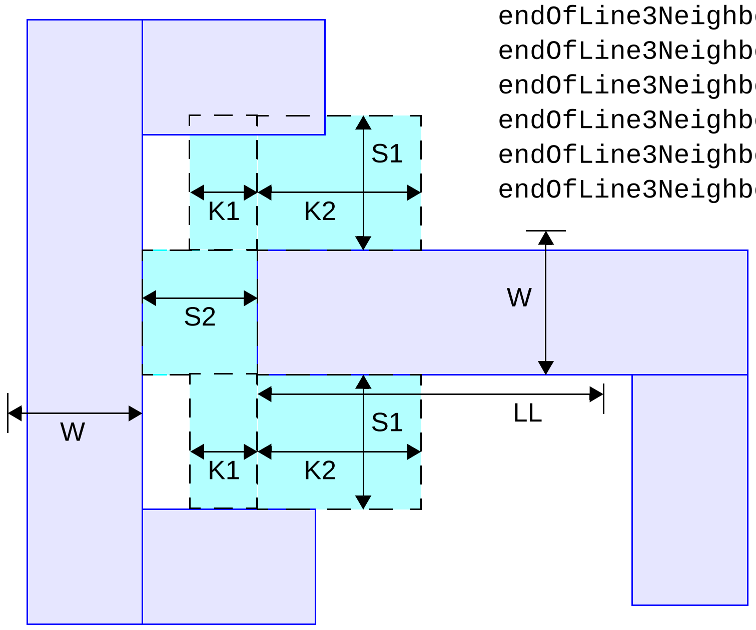
Note:

This rule is supported only by Zroute.

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 2-121, 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 2-121 Three-Neighbor End-of-Line Cap Rule

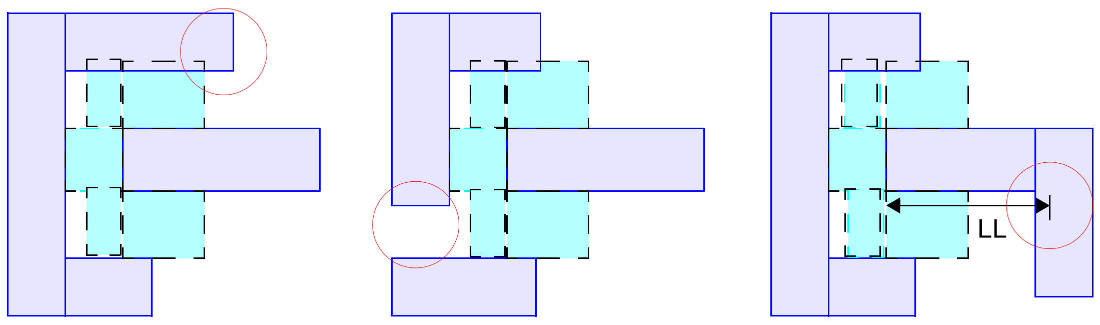
```
endOfLine3NeighborThreshold          = W
endOfLine3NeighborMinSpacing         = S2
endOfLine3NeighborCornerKeepoutWidth = K1
endOfLine3NeighborSideMinSpacing     = S1
endOfLine3NeighborSideKeepoutLength  = K2
endOfLine3NeighborMinLength          = LL
endOfLine3NeighborWireConnected      = 1
```



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, the spacing from the line-end to the U-shape must be at least S2.

Figure 2-122 shows some examples of configurations that resemble a U-shaped cap, but the rule does not apply.

Figure 2-122 Three-Neighbor End-of-Line Cap Rule Exclusions



U-shape end extends beyond error region

U-shape is not continuous

Length of metal facing U-shape is less than LL

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 "MetalX" {
  endOfLine3NeighborThreshold           = 0.061
  endOfLine3NeighborMinSpacing          = 0.127
  endOfLine3NeighborCornerKeepoutWidth = 0.000
  endOfLine3NeighborSideMinSpacing      = 0.068
  endOfLine3NeighborSideKeepoutLength  = 0.136
  endOfLine3NeighborMinLength           = 0.168
  endOfLine3NeighborWireConnected       = 1
}
```

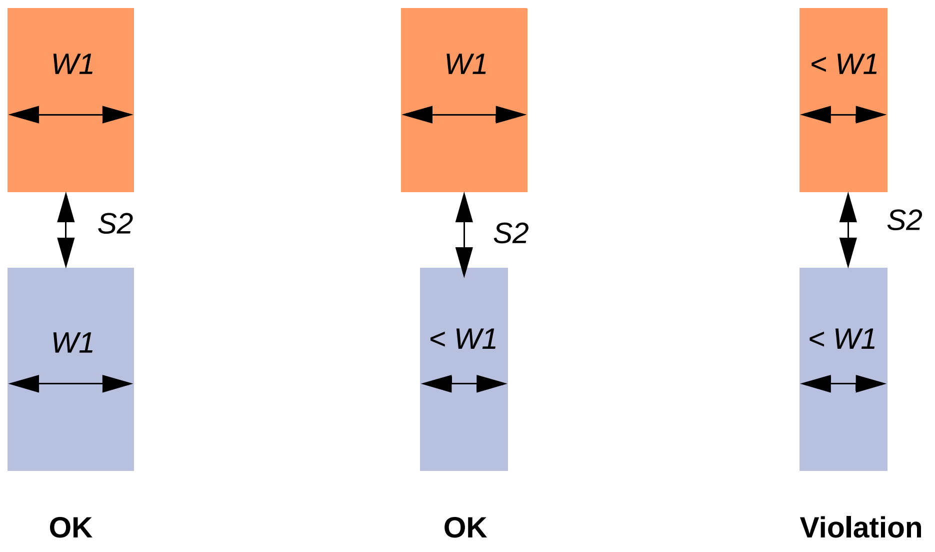
Single-Side Isolated Neighbor End-of-Line Spacing Rule

Note:
This rule is supported only by Zroute.

The single-side isolated neighbor end-of-line spacing rule allows a smaller line-end spacing between two metal shapes in the preferred routing direction. This rule applies to same or different metal colors and is a variation of One-Neighbor End-of-Line Spacing Rule. If the two metal shapes are different colors and the width of one metal shape is W1 and the width of the second metal shape is less than W1, then the smaller spacing rule applies.

Figure 2-123 shows some examples. In the third example, it is a violation because both metal shape widths are less than W1.

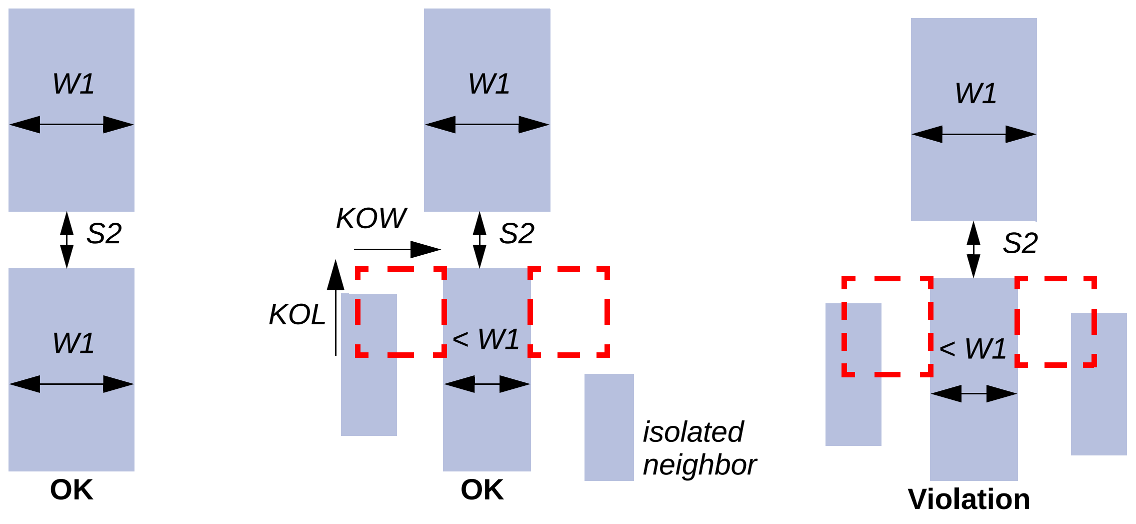
Figure 2-123 Different Color Neighbor Line-End Spacing Rule



In Figure 2-124, if the two metal shapes are the same color, then you can have a smaller line-end spacing if at least one of the following conditions are met:

- The widths of the two shapes are the same (= W1)
- The width of one shape is at a specific value (W1) and the width of the other shape is less than that value (< W1 when the smaller-width shape has an isolated neighbor on one side

Figure 2-124 Same-Color Single-Side Isolated Neighbor End-of-Line Spacing



Define this rule in the `Metal Layer` section of the technology file:

```
Layer MetalX {
   # ordinary line-end spacing
   endOfLine1NeighborThreshold           = W1
   endOfLine1NeighborCornerKeepoutWidth = PRL
   endOfLineNeighborMinSpacing           = S1
   # spacing exception
   endOfLine1NeighborSameColorPrefSparseKeepoutWidth = KOW
   endOfLine1NeighborSameColorPrefSparseKeepoutLength = KOL
   endOfLine1NeighborSameColorPrefMinSpacing = S2 # S2 <= S1
   endOfLine1NeighborDiffColorPrefMinSpacing = S2
}
```

The following is an example of a technology file entry:

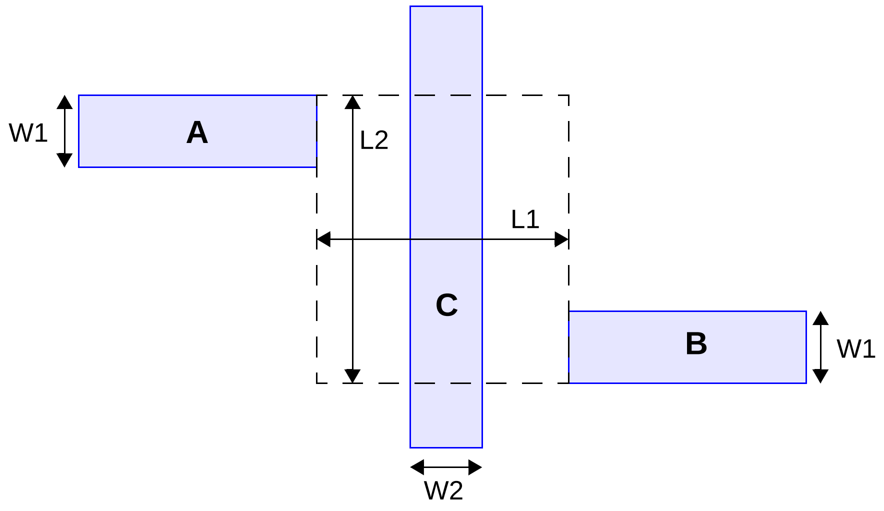
```
Layer "Metal3" {
...
   endOfLine1NeighborSameColorPrefSparseKeepoutWidth  = 0.015
   endOfLine1NeighborSameColorPrefSparseKeepoutLength = 0.055
   endOfLine1NeighborSameColorPrefMinSpacing          = 0.205
   endOfLine1NeighborDiffColorPrefMinSpacing          = 0.205
...
}
```

Two-Sided End-of-Line Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-125, 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 2-125 Two-Sided End-of-Line Spacing Rule



To invoke this rule, use the following syntax in the `Layer` section:

```
tJunctionStubWireMaxThreshold   = W1
tJunctionOrthoWireMaxThreshold  = W2
tJunctionStubKeepoutMinSpacing  = L1
tJunctionStubKeepoutMinWidth    = L2
```

For example,

```
tJunctionStubWireMaxThreshold   = 0.20
tJunctionOrthoWireMaxThreshold  = 0.22
tJunctionStubKeepoutMinSpacing  = 0.80
tJunctionStubKeepoutMinWidth    = 0.70
```

Figure 2-126 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 2-126 Two-Sided End-of-Line Spacing Violation Examples

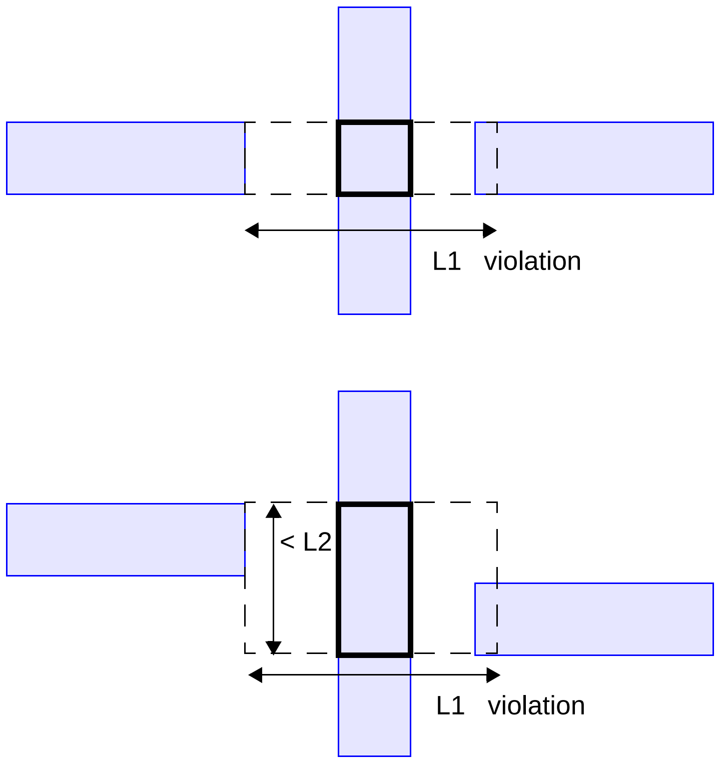
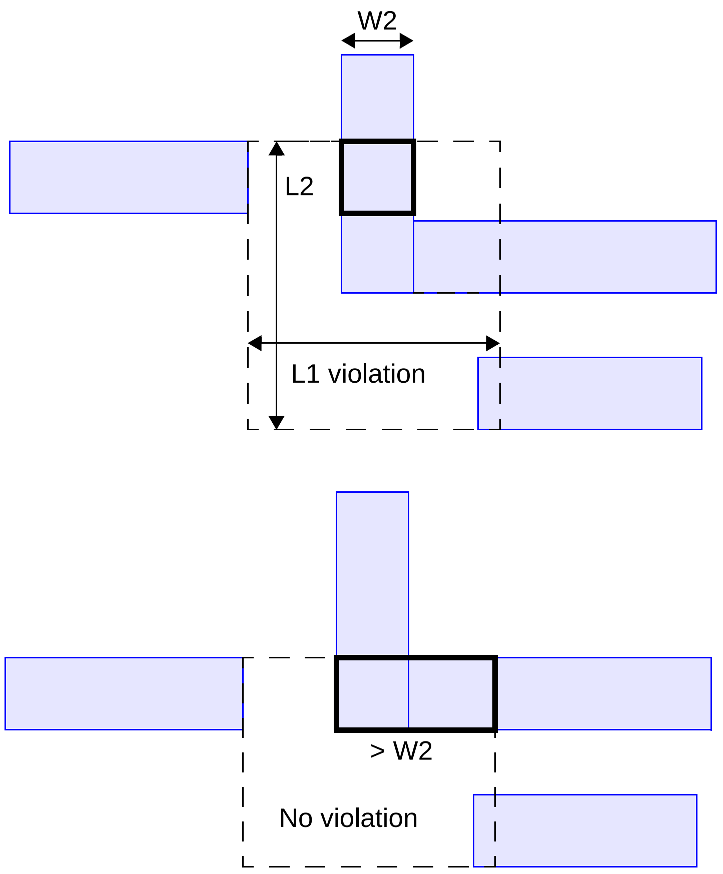


Figure 2-127 shows two examples of end-of-line wires approaching an L-shaped wire. The black rectangle represents the overlap of the two approaching wires with the L-shaped wire. The first example is a violation. However, the second example is not a violation because the overlap applies to the long section 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 2-127 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 2-128 shows two wire ends approaching opposite sides of a horizontal wire and two joints located on opposite sides of the same horizontal wire. Figure 2-129 shows examples of a wire end and a joint located on opposite sides of the same horizontal wire.

Figure 2-128 Two Wire Ends and Two Wire Joints on Opposite Sides of a Wire

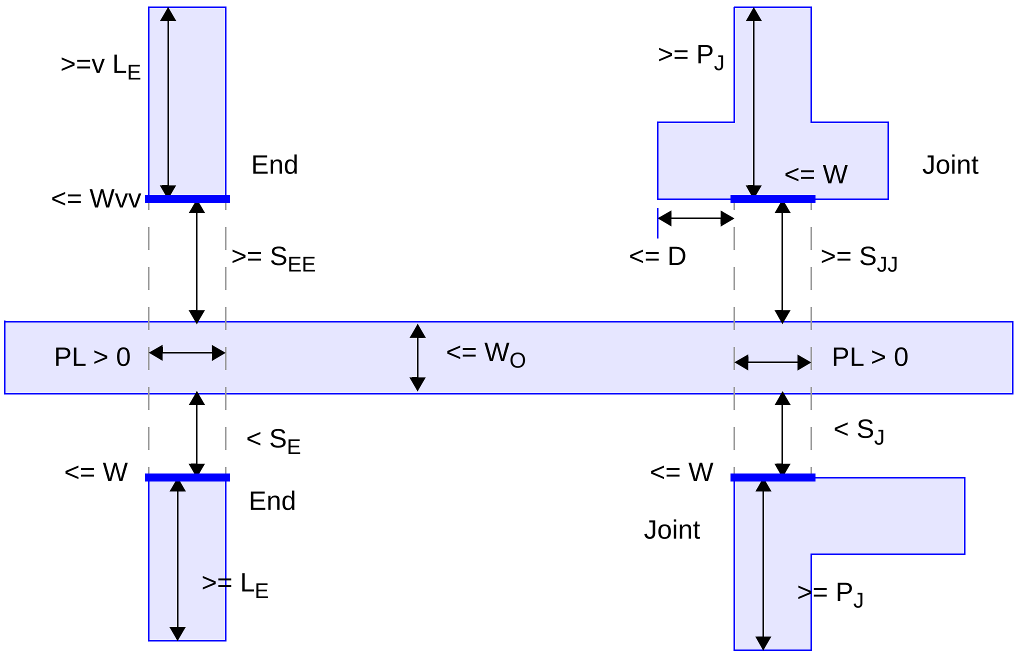
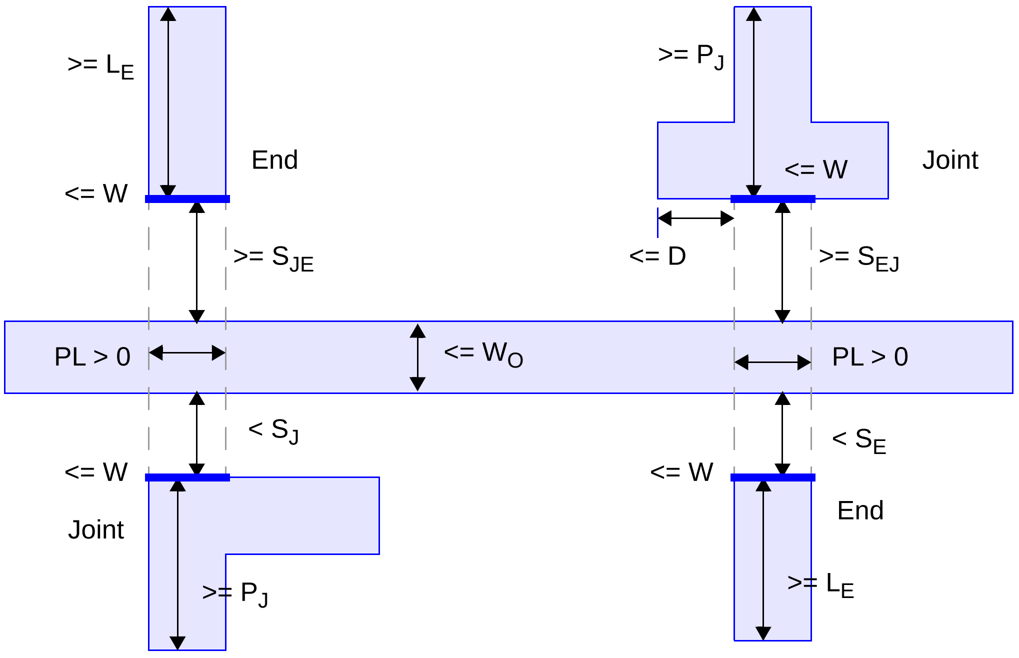


Figure 2-129 A Wire End and a Joint on Opposite Sides of a Wire



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.

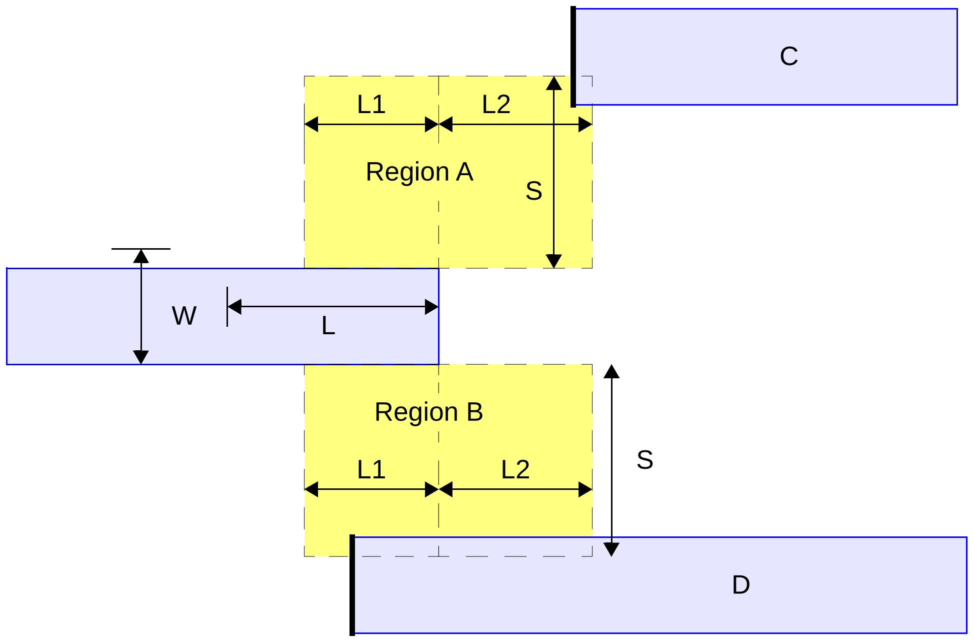
The rule is defined in the metal `Layer` section as follows:

```
Layer "MetalX" {
  endJointOrthoWireMaxThreshold          = $W_O$
  endWireSideEdgeLengthThreshold         = $L_E$
  jointWireSpanThreshold                 = $P_J$
  jointToConvexCornerMaxDist             = D
  endJointWireThresholdTblSize           = N
  endJointWireWidthMaxThresholdTbl       = ($W_1$,$W_2$,...)
  endJointWireParallelWidthThresholdTbl = ($PL_1$,$PL_2$,...)
  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 2-130, 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 2-130 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:

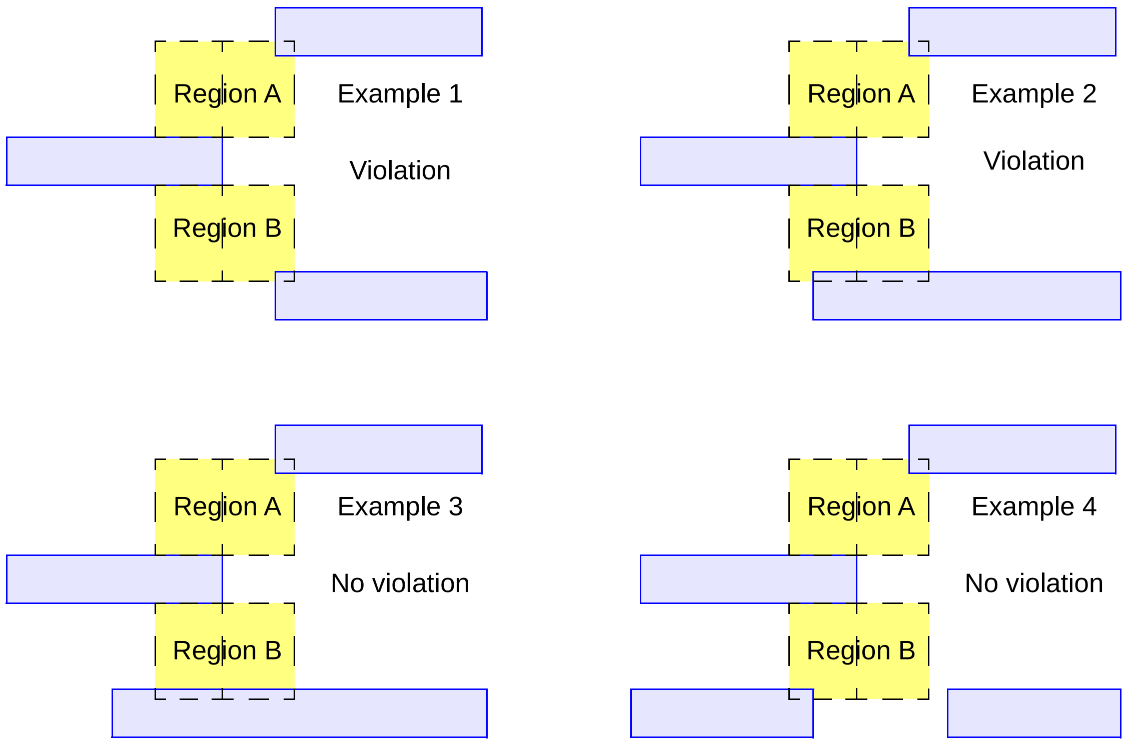
```
Layer "Mx" {
  endOfLine2CornerThreshold         = W
  endOfLine2CornerMinLength         = L
  endOfLine2CornerKeepoutWidth      = S (optional)
  endOfLine2CornerKeepoutLength     = L2
  endOfLine2CornerSideKeepoutLength = L1
}
```

For example,

```
Layer "Mx" {
  endOfLine2CornerThreshold         = 0.22
  endOfLine2CornerKeepoutWidth      = 0.43
  endOfLine2CornerKeepoutLength     = 0.31
  endOfLine2CornerSideKeepoutLength = 0.26
}
```

The examples in Figure 2-131 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 2-131 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.

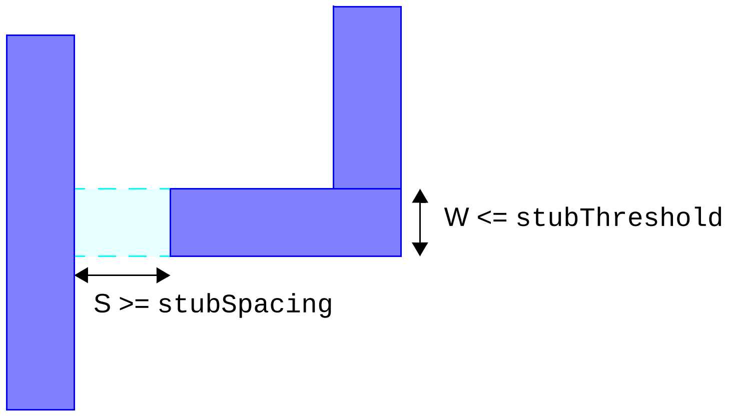
Note:

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.

You cannot use both old and new syntax for end-of-line spacing rules in the same technology file.

Figure 2-132 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 2-132 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.

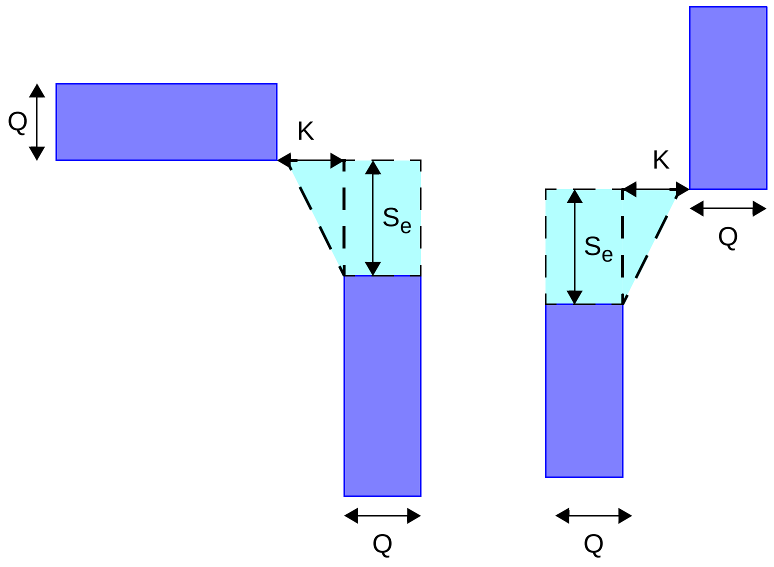
For example,

```
Technology {
  stubMode = 1
}

Layer "MetalX"  {
  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 2-133, the upper blue metal line must stay outside of the lightly shaded regions between it and the lower-metal line.

Figure 2-133 Stub Mode 1



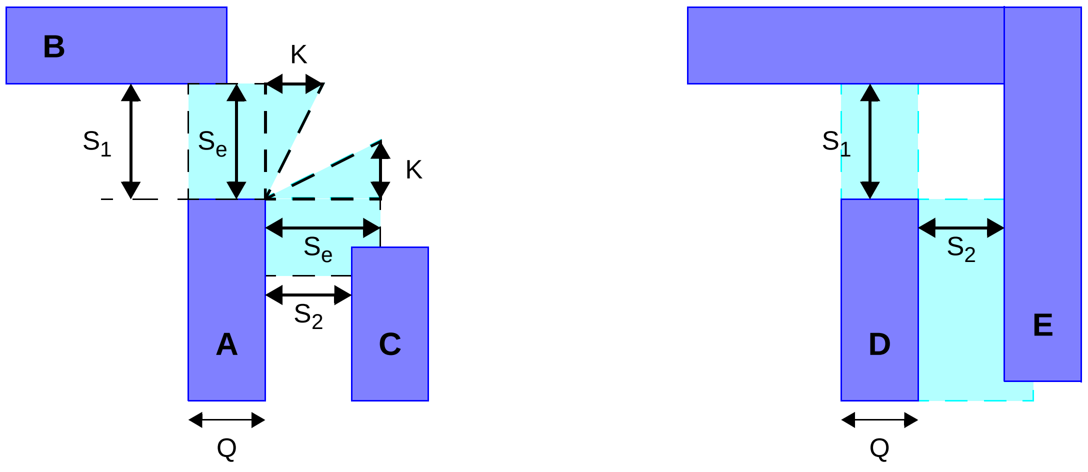
This is the syntax for specifying the end-of-line rule in stub mode 1:

```
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, at least one of the two spacing values must be greater than the specified stub spacing value. Two examples are illustrated in Figure 2-134.

Figure 2-134 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.

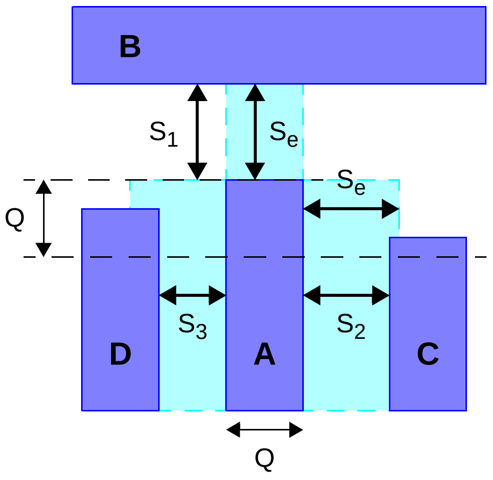
This is the syntax for specifying the end-of-line rule in stub mode 2:

```
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, at least one of the three spacing values must be greater than the specified stub spacing value. An example is shown in Figure 2-135.

Figure 2-135 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:

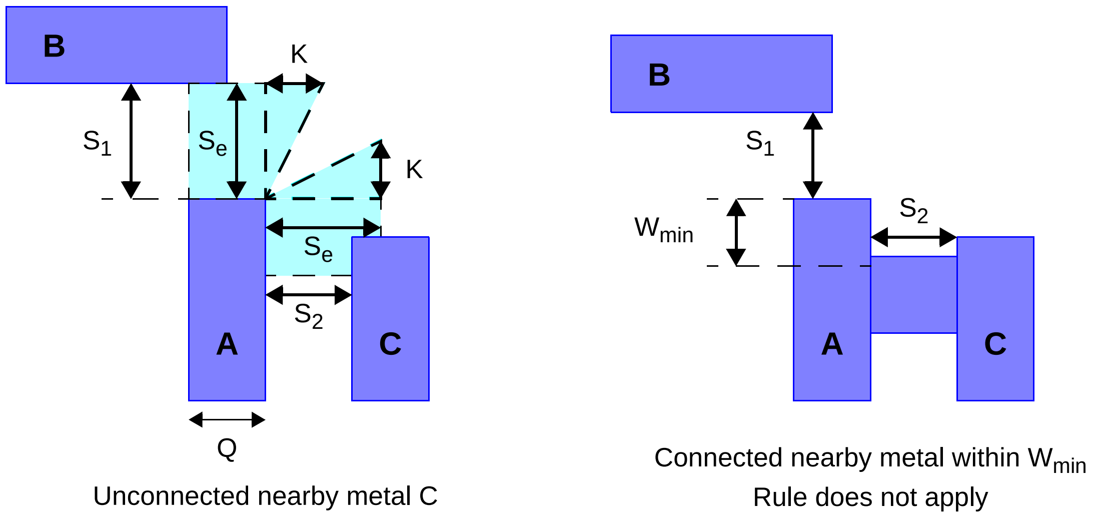
```
Technology {
  stubMode = 3
  }

Layer "Metal1" {
  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 2-136, 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}, the rule still applies and stub mode 4 still works like stub mode 2.

Figure 2-136 Stub Mode 4


B
K
S1
Se
K
Se
S2
A
C
Q
Unconnected nearby metal C
B
S1
Wmin
S2
A
C
Connected nearby metal within Wmin
Rule does not apply

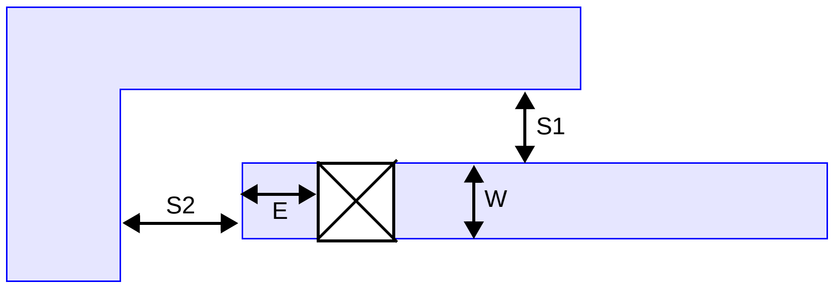
This is the syntax for specifying the end-of-line rule in stub mode 4:

```
Technology {
  stubMode = 4
  }

Layer "Metal1" {
  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 2-137 shows the attributes associated with this rule.

Figure 2-137 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.

In the following example, the S1 spacing requirement is 0.12.

