

# **LEF/DEF 5.8 Language Reference**

**Product Version 5.8**  
**March 2018**

© 2018 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

**Trademarks:** Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

All other trademarks are the property of their respective holders.

**Restricted Print Permission:** This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

**Disclaimer:** Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

**Restricted Rights:** Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

---

# Contents

---

<u>Preface</u> .....	7
<u>What's New</u> .....	7
<u>Typographic and Syntax Conventions</u> .....	7
 <b>1</b>	
<u>LEF Syntax</u> .....	9
<u>About Library Exchange Format Files</u> .....	10
<u>General Rules</u> .....	10
<u>Character Information</u> .....	10
<u>Managing LEF Files</u> .....	11
<u>Order of LEF Statements</u> .....	12
<u>LEF Statement Definitions</u> .....	12
<u>Bus Bit Characters</u> .....	12
<u>Clearance Measure</u> .....	13
<u>Divider Character</u> .....	13
<u>Extensions</u> .....	13
<u>FIXEDMASK</u> .....	14
<u>Layer (Cut)</u> .....	15
<u>Layer (Implant)</u> .....	213
<u>Layer (Masterslice or Overlap)</u> .....	221
<u>Layer (Routing)</u> .....	251
<u>Library</u> .....	510
<u>Macro</u> .....	545
<u>Manufacturing Grid</u> .....	602
<u>Maximum Via Stack</u> .....	602
<u>Nondefault Rule</u> .....	603
<u>Property Definitions</u> .....	608
<u>Site</u> .....	609
<u>Units</u> .....	612
<u>Use Min Spacing</u> .....	615
<u>Version</u> .....	616

## LEF/DEF 5.8 Language Reference

---

<u>Via</u> .....	616
<u>Via Rule</u> .....	627
<u>Via Rule Generate</u> .....	629

## 2

### ALIAS Statements .....

<u>ALIAS Statements</u> .....	635
<u>ALIAS Definition</u> .....	636
<u>ALIAS Examples</u> .....	636
<u>ALIAS Expansion</u> .....	637

## 3

### Working with LEF .....

<u>Incremental LEF</u> .....	639
<u>Error Checking</u> .....	640
<u>Message Facility</u> .....	641
<u>Error-Checking Facility</u> .....	643

## 4

### DEF Syntax .....

<u>About Design Exchange Format Files</u> .....	646
<u>General Rules</u> .....	647
<u>Character Information</u> .....	647
<u>Order of DEF Statements</u> .....	651
<u>DEF Statement Definitions</u> .....	652
<u>Blockages</u> .....	652
<u>Bus Bit Characters</u> .....	656
<u>Component Mask Shift</u> .....	656
<u>Components</u> .....	657
<u>Design</u> .....	664
<u>Die Area</u> .....	665
<u>Divider Character</u> .....	665
<u>Extensions</u> .....	665
<u>Fills</u> .....	666

## LEF/DEF 5.8 Language Reference

---

<u>GCell Grid</u>	670
<u>Groups</u>	672
<u>History</u>	673
<u>Nets</u>	673
<u>Nondefault Rules</u>	696
<u>Pins</u>	700
<u>Pin Properties</u>	717
<u>Property Definitions</u>	718
<u>Regions</u>	719
<u>Rows</u>	720
<u>Scan Chains</u>	722
<u>Slots</u>	728
<u>Special Nets</u>	729
<u>Styles</u>	746
<u>Technology</u>	758
<u>Tracks</u>	758
<u>Units</u>	760
<u>Version</u>	761
<u>Vias</u>	761

## A

<u>Examples</u>	775
<u>LEF</u>	775
<u>DEF</u>	786
<u>Scan Chain Synthesis Example</u>	791

## B

<u>Optimizing LEF Technology for Place and Route</u>	793
<u>Overview</u>	793
<u>Guidelines for Routing Pitch</u>	794
<u>Guidelines for Wide Metal Spacing</u>	796
<u>Guidelines for Wire Extension at Vias</u>	797
<u>Guidelines for Default Vias</u>	799
<u>Guidelines for Stack Vias (MAR Vias) and Samenet Spacing</u>	801
<u>Example of an Optimized LEF Technology File</u>	805

## C

<u>Calculating and Fixing Process Antenna Violations</u> .....	811
<u>Overview</u> .....	812
<u>What Are Process Antennas?</u> .....	813
<u>What Is the Process Antenna Effect (PAE)?</u> .....	814
<u>What Is the Antenna Ratio?</u> .....	815
<u>What Can Be Done to Improve the Antenna Ratio?</u> .....	815
<u>Using Process Antenna Keywords in the LEF and DEF Files</u> .....	816
<u>Calculating Antenna Ratios</u> .....	817
<u>Calculating the Antenna Area</u> .....	817
<u>Calculating a PAR</u> .....	818
<u>Calculating a CAR</u> .....	823
<u>Calculating Ratios for a Cut Layer</u> .....	831
<u>Checking for Antenna Violations</u> .....	834
<u>Area Ratio Check</u> .....	835
<u>Side Area Ratio Check</u> .....	835
<u>Cumulative Area Ratio Check</u> .....	836
<u>Cumulative Side Area Ratio Check</u> .....	837
<u>Cut Layer Process Antenna Model Examples</u> .....	837
<u>Routing Layer Process Antenna Model Examples</u> .....	838
<u>Example Using the Antenna Keywords</u> .....	844
<u>Using Antenna Diode Cells</u> .....	845
<u>Changing the Routing</u> .....	846
<u>Inserting Antenna Diode Cells</u> .....	846
<u>Using DiffUseOnly</u> .....	846
<u>Calculations for Hierarchical Designs</u> .....	847
<u>LEF and DEF Keywords for Hierarchical Designs</u> .....	848
<u>Design Example</u> .....	848
<u>Top-Down Hierarchical Design Example</u> .....	851
<u>Index</u> .....	853

---

# Preface

---

This manual is a language reference for users of the Cadence® Library Exchange Format (LEF) and Design Exchange Format (DEF) integrated circuit (IC) description languages.

LEF defines the elements of an IC process technology and associated library of cell models. DEF defines the elements of an IC design relevant to physical layout, including the netlist and design constraints. LEF and DEF inputs are in ASCII form.

This manual assumes that you are familiar with the development and design of integrated circuits.

This preface provides the following information:

- [What's New](#) on page 7
- [Typographic and Syntax Conventions](#) on page 7

## What's New

For information on what is new or changed in LEF and DEF for version 5.8 see [What's New in LEF/DEF](#).

## Typographic and Syntax Conventions

This list describes the conventions used in this manual.

`text`

Words in `monospace` type indicate keywords that you must enter literally. These keywords represent language tokens. Note that keywords are case insensitive. They are shown in uppercase in this document, but a keyword like `LAYER` can also be `Layer` or `layer` in a LEF or DEF file.

*variable*

Words in *italics* indicate user-defined information for which you must substitute a name or a value.

## LEF/DEF 5.8 Language Reference

### Preface

---

#### *objRegExpr*

An object name with the identifier *objRegExpr* represents a regular expression for the object name.

#### *pt*

Represents a point in the design. This value corresponds to a coordinate pair, such as x y. You must enclose a point within parentheses, with space between the parentheses and the coordinates. For example,

```
RECT ( 1000 2000 ) ( 1500 400 ).
```

|

Vertical bars separate possible choices for a single argument. They take precedence over any other character.

[ ]

Brackets denote optional arguments. When used with vertical bars, they enclose a list of choices from which you can choose one.

{ } . . .

Braces followed by three dots indicate that you must specify the argument at least once, but you can specify it multiple times.

{ }

Braces used with vertical bars enclose a list of choices from which you must choose one.

. . .

Three dots indicate that you can repeat the previous argument. If they are used with brackets, you can specify zero or more arguments. If they are used with braces, you must specify at least one argument, but you can specify more.

, . . .

A comma and three dots together indicate that if you specify more than one argument, you must separate those arguments with commas.

" "

Quotation marks enclose string values. Write quotation marks within a string as `\ "`. Write a backslash within a string as `\\`.

Any characters not included in the list above are required by the language and must be entered literally.



---

# LEF Syntax

---

This chapter contains information about the following topics:

- About Library Exchange Format Files on page 10
  - ❑ General Rules on page 10
  - ❑ Character Information on page 10
    - Name Escaping Semantics for LEF/DEF Files on page 11
  - ❑ Managing LEF Files on page 11
  - ❑ Order of LEF Statements on page 12
- LEF Statement Definitions on page 12
  - ❑ Bus Bit Characters on page 12
  - ❑ Clearance Measure on page 13
  - ❑ Divider Character on page 13
  - ❑ Extensions on page 13
  - ❑ Layer (Cut) on page 15
  - ❑ Layer (Implant) on page 213
  - ❑ Layer (Masterslice or Overlap) on page 221
  - ❑ Layer (Routing) on page 251
  - ❑ Library on page 510
  - ❑ Macro on page 545
    - Layer Geometries on page 582
    - Macro Obstruction Statement on page 588
    - Macro Pin Statement on page 591