```
DesignRule {
   layer1 = "MetalX"
   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)
}
```

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.

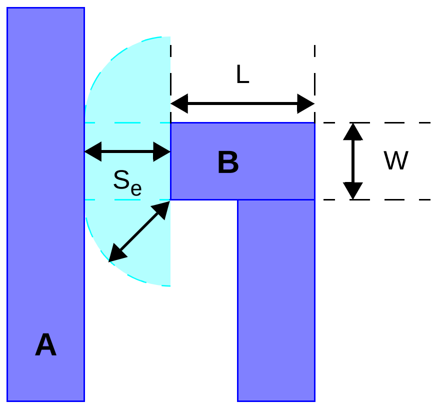
The L-shape rule is applied to the end-of-line segment edge as well as to the diagonal corner spacing, provided that the L-shape geometry and its length meet the given length threshold.

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 2-138, 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, the minimum spacing is S_e (`minSpacing`); otherwise, it is the default metal-to-metal minimum spacing.

Figure 2-138 L-Shaped End-of-Line Spacing Rule

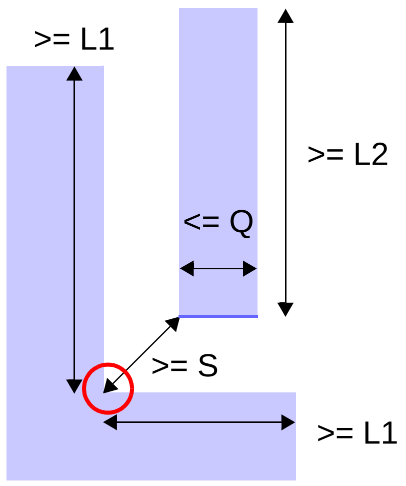


This rule is typically set to be less restrictive than the ordinary end-of-line minimum spacing rule, thus conserving routing resources for the L-shaped case versus the normal case.

End-of-Line to Concave Corner Spacing Rule

The end-of-line 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 2-139.

Figure 2-139 Line-End to Concave Corner Direct Spacing Rule

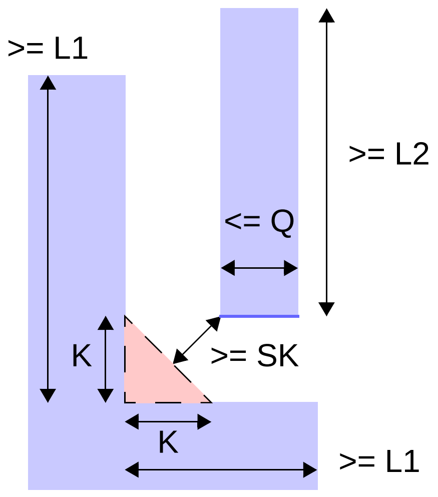


The following example shows the syntax of the rule:

```
Layer "MetalX" {
  endOfLineToConcaveCornerMaxWidthThreshold   = 0.12 # Q
  endOfLineToConcaveCornerEdgeLengthThreshold = 0.54 # L1 (optional)
  endOfLineToConcaveCornerMinLength           = 0.05 # L2 (optional)
  endOfLineToConcaveCornerMinSpacing          = 0.07 # S
}
```

In an alternative form of the rule, the spacing is measured from a triangular keepout region defined by the concave corner and a specified keepout length, as shown in Figure 2-140.

Figure 2-140 Line-End to Concave Corner Keepout Spacing Rule



The following example shows the syntax of corner spacing rule using the triangular keepout region:

```
Layer "MetalX" {
  endOfLineToConcaveCornerMaxWidthThreshold   = 0.12  # Q
  endOfLineToConcaveCornerEdgeLengthThreshold = 0.54  # L1 (optional)
  endOfLineToConcaveCornerMinLength           = 0.05  # L2 (optional)
  sideToConcaveCornerKeepoutLength            = 0.06  # K
  sideToConcaveCornerKeepoutMinSpacing        = 0.04  # SK
}
```

Double-Patterning Metal Spacing Rules

The double-patterning metal 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 a 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 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.

The following topics describe 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 Fat-to-Thin Metal Spacing Rule
- Double-Patterning Fat-to-Fat Metal Spacing Rule

- Color-Based Span Spacing Rule
- Color-Based Line-End Alignment Rule
- Color-Based Metal Corner Keepout Rule
- Minimum Metal to Cut-Metal Keepout and Overlap Rule

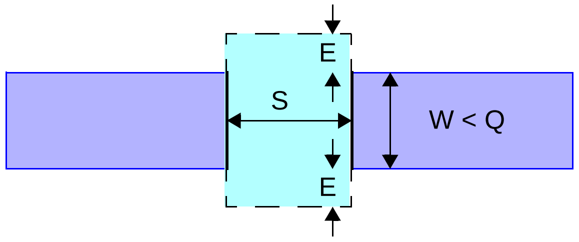
The double-patterning spacing rules are supported only by Zroute.

Double-Patterning Line-End to Line-End Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-141.

Figure 2-141 Double-Patterning Line-End to Line-End Spacing Rule



```
Layer "MetalX" {
   doublePatternEndMaxWidthThreshold  = Q
   doublePatternEndToEndKeepoutLength = 2*E – 0.001
   doublePatternEndToEndMinSpacing    = A
}
```

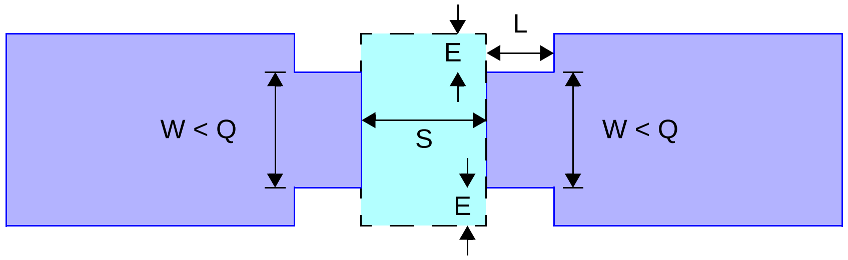
For example,

```
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*E-0.001, where E (keepout attribute) is 0.025. Thus the attribute is specified as 2*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 2-142.

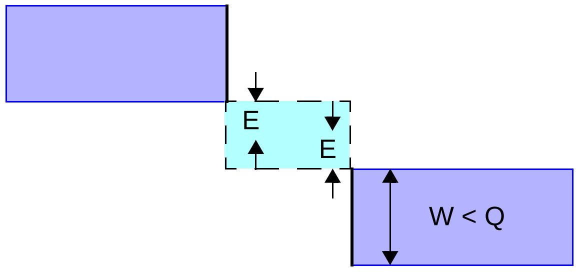
Figure 2-142 Double-Patterning Line-End to Line-End Spacing Rule With Run Length



```
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 2-143, the check still applies as long as the magnitude of the negative run is less than 0.049.

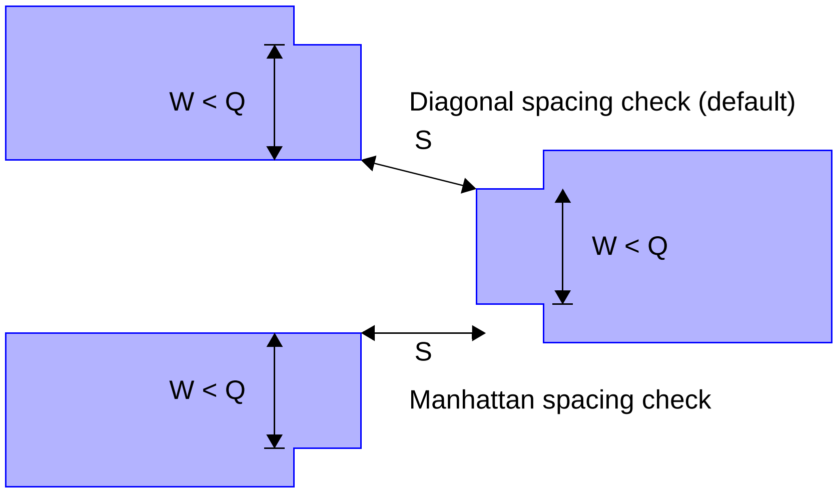
Figure 2-143 Line-End to Line-End Spacing Rule With Negative Parallel Run



With a negative parallel run, the check from corner to corner is performed diagonally by default, as shown in the top portion of Figure 2-144. To perform the check using Manhattan measurement instead, as shown in the bottom portion of the figure, add the following statement:

```
doublePatternCheckManhattanSpacing = 1
```

Figure 2-144 Diagonal and Manhattan Line-End to Line-End Spacing Checks



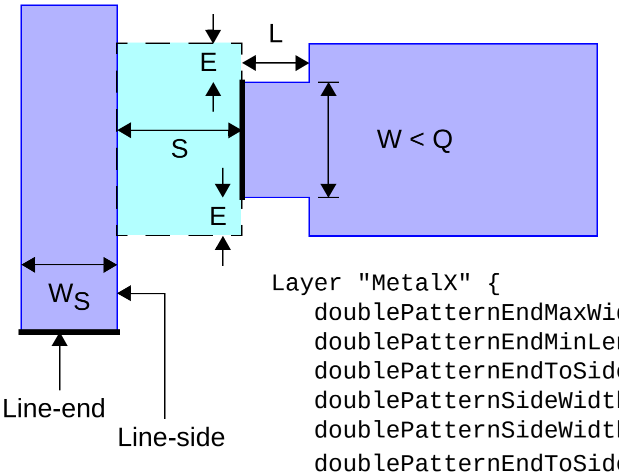
Double-Patterning Line-End to Line-Side Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-145.

Figure 2-145 Double-Patterning Line-End to Line-Side Spacing Rule



```
Layer "MetalX" {
   doublePatternEndMaxWidthThreshold      = Q
   doublePatternEndMinLength              = L
   doublePatternEndToSideKeepoutLength    = E
   doublePatternSideWidthThresholdTblSize = 2
   doublePatternSideWidthThresholdTbl     = (0, W_s)
   doublePatternEndToSideMinSpacingTbl    = (A_11, A_12)
}
```

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)
}
```

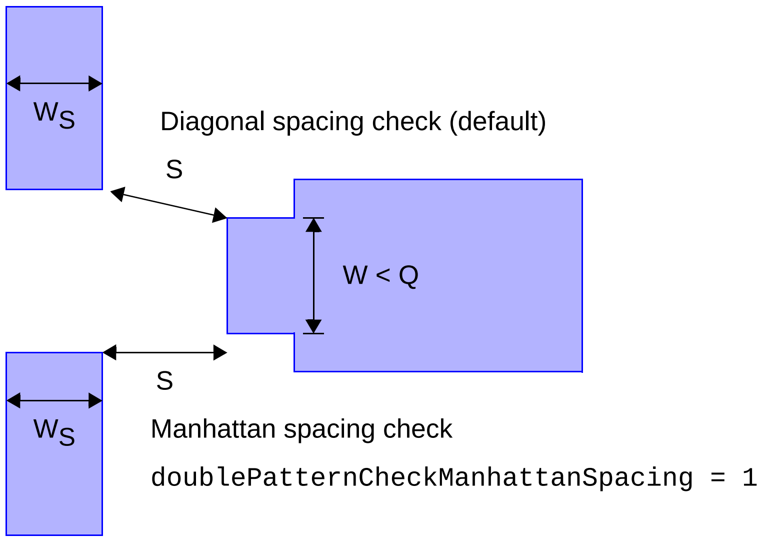
In this example, when the width at the line-end is less than 0.10 and extends at that width for at least 0.04 away from the line-end, and the width of the nearby line is at least 0.00 but less than 0.05, the minimum spacing is 0.10. If the width of the nearby line is at least 0.05, the minimum spacing 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 2-146.

Figure 2-146 Diagonal and Manhattan Line-End to Line-Side Spacing Checks


W_S
Diagonal spacing check (default)
S
W < Q
S
W_S
Manhattan spacing check
doublePatternCheckManhattanSpacing = 1

To perform the check using Manhattan measurement, as shown in the bottom portion of the figure, add the following statement:

```
doublePatternCheckManhattanSpacing = 1
```

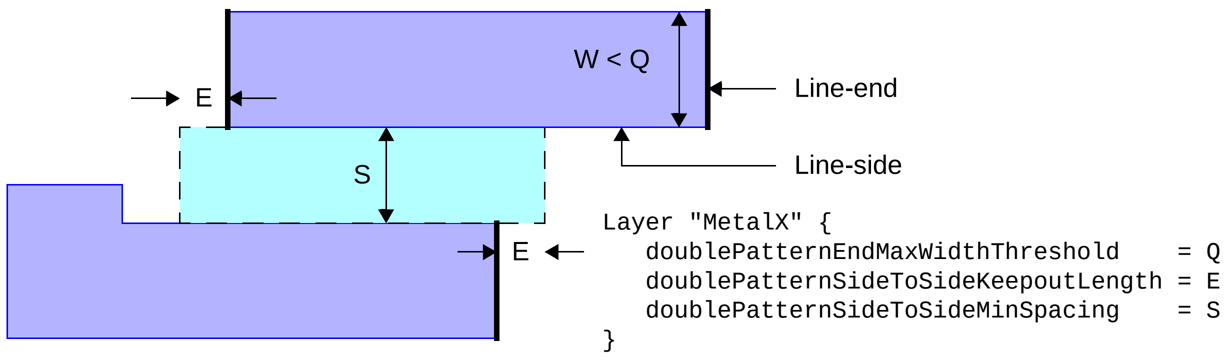
Double-Patterning Line-Side to Line-Side Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-147.

Figure 2-147 Double-Patterning Line-Side to Line-Side Spacing Rule



```
Layer "MetalX" {
   doublePatternEndMaxWidthThreshold    = Q
   doublePatternSideToSideKeepoutLength = E
   doublePatternSideToSideMinSpacing    = S
}
```

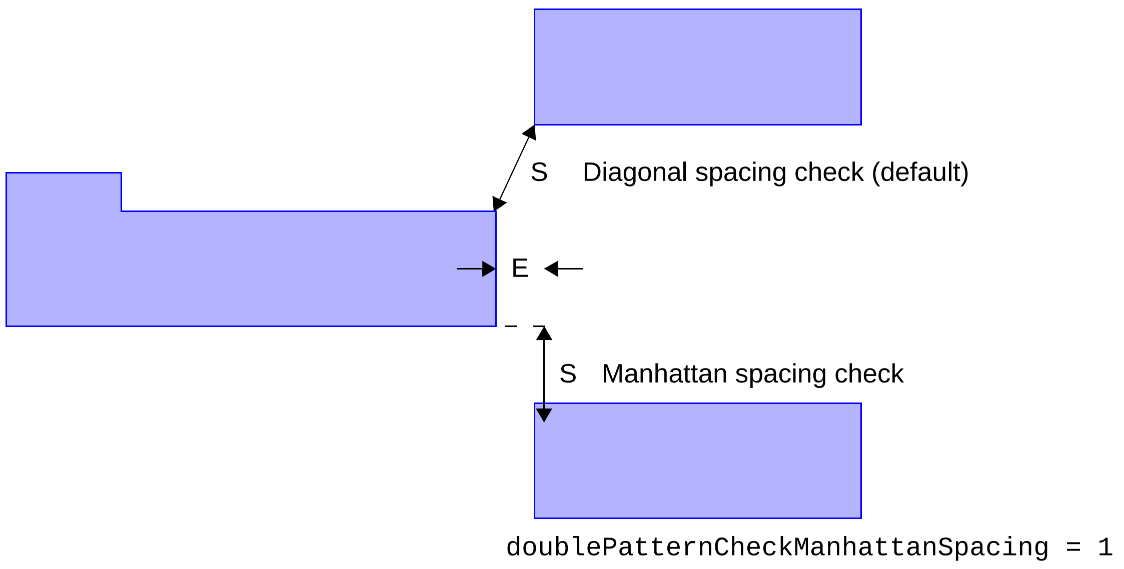
For example,

```
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 2-148.

Figure 2-148 Diagonal and Manhattan Line-Side to Line-Side Spacing Checks



```
doublePatternCheckManhattanSpacing = 1
```

To perform the check using Manhattan measurement instead, as shown in the bottom portion of the figure, add the following statement:

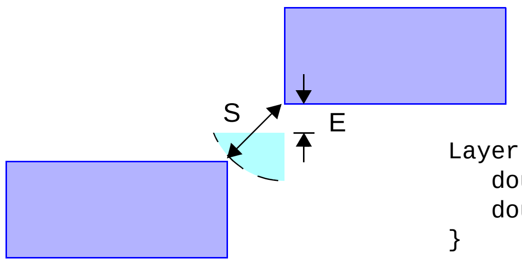
```
doublePatternCheckManhattanSpacing = 1
```

Double-Patterning Corner-to-Corner Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-149.

Figure 2-149 Double-Patterning Corner-to-Corner Spacing Rule



```
Layer "MetalX" {
   doublePatternCornerMinSpacing   = S
   doublePatternCornerKeepoutWidth = E
}
```

For example,

```
Layer "MetalX" {
  doublePatternCornerMinSpacing   = 0.10
  doublePatternCornerKeepoutWidth = 0.02
}
```

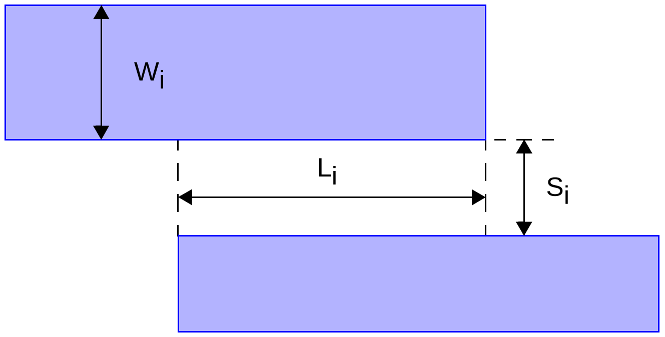
When the corners lie within the specified keepout width, the rule does not apply, but other rules might apply such as line-side to line-side, line-side to line-end, and line-end to line-end.

Double-Patterning Fat Metal Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-150. This rule applies to both line-ends and line-sides, as long as the metal widths meet the threshold requirements.

Figure 2-150 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)
}
```

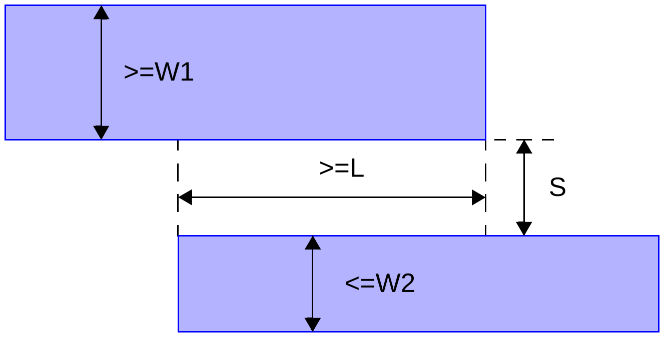
In this example, when the wire width is at least 0.08 but less than 0.10, and the parallel overlap is at least 0.10, the minimum spacing is 0.10. When the wire width is at least 0.10, and the parallel overlap is at least 0.11, the minimum spacing is 0.12.

Double-Patterning Fat-to-Thin Metal Spacing Rule

Note:
This rule is supported only by Zroute.

The double-patterning fat-to-thin metal spacing rule specifies the minimum spacing between two same-mask metal shapes, one fat (width >= W1) and the other thin (width <= W2), with a parallel overlap of at least L. See Figure 2-151.

Figure 2-151 Double-Patterning Fat-to-Thin Metal Spacing Rule



This is the syntax of the rule:

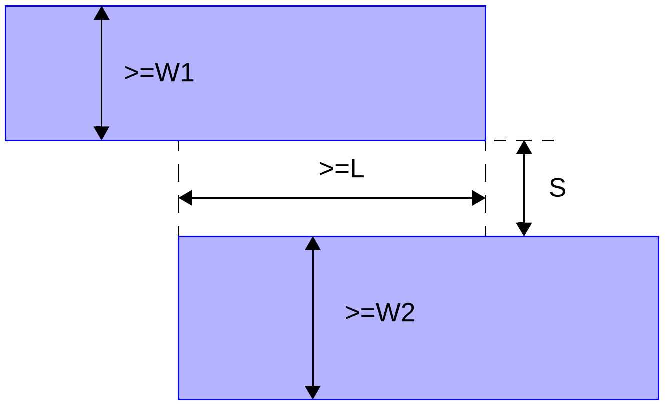
```
Layer "MetalX" {
  doublePatternFatWidthThreshold        = W1
  doublePatternThinMaxWidthThreshold    = W2
  doublePatternFatToThinParallelLength = L
  doublePatternFatToThinMinSpacing      = S
}
```

Double-Patterning Fat-to-Fat Metal Spacing Rule

Note:
This rule is supported only by Zroute.

The double-patterning fat-to-fat metal spacing rule specifies the minimum spacing between two same-mask metal shapes, both fat as measured by two width thresholds (width >= W1 and width >= W2), with a parallel overlap of at least L. See Figure 2-152.

Figure 2-152 Double-Patterning Fat-to-Fat Metal Spacing Rule



This is the syntax of the rule:

```
Layer "MetalX" {
  doublePatternFatToFatWidthThreshold  = W1
  doublePatternFatToFatWidthThreshold2 = W2
  doublePatternFatToFatParallelLength  = L
  doublePatternFatToFatMinSpacing      = S
}
```

Color-Based Span Spacing Rule

Note:

This rule is supported only by Zroute.

The color-based span spacing rule is an extension of the basic metal span spacing rules described in Fat Metal Span Spacing Rule. With the color- based rule, you can specify the spacing constraints for a nearby metal shape of the same color or a different color. As with basic span spacing rules, the span spacing is checked in the same direction as the span. The span is defined in either the X or a Y direction. The parallel run length is always measured perpendicular to the span.

In the technology file, the syntax for same-colored metal shapes is:

```
Layer "MetalX" {
   sameColorSpanTblXDimension = M
   sameColorSpanTblXThreshold = (W1, W2, W3)
   sameColorSpanTblYParallelLengthDimension = N
   sameColorSpanTblYParallelLength         = (-L1, 0, L2)
   sameColorSpanTblXMinSpacing             = (... ... ...) # M x N table
   sameColorSpanRuleExcludedForEndOfLineMaxWidthThreshold
                                           = We
   sameColorSpanRuleExcludedForEndOfLineMinLength
                                           = 0 # optional
...
   sameColorSpanTblYDimension = My
   sameColorSpanTblYThreshold = (Wy1, Wy2, Wy3)
   sameColorSpanTblXParallelLengthDimension = Ny
   sameColorSpanTblXParallelLength = (-Lx1, 0, Lx2)
   sameColorSpanTblYMinSpacing  = (... ... ...)    My x Ny table
}
```

Similarly, the syntax for different-colored metal shapes is:

```
Layer "MetalX" {
   diffColorSpanTblXDimension = M
   diffColorSpanTblXThreshold = (W1, W2, W3)
   diffColorSpanTblYParallelLengthDimension = N
   diffColorSpanTblYParallelLength         = (-L1, 0, L2)
   diffColorSpanTblXMinSpacing             = (... ... ...) # M x N table
   diffColorSpanRuleExcludedForEndOfLineMaxWidthThreshold
                                           = Q
   diffColorSpanRuleExcludedForEndOfLineMinLength
                                           = 0 # optional
...
   diffColorSpanTblYDimension = My
   diffColorSpanTblYThreshold = (Wy1, Wy2, Wy3)
   diffColorSpanTblXParallelLengthDimension = Ny
   diffColorSpanTblXParallelLength = (-Lx1, 0, Lx2)
   diffColorSpanTblYMinSpacing  = (... ... ...)    My x Ny table
}
```

Line-End Exceptions

The color-based span spacing rule is waived for a line-end when both of the following conditions are met:

- The line-end width is less than or equal to the attribute We or Q (same or different colored metal shape)
- The line-end X or Y span is at least the attribute Wx or Wy, respectively

The syntax in the technology file is:

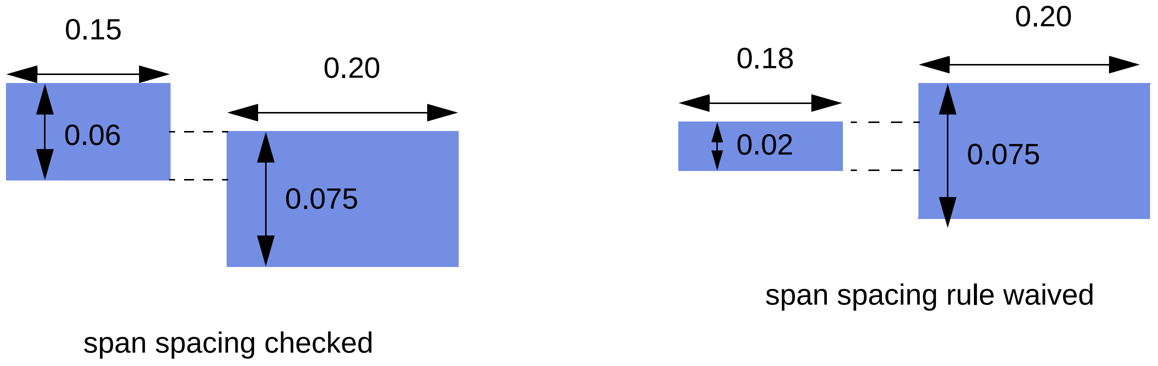
```
Layer "MetalX" {
   ...
   sameColorSpanRuleExcludedForEndOfLineXThreshold = Sx
   sameColorSpanRuleExcludedForEndOfLineYThreshold = Sy
   diffColorSpanRuleExcludedForEndOfLineXThreshold = Dx
   diffColorSpanRuleExdludedForEndOfLineYThreshold = Dy
   ...
}
```

For example, if a technology file has the following entries:

```
sameColorSpanRuleExcludedForEndOfLineMaxWidthThreshold = 0.05
sameColorSpanRuleExcludedForEndOfLineXThreshold = 0.15
```

For the figure on the left in Figure 2-153, the tool checks span spacing since the widths of the line-ends, 0.06 and 0.075, are greater than 0.05 and the spans 0.15 and 0.20 are at least 0.15. For the example on the right, the tool waives checking of the span spacing rule because one of the widths of the line-ends is less than 0.05.

Figure 2-153 Color-Based Span Spacing Line-End Exception

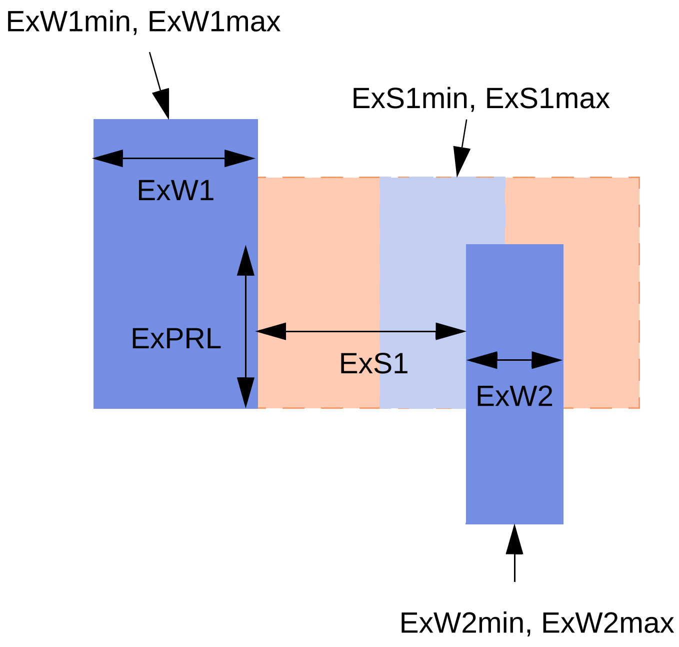


Spacing Range Exceptions

Like Spacing Range Exceptions for the metal span spacing rule, the color-based 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.

Figure 2-154 shows an example of a color-based span spacing range exception.

Figure 2-154 Color-Based Span Spacing Range Exception



In Figure 2-154, the syntax for same-color metal shapes is as follows:

```
sameColorSpanTblXExcludedSpacingRange = (
   "ExW1min, ExW1max", "ExW2min, ExW2max", "-L0, L2", "ExS1min, ExS1max")
```

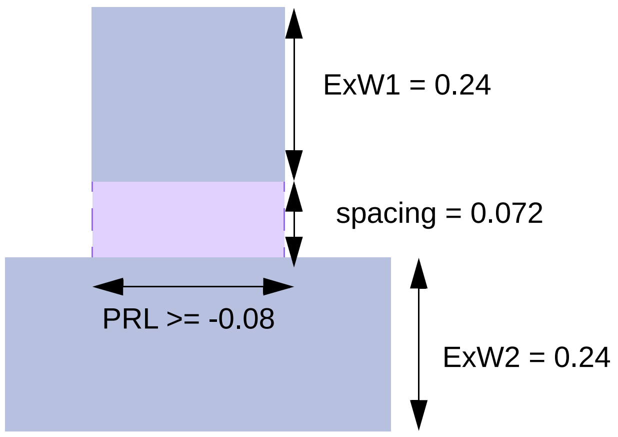
Similarly, for different-color metal shapes:

```
diffColorSpanTblXExcludedSpacingRange = (
   "ExW1min, ExW1max", "ExW2min, ExW2max", "-L0, L2", "ExS1min, ExS1max")
```

where "ExW1min, ExW1max" and "ExW2min, ExW2max" are the ranges for the metal shape spans and "S1min, S1max" is the excluded range. The values for ExW1, ExW2, -L0, and L2 that are specified in the excluded spacing range must match the values previously defined by `sameColorSpanTbl*` or `diffColorSpanTbl*`.

Figure 2-155 shows another example of this rule exception where the range of values is zero.

Figure 2-155 Example of Keepout Metal With Zero Range



In Figure 2-155, the following entires are included in the technology file:

```
sameColorSpanTblXParallelLength = (-0.08, 0, 0.08)
sameColorSpanTblYExcludedSpacingRange =
   ("0.24, 0.24", "0.24, 0.24", "-0.08. 0.08," "0.072, 0.072")
```

The spacing for the metal shapes is 0.072 for all parallel run lengths with a value of at least -0.08. In this example, the metal shape widths have a range of zero because the width range upper and lower boundaries are the same (0.24 and 0.24). However, the parallel run length values represent a minimum boundary value. The -0.08 is a range of at least -0.08 and above. The value of 0.08 is a boundary of 0.08 and above. The spacing is also a range of zero because the range pair is 0.072 and 0.072.

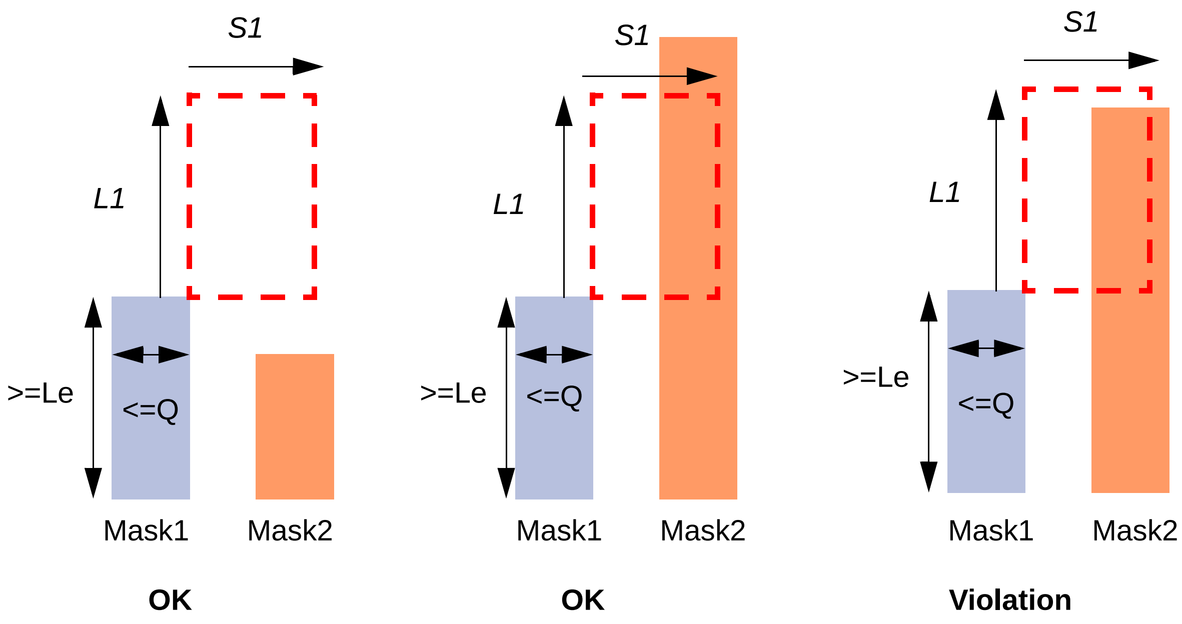
Color-Based Line-End Alignment Rule

Note:

This rule is supported only by Zroute.

The color-based line-end alignment rule specifies a corner keepout area that prevents two line-ends from being too close to each other when they are different-colored metal shapes. The keepout area dimensions are L1 x S1. For example, in Figure 2-156, the first two examples conform to the rule. However, the third example is a violation because Mask2's line-end falls within the keepout area. In all three examples, the metal shapes are parallel to each other so, their line-end corners are also considered to be parallel.

Figure 2-156 Color-Based Line-End Corner Keepout With Parallel Line Ends

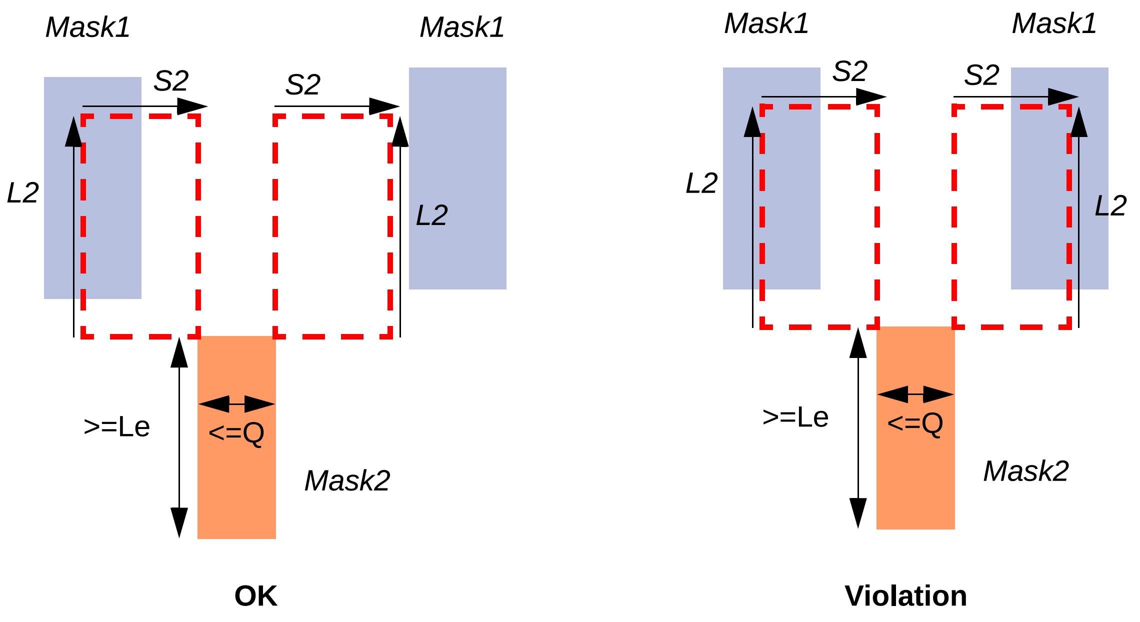


To specify this rule for layer MetalX:

```
Layer "MetalX" {
   color1EndToEndSpaceMaxWidthThreshold = Q
   color1EndToEndSpaceMinLength         = Le # optional
   color1EndToEndSpaceKeepoutWidth      = S1
   color1EndToEndSpaceKeepoutLength     = L1
}
```

For line-ends approaching from opposite directions (facing line-ends), the keepout corner constraint is more relaxed; a line-end is allowed in the keepout region on one side, but not both sides, as shown in Figure 2-157.

Figure 2-157 Color-Based Line-End Corner Keepout With Facing Line Ends



You define the rules in the metal layer section as follows:

```
Layer "MetalX" {
   color2EndToEndSpaceMaxWidthThreshold = Q
   color2EndToEndSpaceMinLength         = Le      # optional
   color2EndToEndSpaceKeepoutWidth      = S2
   color2EndToEndSpaceKeepoutLength     = L2
}
```

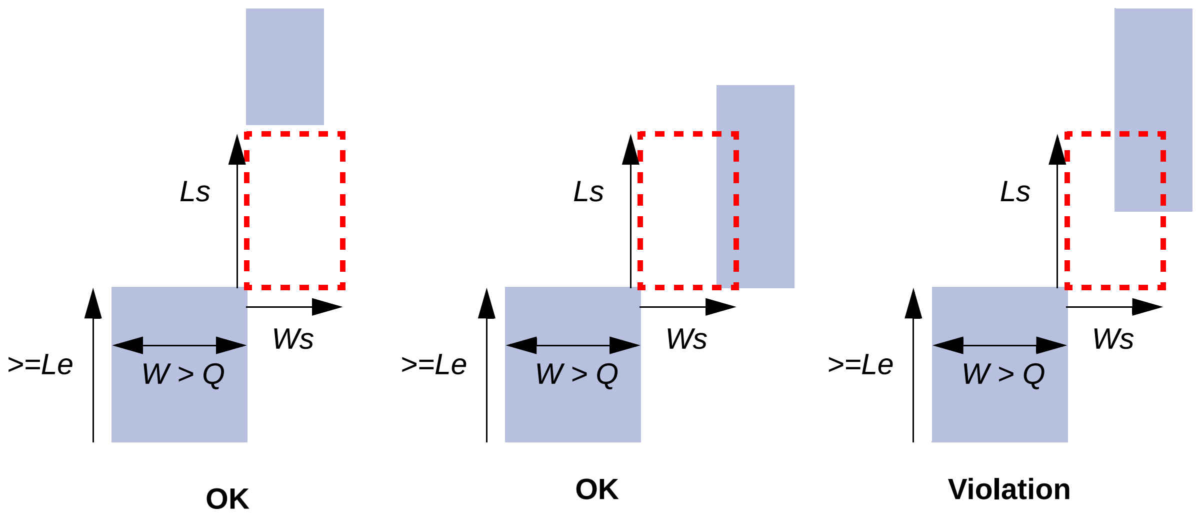
Color-Based Metal Corner Keepout Rule

Note:
   This rule is supported only by Zroute.

The color-based metal corner keepout rule allows you to specify two different keepout areas for corners of two metal shapes that approach each other (facing metal shapes). One keepout area applies to metal shapes of the same color and another applies to metal shapes of different colors. Although the two different keepout areas might be different sizes, they share the same line-end width and length threshold attributes.

Figure 2-158 illustrates the rule. A violation occurs when a metal shape ends inside the keepout region. There is no violation if the metal shape is outside of the keepout area, merely touches the keepout boundary, or passes entirely through the keepout area. It is a violation if a metal shape's corner resides i the keepout area. This example shows same-color metal shapes, but the rule also applies to different-colored metal shapes.

Figure 2-158 Color-Based Metal Corner Keepout Rule



The syntax for this rule is as follows:

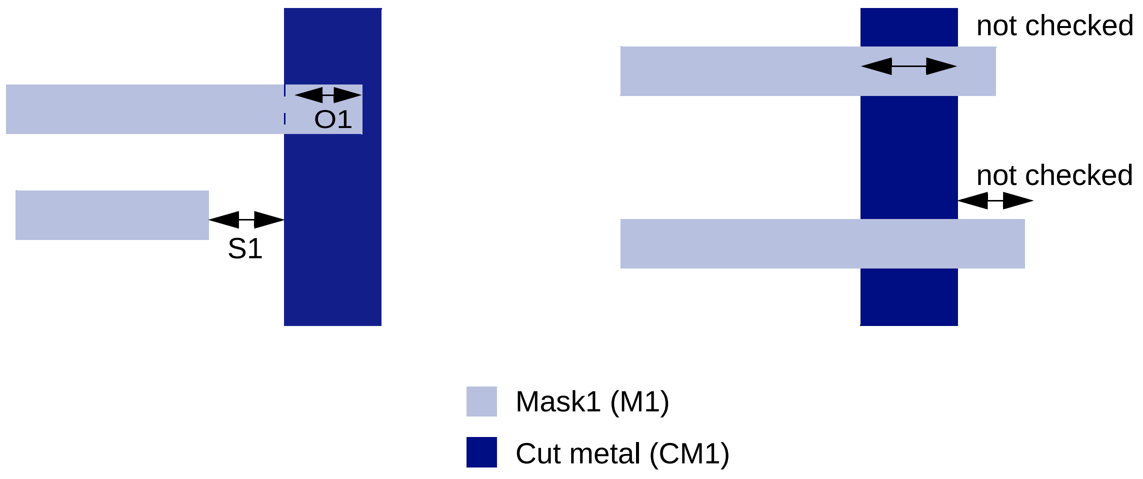
```
Layer "MetalX" {
   sameColorCornerKeepoutWidth   = Ws
   sameColorCornerKeepoutLength  = Ls
   diffColorCornerKeepoutWidth   = Wd
   diffColorCornerKeepoutLength  = Ld
   cornerEndOfLineWidthThreshold = Q
   cornerEndOfLineMinLength      = Le
}
```

Minimum Metal to Cut-Metal Keepout and Overlap Rule

Note:
   This rule is supported only by Zroute.