- ❑ [Manufacturing Grid](#) on page 602
- ❑ [Maximum Via Stack](#) on page 602
- ❑ [Nondefault Rule](#) on page 603
- ❑ [Property Definitions](#) on page 608
- ❑ [Site](#) on page 609
- ❑ [Units](#) on page 612
- ❑ [Use Min Spacing](#) on page 615
- ❑ [Version](#) on page 616
- ❑ [Via](#) on page 616
- ❑ [Via Rule](#) on page 627
- ❑ [Via Rule Generate](#) on page 629

## About Library Exchange Format Files

A Library Exchange Format (LEF) file contains library information for a class of designs. Library data includes layer, via, placement site type, and macro cell definitions. The LEF file is an ASCII representation using the syntax conventions described in [“Typographic and Syntax Conventions”](#) on page 7.

### General Rules

Note the following information about creating LEF files:

- Identifiers like net names and cell names are limited to 2,048 characters.
- Distance is specified in microns.
- Distance precision is controlled by the `UNITS` statement.
- LEF statements end with a semicolon ( ; ). You must leave a space between the last character in the statement and the semicolon.

### Character Information

For information, see [Character Information](#) on page 647.

## **Name Escaping Semantics for LEF/DEF Files**

For information, see [Name Escaping Semantics for Identifiers](#) on page 648.

## **Managing LEF Files**

You can define all of your library information in a single LEF file; however this creates a large file that can be complex and hard to manage. Instead, you can divide the information into two files, a “technology” LEF file and a “cell library” LEF file.

A technology LEF file contains all of the LEF technology information for a design, such as placement and routing design rules, and process information for layers. A technology LEF file can include any of the following LEF statements:

```
[VERSION statement]  
[BUSBITCHARS statement]  
[DIVIDERCHAR statement]  
[UNITS statement]  
[MANUFACTURINGGRID statement]  
[USEMINSPACING statement]  
[CLEARANCEMEASURE statement ;]  
[PROPERTYDEFINITIONS statement]  
[FIXEDMASK ;]  
[LAYER (Nonrouting) statement  
 | LAYER (Routing) statement] ...  
[MAXVIASTACK statement]  
[VIA statement] ...  
[VIARULE statement] ...  
[VIARULE GENERATE statement] ...  
[NONDEFAULTRULE statement] ...  
[SITE statement] ...  
[BEGINEXT statement] ...  
[END LIBRARY]
```

A cell library LEF file contains the macro and standard cell information for a design. A library LEF file can include any of the following statements:

```
[VERSION statement]  
[BUSBITCHARS statement]  
[DIVIDERCHAR statement]  
[VIA statement] ...  
[SITE statement]  
[MACRO statement  
 | [PIN statement] ...  
 | [OBS statement ...] ] ...  
[BEGINEXT statement] ...  
[END LIBRARY]
```

When reading in LEF files, always read in the technology LEF file first.

## Order of LEF Statements

LEF files can contain the following statements. You can specify statements in any order; however, data must be defined before it is used. For example, the `UNITS` statement must be defined before any statements that use values that are dependent on `UNITS` values, `LAYER` statements must be defined before statements that use the layer names, and `VIA` statements must be defined before referencing them in other statements. If you specify statements in the following order, all data is defined before being used.

```
[VERSION statement]  
[BUSBITCHARS statement]  
[DIVIDERCHAR statement]  
[UNITS statement]  
[MANUFACTURINGGRID statement]  
[USEMINSPACING statement]  
[CLEARANCEMEASURE statement ;]  
[PROPERTYDEFINITIONS statement]  
[FIXEDMASK ;]  
[ LAYER (Nonrouting) statement  
 | LAYER (Routing) statement] ...  
[MAXVIASTACK statement]  
[VIARULE GENERATE statement] ...  
[VIA statement] ...  
[VIARULE statement] ...  
[NONDEFAULTRULE statement] ...  
[SITE statement] ...  
[MACRO statement  
  [PIN statement] ...  
  [OBS statement ...]] ...  
[BEGINEXT statement] ...  
[END LIBRARY]
```

## LEF Statement Definitions

The following definitions describe the syntax arguments for the statements that make up a LEF file. Statements are listed in alphabetical order, *not* in the order they should appear in a LEF file. For the correct order, see [“Order of LEF Statements”](#) on page 12.