The minimum metal to cut-metal keepout and overlap rule defines the minimum keepout distance between a metal shape and a cut-metal shape. A cut metal layer is used for self-aligned double-patterning processes. For a metal shape on the same mask, there is a minimum overlap length as shown in Figure 2-159. This rule only applies when the metal shape is facing the cut metal. Any extension over the cut metal is not checked.

Figure 2-159 Minimum Metal to Cut Metal Keepout and Overlap



Define the cut metal layer in the `Layer` section of the technology file:

```
Layer "CM1" {
   layerNumber   = N
   maskName      = "cutMetal1"
}
```

Define the rule in the `DesignRule` section:

```
DesignRule {
   layer1            = "CM1"
   layer2            = "M1"
   mask1KeepoutMinDist = S1
   mask1OverlapMinLength = O1
}
```

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.

The following topics describe the bridge rules:

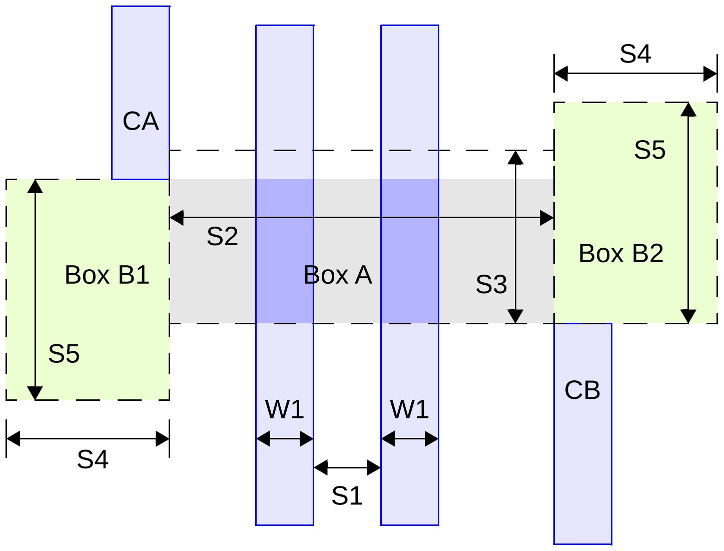
- Bridge Rule 1
- Bridge Rule 2
- Bridge Rule 3

- Via-Induced Metal Bridge Rule

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 2-160.

Figure 2-160 Bridge Rule 1

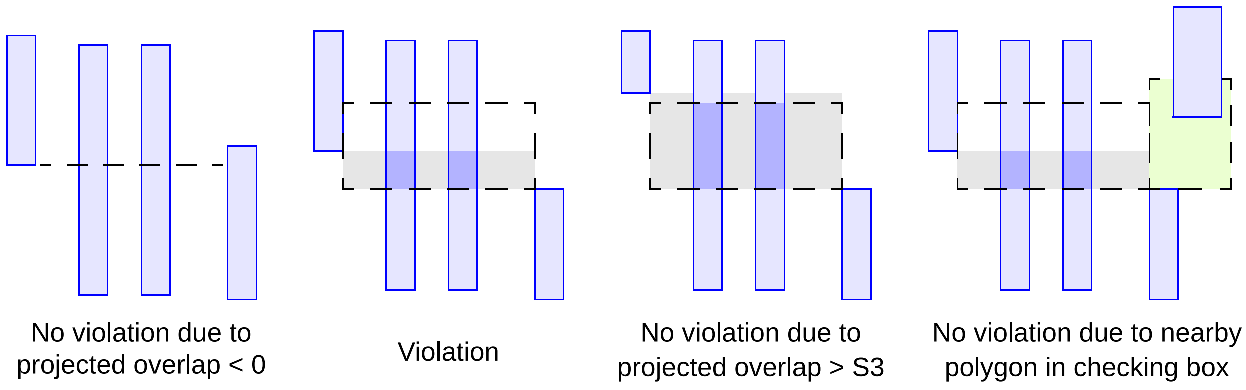


The two parallel metal lines have a width of exactly W1 and a spacing of exactly S1, and they fully cross the projection area between the two outside line-ends, shown as Box A in the figure. The two line-ends of metal CA and CB, if extended longer, would have a spacing of exactly S2.

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 2-161 shows some examples of where the rule does and does not apply.

Figure 2-161 Bridge Rule 1 Examples


No violation due to projected overlap < 0
Violation
No violation due to projected overlap > S3
No violation due to nearby polygon in checking box

This is the general syntax of the rule:

```
Layer "MetalX" {
   diffSideKeepoutMidWireExactWidth   = W1
   diffSideKeepoutMidWireExactSpacing = S1
   diffSideKeepoutNumMidWires         = 2
   diffSideKeepoutSideExactSpacing    = S2
   diffSideKeepoutEndMinSpacing       = S3
   diffSideKeepoutWidth               = S4
   diffSideKeepoutLength              = S5
}
```

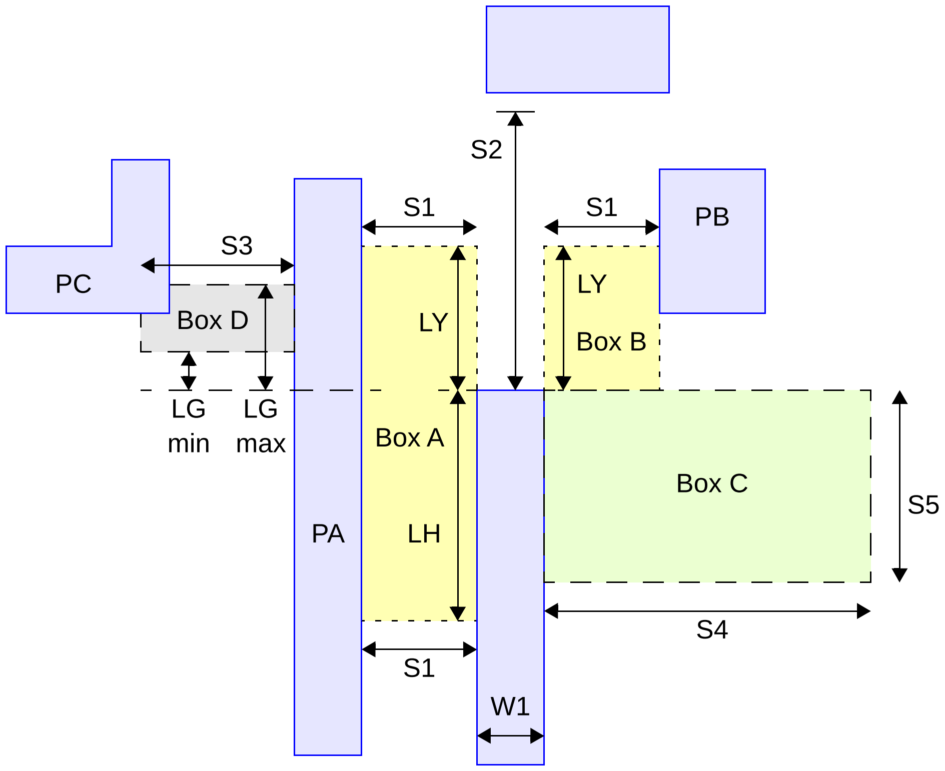
For example,

```
Layer "MetalX" {
   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 2-162.

Figure 2-162 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 abutted 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
}
```

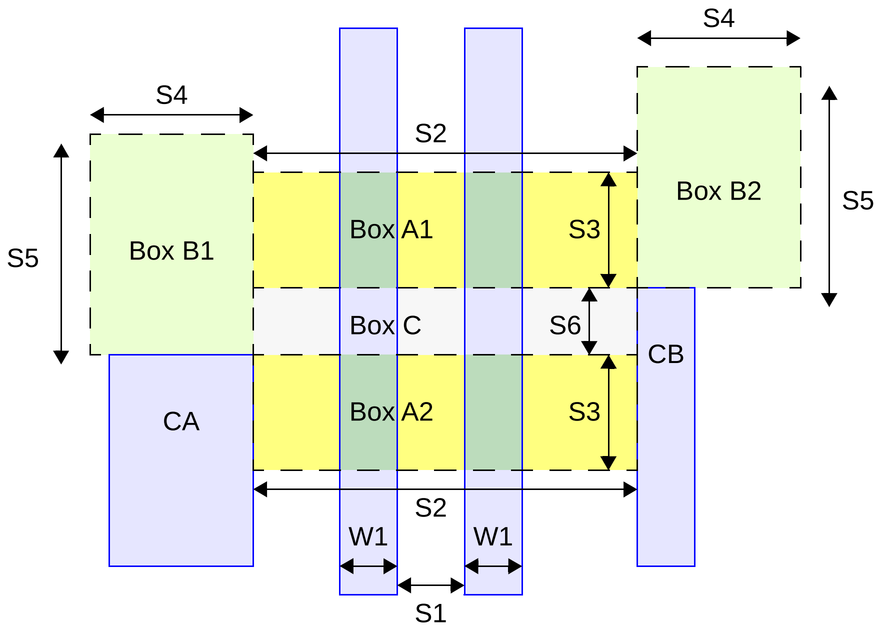
For example,

```
Layer "MetalX" {
  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 2-163.

Figure 2-163 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 "MetalX" {
  sameSideKeepoutMidWireExactWidth    = 0.06
  sameSideKeepoutMidWireExactSpacing  = 0.06
  sameSideKeepoutNumMidWires          = 2
  sameSideKeepoutSideExactSpacing     = 0.30
  sameSideKeepoutEndExtensionRange    = 0.11
  sameSideKeepoutEndMinOffset         = 0.06
  sameSideKeepoutWidth                = 0.20
  sameSideKeepoutLength               = 0.14
}
```

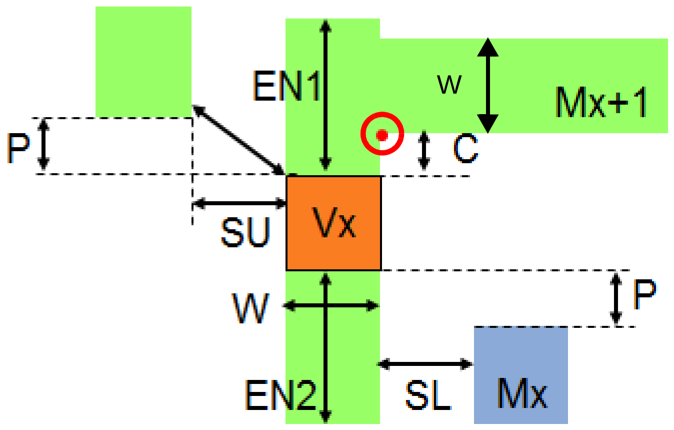
Via-Induced Metal Bridge Rule

Note:
 This rule is supported only by Zroute.

The via-induced metal bridge rule specifies a minimum spacing between a via 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 2-164, 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 2-164 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)
}
```

Fat Metal Rules

Fat metal rules are rules that are based on the width of the metal shapes. You define the fat metal width thresholds for a layer by setting the `fatTblThreshold` attribute in the `Layer` section.

The following topics describe the fat metal rules:

- Fat Metal Minimum Enclosed Area Rule
- Fat Metal Spacing Rule
- Fat Metal Span Spacing Rule
- Fat Metal Orthogonal Spacing Rule
- Fat Metal Parallel Length Rule
- Fat Metal Extension Spacing Rule
- Fat Metal Corner Keepout Rule
- Fat Metal Jog Rule
- Fat Metal Forbidden Spacing Rule

Fat Metal Minimum Enclosed Area Rule

The fat metal minimum enclosed area rule defines a set of minimum enclosed areas based on the wire width. 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 consider the created wire widths to avoid requiring a larger enclosed area.

In the following example, A_0 applies to thin wires (those less than T_1). A_1 applies to wires wider than or equal to T_1, but less than T_2. A_2 applies to wires wider than or equal to T_2.

The syntax for this rule is

```
Layer "MetalX" {
   fatTblDimension     = 3
   fatTblThreshold     = (0, $T_1$, $T_2$)
   fatTblMinEnclosedArea = ($A_0$, $A_1$, $A_2$)
}
```

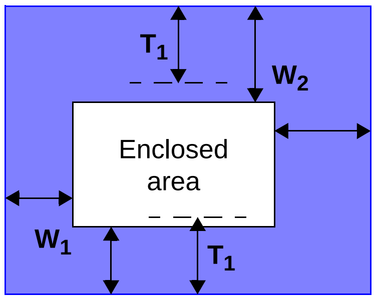
The fat metal minimum enclosed area rule has two modes, which are controlled by the setting of the `fatTblMinEnclosedAreaMode` attribute in the `Technology` section.

- When `fatTblMinEnclosedAreaMode = 0`, which is the default, the tool uses the minimum enclosed area defined for the narrowest wire.
- When `fatTblMinEnclosedAreaMode = 1`, the tool uses the minimum enclosed area defined for the widest wire.

For example, in Figure 2-165, the width of the metal surrounding the enclosed area is W_1 on two sides and W_2 on the other two sides. W_1 is less than the fat wire threshold T_1 and W_2 is greater than the fat wire threshold T_1.

- When `fatTblMinEnclosedAreaMode = 0` (the default), A_0 is applied because there is at least one surrounding width below the threshold.
- When `fatTblMinEnclosedAreaMode = 1`, A_1 is applied because at least one wire is above the threshold.

Figure 2-165 Minimum Enclosed Area Rule Modes



See Also

- Minimum Enclosed Area Rule

Fat Metal Spacing Rule

The fat metal spacing 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.

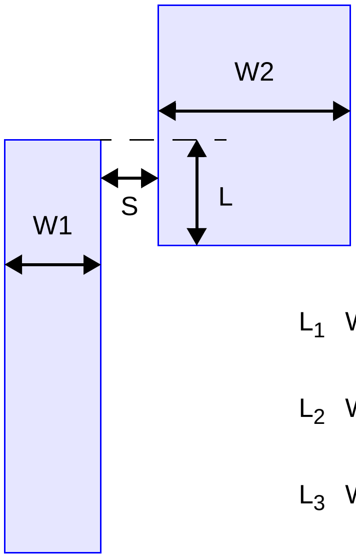
There are two forms of this rule:

- 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 2-166 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 2-166 Width Versus Width and Width Versus Parallel Length Forms



Width versus width,
N=3

		L_1 W_{11}	L_2 W_{12}	L_3 W_{13}
L_1	W_{21}	S_{11}	S_{12}	S_{13}
L_2	W_{22}	S_{21}	S_{22}	S_{23}
L_3	W_{23}	S_{31}	S_{32}	S_{33}

Width versus parallel length,
N=3, M=4

	L_1	L_2	L_3	L_4
W_1	S_{11}	S_{12}	S_{13}	S_{14}
W_2	S_{21}	S_{22}	S_{23}	S_{24}
W_3	S_{31}	S_{32}	S_{33}	S_{34}

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,
                           S21, S22, ..., S2N,
                           ...
                           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. By default, there is no restriction on the amount of downgrade that can result from having a smaller parallel overlap. To restrict the amount of the downgrade to no more than one row and one column in the table, set the `fatTblSpacingMode` attribute to 1.

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:

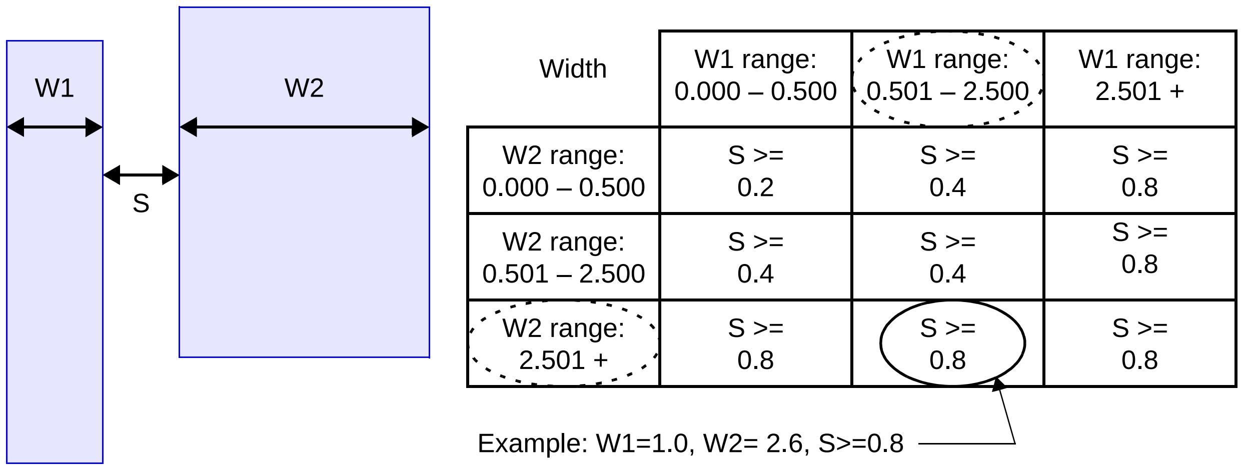
- A minimum spacing of 0.2 between any two parallel metal segments.
- A minimum spacing of 0.4 between two parallel metal segments when either one is greater than 0.5 and the parallel length is greater than 1.5.
- A minimum spacing of 0.8 between two parallel metal segments when either one is greater than 2.5 and the parallel length is greater than 2.5.

To specify this in the technology file, enter

```
Layer "MetalX" {
   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* x *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 2-167.

Figure 2-167 Fat Metal Minimum Spacing Table, Width Versus Width

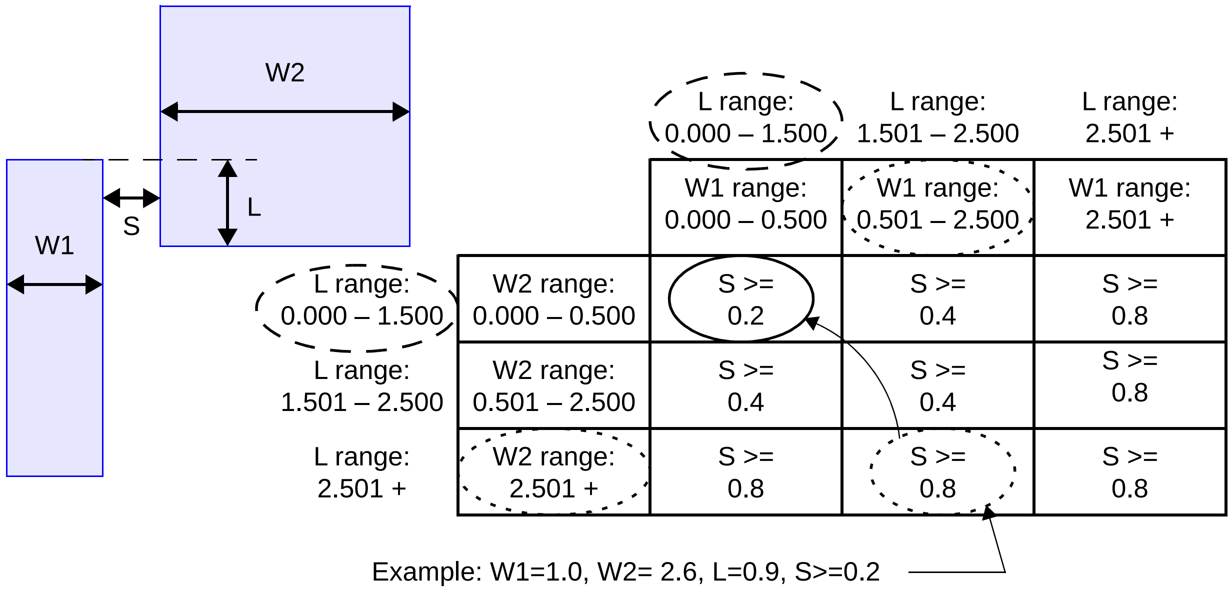


Width	W1 range: 0.000 – 0.500	W1 range: 0.501 – 2.500	W1 range: 2.501 +
W2 range: 0.000 – 0.500	S >= 0.2	S >= 0.4	S >= 0.8
W2 range: 0.501 – 2.500	S >= 0.4	S >= 0.4	S >= 0.8
W2 range: 2.501 +	S >= 0.8	S >= 0.8	S >= 0.8

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`, 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 2-168 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 2-168 Parallel Overlap Length Downgrade of Minimum Spacing



		L range: 0.000 – 1.500	L range: 1.501 – 2.500	L range: 2.501 +
		W1 range: 0.000 – 0.500	W1 range: 0.501 – 2.500	W1 range: 2.501 +
L range: 0.000 – 1.500	W2 range: 0.000 – 0.500	S >= 0.2	S >= 0.4	S >= 0.8
L range: 1.501 – 2.500	W2 range: 0.501 – 2.500	S >= 0.4	S >= 0.4	S >= 0.8
L range: 2.501 +	W2 range: 2.501 +	S >= 0.8	S >= 0.8	S >= 0.8

You can optionally restrict the amount of the downgrade to no more than one row and one column in the table by setting the `fatTblSpacingMode` attribute to 1. In that case, in Figure 2-168, 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.

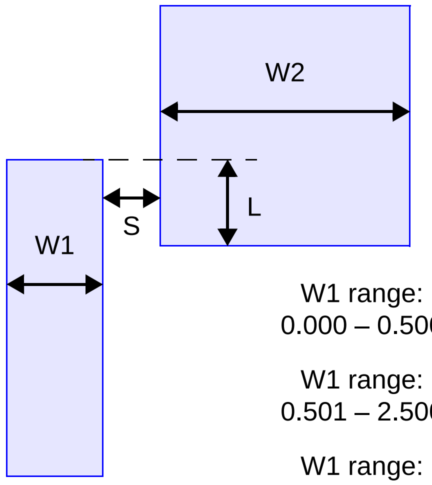
The `fatTblSpacing` attribute is a list of *N* by *M* values specifying the minimum spacing requirement for each combination of segment width and parallel overlap. The rule considers only the width of the wider of the two segments.

For example, consider the following rule:

```
Layer "MetalX" {
   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 2-169.

Figure 2-169 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.

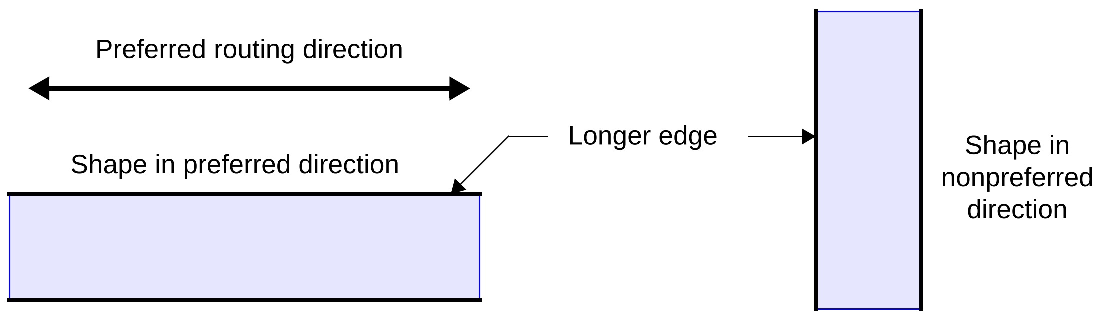
```
fatTblDimension              = 5
fatTblThreshold              = (0.000, 0.050, 0.140, 0.201, 0.221)
fatTblParallelLengthDimension = 4
fatTblParallelLength         = (0.000, 0.000, 0.000, 0.500)
orthoSpacingExcludeCorner    = 1
fatTblSpacing                = ( 4-by-5 table )
```

Preferred and Nonpreferred Fat Metal Spacing Rule

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.

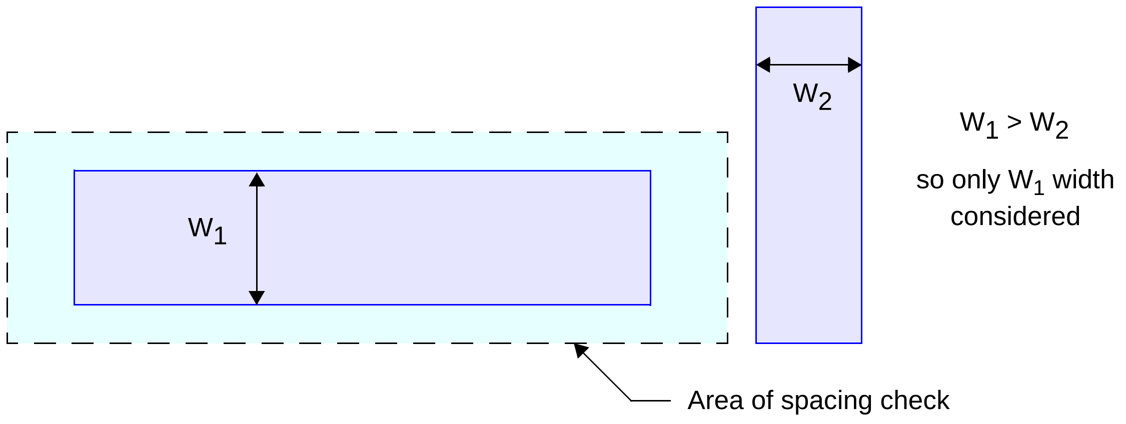
A rectangle is considered to be running in the preferred routing direction if the larger edges (sides) of the rectangle are parallel to that direction, as shown in Figure 2-170. Otherwise, the rectangle is considered to be running in the nonpreferred direction.

Figure 2-170 Rectangle's Routing Direction


Preferred routing direction
Shape in preferred direction
Longer edge
Shape in nonpreferred direction

In checking the spacing between two metal shapes of different widths, the rule considers the only the wider shape's width. See Figure 2-171.

Figure 2-171 Only Greater Width Considered

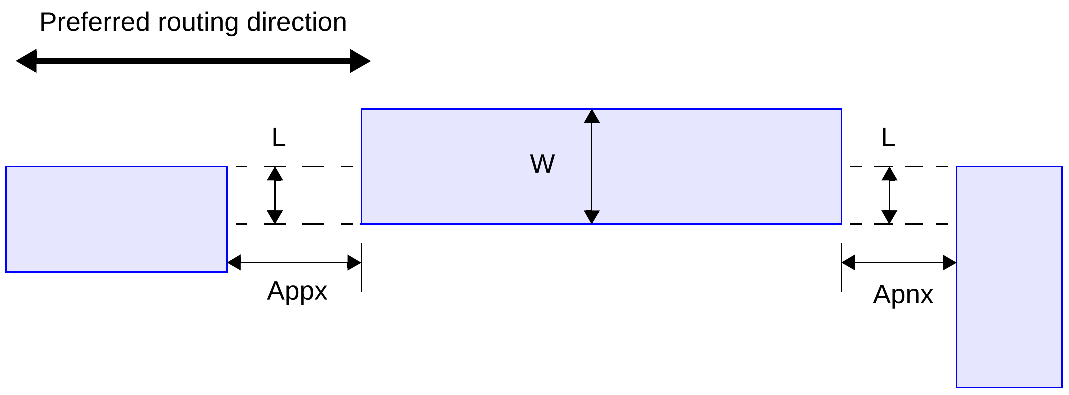


For the wider shape running in the preferred direction, the minimum spacing in the x-direction depends on the width, the length of the parallel overlap between the metal and the nearby metal, and whether the nearby metal is also running in the preferred direction.

X-Spacing From Shape in Preferred Direction

In Figure 2-172, 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 2-172 Fat Metal X Spacing Rule, Wider Metal Running in Preferred Direction



In the diagram, the Appx notation 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 Apnx notation 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,...)
  ...
```

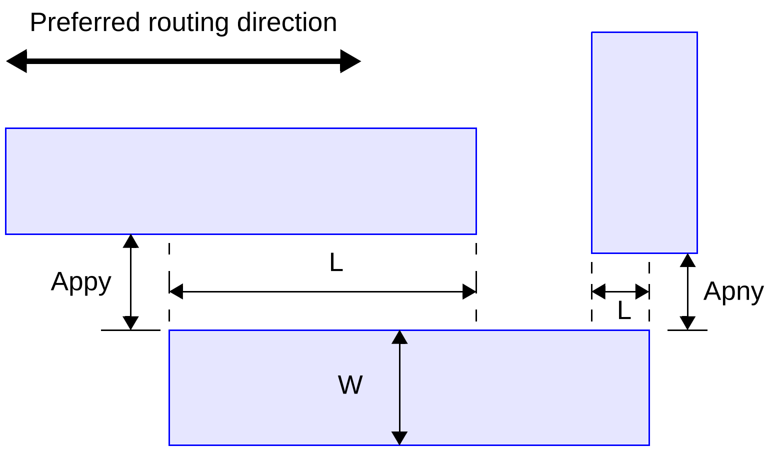
Each spacing attribute is a list of (M x N) values corresponding to each combination of fat metal width and parallel overlap. For example,

```
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 2-173, 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 2-173 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:

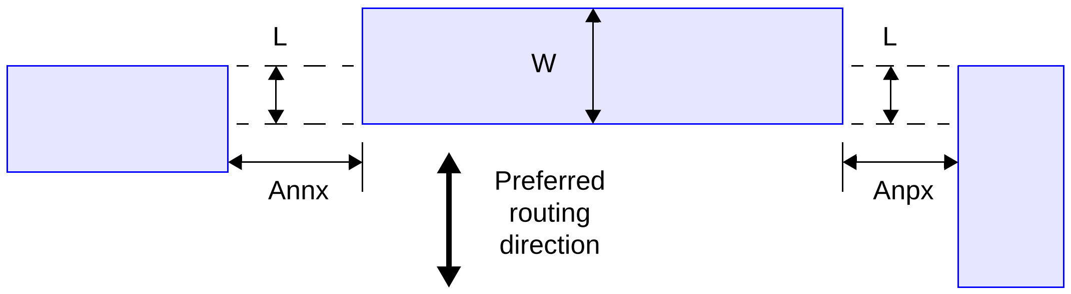
```
Layer "Mx" {
  fatTblPrefWidthDimension            = M
  fatTblPrefWidthThreshold            = (W1,W2,...,Wm)
  fatTblPrefXParallelLengthDimension = N
  fatTblPrefXParallelLengthThreshold = (L1,L2,...,Ln)
  fatTblPrefToPrefYMinSpacing         = (Appy1,Appy2,...)
  fatTblPrefToNonPrefYMinSpacing      = (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 2-174, 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 2-174 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:

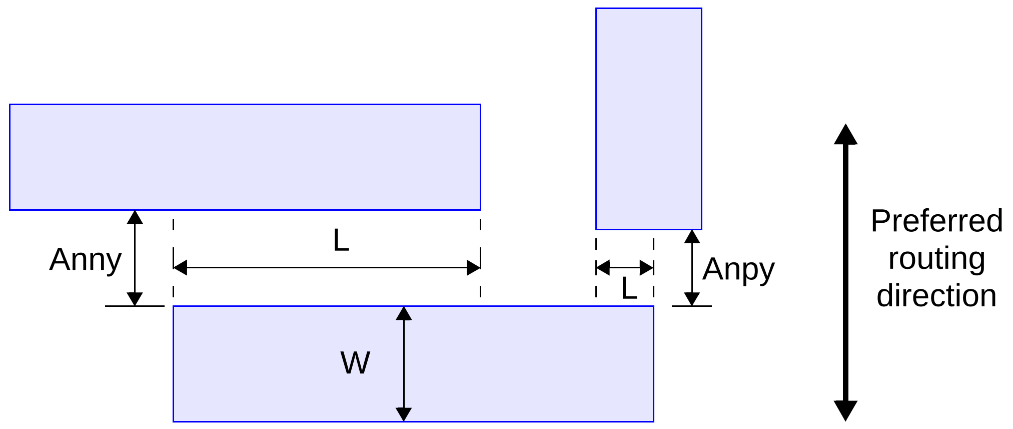
```
Layer "Mx" {
  fatTblNonPrefWidthDimension            = R
  fatTblNonPrefWidthThreshold            = (W1,W2,...,Wr)
  fatTblNonPrefYParallelLengthDimension = S
  fatTblNonPrefYParallelLengthThreshold = (L1,L2,...,Ls)
  fatTblNonPrefToPrefXMinSpacing         = (Anpx1,Anpx2,...)
  fatTblNonPrefToNonPrefXMinSpacing      = (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 2-175, 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 2-175 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:

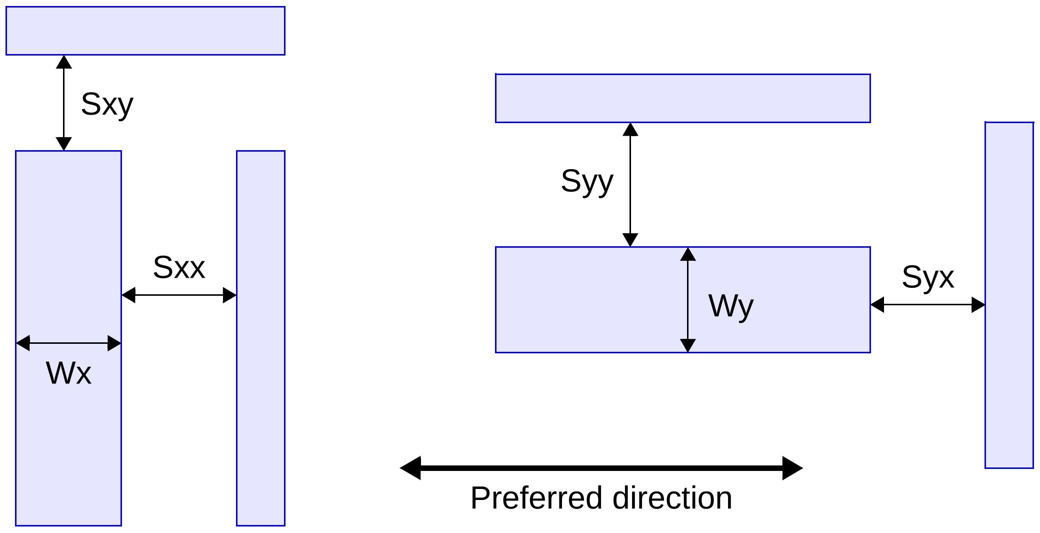
```
Layer "Mx" {
  fatTblNonPrefWidthDimension           = R
  fatTblNonPrefWidthThreshold           = (W1,W2,...,Wr)
  fatTblNonPrefXParallelLengthDimension = S
  fatTblNonPrefXParallelLengthThreshold = (L1,L2,...,Ls)
  fatTblNonPrefToPrefYMinSpacing        = (Anpy1,Anpy2,...)
  fatTblNonPrefToNonPrefYMinSpacing     = (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 2-176.

Figure 2-176 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 = ("L_{11}, U_{11},…", …)_{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)
}
```

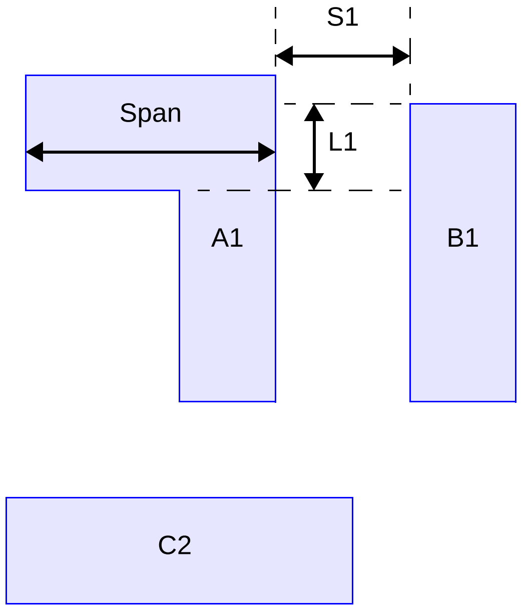
Fat Metal Span Spacing Rule

The fat metal span spacing rule is similar to the fat metal spacing rule described in Fat Metal Spacing Rule. 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.

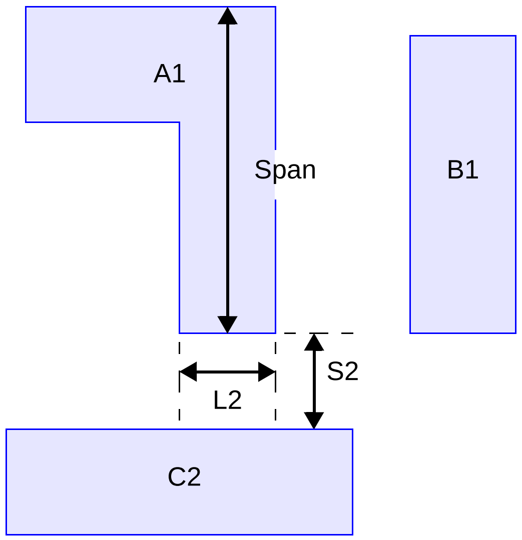
In Figure 2-177, when the metal span spacing rule considers the horizontal spacing S1 between metal A1 and metal B1, it considers the horizontal span across A1, perpendicular to the edges separating A1 and B1. It also considers the parallel length L1 where the wider top portion of metal A1 overlaps the vertical run of B1.

Figure 2-177 Metal Span Attributes for a Horizontal Spacing Check



In Figure 2-178, when the metal span spacing rule considers the vertical spacing S2 between metal A1 and metal C2, 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 2-178 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 "MetalX" {
  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 one-dimensional or two-dimensional table. A one-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 two-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 two-dimensional minimum spacing table:

```
Layer "MetalX" {
  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, the minimum spacing between the shapes must be at least 0.085.