### Bus Bit Characters

```
[BUSBITCHARS "delimiterPair" ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the pair of characters used to specify bus bits when LEF names are mapped to or from other databases. The characters must be enclosed in double quotation marks. For example:

```
BUSBITCHARS "[]" ;
```

If one of the bus bit characters appears in a LEF name as a regular character, you must use a backslash (\) before the character to prevent the LEF reader from interpreting the character as a bus bit delimiter.

If you do not specify the `BUSBITCHARS` statement in your LEF file, the default value is `"[]"`.

## Clearance Measure

```
[CLEARANCEMEASURE {MAXXY | EUCLIDEAN} ;]
```

Defines the clearance spacing requirement that will be applied to all object spacing in the `SPACING` and `SPACINGTABLE` statements. If you do not specify a `CLEARANCEMEASURE` statement, euclidean distance is used by default.

<code>MAXXY</code>	Uses the largest x or y distances for spacing between objects.
<code>EUCLIDEAN</code>	Uses the euclidean distance for spacing between objects. That is, the square root of $x^2 + y^2$ .

## Divider Character

```
[DIVIDERCHAR "character" ;]
```

Specifies the character used to express hierarchy when LEF names are mapped to or from other databases. The character must be enclosed in double quotation marks. For example:

```
DIVIDERCHAR "/" ;
```

If the divider character appears in a LEF name as a regular character, you must use a backslash (\) before the character to prevent the LEF reader from interpreting the character as a hierarchy delimiter.

If you do not specify the `DIVIDERCHAR` statement in your LEF file, the default value is `"/"`.

## Extensions

```
[BEGINEXT "tag"  
         extension
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

ENDEXT]

Adds customized syntax to the LEF file that can be ignored by tools that do not use that syntax. You can also use extensions to add new syntax not yet supported by your version of LEF/DEF, if you are using version 5.1 or later.

*extension*

Specifies the contents of the extension.

*"tag"*

Identifies the extension block. You must enclose *tag* in double quotation marks.

#### Example 1-1 Extension Statement

```
BEGINEXT "1VSI Signature 1.0"
    CREATOR "company name"
    DATE "timestamp"
    REVISION "revision number"
ENDEXT
```

## FIXEDMASK

[FIXEDMASK ;]

Does not allow mask shifting. All the LEF macro pin mask assignments must be kept fixed and cannot be shifted to a different mask. The LEF macro pin shapes should all have MASK assignments, if FIXEDMASK is present. This statement should be included before the LAYER statements.

For example,

```
...
MANUFACTURINGGRID 0.001 ;
FIXEDMASK ;
LAYER xxx
```

Some technologies do not allow mask shifting for cells using multi-mask patterning. For example, the pin and routing shapes are all pre-colored and must not be shifted to other masks.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### Layer (Cut)

```
LAYER layerName
  TYPE CUT ;
  [MASK maskNum ;]
  [SPACING cutSpacing
    [CENTERTOCENTER]
    [SAMENET]
    [ LAYER secondLayerName [STACK]
      | ADJACENTCUTS {2 | 3 | 4} WITHIN cutWithin [EXCEPTSAMEPGNET]
      | PARALLELOVERLAP
      | AREA cutArea
    ]
  ;] ...
  [SPACINGTABLE ORTHOGONAL
    {WITHIN cutWithin SPACING orthoSpacing} ... ;]
  [ARRAYSPACING [LONGARRAY] [WIDTH viaWidth] CUTSPACING cutSpacing
    {ARRAYCUTS arrayCuts SPACING arraySpacing} ... ;]
  [WIDTH minWidth ;]
  [ENCLOSURE [ABOVE | BELOW] overhang1 overhang2
    [ WIDTH minWidth [EXCEPTEXTRACUT cutWithin]
    | LENGTH minLength
  ]
  ;] ...
  [PREFERENCLOSURE [ABOVE | BELOW] overhang1 overhang2 [WIDTH minWidth] ;] ...
  [RESISTANCE resistancePerCut ;]
  [PROPERTY propName propVal ;] ...
  [ACCURRENTDENSITY {PEAK | AVERAGE | RMS}
    { value
      | FREQUENCY freq_1 freq_2 ... ;
      | CUTAREA cutArea_1 cutArea_2 ... ;]
    TABLEENTRIES
      v_freq_1_cutArea_1 v_freq_1_cutArea_2 ...
      v_freq_2_cutArea_1 v_freq_2_cutArea_2 ...
      ...
    } ;]
  [DCCURRENTDENSITY AVERAGE
    { value
      | CUTAREA cutArea_1 cutArea_2 ... ;
      TABLEENTRIES value_1 value_2 ...
    } ;]
  [ANTENNAMODEL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4} ;] ...
  [ANTENNAAREARATIO value ;] ...
  [ANTENNADIFFAREARATIO {value | PWL ( ( d1 r1 ) ( d2 r2 ) ... )} ;] ...
  [ANTENNACUMAREARATIO value ;] ...
  [ANTENNACUMDIFFAREARATIO {value | PWL ( ( d1 r1 ) ( d2 r2 ) ... )} ;] ...
  [ANTENNAAREAFACOR value [DIFFUSEONLY] ;] ...
  [ANTENNACUMROUTINGPLUSCUT ;]
  [ANTENNAGATEPLUSDIFF plusDiffFactor ;]
  [ANTENNAAREAMINUSDIFF minusDiffFactor ;]
  [ANTENNAAREADIFFREDUCEPWL
    ( ( diffArea1 diffAreaFactor1 ) ( diffArea2 diffAreaFactor2 ) ... ) ; ]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[PROPERTY LEF58 TYPE
  "TYPE [TSV [LAYER bottomLayer topLayer] | PASSIVATION | MIMCAP | HIGHR]
  ;" ;]

[PROPERTY LEF58 ADJACENTFOURCUTS
  "ADJACENTFOURCUTS cutSpacing CUTCLASS {className | ALL}
    {ABOVE | BELOW}
    PARALLEL parLength WITHIN parWithin [HORIZONTAL|VERTICAL]
  ;" ;]

[PROPERTY LEF58 ANTENNAGATEPWL
  "ANTENNAGATEPWL OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}
  ((gateArea1 effectiveGateArea1) (gateArea2 effectiveGateArea2) ... )
  ;" ;]

[PROPERTY LEF58 ANTENNAGATEPLUSDIFF
  "ANTENNAGATEPLUSDIFF OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}
  {plusDiffFactor |
    PWL ((diffArea1 plusDiffProtect1) (diffArea1 plusDiffProtect2) ...)}
  ;" ;]

[PROPERTY LEF58 ARRAYSPACING
  "ARRAYSPACING [CUTCLASS className] [PARALLELOVERLAP]
    [LONGARRAY] [WIDTH viaWidth]
    [WITHIN within ARRAYWIDTH arrayWidth] CUTSPACING cutSpacing
    {ARRAYCUTS arrayCuts SPACING arraySpacing} ...
  ;..." ;]

[PROPERTY LEF58 BACKSIDE
  "BACKSIDE ;" ;]

[PROPERTY LEF58 CUTCLASS
  "CUTCLASS className WIDTH viaWidth [LENGTH viaLength] [CUTS numCut]
  ;..." ;]

[PROPERTY LEF58 CUTONCENTERLINE
  "CUTONCENTERLINE width CUTCLASS className [ABOVE | BELOW]
    [EXTRACUT cutWithin WIDTH exactWidth]
  ; " ;]

[PROPERTY LEF58 DIRECTIONALSPACING
  "DIRECTIONALSPACING cutSpacing {HORIZONTAL | VERTICAL}
    PRL prl
    CUTCLASS className1 TO className2
    [PARALLEL parLength WITHIN parWithin
      [VIAGROUP groupName]]
  ; " ;]

[PROPERTY LEF58 ENCLOSURE
  "ENCLOSURE [CUTCLASS className] [ABOVE | BELOW]
    [MINCORNER]
    {EOL eolWidth [MINLENGTH minLength]
      [EOLONLY] [SHORTEDGEONEOL] eolOverhang otherOverhang
      [SIDESPACING spacing EXTENSION backwardExt forwardExt
      |ENDSPACING spacing EXTENSION extension
    }
    [{overhang1 overhang2
      |[OFFCENTERLINE] END overhang1 SIDE overhang2}
      | HORIZONTAL overhang1 VERTICAL overhang2}
    [JOGLENGTHONLY length [INCLUDELSHAPE] ]
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[HOLLOW {HORIZONTAL|VERTICAL} length]