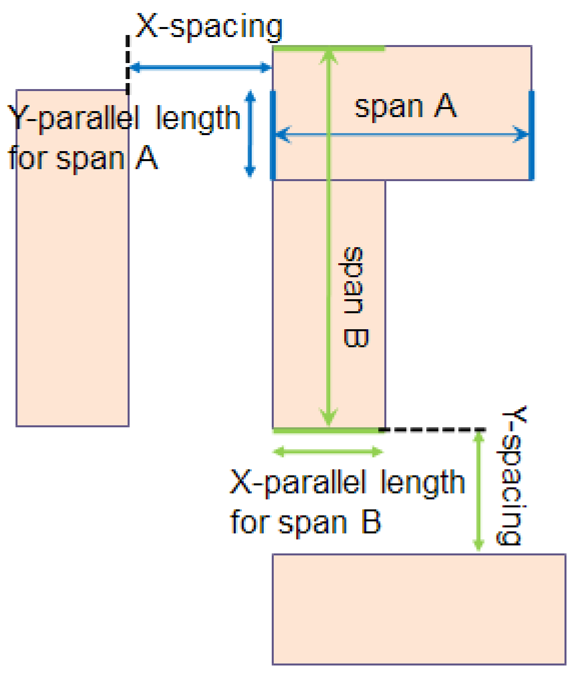
The contents of a two-dimensional table must be symmetrical with respect to a diagonal through the square matrix:

```
  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)
```

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 2-179.

Note:
This advanced form of the rule is supported only by Zroute.

Figure 2-179 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.

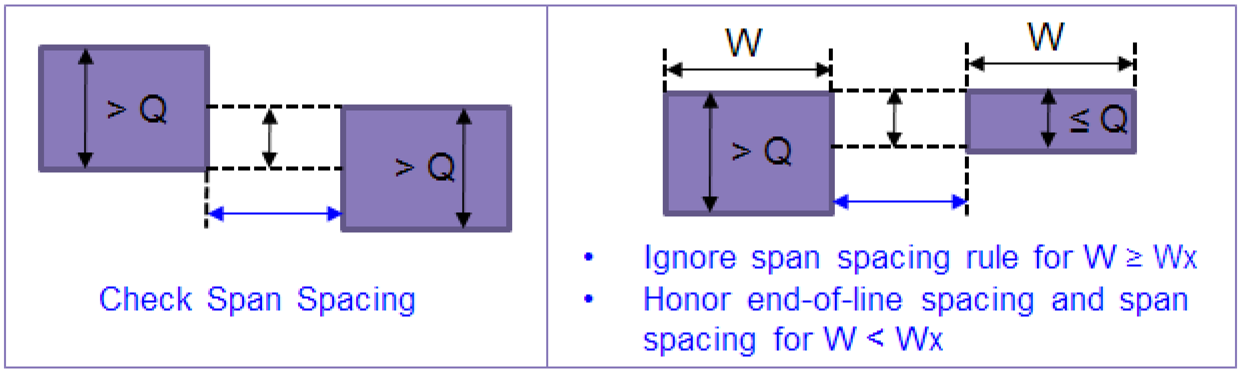
Line-End Exceptions

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 2-180.

Figure 2-180 Metal Span Orthogonal Spacing Line-End Exceptions


> Q
> Q
Check Span Spacing
W
W
> Q
≤ Q
• Ignore span spacing rule for W ≥ Wx
• Honor end-of-line spacing and span spacing for W < Wx

For example,

```
  spanRuleExcludedForEndOfLineMaxWidthThreshold = 0.09
  spanRuleExcludedForEndOfLineXThreshold        = 0.18
  spanRuleExcludedForEndOfLineYThreshold        = 0.23
```

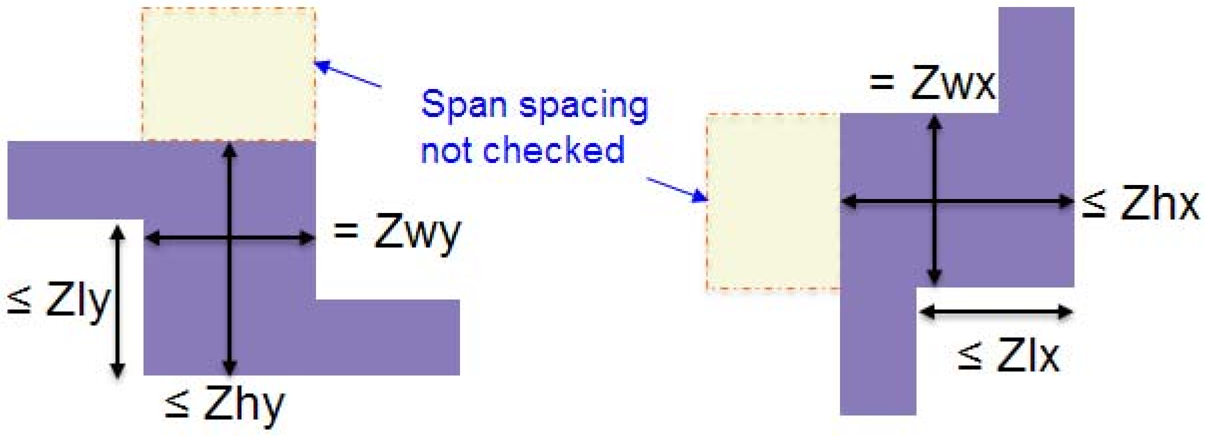
You can optionally specify a line-end minimum length for which the rule exclusion applies, for example,

```
spanRuleExcludedForEndOfLineMinLength = 0.09
```

Z-Shape Exceptions

The span spacing rule can be waived for any Z-shape with width equal to Zw, height no more than Zh, and edge length no more than Zl, as shown in Figure 2-181.

Figure 2-181 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,

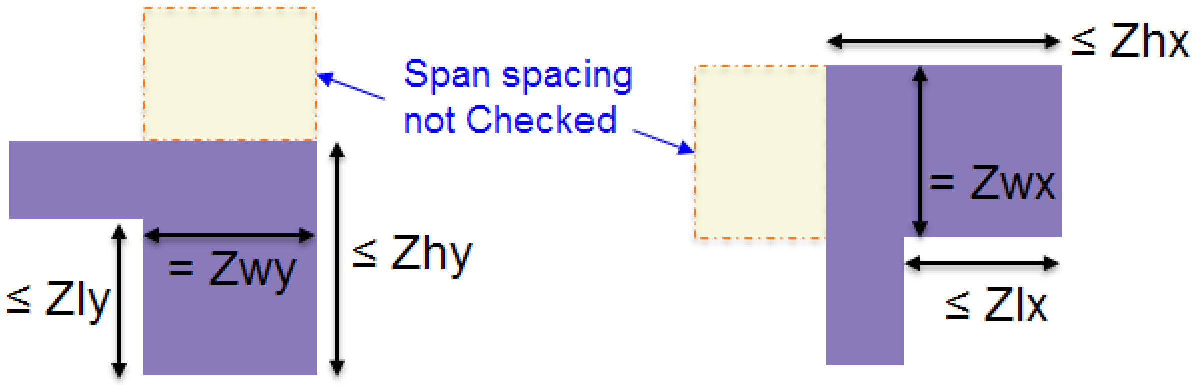
```
spanTblYDimension = 5
...
spanTblYExcludedForZWireWidth         = ( -1, -1, -1, 0.09, 0.09)
spanTblYExcludedForZWireMaxHeight     = ( -1, -1, -1, 0.14, 0.14)
spanTblYExcludedForZWireMaxEdgeLength = ( -1, -1, -1, 0.10, 0.10)
```

In this example, the rule is waived for the fourth and fifth span ranges when the Z-shape dimensions meet the Zwy, Zhy, and Zly 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 Zw, height no more than Zh, and edge length no more than Zl, as shown in Figure 2-182.

Figure 2-182 Metal Span Orthogonal Spacing L-Shape Exceptions



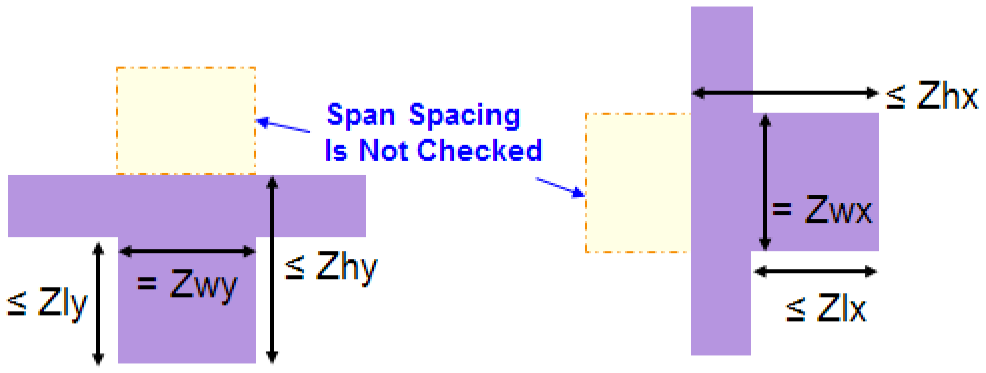
To specify the L-shape exception, add a line to the syntax for the Z-shape exception:

```
spanTblXExcludedForZWireWidth         = (Zwx, -1, …)
spanTblXExcludedForZWireMaxHeight     = (Zhx, -1, …)
spanTblXExcludedForZWireMaxEdgeLength = (Zlx, -1, …)
spanTblYExcludedForZWireWidth         = (Zwy, -1, …)
spanTblYExcludedForZWireMaxHeight     = (Zhy, -1, …)
spanTblYExcludedForZWireMaxEdgeLength = (Zly, -1, …)
spanRuleZWireIncludeLWire             = 1  #default=0
```

T-Shape Exceptions

The span spacing rule can be waived for any T-shape with width equal to Zw, height no more than Zh, and edge length no more than Zl, as shown in Figure 2-183.

Figure 2-183 Metal Span Orthogonal Spacing T-Shape Exceptions



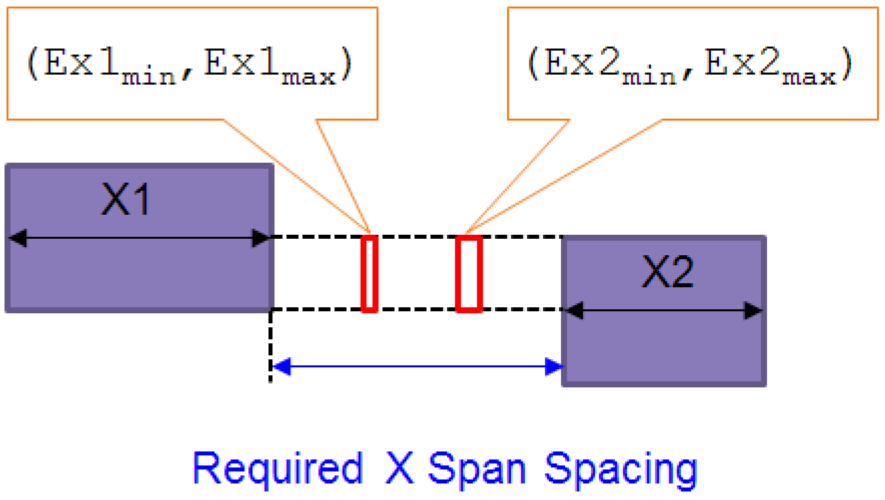
To specify the T-shape exception, add a line to the syntax for the Z-shape exception:

```
spanTblXExcludedForZWireWidth         = (Zwx, -1, …)
spanTblXExcludedForZWireMaxHeight     = (Zhx, -1, …)
spanTblXExcludedForZWireMaxEdgeLength = (Zlx, -1, …)
spanTblYExcludedForZWireWidth         = (Zwy, -1, …)
spanTblYExcludedForZWireMaxHeight     = (Zhy, -1, …)
spanTblYExcludedForZWireMaxEdgeLength = (Zly, -1, …)
spanRuleZWireIncludeTWire             = 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 2-184.

Figure 2-184 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.

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.

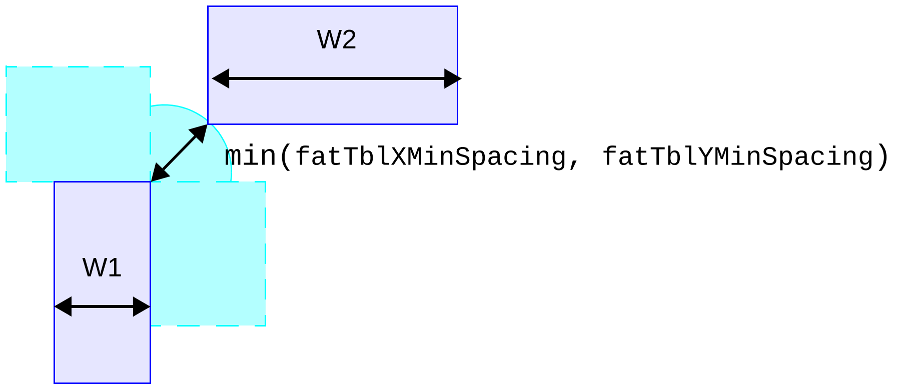
```
Layer "MetalX" {
```

```
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 = 0
checkManhattanSpacing       = 0
cornerSpacingCheckManhattan = 0
}
```

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. By default, when the corner spacing is checked, it is measured diagonally and is the smaller of the `fatTblXMinSpacing` and `fatTblYMinSpacing` value, as shown in Figure 2-185.

Figure 2-185 Fat Metal Orthogonal Spacing Check



To use Manhattan measurement instead of diagonal measurement, set the `checkManhattanSpacing` attribute to 1. To use Manhattan measurement and use the `fatTblXMinSpacing` for x-direction spacing and `fatTblYMinSpacing` for y-direction spacing, set the `cornerSpacingCheckManhattan` attribute to 1.

You can use either the simpler fat table syntax described in Fat Metal Spacing Rule 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) 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.

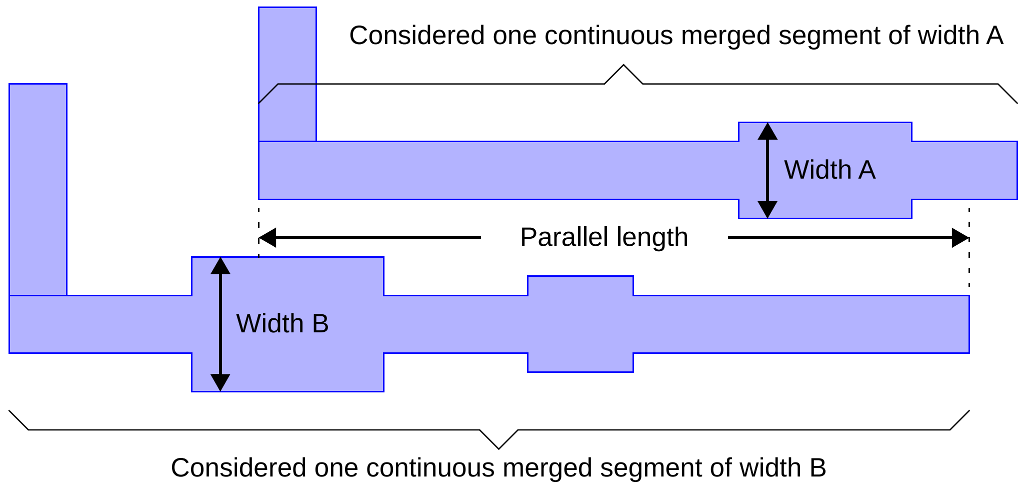
Note:

If you use the fat metal orthogonal spacing rule and the fat metal enclosed minimum area rule (Fat Metal Minimum Enclosed Area Rule) 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 Rule

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 2-186. The minimum spacing between a merged wire and adjacent parallel wires depends on the width and length of the merged wire.

Figure 2-186 Parallel Length Determination


Considered one continuous merged segment of width A
Width A
Parallel length
Width B
Considered one continuous merged segment of width B

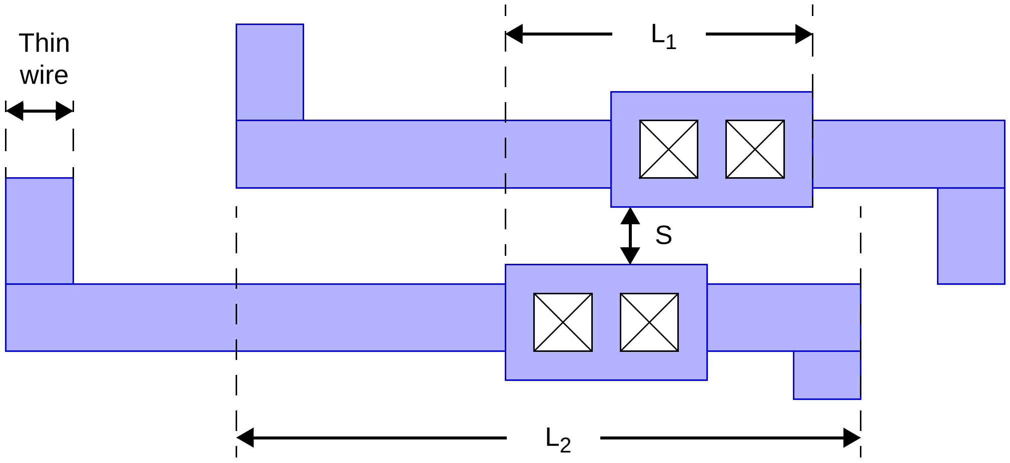
You can control the merging behavior by setting the `parallelLengthMode` attribute in the `Technology` section of the technology file. By default (`parallelLengthMode` = 0), the tool merges the wire segments with fat neighboring shapes only. To merge the wire segments with all neighboring shapes, set the `parallelLengthMode` attribute to 1.

For example,

```
Technology {
   parallelLengthMode = 1
}
```

Figure 2-187 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 2-187 Parallel Length Rule Example

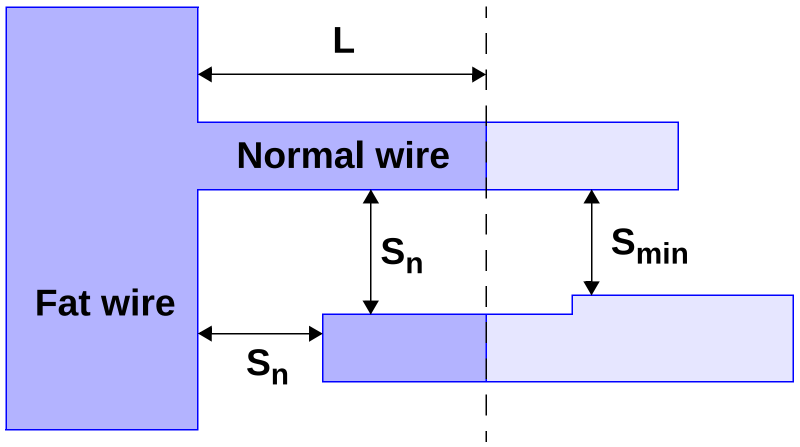


The minimum spacing between these wires depends on the parallel overlap length between them, as defined by the `FatTblParallelLength` and `fatTblSpacing` attributes described in Fat Metal Spacing Rule. In Figure 2-187, the parallel length is either L_1 or L_2, depending on the `ParallelLengthMode` setting. 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 2-188 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 2-188 Fat Metal Wire Having Extension Wire With Normal Width



For example,

```
Layer "MetalX" {
   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 2-4.

Table 2-4 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 2-189 on page 2-201.
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 2-190 on page 2-201.
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 2-191 on page 2-202.

Table 2-4 fatWireExtensionMode Attribute Values (Continued)

Mode value	**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. See Figure 2-192 on page 2-202.
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 2-193 on page 2-203.

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 the extension range of a fat wire edge.
- S_{min}: The default minimum spacing between 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.

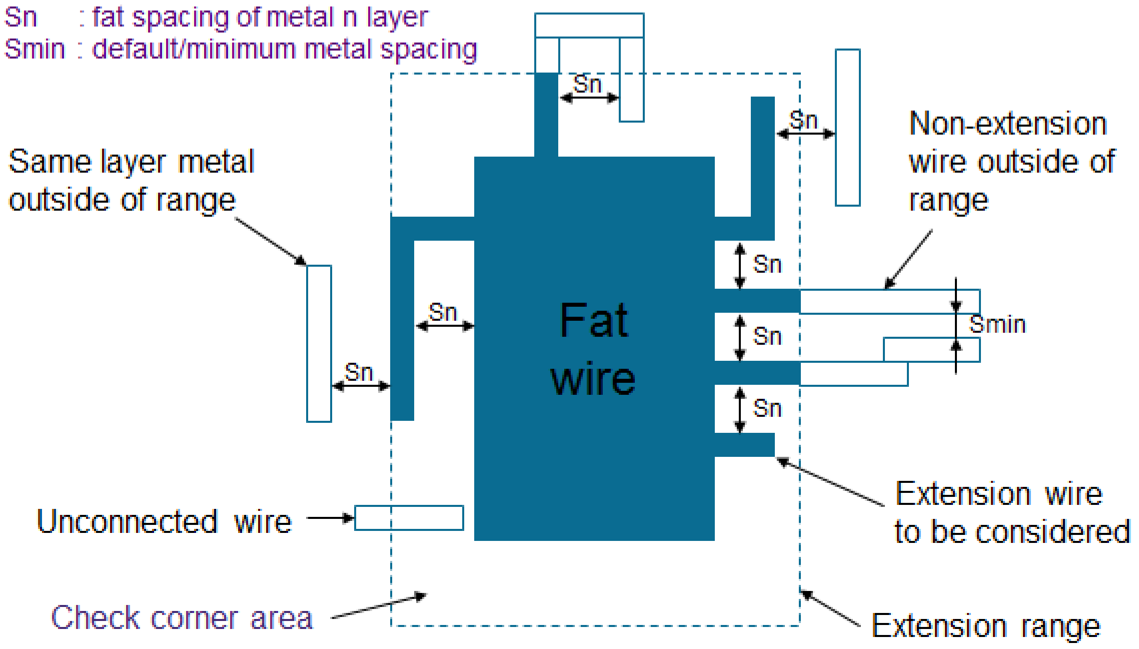

Sn : fat spacing of metal n layer
Smin : default/minimum metal spacing
Same layer metal outside of range
Non-extension wire outside of range
Sn
Sn
Sn
Sn
Sn
Sn
Sn
Smin
Fat wire
Unconnected wire
Extension wire to be considered
Check corner area
Extension range

Figure 2-189 fatWireExtensionMode = 0

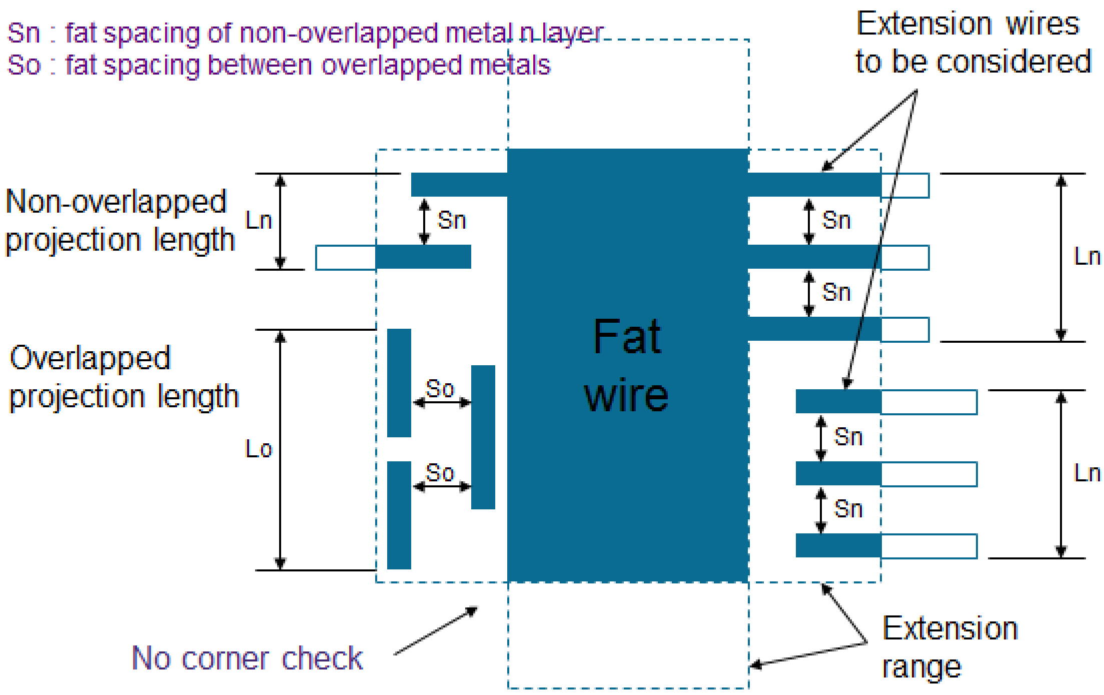

Sn : fat spacing of non-overlapped metal n layer
So : fat spacing between overlapped metals
Extension wires to be considered
Non-overlapped projection length
Ln
Sn
Sn
Sn
Ln
Overlapped projection length
Lo
So
So
Fat wire
Sn
Sn
Ln
No corner check
Extension range

Figure 2-190 fatWireExtensionMode = 1

Figure 2-191 fatWireExtensionMode = 2

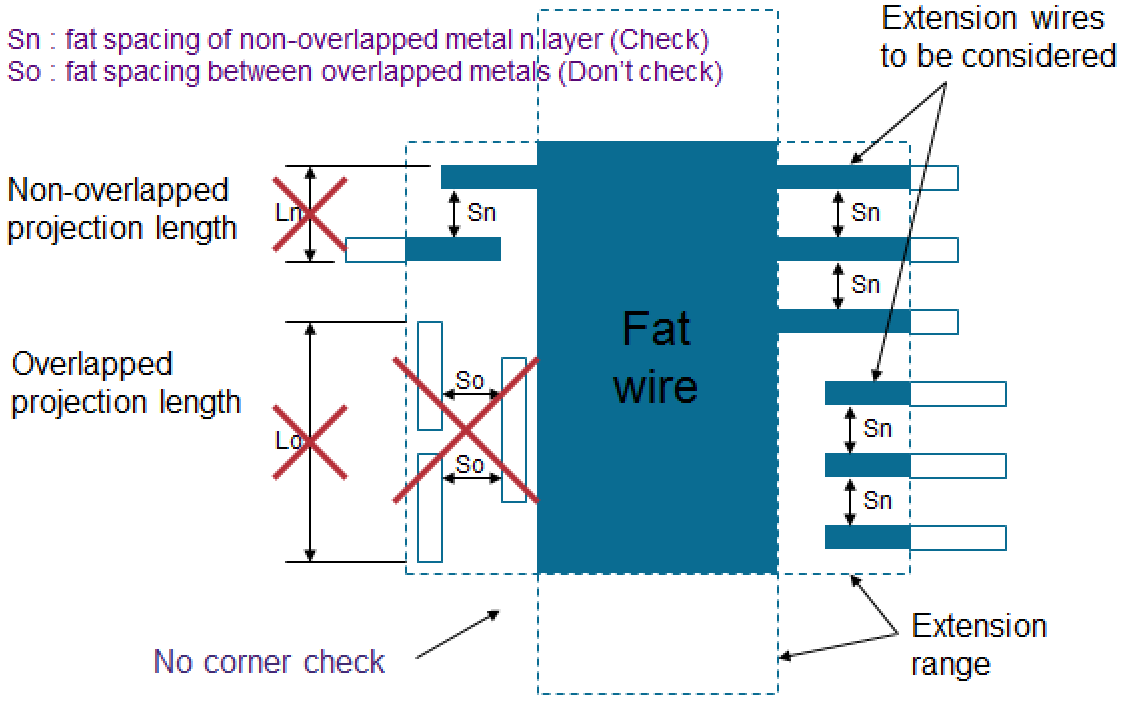

Sn : fat spacing of non-overlapped metal n layer (Check)
So : fat spacing between overlapped metals (Don't check)
Extension wires to be considered
Non-overlapped projection length
Ln
Sn
Overlapped projection length
Lo
So
So
Fat wire
No corner check
Extension range

Figure 2-192 fatWireExtensionMode = 3

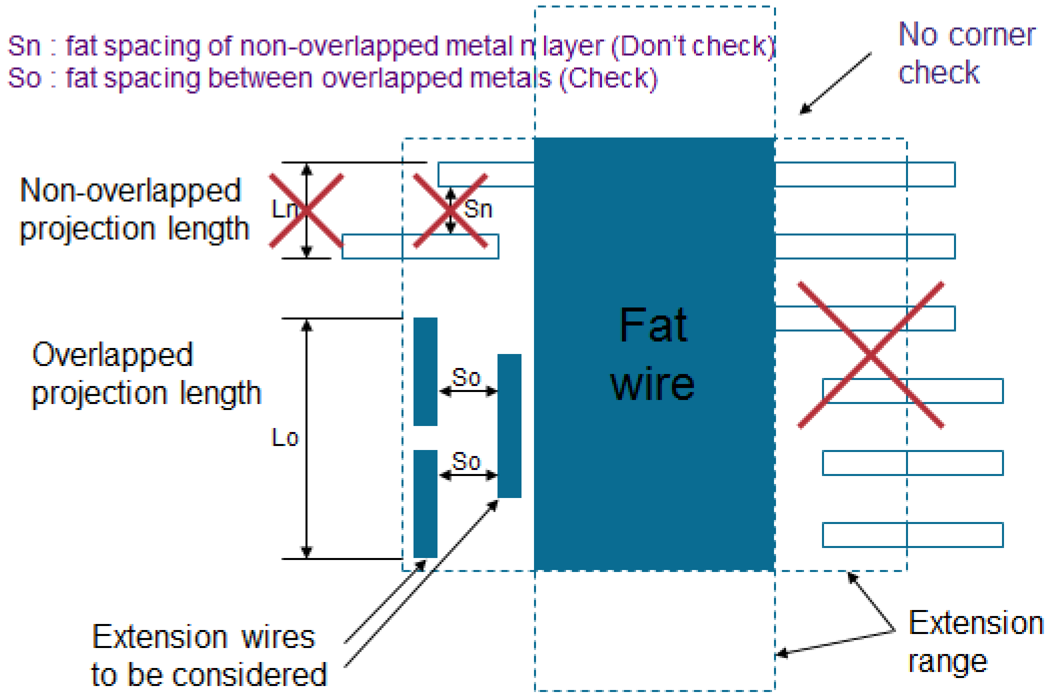
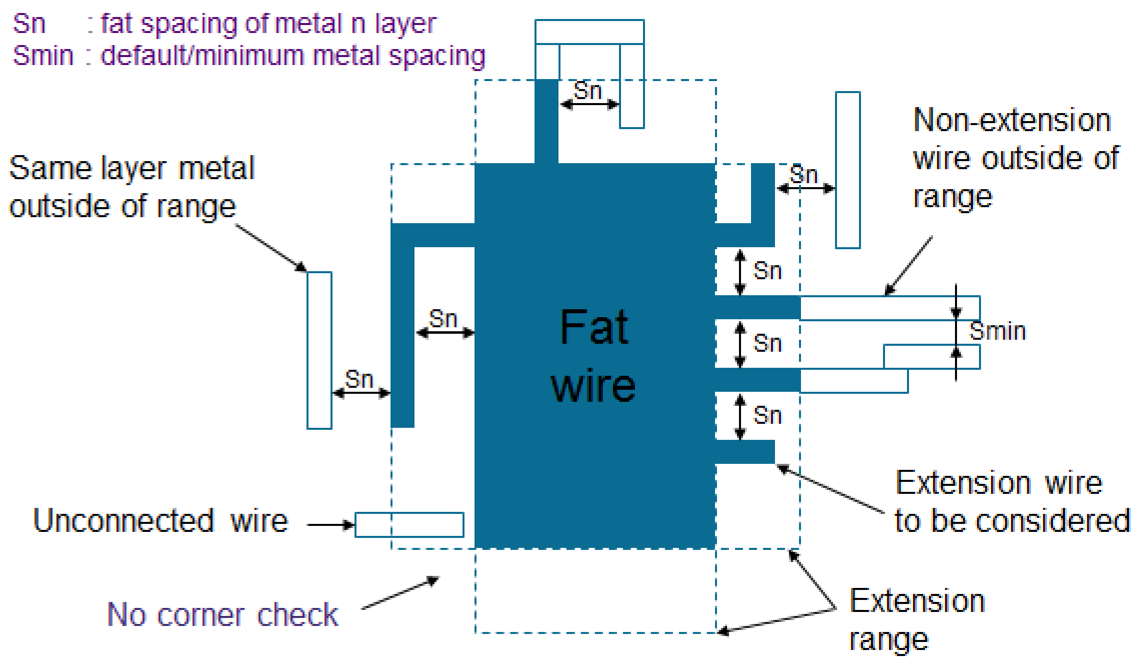

Sn : fat spacing of non-overlapped metal n layer (Don't check)
So : fat spacing between overlapped metals (Check)
No corner check
Non-overlapped projection length
Ln
Sn
Overlapped projection length
Lo
So
So
Fat wire
Extension wires to be considered
Extension range

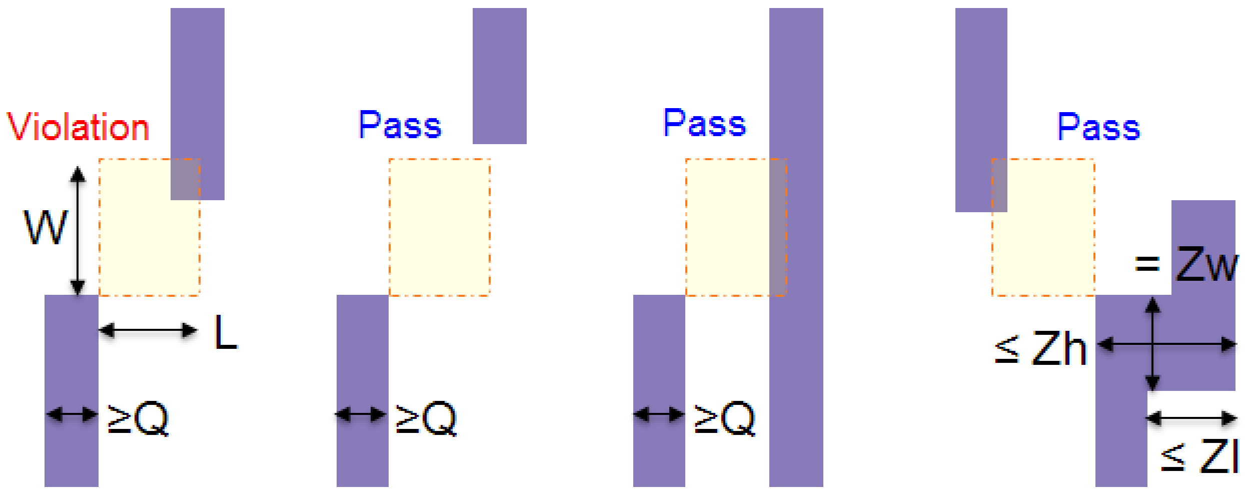
Figure 2-193 fatWireExtensionMode = 4


Sn : fat spacing of metal n layer
Smin : default/minimum metal spacing
Sn
Same layer metal outside of range
Sn
Non-extension wire outside of range
Sn
Sn
Smin
Fat wire
Sn
Sn
Sn
Extension wire to be considered
Unconnected wire
No corner check
Extension range

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 2-194. The rule does not apply for a corner belonging to a Z-shape with width equal to Zw, height no more than Zh, and longer concave-convex edge of no more than Zl.

Figure 2-194 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)
}
```

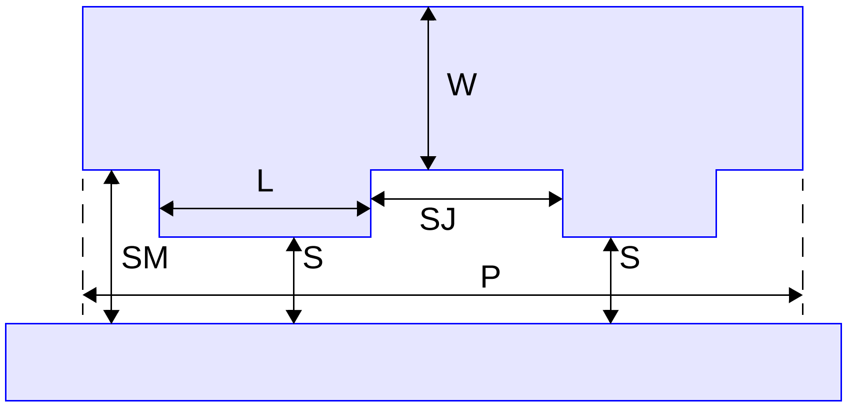
Fat Metal Jog Rule

Note:

This rule is supported only by Zroute.

The fat 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 2-195. 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 <= S <= SB).

Figure 2-195 Fat 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)
}
```

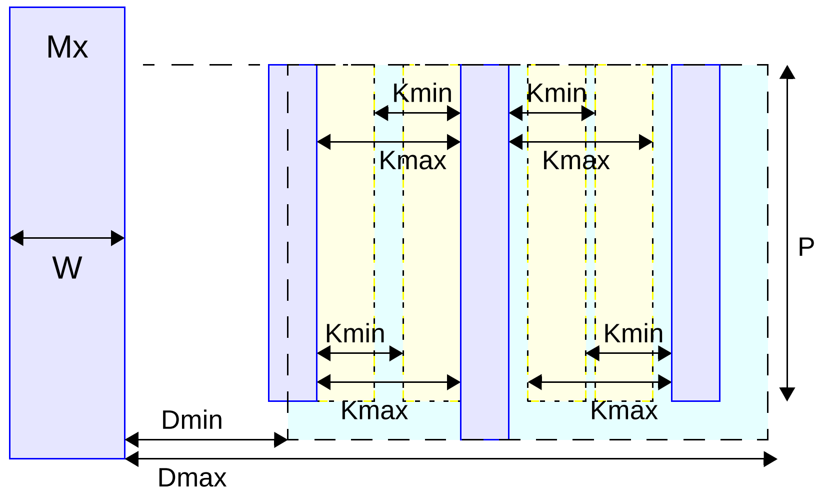
Fat Metal Forbidden Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-196, 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 Dmin and Dmax, each metal shape in that area cannot occupy the space from Kmin to Kmax away from the other metal shapes (the yellow areas).

Figure 2-196 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
}
```

Via Placement Rules

The following types of rules restrict the placement of vias:

- No Vias on Nonpreferred Direction Rule
- Via Stack Rule

- Via Array Rule
- Via Density Rule

No Vias on Nonpreferred 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.

See Also

- No Nonpreferred Direction Routing Rule

Via Stack Rule

The `stackable` attribute in the `DesignRule` section 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 a spacing rule exists between the two layers and `stackable` is set to 1, the router has the flexibility to either stack the two vias with their centers exactly aligned or enforce the minimum spacing rules that have been defined for the layers. By default, the router can violate a minimum spacing rule only by exactly aligning the via centers.

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 2-197 and Table 2-5 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 2-197 Via Stack Diagram

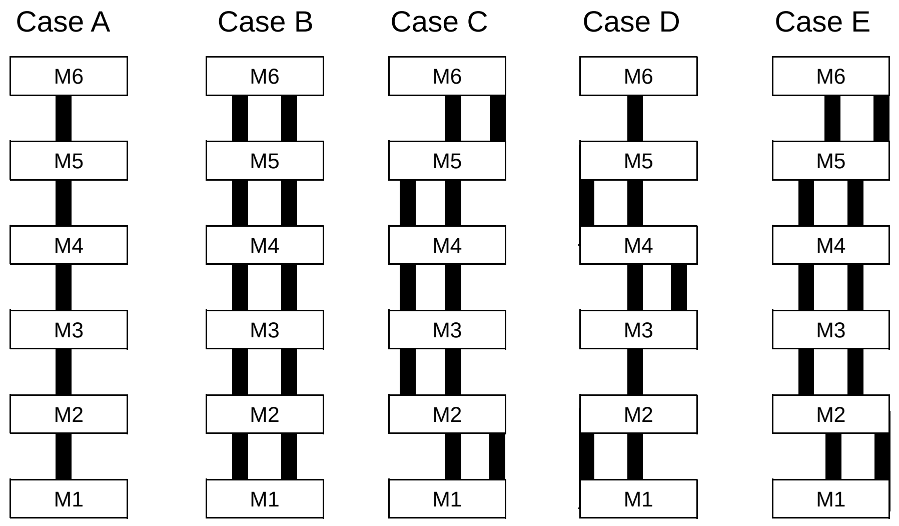


Table 2-5 Via Stack Violation Conditions

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, a via in layer VIA1 and a via in layer VIA2 are considered stacked vias when the center-to-center offset between them is no more than 0.01. Otherwise, they are not stacked vias and ordinary via-to-via spacing rules apply.

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 Rule

The via array 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.

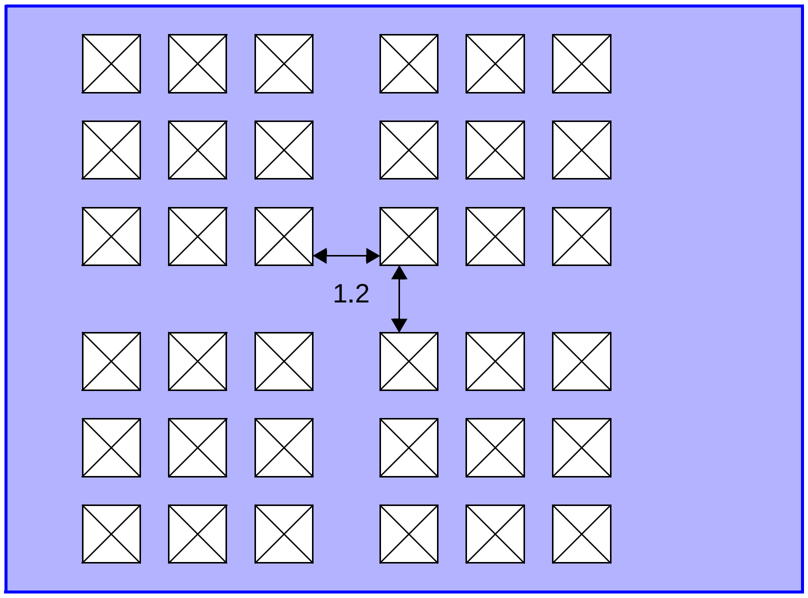
This rule is defined in the `ContactCode` section of the technology file. Each via array requires a `ContactCode` definition. To specify the via array size, use the `maxNumRows` attribute. To specify the spacing between two via arrays, use the `viaFarmSpacing` attribute.

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 2-198.

Figure 2-198 Via Array Rule



By default, the maximum number of rows and via array 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
}
```

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
- Enclosed Via Metal Minimum Area Rule
- Fat Wire Via Enclosure Rule
- Fat Metal Via Keepout Rules
- Concave Metal Corner Via Enclosure Rule
- Convex Metal Corner Via Enclosure Rule
- Via to Joint Distance Rule
- Line-End Cut Enclosure Rule
- Convex Metal Via Keepout Rule
- Concave-Convex Edge Via Enclosure Rule
- Fat Metal Contact Asymmetric Enclosure Rules

- Sparse-Dense Via Enclosure Rule
- Via-to-Jog Metal Enclosure Rule
- Interlayer Via Enclosure Rule
- Via to Concave Corners Maximum Limit Rule
- Color-Based Via Enclosure Rule
- Enclosure-Dependent Metal Spacing Rule

Minimum Enclosure Rule

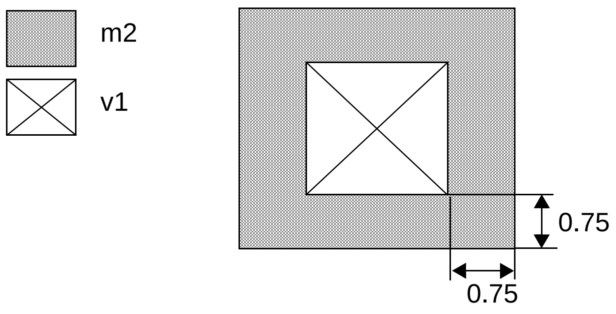
The minimum enclosure rule specifies the minimum distance at which layer1 must enclose layer2 when the two layers overlap.

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 2-199, enter

```
DesignRule {
  layer1          = "MetalX"
  layer2          = "ViaX"
  minEnclosure    = 0.05
}
```

Figure 2-199 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 2-199 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 in the IC Compiler tool, set the `ignoreMetalExtensionRule` detail route option. The option setting is stored with the cell.

```
icc_shell> set_droute_options -name ignoreMetalExtensionRule -value value
```

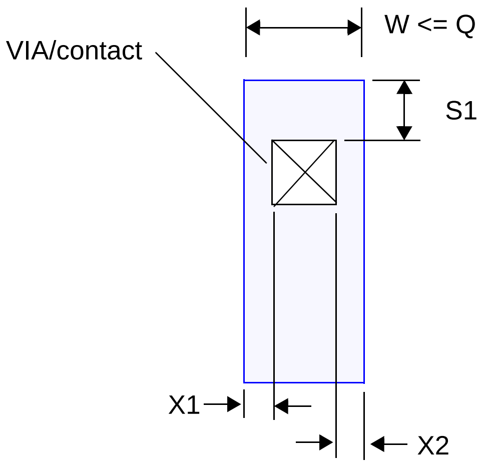
The following table lists the valid values for the `ignoreMetalExtensionRule` detail route option.

Value	Description
0	Checks the rules (the default)
1	Ignores the rules, even when they are set in the technology file

Zero-Neighbor 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 2-200 shows the attributes associated with this rule.

Figure 2-200 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)
}
```

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 nm <= X1/X2 < 0.015 nm, S1 is greater than 0.08 nm for the via
- If 0.015 nm <= X1/X2 < 0.025 nm, S1 is greater than 0.06 nm for the via

If the metal overlapping the via has two adjacent short edges of different lengths no more than Q, by default, the rule considers only the overlap of the shorter edge. To have the rule check the overlap of the longer edge as well, add the following line:

```
  endOfLineEncLongEdgeIncluded = 1
```

By default, the rule applies to both via arrays and single vias. To waive the rule for via arrays, add the following line:

```
  endOfLineEncViaArrayExcluded = 1
```

T-Shape Metal Extension for End-of-Line Via Enclosure Rule

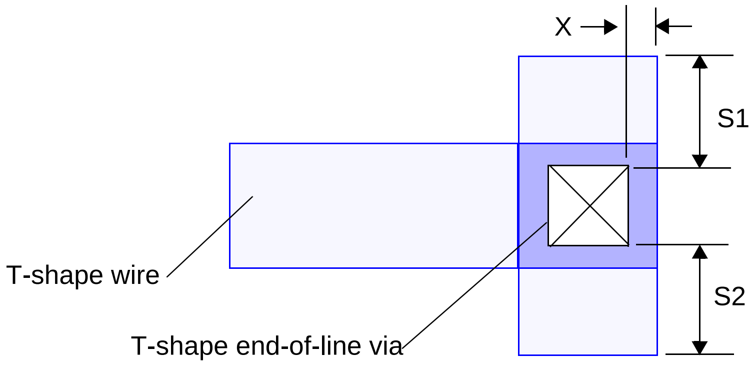
Use the `endOfLineTShapeEncTblSize`, `endOfLineTShapeCornerMinSpacing`, `endOfLineTShapeEncSideThreshold`, and `endOfLineTShapeEncTbl` attributes to specify the T-shape end-of-line via enclosure rule. Define these attributes in the `DesignRule` section of the technology file.

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 2-201 shows the attributes associated with the metal extension for end-of-line via enclosure rule.

Figure 2-201 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

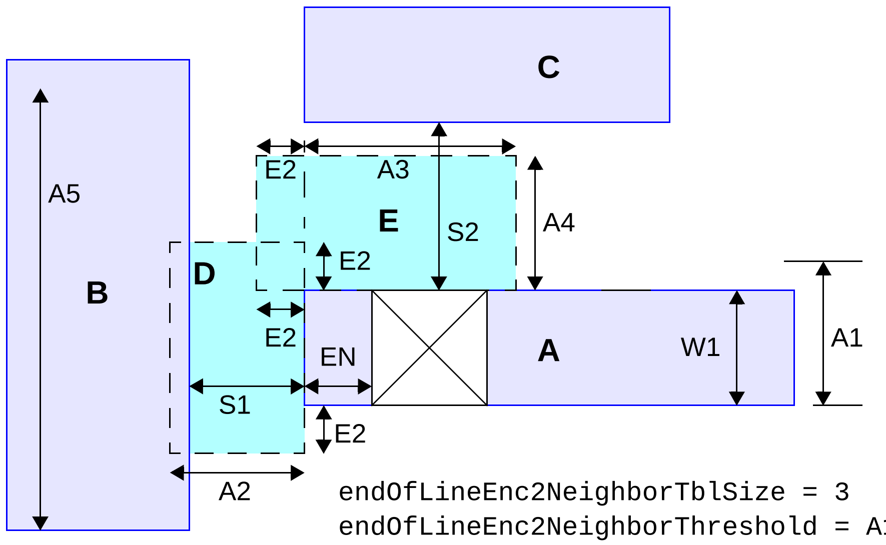
Note:

This rule is supported only by Zroute. For a similar rule supported by the classic router, see Enhanced Dense End-of-Line Spacing Rule.

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.

In Figure 2-202, if the metal width W1 is less than or equal to the width threshold A1, 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 2-202 Two-Neighbor End-of-Line Via Enclosure Rule



```
endOfLineEnc2NeighborTblSize = 3
endOfLineEnc2NeighborThreshold = A1
endOfLineEnc2NeighborCornerKeepoutWidth = E2
endOfLineEnc2NeighborSideKeepoutLength = A3
endOfLineEnc2NeighborSideMinSpacing = A4
endOfLineEnc2NeighborTbl = (EN1, EN2, EN3)
endOfLineEnc2NeighborSpacingTbl = (A21, A22, A23)
endOfLineEnc2NeighborViaArrayExcludedTbl=(1, 1, 1)
endOfLineEnc2NeighborWireMinThreshold = A5
```

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:

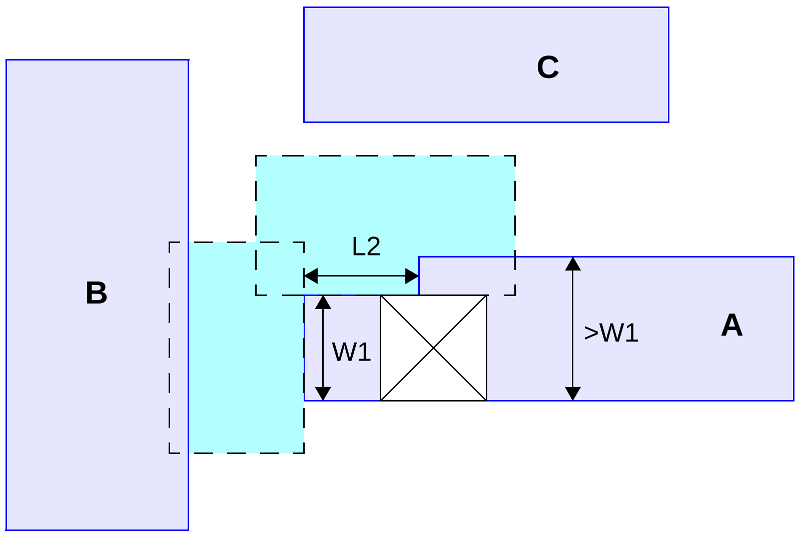
```
DesignRule {
  layer1 = "MetalX"
  layer2 = "VIA1"
  endOfLineEnc2NeighborTblSize = 2
  endOfLineEnc2NeighborThreshold = 0.07
  endOfLineEnc2NeighborCornerKeepoutWidth = 0.03
  endOfLineEnc2NeighborSideKeepoutLength = 0.08
  endOfLineEnc2NeighborSideMinSpacing = 0.068
  endOfLineEnc2NeighborTbl = (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 shapes 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 2-203.

Figure 2-203 Two-Neighbor End-of-Line Via With Adjacent Short Edge



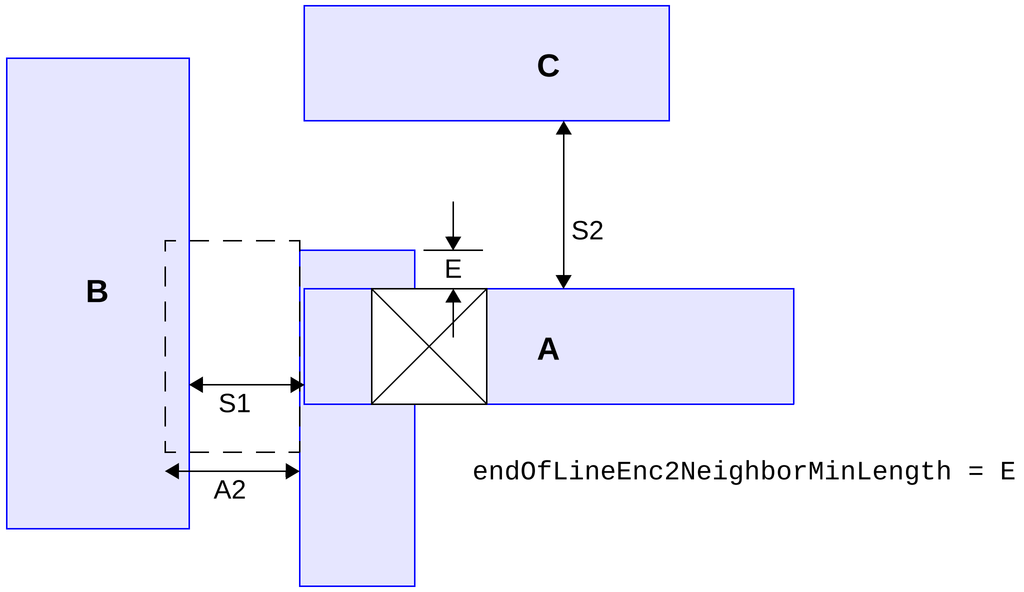
By default, the adjacent edge L2 is considered short if it is less than or equal to the `minWidth` value defined for the metal. To define a different L2 threshold for the two-neighbor end-of-line via enclosure rule, use the following attribute:

```
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 2-204.

Figure 2-204 Two-Neighbor End-of-Line Via Enclosure On Side

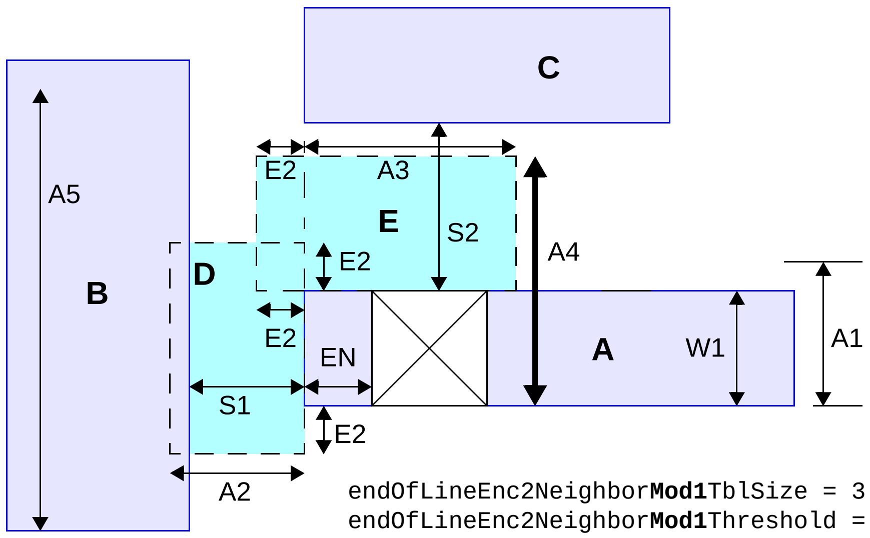


For example,

```
DesignRule {
  layer1 = "MetalX"
  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 2-205 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 2-205 Modified Two-Neighbor End-of-Line Via Enclosure Rule



```
endOfLineEnc2NeighborMod1TblSize = 3
endOfLineEnc2NeighborMod1Threshold = A1
endOfLineEnc2NeighborMod1CornerKeepoutWidth = E2
endOfLineEnc2NeighborMod1SideKeepoutLength = A3
endOfLineEnc2NeighborMod1SideKeepoutWidth = A4
endOfLineEnc2NeighborMod1Tbl = (EN1, EN2, EN3)
endOfLineEnc2NeighborMod1SpacingTbl = (A21, A22, A23)
endOfLineEnc2NeighborMod1ViaArrayExcludedTbl = (0,0,0)
endOfLineEnc2NeighborMod1WireMinThreshold = A5
```

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
}
```

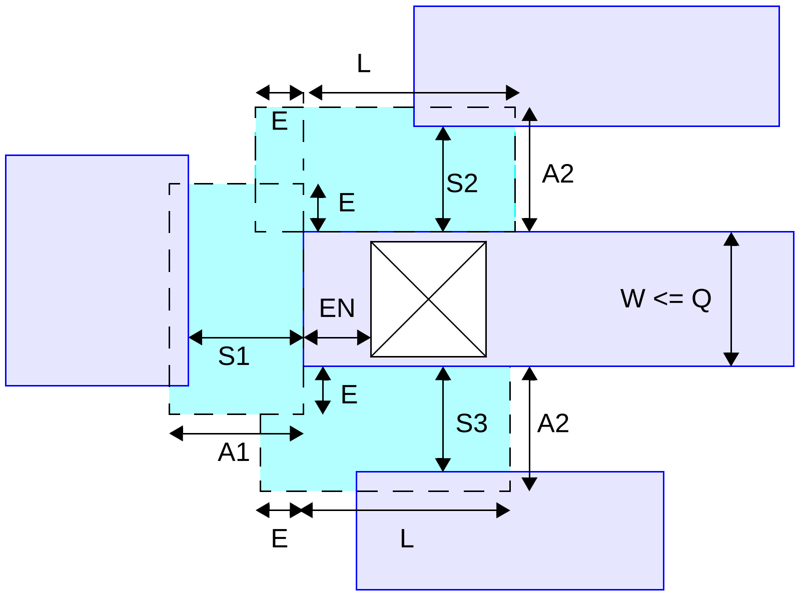
Three-Neighbor End-of-Line Via Enclosure Rule

Note:
  This rule is supported only by Zroute.

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.

In Figure 2-206, 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, the end of the wire must enclose the via by at least the distance EN.

Figure 2-206 Three-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, 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.

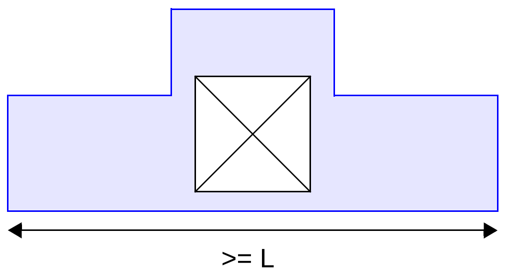
Here is an example of the syntax used for this rule:

```
DesignRule {
  layer1 = "MetalX"
  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 2-207.

Figure 2-207 Enclosed Via Metal 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.

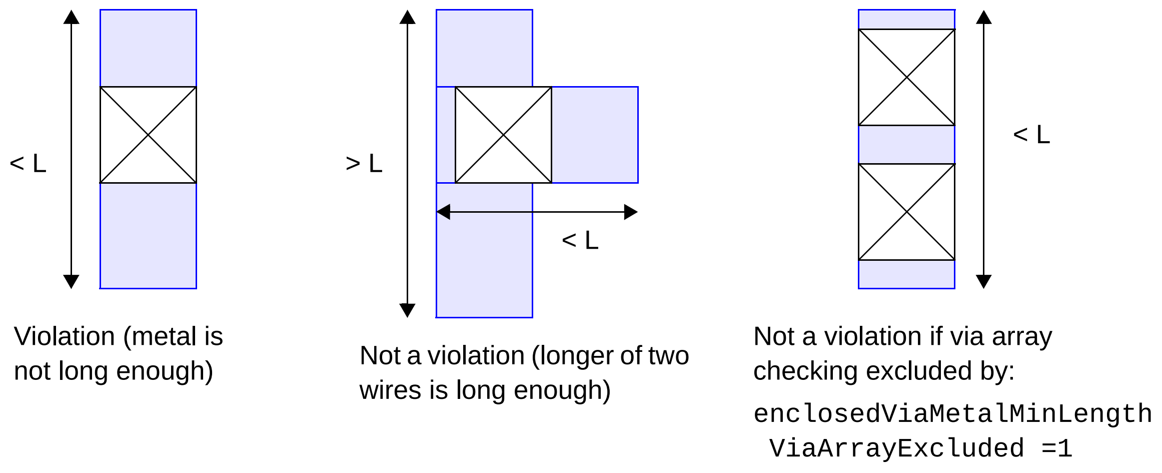
This is the syntax of the rule:

```
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 2-207 demonstrate how the rule works.

Figure 2-208 Enclosed Via Metal Minimum Length Rule With Via Array Exclusion

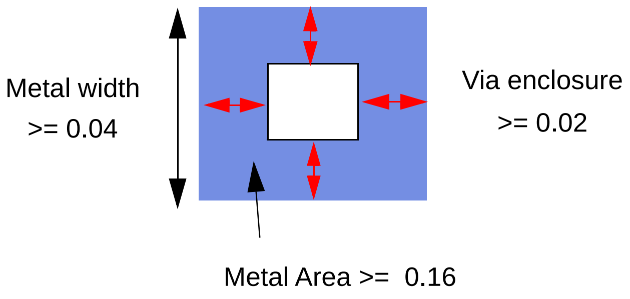

< L
Violation (metal is not long enough)
> L
< L
Not a violation (longer of two wires is long enough)
< L
Not a violation if via array checking excluded by:
enclosedViaMetalMinLength
ViaArrayExcluded =1

Enclosed Via Metal Minimum Area Rule

Note:
This rule is supported only by Zroute.

The enclosed via metal minimum area rule specifies the minimum area for a metal shape that encloses a via under certain conditions. Figure 2-209 shows an example of the rule.

Figure 2-209 Via Enclosure-Based Minimum Area Rule



If the width of the metal shape is at least the `encMetalWidthThreshold`, and the via enclosure on each of the four sides is at least the `encMetalMinEnclosure`, the area of the metal shape must be at least `encMetalMinArea`. An example of the rule definition is as follows:

```
DesignRule {
   layer1                  = "Mx"
   layer2                  = "Vx"
   ...
   encMetalWidthThreshold = 0.04
   encMetalMinEnclosure   = 0.02
   encMetalMinArea        = 0.16
}
```

Fat Wire Via Enclosure Rule

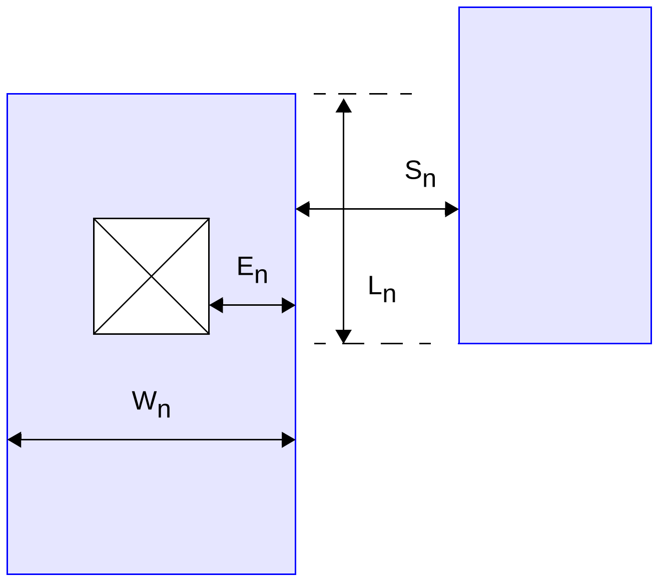
The fat wire via enclosure rule specifies the minimum amount of fat metal overlap over a via for the metal layer either above or below the via. The minimum overlap requirement depends on the width of the fat metal enclosing the via, the spacing to any nearby metal, and the parallel length of that nearby 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 2-210 shows how the minimum enclosure depends on the rule attributes.

Figure 2-210 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)
}
```

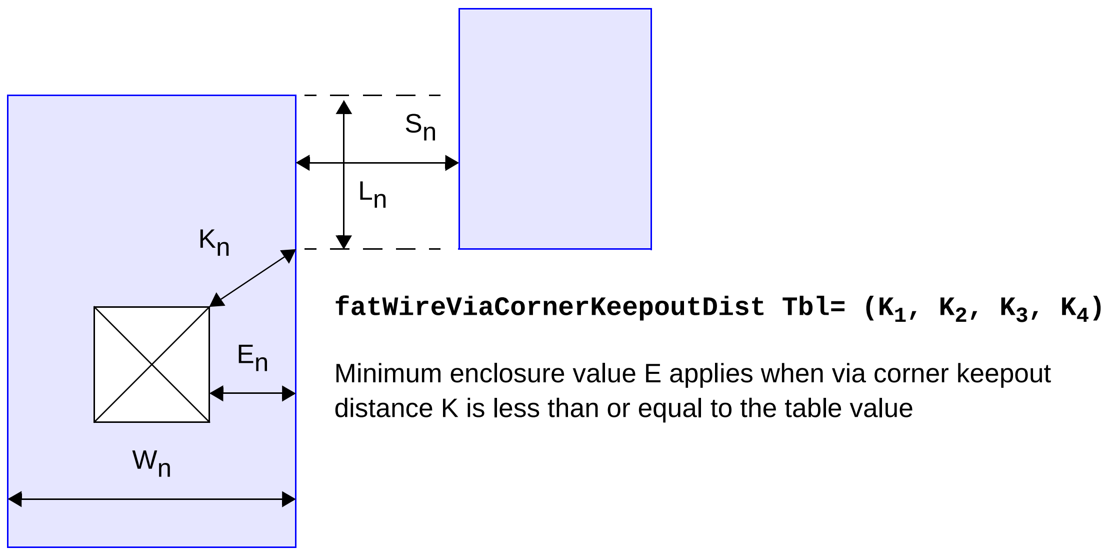
In this example, if the fat wire width W is greater than or equal to 0.055 but less than 0.060, the parallel length L of the nearby metal is greater than 0.12, and the spacing S to the nearby metal is less than 0.062, the minimum enclosure E is 0.008. The last attribute, `fatWireViaArrayExcludedTbl`, is set to 1 to waive the rule for a double via for the corresponding set of values for W, L, S, and E.

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 2-211.

Figure 2-211 Fat Wire Via Enclosure Rule With Corner Keepout Distance

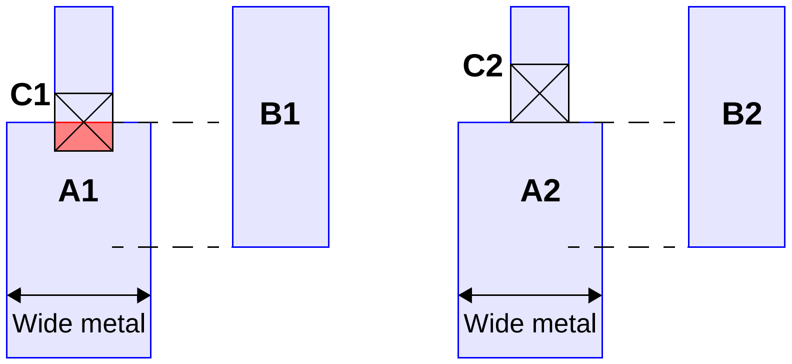


fatWireViaCornerKeepoutDist Tbl= (K_1, K_2, K_3, K_4)

Minimum enclosure value E applies when via corner keepout distance K is less than or equal to the table value

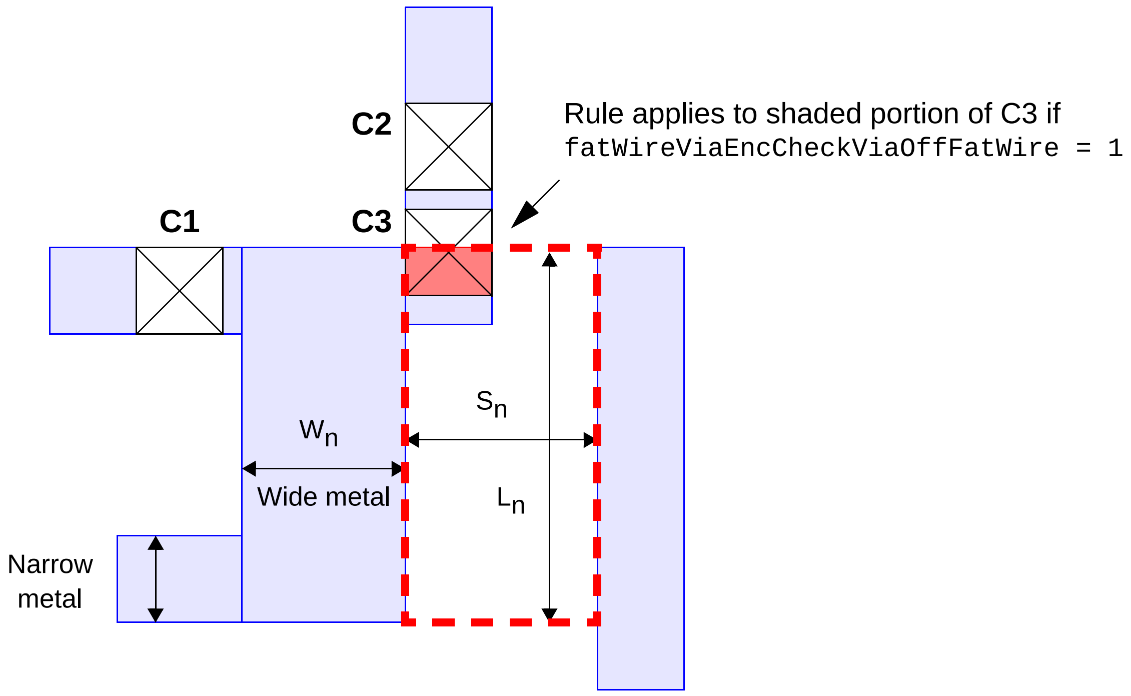
In Figure 2-212, the corner keepout attribute is not used, and via C1 is half inside and half outside the parallel overlap region of the wide metal, so the fat wire via enclosure rule applies only to the lower portion of the via, highlighted in red in the figure. Via C2 is entirely outside the parallel overlap region of the wide metal, so the fat wire via enclosure rule does not apply at all to that via.

Figure 2-212 Via Inside and Outside Fat Metal Parallel Length Region



In Figure 2-213, 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 2-213 Exception to Via Outside Fat Metal Parallel Length Region


C2
Rule applies to shaded portion of C3 if
fatWireViaEncCheckViaOffFatWire = 1
C1
C3
Sn
Wn
Wide metal
Ln
Narrow metal

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.

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 the 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:

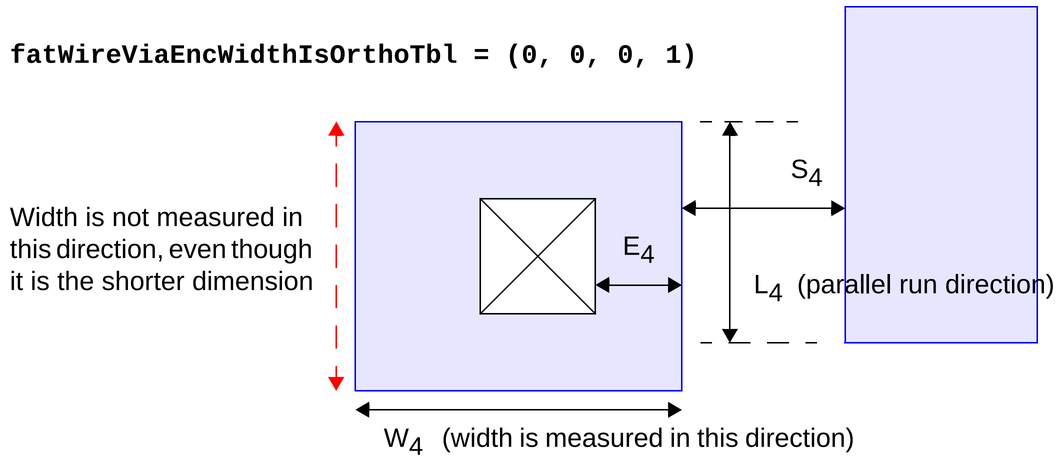
```
DesignRule {
  layer1                                 = "MetalX"
  layer2                                 = "ViaX"
  fatWireViaEncTblSize                   = 4
  fatWireViaEncWidthThresholdTbl         = (W1, W2, W3, W4)
  fatWireViaEncParallelLengthThresholdTbl  = (L1, L2, L3, L4)
  fatWireViaEncMaxSpacingThresholdTbl    = (S1, S2, S3, S4)
  fatWireViaEnclosureTbl                 = (E1, E2, E3, E4)
  fatWireViaArrayExcludedTbl             = (0, 0, 1, 1)
  fatWireViaEncWidthIsOrthoTbl           = (0, 0, 1, 1)
}
```

For example,

```
DesignRule {
  layer1                                 = "MetalX"
  layer2                                 = "ViaX"
  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 2-210.

Figure 2-214 Fat Wire Orthogonal Width Measurement


fatWireViaEncWidthIsOrthoTbl = (0, 0, 0, 1)
Width is not measured in this direction, even though it is the shorter dimension
E_4
S_4
L_4 (parallel run direction)
W_4 (width is measured in this direction)

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 `fatWireViaEncMaxWidthThresholdTbl` and `fatWireViaEncMinWidthThresholdTbl` attributes replace the single `fatWireViaEncWidthThresholdTbl` attribute.

For example,

```
DesignRule {
  layer1                                 = "MetalX"
  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 `fatWireViaEncWidthThresholdTbl` attribute, the width ranges are:

0.0–W1, W1–W2, W2–W3, W3–W4

When you use the `fatWireViaEncMaxWidthThresholdTbl` and `fatWireViaEncMinWidthThresholdTbl` attributes, the width ranges are:

W1–W1’, W2–W2’, W3–W3’, W4–W4’

The list of values in the `fatWireViaEncMinWidthThresholdTbl` attribute must be monotonically increasing; each value in the list must be larger than one before it. This restriction does not apply to the `fatWireViaEncMaxWidthThresholdTbl` 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                                  = "MetalX"
  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 Via Keepout Rules

The fat metal via keepout rules are specified in the `DesignRule` section of the technology file by the following attributes:

```
fatWireViaKeepoutTblSize
fatWireViaKeepoutWidthThreshold
fatWireViaKeepoutMinSize
fatWireViaKeepoutEnclosure
fatWireViaKeepoutParallelLengthThreshold
fatWireViaKeepoutMaxSpacingThreshold
```

You can use these attributes to define the following rules:

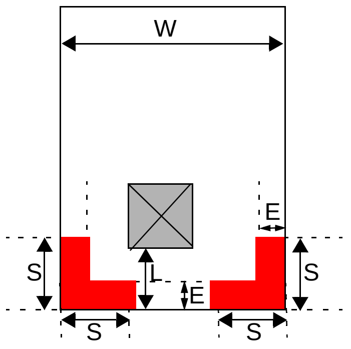
- The fat metal via keepout area rule
- The via enclosure rule
- The poly contact enclosure rule

Note:
For any new technology file, you should use the newer, more advanced rules described in Fat Wire Via Enclosure Rule. Zroute does not support usage of both the older and newer 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 2-215). 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 2-215 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                          = "ViaX"
  layer2                          = "MetalX"
  fatWireViaKeepoutTblSize        = 2
  fatWireViaKeepoutWidthThreshold = (0.5,1.0)
  fatWireViaKeepoutMinSize        = (0.18,0.36)
  fatWireViaKeepoutEnclosure      = (0.05,0.10)
}

DesignRule     {
  layer1                          = "ViaX"
  layer2                          = "MetalX"
  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.

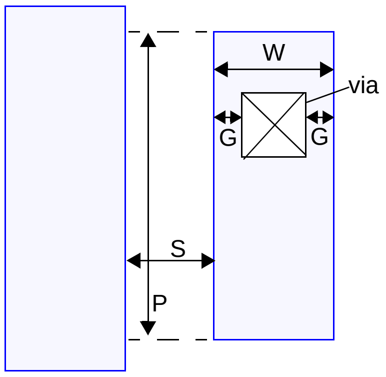
- The parallel length, P, between the enclosing metal segment and the neighboring metal segment is greater than the value specified in the `fatWireViaKeepoutParallelLengthThreshold` attribute and the parallel run covers the poly contact edge partially or completely.
- The enclosing metal does not contain 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 2-216 shows the measurements involved in the fat via enclosure rule.