[ WIDTH minWidth
  [INCLUDEABUTTED]
  [EXCEPTEXTRACUT cutWithin [PRL | NOSHAREDEDGE | EXACTPRL prl]]
| LENGTH minLength
| EXTRACUT [EXTRAONLY [PRL prl]]
| REDUNDANTCUT cutWithin
| PARALLEL parLength [parLength2] WITHIN parWithin [parWithin2]
  [BELOWENCLOSURE belowEnclosure
    [ALLSIDES enclosure1 enclosure2]
  | ABOVEENCLOSURE aboveEnclosure]
| CONCAVECORNERS numCorner]}
;..." ;]

[PROPERTY LEF58 ENCLOSUREEDGE
  "ENCLOSUREEDGE [CUTCLASS className][ABOVE | BELOW] overhang
    {OPPOSITE
      {[EXCEPTEOL eolWidth] [NOCONCAVECORNER within]
        [CUTTOBELOWSPACING spacing
          [ABOVEMETAL extension]]
      | WRONGDIRECTION
    }
  | [INCLUDECORNER]
    {WIDTH [BOTHWIRE] minWidth [maxWidth]
  | SPANLENGTH minSpanLength [maxSpanLength]}
    PARALLEL parLength
    {WITHIN parWithin | WITHIN minWithin maxWithin}
    [EXCEPTEXTRACUT [cutWithin]]
    [EXCEPTTWOEDGES [exceptWithin]]
  | CONVEXCORNERS convexLength adjacentLength
    PARALLEL parWithin LENGTH length
  }
;..." ;]

[PROPERTY LEF58 ENCLOSURETABLE
  "ENCLOSURETABLE [CUTCLASS className] [MASK maskNum]
    [USEMAXWIDTH | WITHINFIRSTWIDTH]
    [DEFAULT {[ABOVE| BELOW]
      overhang1 overhang2 overhang3 overhang4}...]
  {WIDTH width {[ABOVE| BELOW]
    overhang1 overhang2 overhang3 overhang4 [MINSUM]
    [MAXLENGTH length]}
    [LAYER trimLayer OVERLAP {1|2}
      [TRIMLENGTHOUTERMETAL trimLength]]}...
  | OTHERWIDTH otherWidth [PARALLEL parLength WITHIN parWithin]
    {[ABOVE| BELOW]
      overhang1 overhang2 overhang3 overhang4 [MINSUM]}...}...
; " ;]

[PROPERTY LEF58 ENCLOSURETOJOINT
  "ENCLOSURETOJOINT [CUTCLASS className] [ABOVE | BELOW]
    toOneJointOverhang [toBothJointOverhang]
    [EOLMINLENGTH minLength]
    JOINTWIDTH jointWidth JOINTLENGTH spanLength
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
    ; " ;]
[PROPERTY LEF58 ENCLOSUREWIDTH
    "ENCLOSUREWIDTH {VIAOVERLAPONLY | USEMINWIDTH}
    ;... " ;]
[PROPERTY LEF58 EOENCLOSURE
    "EOENCLOSURE eolWidth [MINEOLWIDTH minEolWidth]
    [EQUALRECTWIDTH] [CUTCLASS className] [ABOVE | BELOW]
    {{LONGEDGEONLY | SHORTEDEGEONLY} overhang
    |overhang
        [exactOverhang
        |PARALLELEDGE parSpace EXTENSION backwardExt forwardExt
        [MINLENGTH minLength]
        |MINLENGTH minLength
        ]
    }
    ; " ;]
[PROPERTY LEF58 EOLSPACING
    "EOLSPACING cutSpacing1 cutSpacing2
    [CUTCLASS className1 [{TO className2 cutSpacing1 cutSpacing2}...]]
    ENDWIDTH eolWidth PRL prl
    ENCLOSURE smallerOverhang equalOverhang
    EXTENSION sideExt backwardExt SPANLENGTH spanLength
    ; " ;]
[PROPERTY LEF58 FORBIDDENSPACING
    "FORBIDDENSPACING CUTCLASS className minSpacing maxSpacing
    [SAMEMASK]
    [SHORTEDEGEONLY | LONGEDGEONLY]
    [PRL {prl TO prlClassName}...]
    ; " ;]
[PROPERTY LEF58 KEEPOUTZONE
    "KEEPOUTZONE CUTCLASS className1 [TO className2]
    [SAMEMASK]
    [EXCEPTEXACTALIGNED [SIDE | END] spacing]
    {EXTENSION sideExtension forwardExtension |
    ENDEXTENSION endSideExtension endForwardExtension
    SIDEEXTENSION sideSideExtension sideForwardExtension |
    HORIZONTALEXTENSION horzSideExtension horzForwardExtension
    VERTICALEXTENSION vertSideExtension vertForwardExtension}
    SPIRALEXTENSION extension [CORNERONLY]
    ; " ;]
[PROPERTY LEF58 MANUFACTURINGGRID
    "MANUFACTURINGGRID value
    ; " ;]
[PROPERTY LEF58 MAXSPACING
    "MAXSPACING spacing [CUTCLASS className]
    ; " ;]
[PROPERTY LEF58 ONEDARRAY
    "ONEDARRAY CUTCLASS className CUTSPACING cutSpacing
    ARRAYCUTS arrayCuts SPACING spacing
    LAYER secondLayerName SPACING interLayerSpacing
    ENCLOSURE overhang1 overhang2
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
    ; " ;]
[PROPERTY LEF58_ORTHOGONALSPACING
    "ORTHOGONALSPACING cutSpacing
        [CUTCLASS className] {ABOVE | BELOW}
        PARALLEL parLength WITHIN parWithin WIDTH width
        [METALPRL metalPrl METALWITHIN metalWithin]
        [HORIZONTAL | VERTICAL]
    ; " ;]
[PROPERTY LEF58_PRLTWO SIDESPACING
    "PRLTWO SIDESPACING prlSpacing nonPrlSpacing
        CUTCLASS className1 TO className2