Figure 2-216 Fat 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 greater than 0.27 microns

```
DesignRule {
  layer1                                  = "MetalX"
  layer2                                  = "ViaX"
  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                                  = "MetalX"
  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

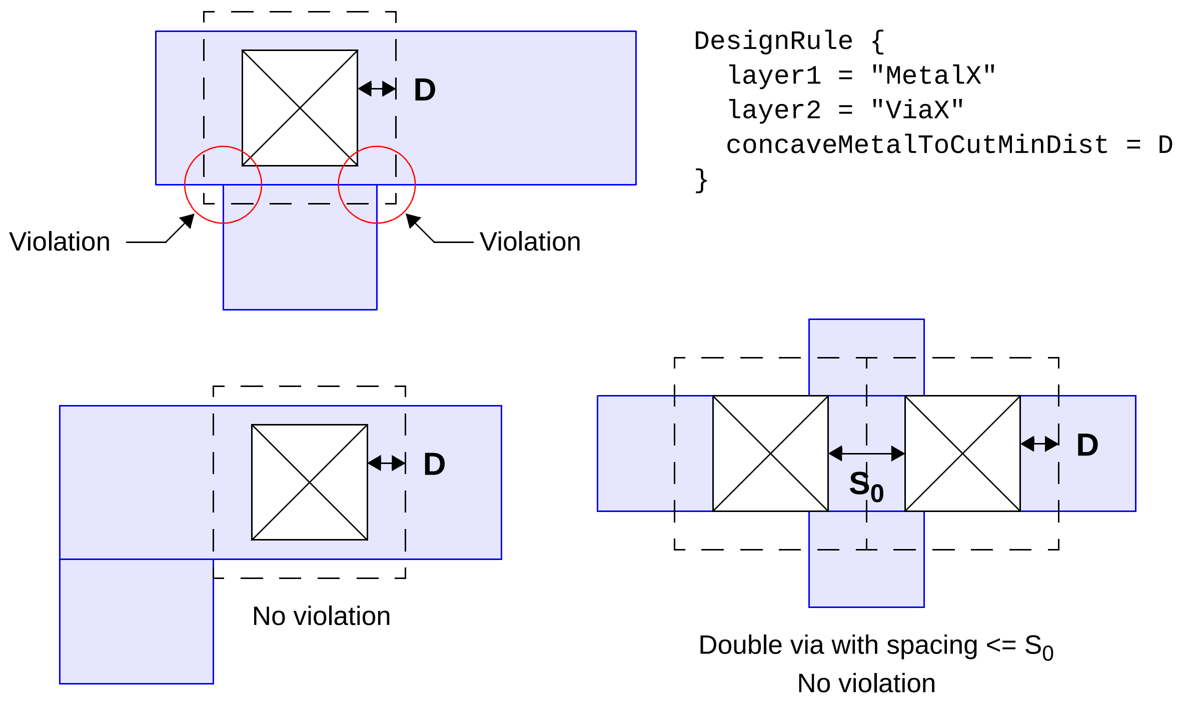
Note:

This rule is supported only by Zroute.

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 2-217, 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 S0, the `minCutSpacing` value specified in the `ContactCode` section of the technology file.

Figure 2-217 Concave Metal Corner Via Enclosure Rule



```
DesignRule {
  layer1 = "MetalX"
  layer2 = "ViaX"
  concaveMetalToCutMinDist = D
}
```

Here is an example of the syntax used for this rule:

```
DesignRule {
  layer1 = "MetalX"
  layer2 = "ViaX"
  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.

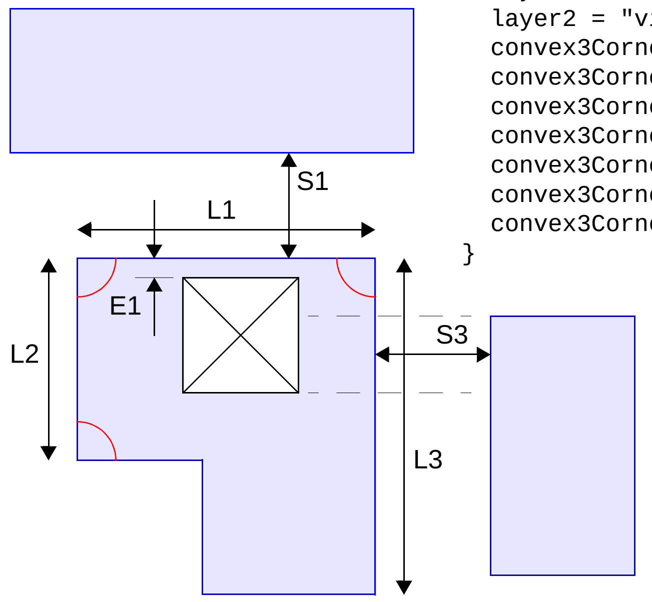
Convex Metal Corner Via Enclosure Rule

Note:
This rule is supported only by Zroute.

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 2-218, 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 2-218 Convex Metal Corner Via Enclosure and Spacing Rule



```
Layer DesignRule {
  layer1 = "metal1"
  layer2 = "via1"
  convex3CornerEncCutTblSize                    = 1
  convex3CornerEncCutNameTbl                    = (Vsm)
  convex3CornerEncMaxEdgeLengthThreshold          = LA
  convex3CornerEncAdjacentMaxEdgeLengthThreshold = LB
  convex3CornerEncAdjacentMinEdgeLengthThreshold = LC
  convex3CornerEncMaxSpacingThreshold             = S
  convex3CornerEnclosure                          = E
}
```

if L1 <= LA
  L2 <= LB
  L3 >= LC
  S1 <= S
  S3 <= S
then E1 >= E

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 .

Via to Joint Distance Rule

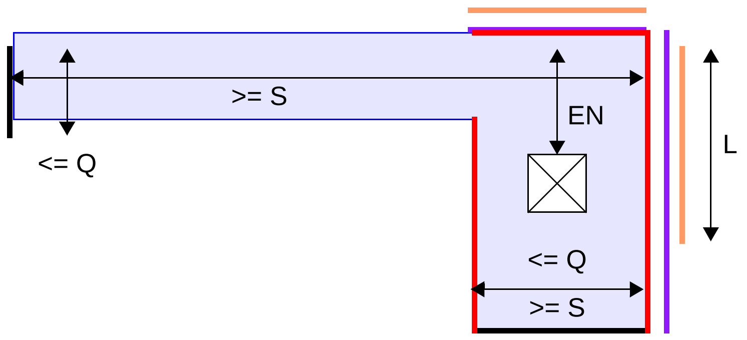
Note:

This rule is supported only by Zroute.

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 2-219.

Figure 2-219 Via to Joint Distance Rule



⬛ End-of-line

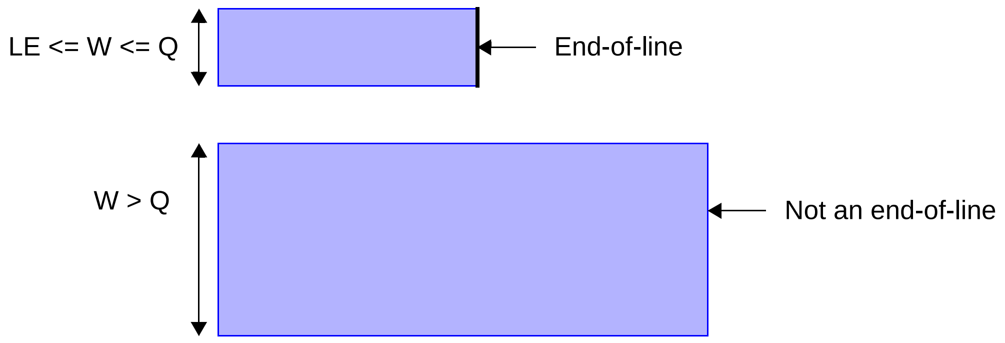
🟥 Joint (not end-of line and span >= S)

🟪 Joint that meets another joint to form a convex corner

🟧 Portion of joint at convex corner checked for minimum enclosure EN

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 2-220, the narrower line has an end-of-line and the wider wire does not.

Figure 2-220 Definition of an End-of-Line for Joint Distance Rule

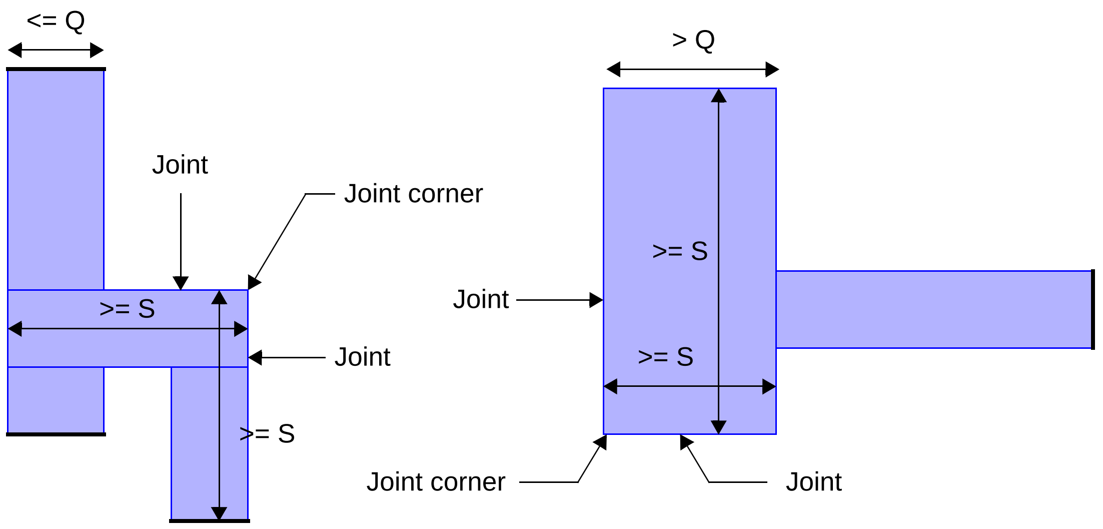


This is the syntax for defining an end-of-line for the via to joint distance rule:

```
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 2-221.

Figure 2-221 Joint Corner Examples

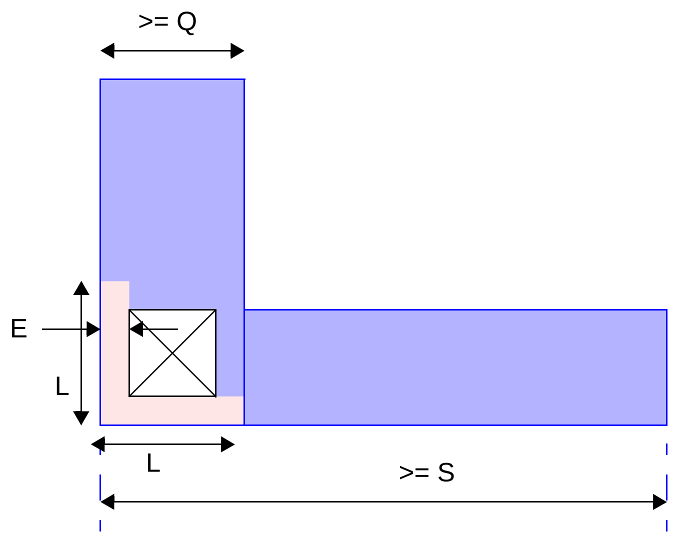


This is the syntax for defining the span S for the definition of a joint:

```
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 2-222, the rule checks for the minimum enclosure in the pink area.

Figure 2-222 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 `Vsm` in this example, must be defined in the `Layer` section in for the via layer earlier in the technology file.

For example,

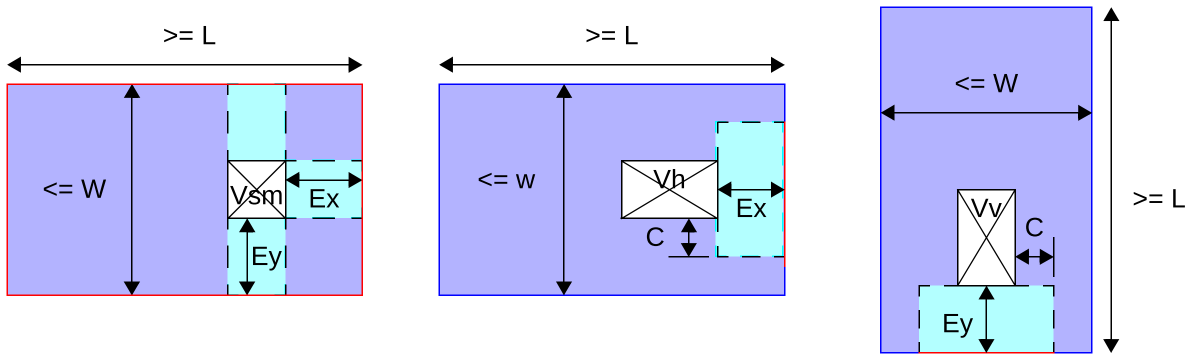
```
DesignRule {
   ...
   jointWireViaEndWidthMaxThreshold   = 0.24
   jointWireViaEndMinLength           = 0.05
   jointWireViaMinSpanThreshold       = 0.08
   jointWireViaMaxEdgeLengthThreshold = 0.15
   jointWireViaMinEnclosure           = 0.04
   ...
```

Line-End Cut Enclosure Rule

Note:
    This rule is supported only by Zroute.

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 2-223.

Figure 2-223 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 `Vsm` via is 0.021 in the x-direction and y-direction. For a `Vv` 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 `Vh` 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

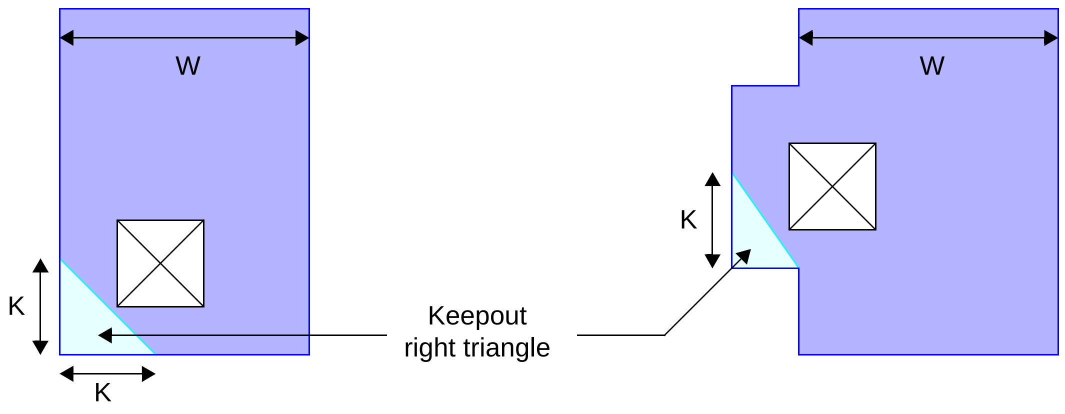
Note:
This rule is supported only by Zroute.

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 2-224. 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 2-224 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:

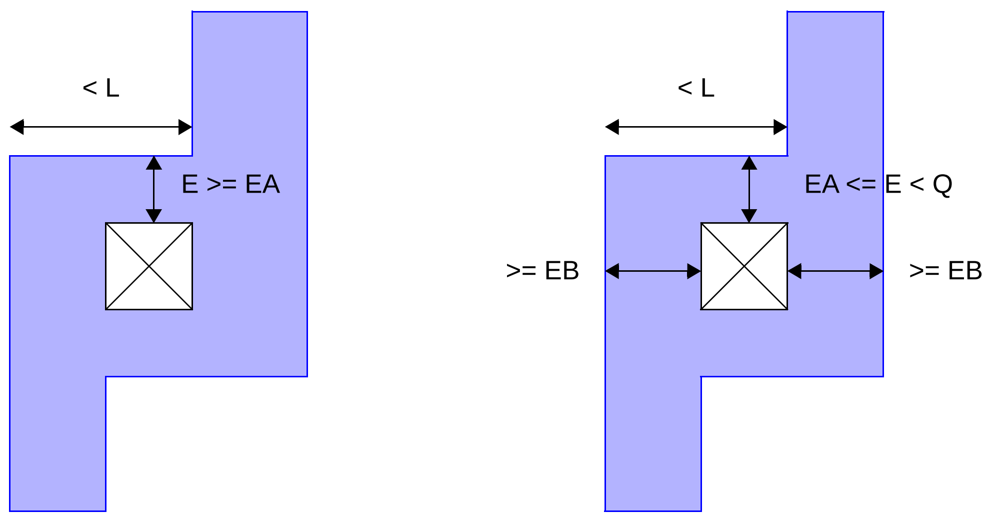
```
DesignRule {
  layer1                                = "metal1"
  layer2                                = "via1"
  convexMetalCutTblSize                 = 3
  convexMetalCutNameTbl                 = (Vsm,Vh,Vv)
  convexMetalEndOfLineMaxWidthThreshold = 0.11
  convexMetalEndOfLineMinLength         = 0.03
  convexMetalCutKeepoutLengthTbl        = (0.30,0.32,0.32)
}
```

Concave-Convex Edge Via Enclosure Rule

Note:
    This rule is supported only by Zroute.

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 2-225.

Figure 2-225 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 2-225.

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 Metal Contact Asymmetric Enclosure Rules

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.

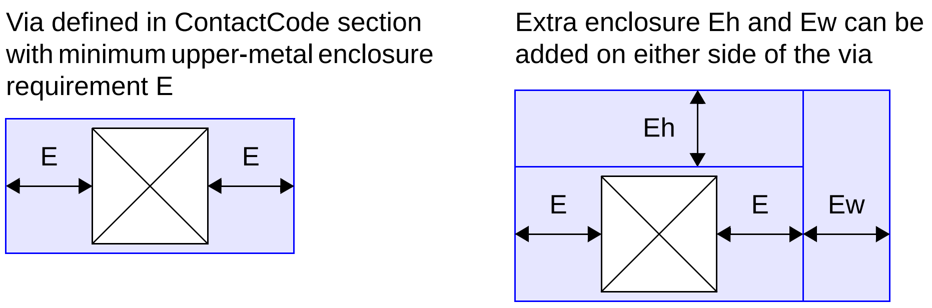
For advanced geometries, you can specify the additional upper-metal enclosure requirement either as an additional distance beyond the minimum enclosure specified in the `ContactCode` section in the height and width directions, or as a total span of metal covering the via in the height and width directions. 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 2-226, 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 2-226 Fat Metal Contact Asymmetric Extra Enclosure Rule


Via defined in ContactCode section with minimum upper-metal enclosure requirement E
E
E
Extra enclosure Eh and Ew can be added on either side of the via
Eh
E
E
Ew

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)
}
```

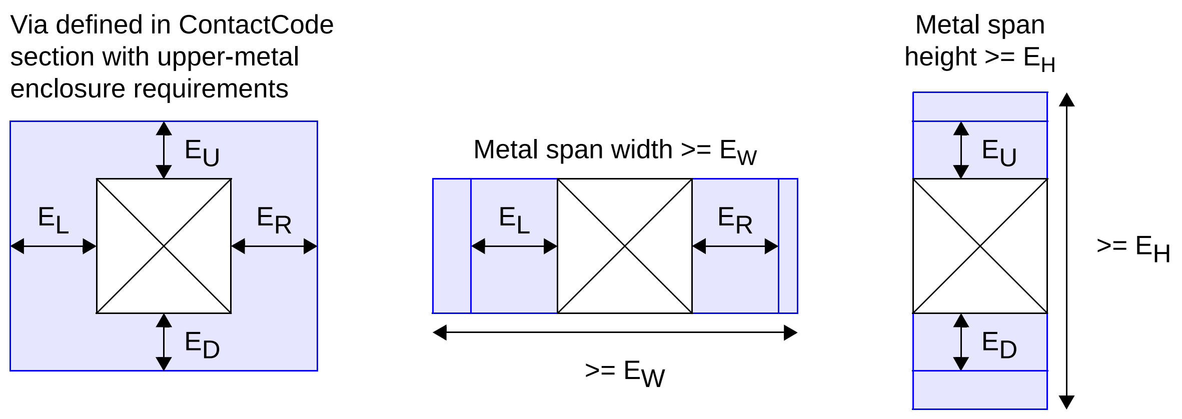
For example,

```
  ...
  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 2-227.

Figure 2-227 Fat Metal Contact Asymmetric Total Enclosure Rule


Via defined in ContactCode section with upper-metal enclosure requirements
E_U
E_L
E_R
E_D
Metal span width >= E_W
E_L
E_R
>= E_W
Metal span height >= E_H
E_U
>= E_H
E_D

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). 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*.

Similarly, for a rule that applies to the lower-metal enclosure around the via:

```
Layer "VIAX" {
  asymLowerEnclosureTblSize  = 1 # number of asymLowerEnclosureTbl rows
  asymLowerEnclosureTbl  = (cutName, xleftEncWidth, xrightEncWidth,
                           xbotEncHeight, xtopEncHeight, botMetalWidth)
```

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 implementation tool internally converts the rule to standard syntax. To verify the accuracy of the converted rules, you can write them out as a new technology file, as follows:

IC Compiler II:

```
% setenv write_converted_tf_syntax true
...
% icc2_shell
...
icc2_shell> write_tech_file my_tech.tf
...
```

IC Compiler:

```
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

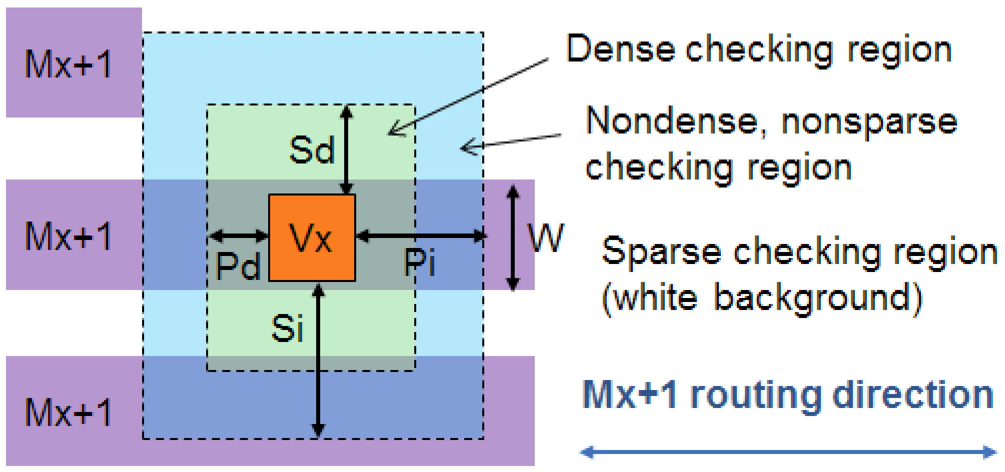
Note:
This rule is supported only by Zroute.

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.

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 vial falls into certain range (Wmin, Wmax) and its neighboring metal shapes form a sparse-dense arrangement on two sides, as demonstrated in Figure 2-228.

Figure 2-228 Sparse and Dense Via Upper and Lower Metal Enclosure Rule Regions


Mx+1
Mx+1
Mx+1
Sd
Vx
Pd
Pi
Si
W
Dense checking region
Nondense, nonsparse checking region
Sparse checking region (white background)
Mx+1 routing direction

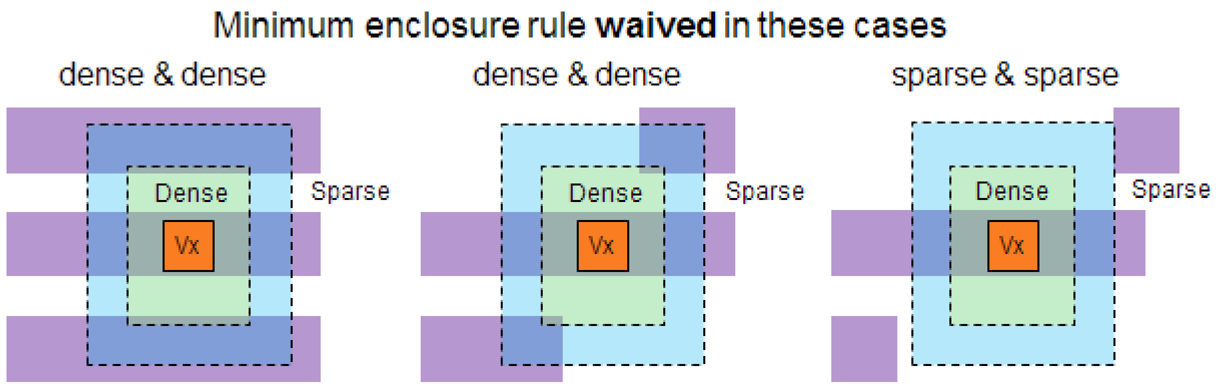
The rule defines the following regions for checking the presence of nearby metal:

- The dense region is a rectangular area expanded from edges of the via by a distance Pd in the upper metal’s routing direction and Sd 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 Pi in the upper metal’s routing direction and Si 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 2-229.

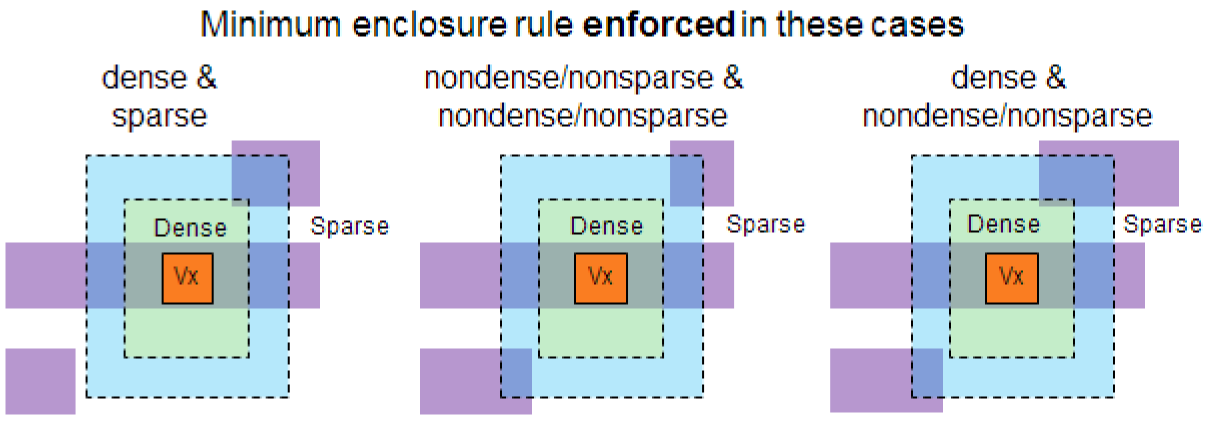
Figure 2-229 Rule Waived for Dense-Dense or Sparse-Sparse Nearby Metal on Two Sides



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 2-230.

Figure 2-230 Rule Enforced for Other Types of Nearby Metal Occupancy on Two Sides


Minimum enclosure rule **enforced** in these cases
dense & sparse
nondense/nonsparse & nondense/nonsparse
dense & nondense/nonsparse
Dense
Sparse
Vx
Dense
Sparse
Vx
Dense
Sparse
Vx

This is the syntax of the rule:

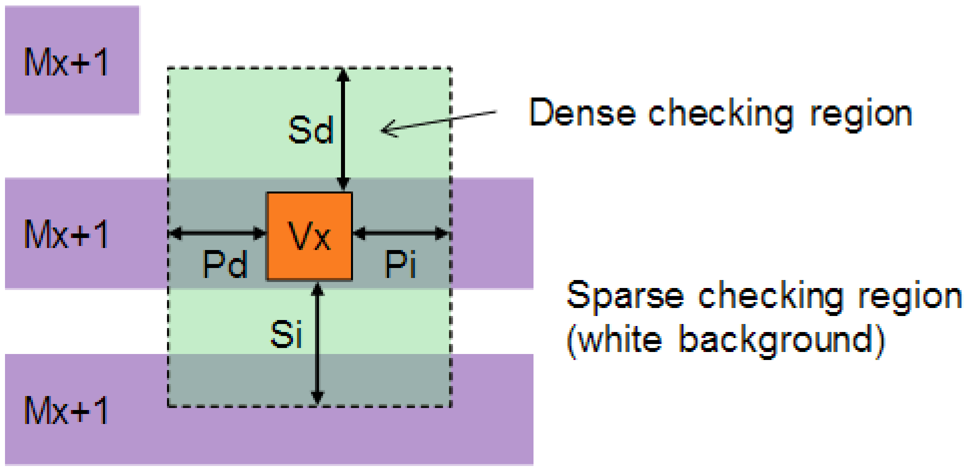
```
DesignRule  {
  layer1                                      = "Mx"
  layer2                                      = "Vx"
  fatWireViaEncCutTblSize                     = 1
  fatWireViaEncCutNameTbl                     = (Vs)
  fatWireViaEncTblSize                        = 1
  fatWireViaEncMinWidthThresholdTbl           = (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

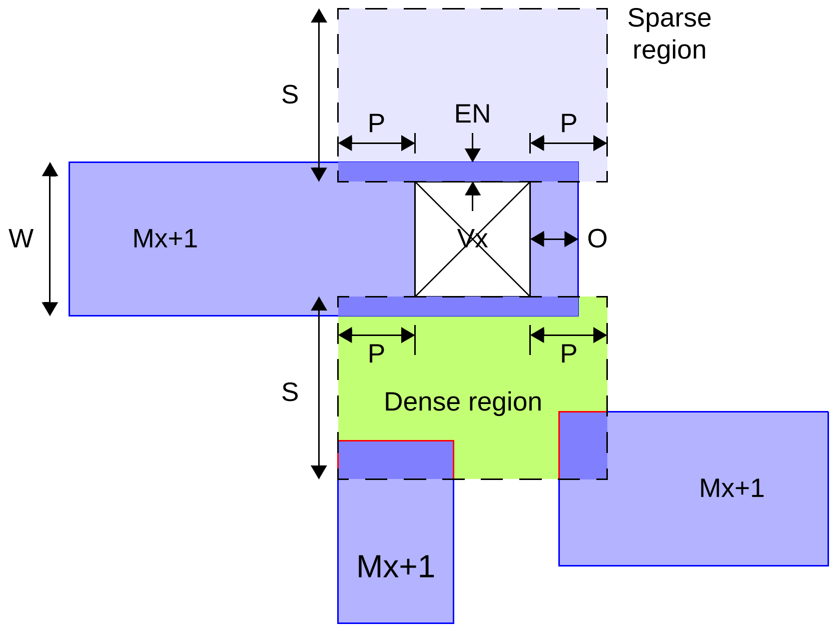
In the case where Sd and Si are the same, and Pd and Pi are the same, the dense region (green) and sparse region (white) are exactly complementary, and there is no blue region between them, as shown in Figure 2-231.

Figure 2-231 Complementary Sparse and Dense Via Regions



In that case, the sparse and dense checking regions have the same dimensions, and the Pi attribute need not be specified because it is assumed to be the same as Pd. The rule checks the regions shown in Figure 2-232.

Figure 2-232 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

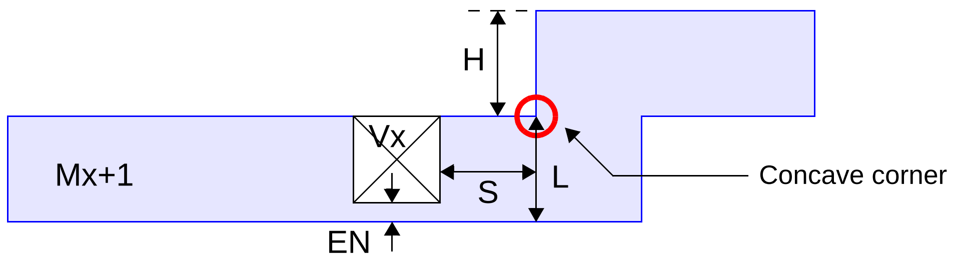
Note:

This rule is supported only by Zroute.

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 2-233, the via in layer Vx 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 2-233 Via-to-Jog Metal 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,

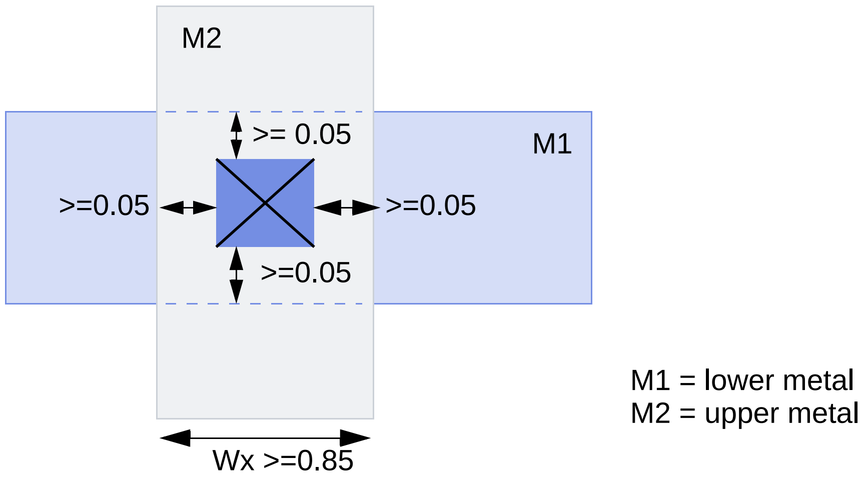
```
DesignRule {
  layer1                                     = "VIA3"
  layer2                                     = "MET4"
  concaveMetalToCutTblSize                 = 1
  concaveMetalToCutNameTbl                 = (Vsm)
  concaveMetalToCutJogMinHeightThresholdTbl = (0.005)
  concaveMetalToCutWireMaxWidthThresholdTbl = (0.041)
  concaveMetalToCutMinEncTbl               = (0.004)
  concaveMetalToCutMinDistTbl              = (0.036)
}
```

Interlayer Via Enclosure Rule

Note:
  This rule is supported only by Zroute.

The interlayer via enclosure rule specifies a minimum enclosure of metal around a lower-level via, as shown in Figure 2-234.

Figure 2-234 Inter-Layer Via Enclosure Rule



The upper layer metal width is at least Wx and the lower layer metal enclosure value is at least a specified distance. Define the rule in the `DesignRule` section of the technology file. The following example is shown Figure 2-234:

```
DesignRule {
   layer1             = M1
   layer2             = V1
   fatWireCutTblSize = 1
   fatWireCutNameTbl = (Vsq)
   fatWireCutUpperWidthThresholdTbl = (0.85)
   fatWireCutLowerMinEnclosureTbl = (0.05)
}
```

Where you previously defined the via layer as:

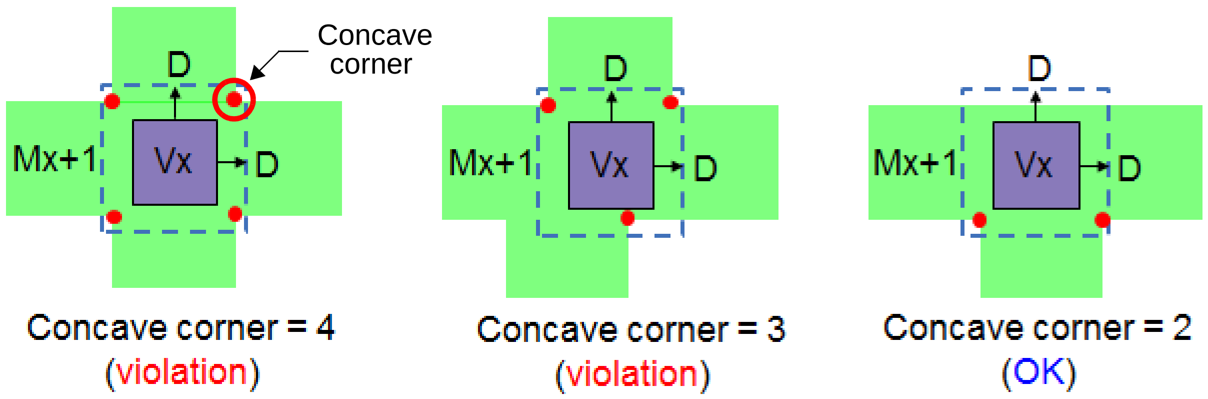
```
Layer "Via1" {
   cutTblSize = 3
   cutNameTlb = (Vsq, Vx, Vm)
   ...
}
```

Via to Concave Corners Maximum Limit Rule

Note:
This rule is supported only by Zroute.

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 2-235.