        PRL prl1 prl2 prl3 prl4
    ; " ;]
[PROPERTY LEF58_REGION
    "REGION regionLayerName BASEDLAYER cutLayerName
    ; " ;]
[PROPERTY LEF58_SAME METALALIGNED CUTS
    "SAME METALALIGNED CUTS numCuts CUTCLASS {className | ALL}
        [ABOVE | BELOW] WIDTH width SPACING spacing
    ; " ;]
[PROPERTY LEF58_SPACING
    "SPACING cutSpacing
        [SAMEMASK
        |MAXXY
        | [CENTERTOCENTER]
        [SAMENET | SAME METAL | SAME VIA]
        [LAYER secondLayerName
            [STACK
            | ORTHOGONALSPACING orthogonalSpacing]
            | CUTCLASS className
                [SHORTEDGEONLY [PRL prl]
                | CONCAVE CORNER
                    [WIDTH width ENCLOSURE enclosure
                    EDGELENGTH edgeLength
                    | PARALLEL parLength WITHIN parWithin
                    ENCLOSURE enclosure
                    | EDGELENGTH edgeLength
                    ENCLOSURE edgeEnclosure adjEnclosure]
                | EXTENSION extension
                | NONEOLCONVEXCORNER eolWidth
                    [MINLENGTH minLength]
                | ABOVEWIDTH width [ENCLOSURE enclosure]
                | MASKOVERLAP
                | WRONGDIRECTION ]]]
        | ADJACENTCUTS {2 | 3 | 4}
            [EXACTALIGNED exactAlignedCut]
            [TWO CUTS twoCuts [TWO CUTSSPACING twoCutsSpacing]
            [SAME CUT] ]
        WITHIN cutWithin | cutWithin1 cutWithin2
        [EXCEPT SAME PGNET]
        [EXCEPT ALL WITHIN exceptAllWithin]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[ENCLOSURE [ABOVE|BELOW] enclosure]
[CUTCLASS className [TO ALL]]
[NOPRL | SIDEPARALLELOVERLAP]
[SAMEMASK]
| PARALLELOVERLAP [EXCEPTSAMENET | EXCEPTSAMEMETAL
| EXCEPTSAMEMETALOVERLAP | EXCEPTSAMEVIA]
| PARALLELWITHIN within [EXCEPTSAMENET]
| [CUTCLASS className
| [LONGEDGEONLY
| ENCLOSURE enclosure {ABOVE | BELOW}
PARALLEL parLength WITHIN parWithin]]
| SAMEMETALSHAREDEDGE parwithin [ABOVE][CUTCLASS className]
| [EXCEPTTWOEDGES] [EXCEPTSAMEVIA numCut]
| AREA cutArea]
;..." ;]
[PROPERTY LEF58_SPACINGTABLE
"SPACINGTABLE CENTERSPACING
LAYER secondLayerName
CUTCLASS {{className1 | ALL} }...
{{className2 | ALL} {cutSpacing}...}...
; " ;]
[PROPERTY LEF58_SPACINGTABLE
"SPACINGTABLE
[ORTHOGONAL
{WITHIN cutWithin SPACING orthoSpacing} ... ;
|[DEFAULT defaultCutSpacing]
[SAMEMASK]
[SAMENET | SAMEMETAL | SAMEVIA]
[LAYER secondLayerName
[NOSTACK]
[NONZEROENCLOSURE
| PRLFORALIGNEDCUT
{{className1 | ALL} TO {className2 | ALL} }...
| EXCEPTENCLOSURE exceptEnclosure]]
[CENTERTOCENTER
{{className1 | ALL}| TO {className2 | ALL}}...]
[CENTERANDEDGE [NOPRL]
{{className1 | ALL}| TO {className2 | ALL}}...]
[PRL prl [ HORIZONTAL| VERTICAL][MAXXY]
[{{className1 | ALL} TO {className2 | ALL} ccPrl
[ HORIZONTAL| VERTICAL] }...] ]
[PRLTWO SIDES
{prl1 prl2 prl3 prl4 [WITHIN within]
className1 TO className2 spacing}...]
[ENDEXTENSION extension [{TO className classExtension}...]
[SIDEXTENSION {TO className classExtension}...] ]
[EXACTALIGNEDSPACING [HORIZONTAL | VERTICAL] {className
exactAlignedSpacing}...]
[NONOPPOSITEENCLOSURESPACING
{className nonOppositeEnclosureSpacing}...]
[OPPOSITEENCLOSURERESIZESPACING
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
        {className resize1 resize2
          oppositeEnclosureResizeSpacing}...]
CUTCLASS { {className1 | ALL} [SIDE | END]}...
        {{className2 | ALL} [SIDE | END] {-|cutSpacing1}
          {-|cutSpacing2}...}...;
;..." ;
[PROPERTY LEF58 VIACLUSTER
  "VIACLUSTER CUTCLASS className
    CUTS perpendicularNumCut
    [DIAGONAL diagonalNumCut]
      [NOPRLSPACING diagonalSpacing bendSpacing WITHIN within
        EXTENSION sideExtension edgeExtension]]
    WITHIN minWithin maxWithin
  ; " ;]
[PROPERTY LEF58 VIAGROUP
  "VIAGROUP groupName CUTS numCut CUTCLASS className
    PRLTWSIDES prl1 prl2 prl3 prl4
    SPACING minSpacing maxSpacing
  ; " ;]
[PROPERTY LEF58 VIAGROUPSPACING
  "VIAGROUPSPACING spacing VIAGROUP groupName CUTS numCut
    CUTCLASS className
    [PARALLEL parLength WITHIN parWithin
      OTHERCUTCLASS otherClassName]
  ; " ;]
END layerName
```