Figure 2-235 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)
}
```

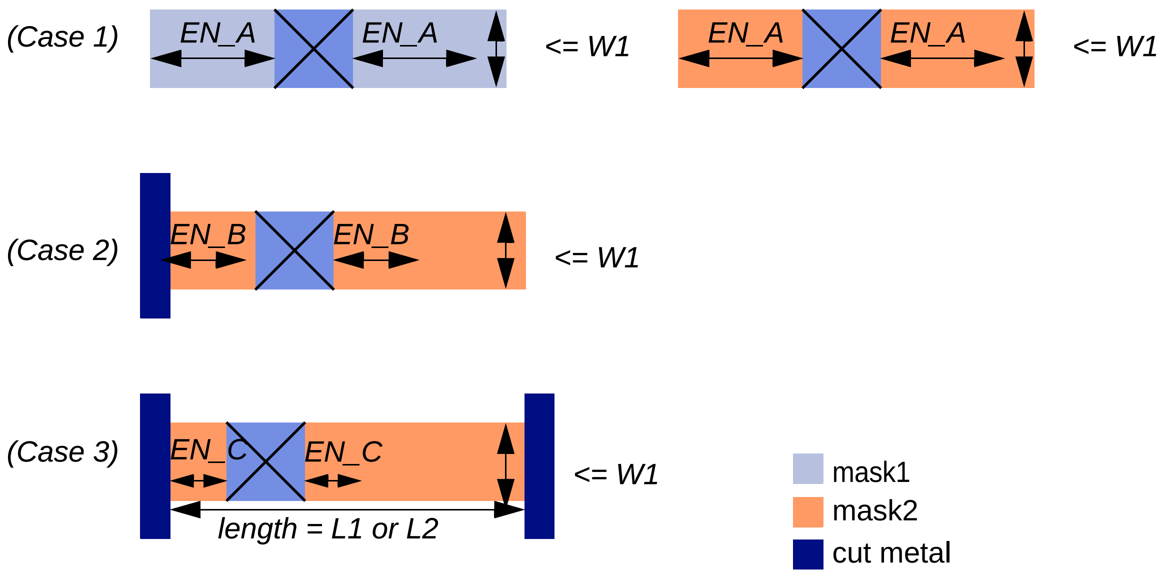
In this example, for the vias named Vs, Vh, and Vv, no more than two concave corners are allowed in the enclosing upper metal within a distance of 0.013 from the via edges.

Color-Based Via Enclosure Rule

Note:
    This rule is supported only by Zroute.

The color-based via enclosure rule specifies different minimum enclosure values that depend on whether the metal in a mask touches cut metal, as shown in Figure 2-236. Cut metal is a mask layer for self-aligned double-patterning processes.

Figure 2-236 Color-Based Via Enclosure



The rule states that for a mask1 with a width smaller than W1, and an enclosure of EN_A or larger, the following conditions apply to mask2 (refer to Figure 2-236):

1. If the mask2 line-ends do not touch a cut metal, then the enclosure is EN_A or larger. In the figure, you can see that the enclosure is EN_A or larger for both mask1 and mask2.
2. If one of the mask2 line-ends touches a cut metal, then the enclosure is EN_B or larger
3. If both line-ends touch cut metal, and the total length of the mask is L1 or L2, then the enclosure is EN_C or larger

The syntax for this rule is in the `DesignRule` section:

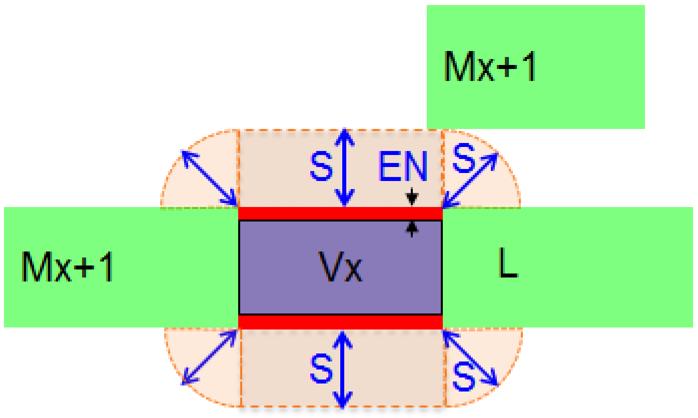
```
DesignRule {
   layer1                               = "MetalX"
   layer2                               = "ViaX"
   maskEndEnclosureCutTblSize           = 1
   maskEndEnclosureCutNameTbl           = (Vsq)
   maskEndEnclosureMaxWidthThresholdTbl  = (W1)
   mask1EndEnclosureTbl                 = (EN_A)
   mask2EndNotTouchCutMetalEnclosureTbl  = (EN_A)
   mask2EndTouchCutMetalEnclosureTbl     = (EN_B)
   mask2EndLengthRangeTbl               = ("L1, L1, L2, L2")
   mask2EndBothTouchCutMetalEnclosureTbl = (EN_C)
}
```

Enclosure-Dependent Metal Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-237.

Figure 2-237 Enclosure-Dependent Metal Spacing 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,

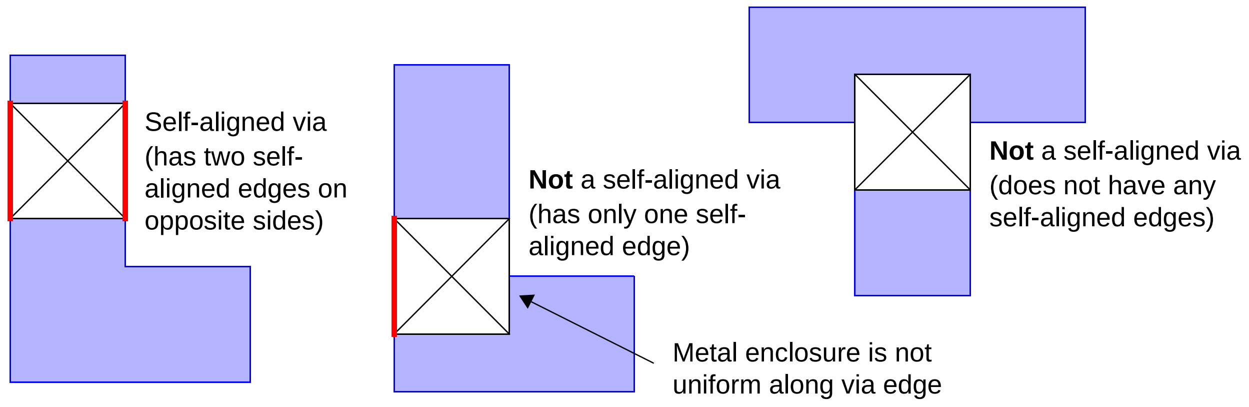
```
DesignRule {
  layer1                       = "VIA1"
  layer2                       = "MET2"
  encMetalCutTblSize           = 2
  encMetalCutNameTbl           = (Vh, Vv)
  encMetalEncThresholdTblSize = 2
  encMetalEncThresholdTbl      = (0.002, 0.006)
  encMetalXMinSpacingTbl       = (    -1,     -1,
                                  0.045, 0.036)
  encMetalYMinSpacingTbl       = (0.045, 0.036,
                                     -1,     -1)
}
```

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 2-238, 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 2-238 Self-Aligned Edges and Self-Aligned Vias

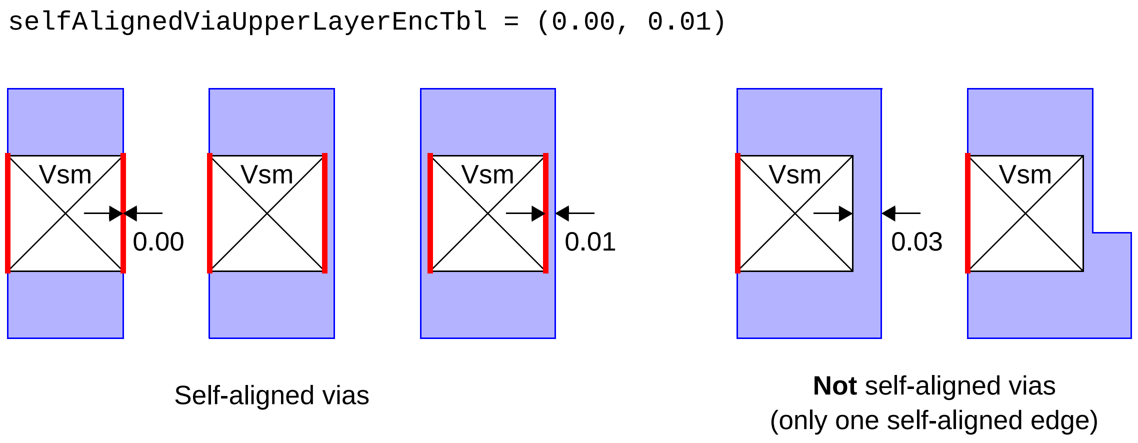

Self-aligned via (has two self-aligned edges on opposite sides)
Not a self-aligned via (has only one self-aligned edge)
Not a self-aligned via (does not have any self-aligned edges)
Metal enclosure is not uniform along via edge

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 2-239.

Figure 2-239 Self-Aligned Via Examples


selfAlignedViaUpperLayerEncTbl = (0.00, 0.01)
Vsm
Vsm
Vsm
Vsm
Vsm
0.00
0.01
0.03
Self-aligned vias
Not self-aligned vias
(only one self-aligned edge)

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 Upper Metal to Other Metal Spacing Rule
- Self-Aligned Via Interlayer Spacing Rules
- Self-Aligned Via Array Rule
- Self-Aligned Via Diagonal Spacing Rule
- Self-Aligned Via Zigzag Spacing Rule
- Double-Patterning Self-Aligned Via Cluster Spacing Rule

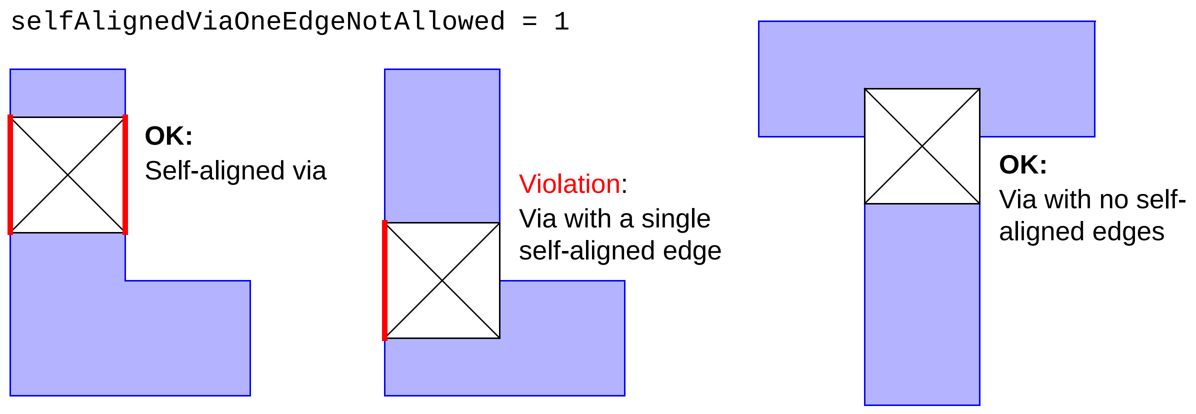
The self-aligned via rules are supported only by Zroute.

Single Self-Aligned Edge Forbidden Rule

Note:
This rule is supported only by Zroute.

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 2-240.

Figure 2-240 Single Self-Aligned Edge Forbidden Rule


selfAlignedViaOneEdgeNotAllowed = 1
OK:
Self-aligned via
Violation:
Via with a single self-aligned edge
OK:
Via with no self-aligned edges

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

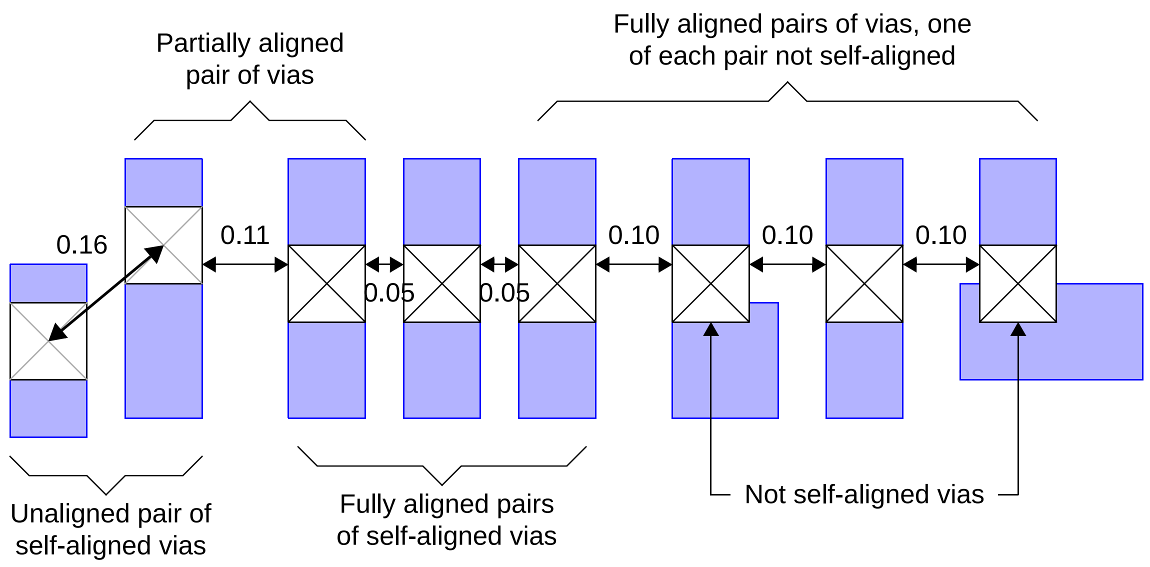
Note:
This rule is supported only by Zroute.

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).

Figure 2-241 shows examples of these alignment configurations.

Figure 2-241 Self-Aligned Via Spacing Rule Examples


Partially aligned pair of vias
Fully aligned pairs of vias, one of each pair not self-aligned
0.16
0.11
0.05
0.05
0.10
0.10
0.10
Unaligned pair of self-aligned vias
Fully aligned pairs of self-aligned vias
Not self-aligned vias

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:

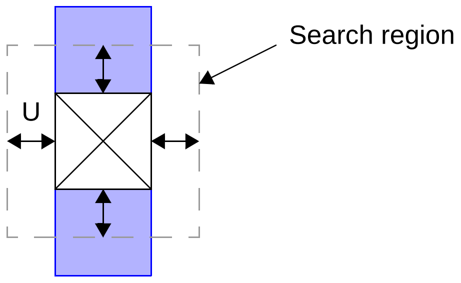
- A fully aligned pair of self-aligned vias: 0.05 (edge-to-edge spacing)
- A fully aligned pair of vias, one self-aligned and one not: 0.10 (edge-to-edge spacing)
- A partially aligned pair of vias (whether or not self-aligned): 0.11 (edge-to-edge spacing)
- An unaligned pair of self-aligned vias: 0.16 (separation measured center-to-center)

Self-Aligned Via Cluster Rules

Note:
This rule is supported only by Zroute.

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 2-242.

Figure 2-242 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 types 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 2-243 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 2-243 Rectangular Self-Aligned Via Cluster

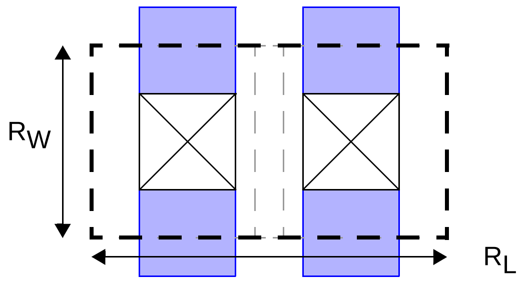
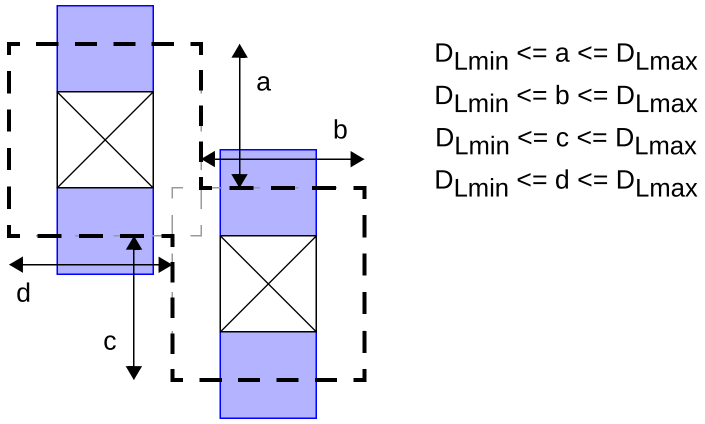


Figure 2-244 shows an example of a diagonal self-aligned via cluster containing two vias. The search regions overlap at the corners, and the lengths of the segments a, b, c, and d must each be at least a minimum D_{Lmin} and no more than a maximum D_{Lmax}.

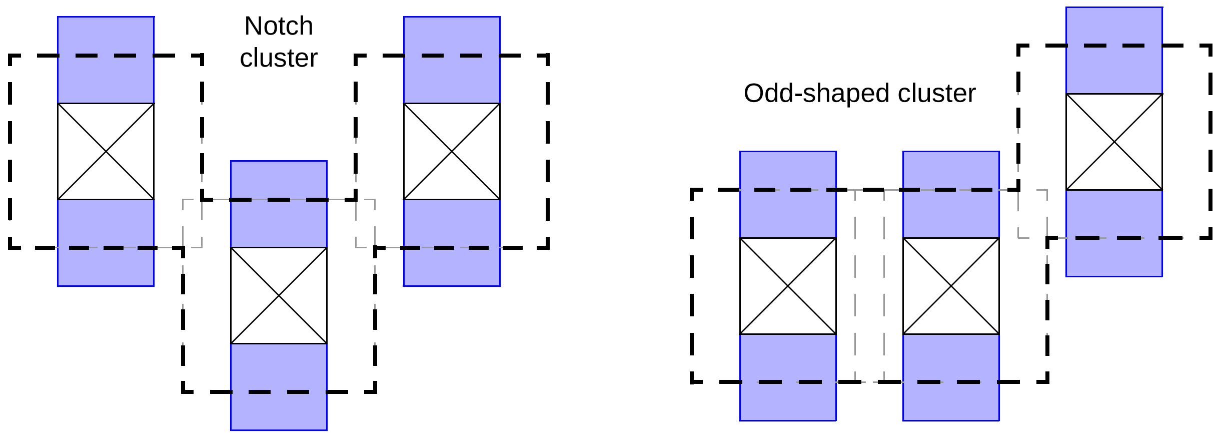
Figure 2-244 Diagonal Self-Aligned Via Cluster



a
b
c
d
$D_{Lmin} <= a <= D_{Lmax}$
$D_{Lmin} <= b <= D_{Lmax}$
$D_{Lmin} <= c <= D_{Lmax}$
$D_{Lmin} <= d <= D_{Lmax}$

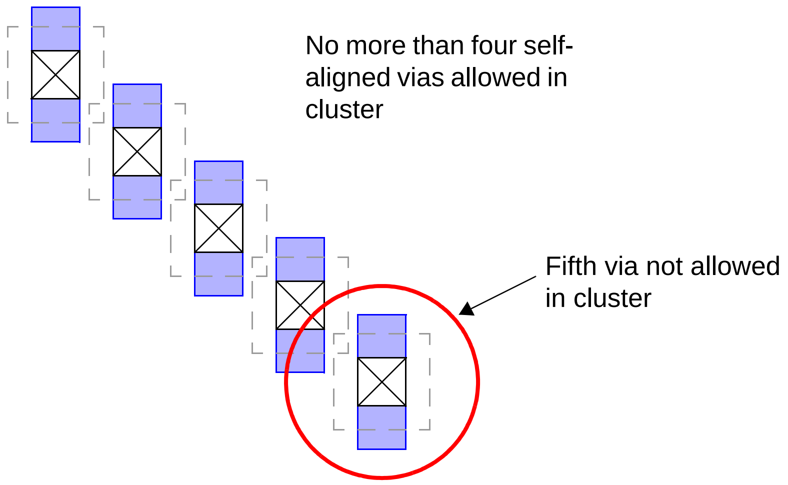
Figure 2-245 shows an example of a notch cluster and an odd-shaped cluster, which is neither a rectangular nor a diagonal cluster. These types of clusters can be forbidden by the cluster rules.

Figure 2-245 Other Types of Self-Aligned Via Clusters



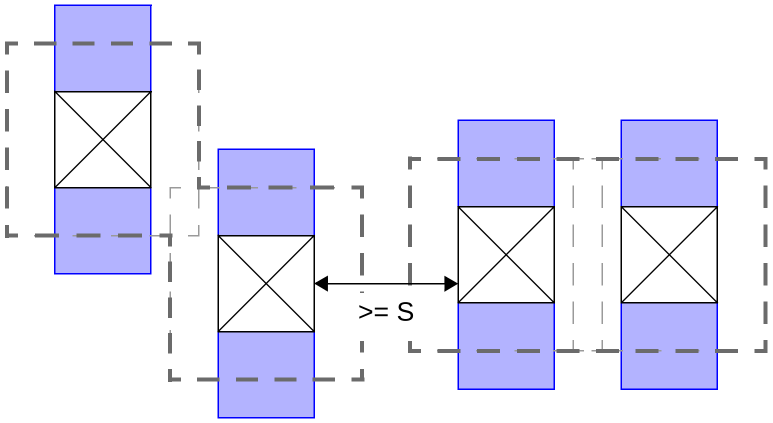
The self-aligned via cluster rules can also limit the number of vias allowed in one cluster. For example, if the limit is set to 4, the 5-via cluster shown in Figure 2-246 would be forbidden.

Figure 2-246 Self-Aligned Via Cluster Limit Rule


No more than four self-aligned vias allowed in cluster
Fifth via not allowed in cluster

A cluster rule can specify a minimum spacing between vias that belong to different self-aligned via clusters. For example, in Figure 2-247, 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 2-247 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 2-244) 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)
}
```

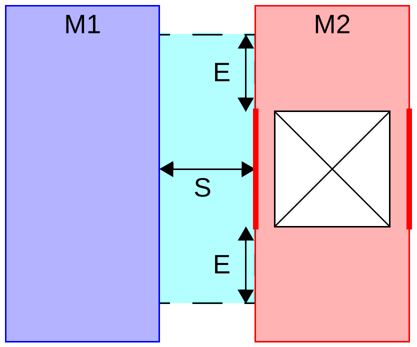
In this example, the minimum spacing requirement applies only when one or both clusters have three or more vias; it does not apply when both clusters only one or two vias each.

Self-Aligned Via Upper Metal to Other Metal Spacing Rule

Note:
This rule is supported only by Zroute.

The self-aligned via upper metal edge to other metal edge rule specifies the minimum spacing S between the upper metal edge along the self-aligned via edge and a neighboring shape in the lower metal layer without a via. The minimum spacing rule applies for an extension distance E away from the adjacent via edges, as shown in the following figure.

Figure 2-248 Self-Aligned Via Upper Metal Edge to Other Metal Edge Rule



This is the syntax of the rule:

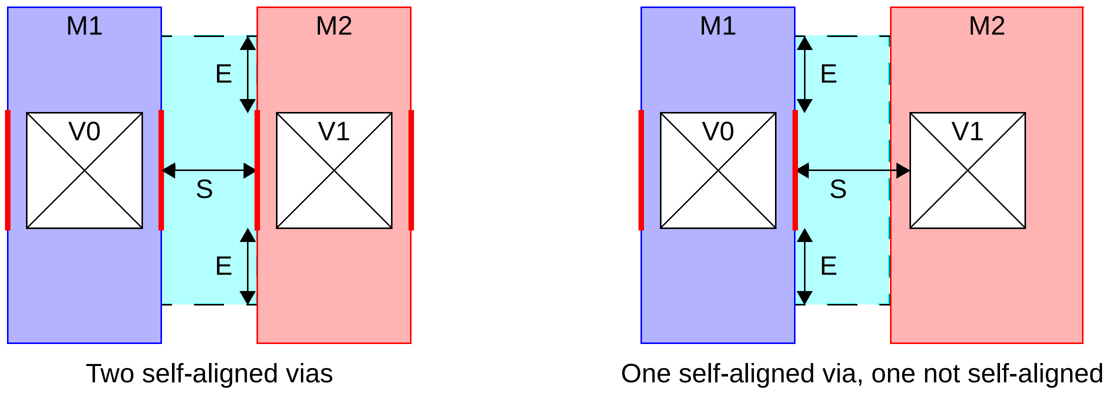
```
DesignRule {
  layer1                                   = "ViaX"
  layer2                                   = "MetalX"
  selfAlignedViaEdgeExtension              = E
  selfAlignedViaLowerToUpperMetalMinSpacing = S
}
```

Self-Aligned Via Interlayer Spacing Rules

Note:
  This rule is supported only by Zroute.

The self-aligned via interlayer spacing rule specifies the minimum spacing S between an upper metal edge by a self-aligned via and a neighboring lower metal edge by its self-aligned via edge. If one of the vias is not a self-aligned via, the minimum required distance is measured from the edge of the non-self-aligned via. This minimum spacing rule applies for an extension distance E from the adjacent via edges, as shown in Figure 2-249.

Figure 2-249 Self-Aligned Via Interlayer Spacing Rule



This is the syntax of the rule:

```
DesignRule {
  layer1           = "ViaX"
  layer2           = "ViaX+1"
  cut1TblSize      = 1
  cut2TblSize      = 1
  cut1NameTbl      = (V0)
  cut2NameTbl      = (Vsm)
  minSpacingYParallelLengthThresholdTbl       = (-E)
  diffNetXMinSpacingTbl                       = (S)
  diffNetSavEncEdgeToNonSavCutXMinSpacingTbl  = (S)
  diffNetSavEncEdgeToSavEncEdgeXMinSpacingTbl = (S)
  minSpacingXParallelLengthThresholdTbl       = (-E)
  diffNetYMinSpacingTbl                       = (S)
  diffNetSavEncEdgeToNonSavCutYMinSpacingTbl  = (S)
  diffNetSavEncEdgeToSavEncEdgeYMinSpacingTbl = (S)
}
```

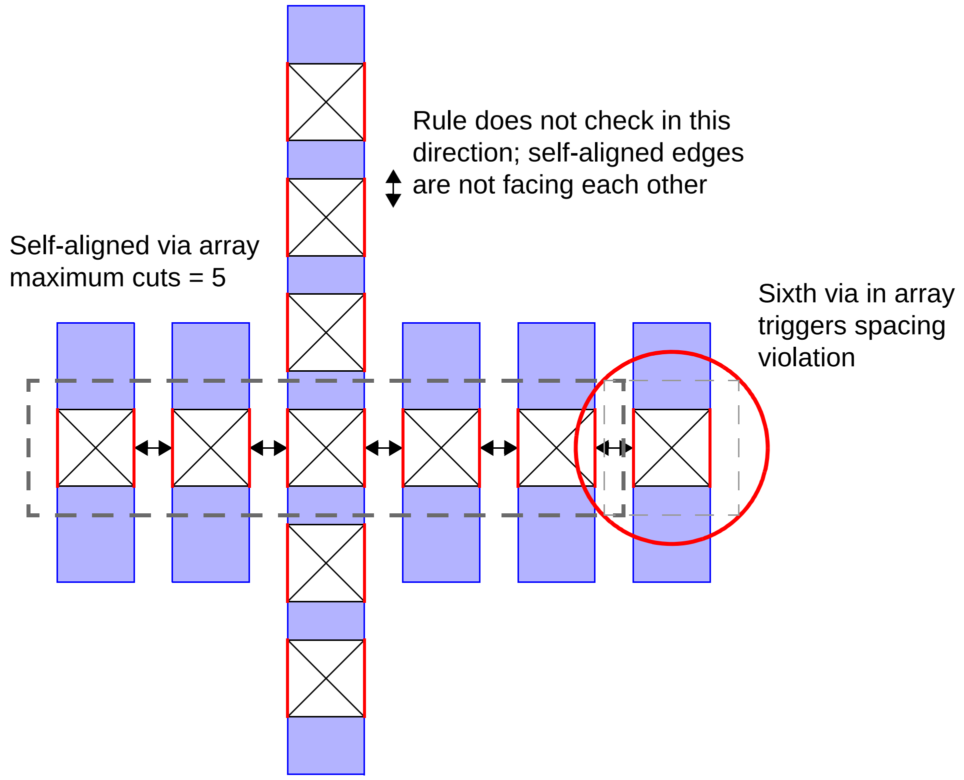
Self-Aligned Via Array Rule

Note:
This rule is supported only by Zroute.

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 2-250, 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 2-250 Self-Aligned Via Array Rule Examples


Rule does not check in this direction; self-aligned edges are not facing each other
Self-aligned via array maximum cuts = 5
Sixth via in array triggers spacing violation

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.

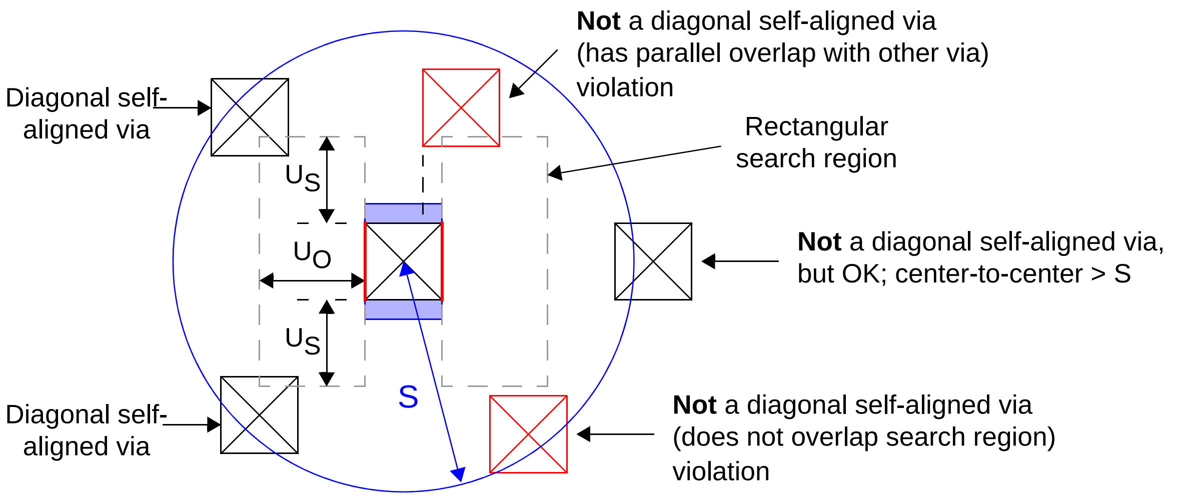
Self-Aligned Via Diagonal Spacing Rule

Note:
This rule is supported only by Zroute.

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 2-251. Any nearby via that has no parallel overlap, but overlaps the search region by any amount, is considered a “diagonal” self-aligned via.

Figure 2-251 Diagonal Self-Aligned Vias


Not a diagonal self-aligned via
(has parallel overlap with other via)
violation
Diagonal self-aligned via
Rectangular search region
U_S
U_O
U_S
Not a diagonal self-aligned via, but OK; center-to-center > S
S
Diagonal self-aligned via
Not a diagonal self-aligned via
(does not overlap search region)
violation

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 the orthogonal direction by 0.073. This forms a rectangular search region measuring 0.073 by (0.055 + 0.055 + via_height) on the two self-aligned sides of the via. Any via that is outside of the search areas or has any 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

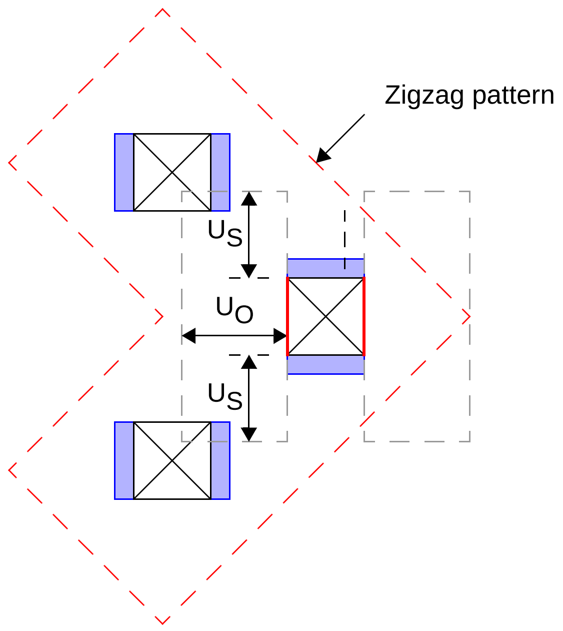
Note:

This rule is supported only by Zroute.

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.

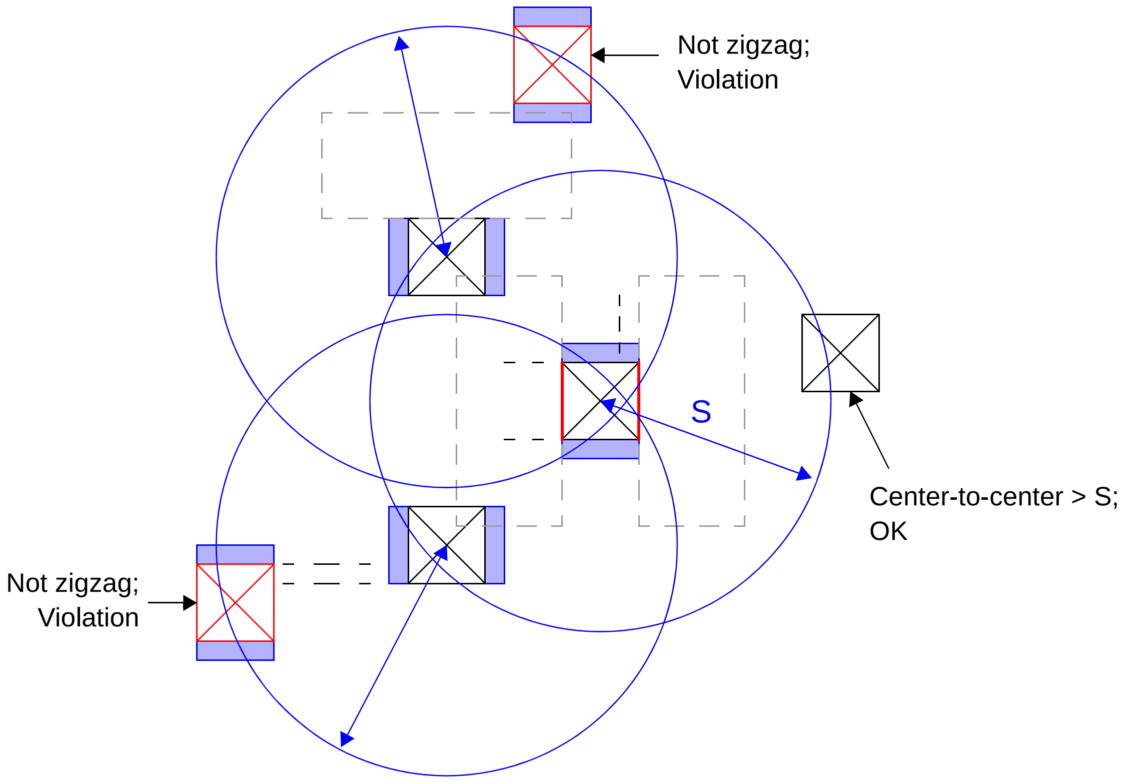
The rule defines two upsize lengths: U_S in the same direction as the self-aligned edge and U_O for the orthogonal direction. These two attributes establish a rectangular search region on two sides of each self-aligned via, as shown in Figure 2-252. A zigzag via pattern exists when exactly two other 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 2-252 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 2-253.

Figure 2-253 Minimum Spacing Requirement in Zigzag Pattern of Self-Aligned Vias



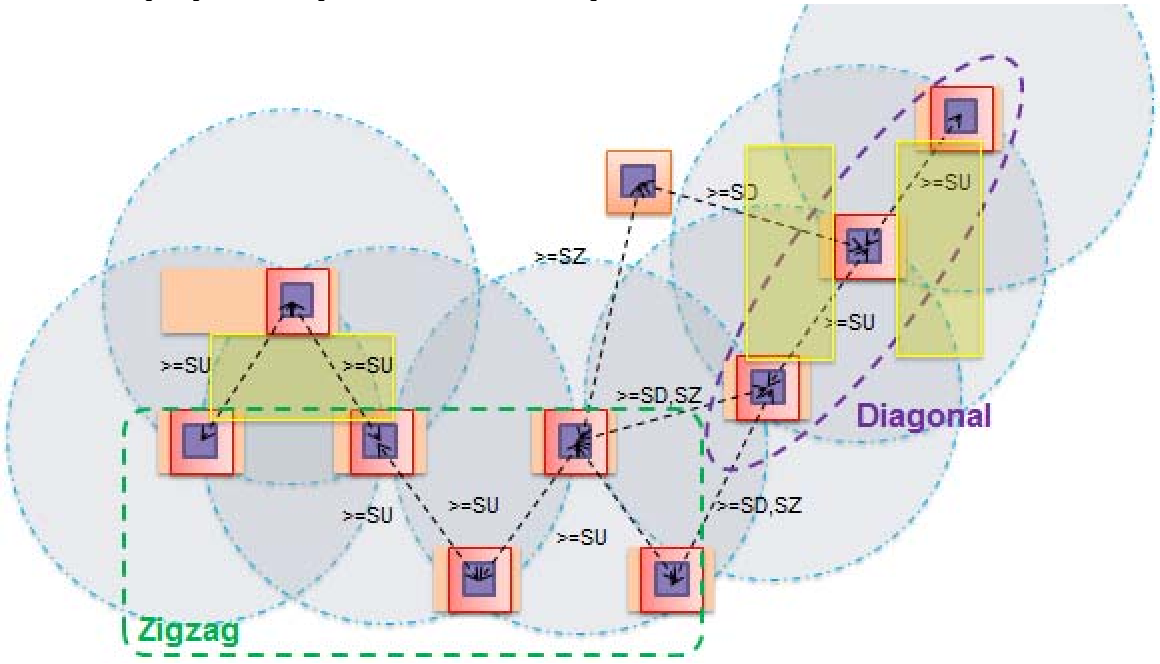
The following example shows the syntax of the rule:

```
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 + 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 2-254 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 2-254 Zigzag and Diagonal Sets of Self-Aligned Via Patterns



Double-Patterning Self-Aligned Via Cluster Spacing Rule

Note:
This rule is supported only by Zroute.

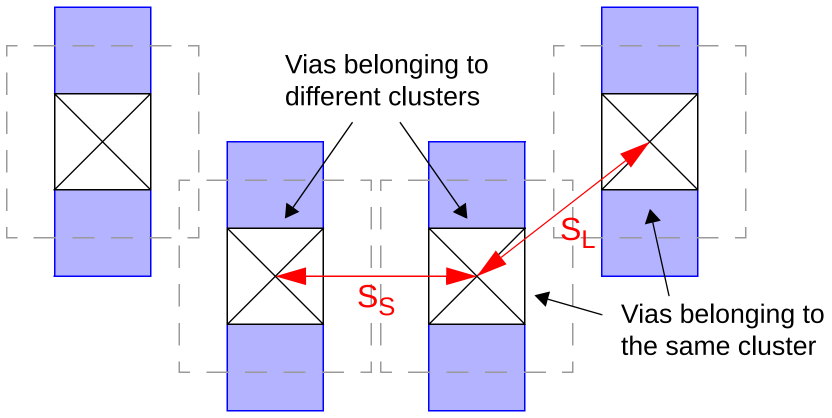
The double-patterning self-aligned via cluster spacing rule specifies the minimum center-to-center spacing of vias of the same layer and same color in a double-patterning technology. The rule applies different spacing requirements according to the clustering status and via dimensions, as follows:

- If one or both of two vias do not belong to a self-aligned via cluster, the minimum center-to-center spacing is SL.
- For each self-aligned via, the tool checks to see if the via is on-grid, and if so, it resizes the via larger by an amount specified in the rule. To check to see if a via is on-grid, the tool creates a list of via edge lengths and adds an upsize amount UP to each edge

length. If all the resulting edge lengths are whole multiples of a specified small unit size UX, the via is on-grid. For each on-grid via, the tool increases the original lengths of the self-aligned via edges by an amount UP and all the other (orthogonal) edges by a smaller amount UO, resulting in larger, not-on-grid via. Vias already not-on-grid are left unchanged.

- Upsizing vias can change the set of vias that belong to a self-aligned via cluster, due to the increase in size of the checking area surrounding each via. For two self-aligned vias belonging to the same cluster, the minimum center-to-center spacing between vias is SL. On the other hand, for two vias belonging to different clusters (clustered with other vias but not with each other), their minimum center-to-center spacing requirement is a smaller value SS. See Figure 2-255.

Figure 2-255 Double-Patterning Self-Aligned Via Cluster Spacing Rule Example


Vias belonging to different clusters
S_L
S_S
Vias belonging to the same cluster

To implement this rule, first define the self-aligned via and cluster attributes in the `ViaRule` section:

```
ViaRule SAVx {
  ...
  selfAlignedViaClusterSavEdgeUpsizeTbl   = (UP)
  selfAlignedViaClusterOrthoEdgeUpsizeTbl = (UO)
  selfAlignedViaClusterCutEdgeXUpsizeTbl   = (UP, UP, UP)
  selfAlignedViaClusterCutEdgeYUpsizeTbl   = (UP, UP, UP)
  selfAlignedViaClusterEdgeXUnitLengthTbl  = (UX, UX, UX)
  selfAlignedViaClusterEdgeYUnitLengthTbl  = (UX, UX, UX)
}
```

Then define the cut spacing rules for clustered vias in the `DesignRule` section:

```
DesignRule {
  layer1                                   = "Vx"
  layer2                                   = "Vx"
  ...
  sameColorSavClusterCenterMinSpacingTbl      = (SS)
  sameColorNonSavClusterCenterMinSpacingTbl   = (SL)
  sameColorSavIntraClusterCenterMinSpacingTbl = (SL)
}
```

Fat Metal Contact Rules

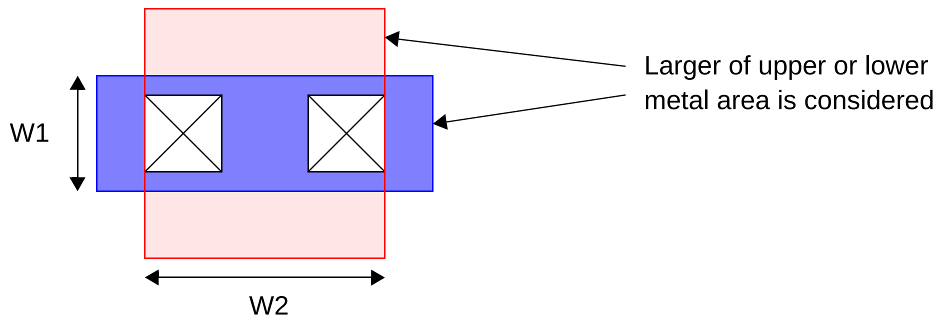
The fat metal contact rules are described in the following sections:

- Area-Based Fat Metal Contact Rule
- Two-Dimensional Fat Metal Contact Rule
- One-Dimensional 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

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 metal 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 2-256.

Figure 2-256 Area-Based Fat Metal Contact Rule



The following example demonstrates usage of the rule.

```
Layer "ViaX" {
  ...
  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")
 fatTblThresholdIncludeEnclosure = 0
 fatTblCutTouchingThinnerWire  = 0
 fatTblCutTouchingThinnerWire2 = 0
}
```

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.

To have the rule consider the via's enclosure shape when computing the metal width for the fat metal contact rule, set the `fatTblThresholdIncludeEnclosure` attribute to 1, as shown in the following example:

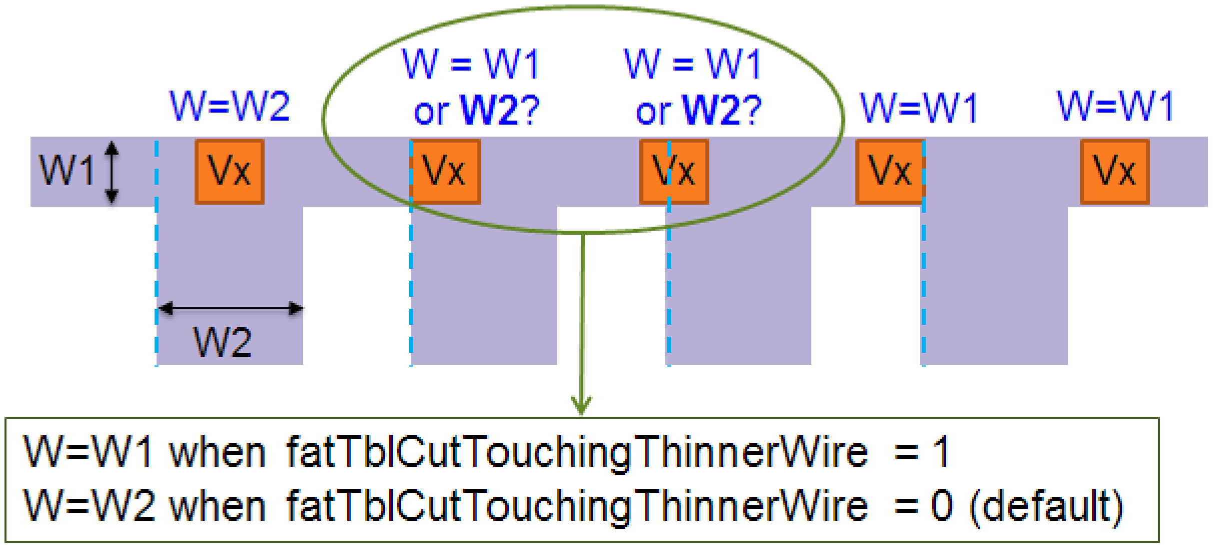
```
Layer "ViaX" {
 ...
 fatTblThresholdIncludeEnclosure = 1
 fatTblDimension                 = 5
 fatTblThreshold                 = (0.0,0.09,0.16,0.35,0.48)
 fatTblDimension2                = 3
 fatTblThreshold2                = (0,0.28,0.50)
 fatTblAreaDimension             = 2
 fatTblAreaThreshold             = (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 one-dimensional, two-dimensional, or three-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.

The `fatTblCutTouchingThinnerWire` and `fatTblCutTouchingThinnerWire2` attributes control how the tool determines the width of the lower and upper metal, respectively, when a via touches or crosses a boundary between two metal shapes of different widths. By default, the tool uses the larger width value. To use the smaller width value, set the attribute to 1.

For example, in Figure 2-257, the second via from the left touches a metal shape boundary, and the third via crosses over such a boundary.

Figure 2-257 Ambiguous Metal Widths Enclosing Vias


W=W2
W = W1 or W2?
W = W1 or W2?
W=W1
W=W1
W1
Vx
W2
W=W1 when fatTblCutTouchingThinnerWire = 1
W=W2 when fatTblCutTouchingThinnerWire = 0 (default)

Two-Dimensional Fat Metal Contact Rule

The two-dimensional 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, where the upper-metal and lower-metal layers have separate thresholds.

Both the power and ground routing and the detail routing operations honor the two-dimensional fat metal contact rule. A rule violation is flagged as a DRC error.

To specify a two-dimensional fat metal contact rule, omit the `fatTblAreaDimension` and `fatTblAreaThreshold` attributes from the area-based fat metal contact rule syntax described in Area-Based Fat Metal Contact Rule. For example,

```
Layer "ViaX" {
 ...
 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")
```

The technology file also supports the older syntax shown in the following example for the two-dimensional fat metal contact rule.

Note:
Support for this older syntax will end in a future release. All new design rule sets should use the previously described syntax.

```
Layer "VIA23" {
  fatTblDimension = 4
  fatTblThreshold = (0.0, 0.421, 0.701, 0.981)
  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)
}
```

One-Dimensional Fat Metal Contact Rule

The one-dimensional 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, where the thresholds are determined by the maximum width of the upper and lower layer metal segments.

To specify a one-dimensional fat metal contact rule, omit the `fatTblAreaDimension`, `fatTblAreaThreshold`, `fatTblDimension2`, and `fatTblThreshold2` attributes from the area-based fat metal contact rule syntax described in Area-Based Fat Metal Contact Rule. For example,

```
Layer "ViaX" {
 ...
 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")
```

Fat Metal Extension Contact Rule

Note:

The newer rule described in Area-Based Fat Metal Extension Contact Rule 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 2-188 on page 2-199, 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`

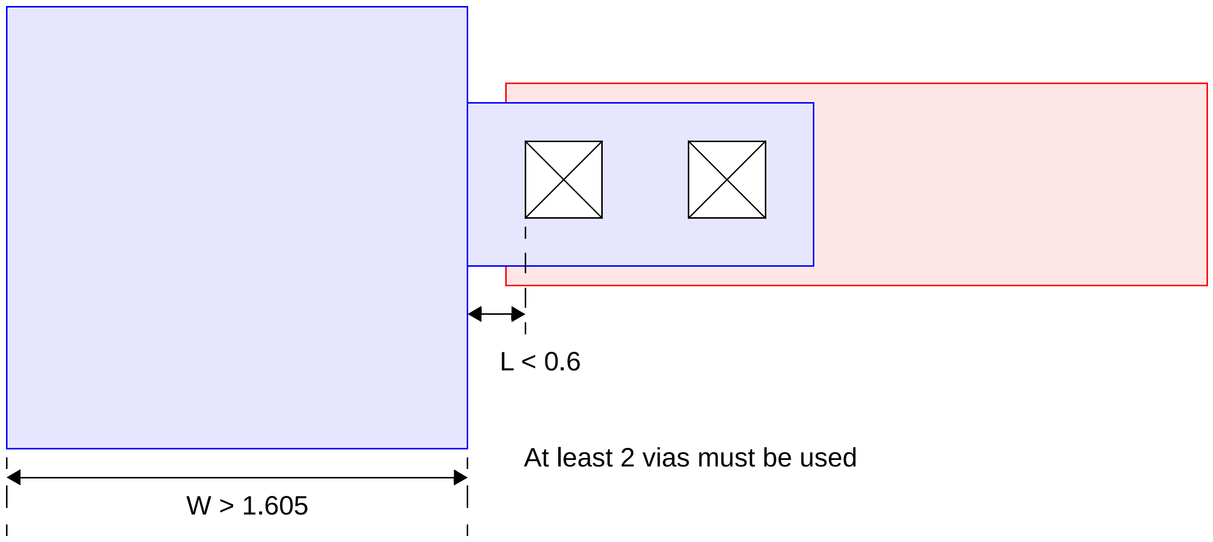
  Specifies the minimum number of vias for the wires within the extension range of fat wires. You must specify *n* values, where *n* is the table dimension.

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 2-258, if the fat metal width is greater than or equal to 1.605 and the fat metal extension range is less than or equal to 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.

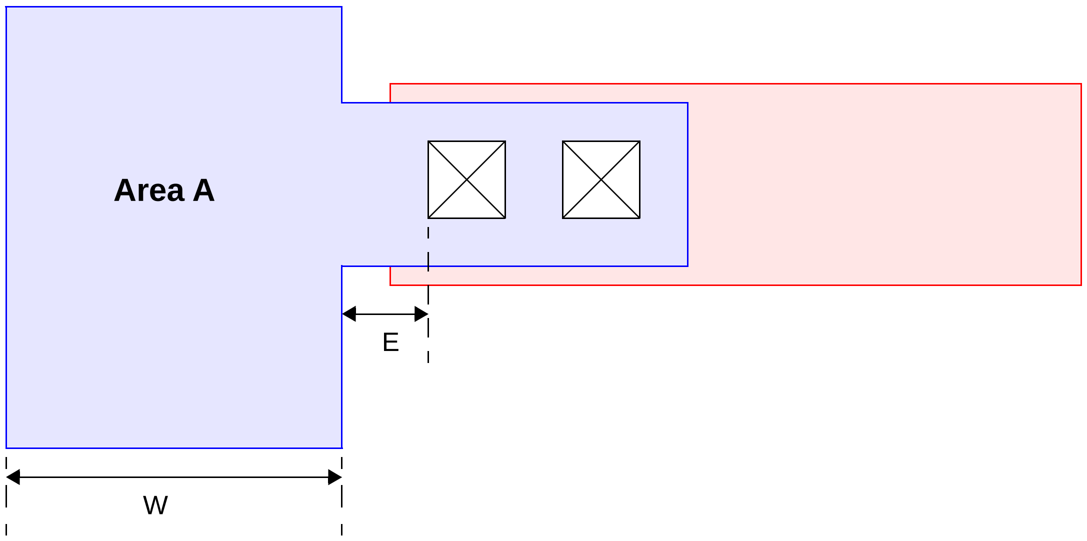
Figure 2-258 Fat Metal Extension Contact Rule



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 2-259.

Figure 2-259 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 "ViaX" {
 ...
 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 means 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 the default-width metal.

In the foregoing example, if the list `7,17,27,37,47` represents the entire list of all contact numbers, 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 "ViaX" {
 ...
 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.

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 "ViaX" {
  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, the tool internally converts the rule to standard syntax described in the foregoing sections, Area-Based Fat Metal Contact Rule and Fat Metal Extension Contact Rule. The tool 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 by setting an environment variable (for IC Compiler II) or tool variable (in IC Compiler) before you write out the technology file:

IC Compiler II:

```
% setenv write_converted_tf_syntax true
...
% icc2_shell
...
icc2_shell> write_tech_file my_tech.tf
...
```

IC Compiler:

```
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 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)
}
```

The physical implementation tool selects the equivalent contact codes from the technology file and obtains the attributes for these contacts:

```
ContactCode "VIA02F" {                ContactCode "VIA02B" {
  contactCodeNumber  = 23               contactCodeNumber   = 24
  cutLayer           = "V2"             cutLayer            = "V2"
  lowerLayer         = "M2"             lowerLayer          = "M2"
  upperLayer         = "M3"             upperLayer          = "M3"
  cutWidth           = 0.06             cutWidth            = 0.12
  cutHeight          = 0.06             cutHeight           = 0.06
  upperLayerEncWidth  = 0.021           upperLayerEncWidth  = 0.021
  upperLayerEncHeight = 0.021           upperLayerEncHeight = 0
  lowerLayerEncWidth  = 0.021           lowerLayerEncWidth  = 0.021
  lowerLayerEncHeight = 0.021           lowerLayerEncHeight = 0
}                                     }
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
}
```

The `minCutsTbl` section and part of the `enclosureTbl` section are converted to via rules in the 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 the tool allows wires and vias connected 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 in the IC Compiler tool. 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.

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:

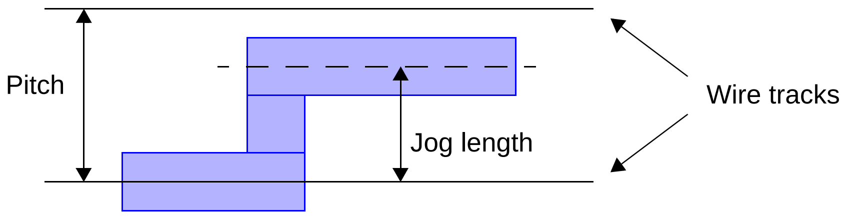
- Small Jog Rule
- Jog Wire End-of-Line Via Rule
- Jog Wire Via Keepout Region Rule
- Line-End to Jog Distance Rule

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.

Figure 2-260 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.

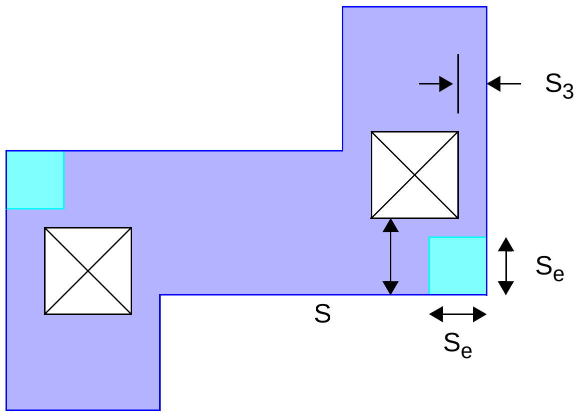
Jog Wire Via Keepout Region Rule

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                          = "MetalX"
  layer2                          = "ViaX"
  minEnclosure                    = 0
   jogWireViaKeepoutTblSize        = 3
   jogWireViaKeepoutEncThreshold   = (0.005,0.015,0.025)
   jogWireViaKeepoutMinSize        = (0.08,0.06,0)
}
```

Figure 2-261 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, 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, the spacing to the corner S_e must be at least 0.06.

Figure 2-261 Jog Wire Via Keepout Region Rule

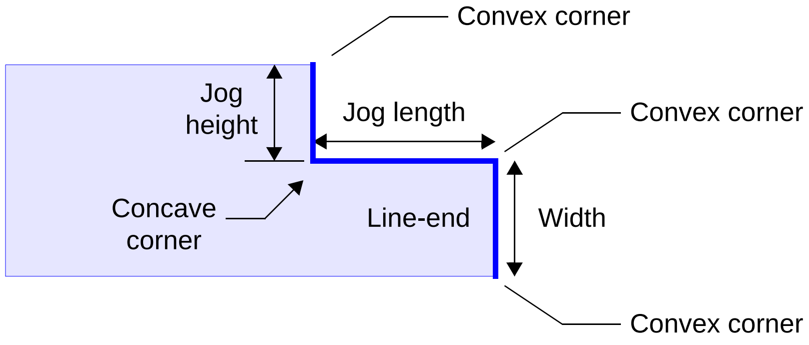


This rule applies only to a single via enclosed within a jog wire. It does not apply to a double via in a given jog wire.

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 2-262.

Figure 2-262 Line-End to Jog Distance 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, 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, 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 "MetalX" {
  minEdgeJogWireMinWidth  = 0.068
  minEdgeJogWireMinLength = 0.136
  minEdgeJogMinHeight     = 0.058
}
```

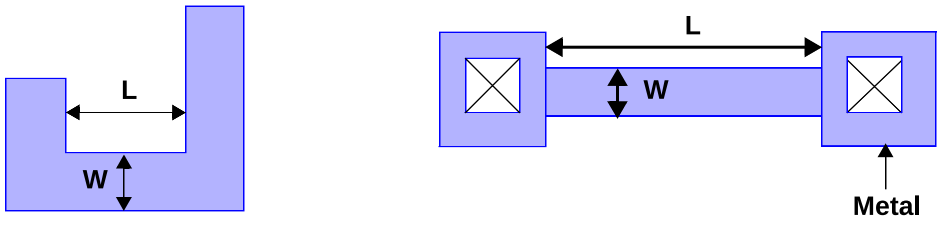
Dog Bone Rule

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 2-263 shows the attributes associated with the dog bone rule.

Figure 2-263 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`

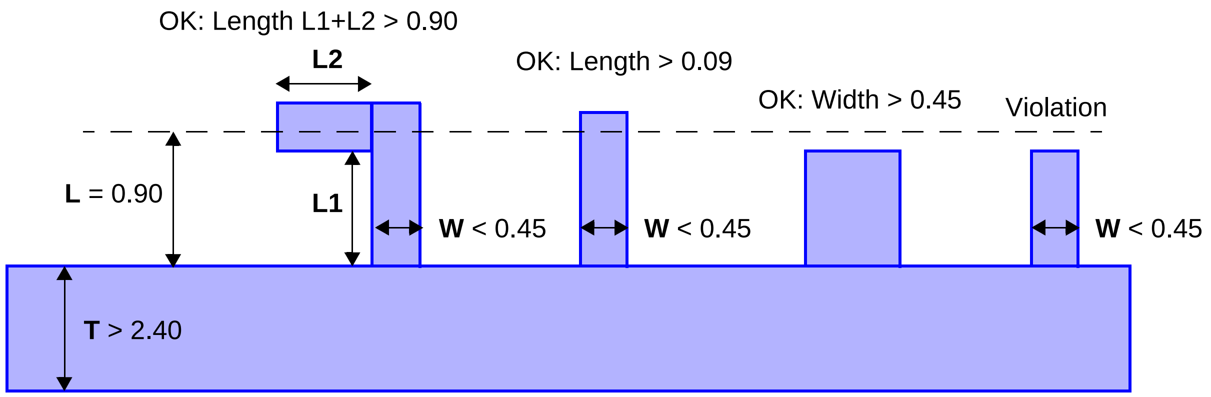
Specifies a floating-point number that represents the minimum width value for the connected thin wire.

Define these attributes in a metal `Layer` section of the technology file. For example,

```
Layer "MetalX"  {
    protrusionTblDim       = 2
    protrusionFatThresholdTbl       = (1.20,2.40)
    protrusionLengthLimitTbl       = (0.60,0.90)
    protrusionMinWidthTbl = (0.30, 0.45)
}
```

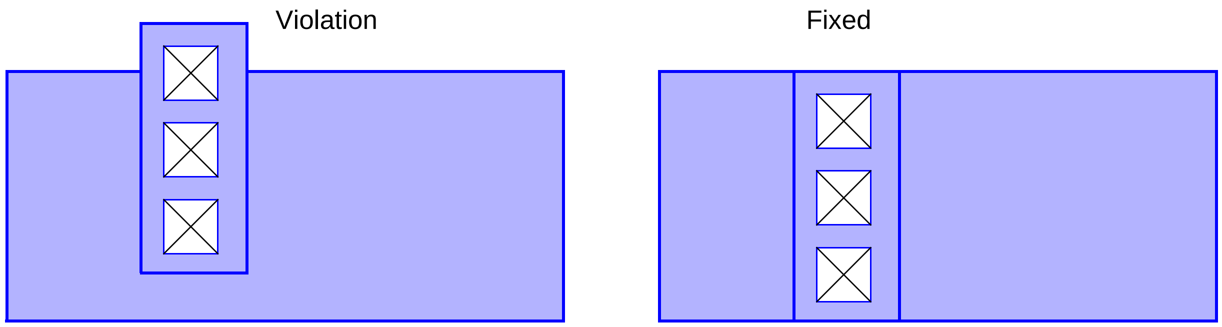
Figure 2-264 shows the attributes associated with the protrusion length rule.

Figure 2-264 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 2-265. A violation such as this can be fixed by adding metal stubs or rerouting the wire.

Figure 2-265 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 =  "MetalX"
  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 in the IC Compiler tool. 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 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 in the IC Compiler tool.

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

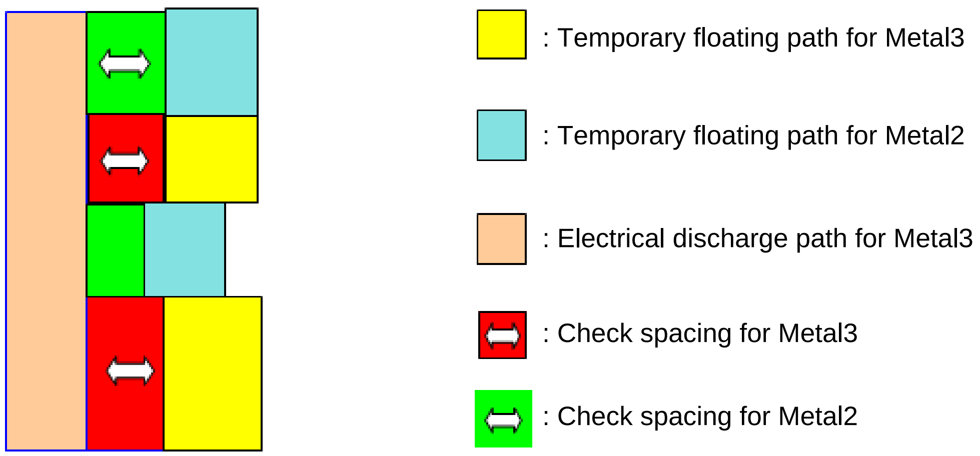
The implementation tool checks floating wire antennas by checking the maximum allowable value for metal areas and the minimum spacing to the adjacent wire. The parallel length-based floating wire antenna rule also considers the parallel length between the floating wire and the adjacent wire. Use the following options to specify the parallel length-based floating wire antenna rule:

```
M1FloatingWirePLength1-5

M2FloatingWirePLength1-5
...

M12FloatingWirePLength1-5

M1FloatingWirePLMinSpc1-5

M2FloatingWirePLMinSpc1-5
...

M12FloatingWirePLMinSpc1-5
```

The floating wires are merged according to the values you specify.

For example, in Figure 2-266, the Metal2 and Metal3 floating wires are merged.

Figure 2-266 Parallel Length-Based Floating Wire Antenna Rule


: Temporary floating path for Metal3
: Temporary floating path for Metal2
: Electrical discharge path for Metal3
: Check spacing for Metal3
: Check spacing for Metal2

Zroute Error Messages

The following table lists some of the DRC errors reported by the Zroute router in the IC Compiler and IC Compiler II tools, and the corresponding rule names in this chapter.

Table 2-6 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 enclosed via spacing	Color-Based Constrained Via Spacing Rule
Constrained via spacing	Constrained Via Spacing Rule
Critical via spacing	Critical Via Spacing Rule
Diff net fat extension spacing	Fat Metal Forbidden Spacing Rule
Diff net spacing	Fat Metal Forbidden Spacing Rule
Diff net spacing	Fat Metal Span Spacing Rule

Table 2-6 Zroute Error Messages and Design Rule Titles (Continued)

Zroute DRC error message	Design rule title
Diff net spacing	Color-Based Span 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 Center Spacing Exception 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
End of line spacing	Preferred and Nonpreferred End-of-Line Spacing Rule
End of line spacing	Single-Side Isolated Neighbor 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	End-of-Line to Concave Corner Spacing Rule
End to end space keepout	Color-Based Line-End Alignment 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 and Extension Rules
Fat wire via asymmetric enclosure	Fat Metal Contact Asymmetric Extra Enclosure Rule

Table 2-6 Zroute Error Messages and Design Rule Titles (Continued)

Zroute DRC error message	Design rule title
Fat wire via keepout enclosure	Fat Wire Via Enclosure Rule
Fat wire via keepout enclosure	Sparse-Dense Via Enclosure Rule
Forbidden mask	Color-Based Forbidden Routing 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
Interlayer via enclosure	Interlayer Via Enclosure Rule
Illegal dimension route	Discrete Metal Dimensions Rule
Less than minimum area	Color-Based Minimum Area Rule
Less than minimum area	Edge-Based Multistage Minimum Area Rule
Less than minimum edge length	Adjacent Edge Length Rule
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
Mask end of line enclosure	Color-Based Via Enclosure Rule
Max number of via-metal concave corners	Via to Concave Corners Maximum Limit Rule
Metal corner keepout	Color-Based Metal Corner Keepout Rule
Metal corner keepout	Fat Metal Corner Keepout Rule
Metal corner preferred-direction keepout	Preferred and Nonpreferred Corner-to-Corner Spacing Rule
Metal to cut metal keepout	Minimum Metal to Cut-Metal Keepout and Overlap Rule

Table 2-6 Zroute Error Messages and Design Rule Titles (Continued)

Zroute DRC error message	Design rule title
Need fat contact	Alternative Syntax for Fat Metal Contact and Extension Rules
Need fat contact	Fat Metal Contact Rules
Need fat contact on extension	Alternative Syntax for Fat Metal Contact and Extension Rules
Polygon not rectangle	Rectangle-Only Rule
Range enclosed via spacing	Range-Enclosed Via Spacing Rule
Same color via-cut spacing	Color-Based Via Centerline Spacing Rule
Same net spacing	Color-Based Span Spacing Rule
Same net spacing	Fat Metal Forbidden Spacing Rule
Same net via cut spacing	Via Center Spacing Exception Rule
Self aligned via cluster	Self-Aligned Via Cluster Rules
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 enclosed metal minimum area	Enclosed Via Metal Minimum Area Rule
Via enclosure to metal spacing	Enclosure-Dependent Metal Spacing Rule
Via metal concave corner	Via-to-Jog Metal Enclosure Rule
Wide metal jog	Fat Metal Jog Rule

A

Classic Router Design Rules

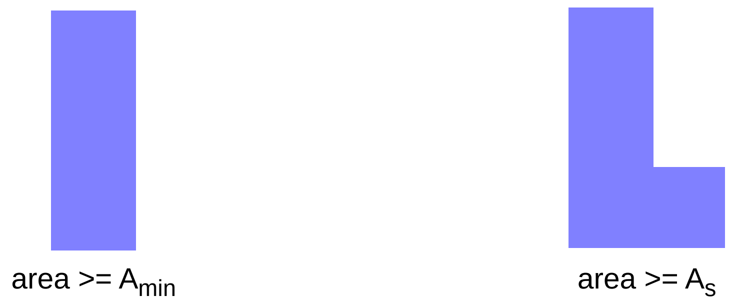
The following topics describe the routing design rules that are supported only by the classic router in the IC Compiler tool.

Most of these routing design rules are specified in the technology file; however, some 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.

Rectangle-Based Special Minimum Area Rule

The rectangle-based special minimum area rule defines a special minimum area, A_s, that applies to nonrectangular polygons. The special minimum area, A_s, must be greater than the general minimum area, A_{min}.

Figure A-1 Rectangle-Based Special Minimum Area Rule



The syntax for this rule is

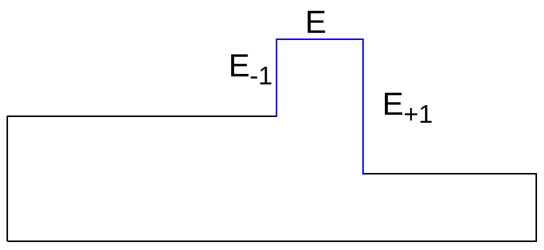
```
Technology {
   minAreaMode = 1
}
Layer "MetalX"  {
   minArea              = Amin
   specialMinArea       = As
}
```

Alternative Three-Adjacent-Edge Minimum Length Rule

Note:
This rule is supported only by the classic router.

The alternative 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 A-2.

Figure A-2 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.

- `Mx`*`MinEdgeLength4`*

  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:

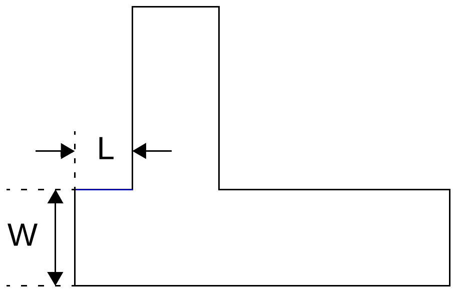
```
icc_shell> set_droute_options -name minEdgeLengthMode -value 0
icc_shell> set_droute_options -name M1MinEdgeLength4 -value 0.4
icc_shell> set_droute_options -name M1MinEdgeLength5 -value 0.26
icc_shell> set_droute_options -name M1MinEdgeLength6 -value 0.3
```

Alternative Short Edge to End-of-Line Rule

Note:
This rule is supported only by the classic router.

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 A-3, when the width of the line-end is less than W, the short edge L must be at least a specified value.

Figure A-3 Short Edge to End-of-Line Rule



For the classic router, you define this rule by using the `minEdgeLengthMode`, `M`*`n`*`MinEdgeLength4`, and `M`*`n`*`MinEdgeLength5` detail route options.

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

- `M`*`n`*`MinEdgeLength4` defines the short edge length
- `M`*`n`*`MinEdgeLength5` 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
```

Alternative End-of-Line to End-of-Line Spacing Rule

Note:
This rule is supported only by the classic router.

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.

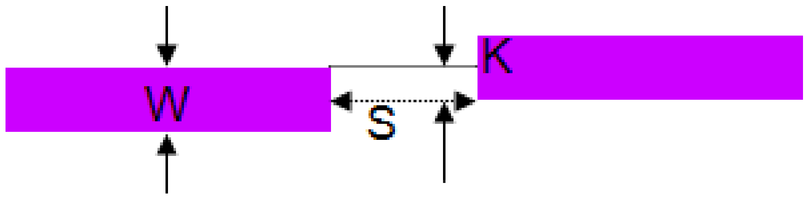
Also, you must specify a value of 1 for the `stubMode` attribute in the `Technology` section of the technology file.

For example,

```
Technology {
  stubMode   = 1
}
Layer "M2" {
  layerNumber                    = 3
  maskName                       = "metal2"
  stubSpacing                    = 0.088
  stubToStubSpacing              = 0.104
  stubThreshold                  = 0.288
  endOfLineCornerKeepoutWidth    = -0.002
}
```

Figure A-4 shows how this syntax is interpreted.

Figure A-4 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.

End-of-Line Depth Rule

Note:
This rule is supported only by the classic router.

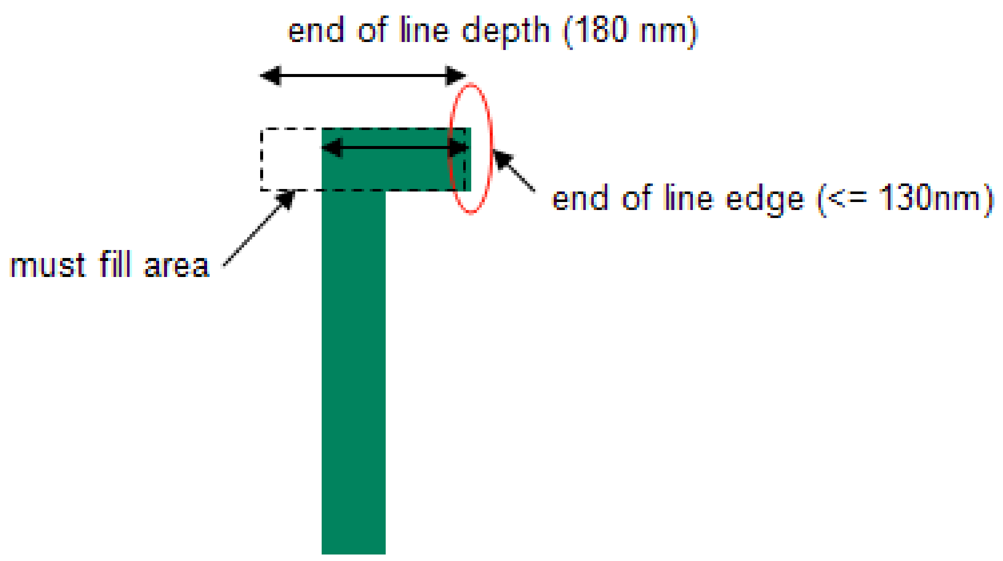
Short segments near line ends cannot always be resolved on the wafer and can create problems for optical control systems. To avoid this situation, use the end-of-line depth rule. This rule specifies that the distance from the end-of-line edge to the opposite internal edge is to be greater than or equal 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 A-5 shows the attributes associated with the end-of-line depth rule.

Figure A-5 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.

Neighboring Layer Fat Metal Extension Spacing Rule

Note:
This rule is supported only by the classic router.

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.

Use the following detail route options to specify the neighboring layer fat extension spacing rule:

```
neighboringLayerFatThreshold
neighboringLayerFatExtensionRange
```

```
neighboringLayerM1RecommendedSpacing
neighboringLayerM2RecommendedSpacing
...
neighboringLayerM12RecommendedSpacing
```

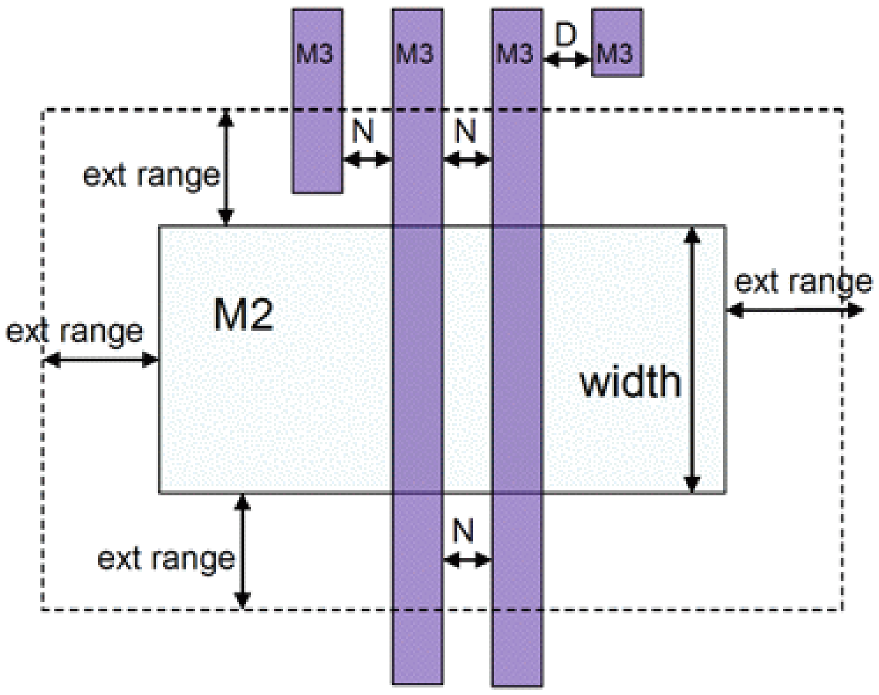
You set these options with the `set_droute_options` command in the IC Compiler tool. These option settings are stored with the cell.

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 A-6, 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 A-6 Neighboring Layer Fat Metal Extension Spacing Rule



Metal and Via Alignment Using the Classic Router

For the classic router in the IC Compiler tool, 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 `V`*n*`NoOffGridRouting` 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 `V`*n*`NoOffGridRouting` 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.