Defines cut layers in the design. Each cut layer is defined by assigning it a name and design rules. You must define cut layers separately, with their own layer statements.

You must define layers in process order from bottom to top. For example:

```
poly      masterslice
cut01     cut
metal1    routing
cut12     cut
metal2    routing
cut23     cut
metal3    routing
```

#### ACCURRENTDENSITY

Specifies how much AC current a cut of a certain area can handle at a certain frequency. For an example using the ACCURRENTDENSITY syntax, see [Example 1-9](#) on page 260.

The ACCURRENTDENSITY syntax is defined as follows:

```
{PEAK | AVERAGE | RMS}
{ value
  | FREQUENCY freq_1 freq_2 ... ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[CUTAREA cutArea_1 cutArea_2 ... ;]
TABLEENTRIES
  v_freq_1_cutArea_1 v_freq_1_cutArea_2 ...
  v_freq_2_cutArea_1 v_freq_2_cutArea_2 ...
  ...
} ;
```

PEAK	Specifies the peak limit of the layer.
AVERAGE	Specifies the average limit of the layer.
RMS	Specifies the root mean square limit of the layer.
<i>value</i>	Specifies a maximum current limit for the layer in milliamps per square micron (mA/μm <sup>2</sup> ). <i>Type:</i> Float
FREQUENCY	Specifies frequency values, in megahertz. You can specify more than one frequency. If you specify multiple frequency values, the values must be specified in ascending order.  If you specify only one frequency value, there is no frequency dependency, and the table entries are assumed to apply to all frequencies. <i>Type:</i> Float
CUTAREA	Specifies cut area values, in square microns (μm <sup>2</sup> ). You can specify more than one cut area. If you specify multiple cut area values, the values must be specified in ascending order.  If you specify only one cut area value, there is no cut area dependency, and the table entries are assumed to apply to all cut areas. <i>Type:</i> Float
TABLEENTRIES	Defines the maximum current for each frequency and cut area pair specified in the FREQUENCY and CUTAREA statements, in mA/μm <sup>2</sup> .  The pairings define each cut area for the first frequency in the FREQUENCY statement, then the cut areas for the second frequency, and so on. The final value for a given cut area and frequency is computed from a linear interpolation of the table values. <i>Type:</i> Float

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

ANTENNAAREADIFFREDUCEPWL ( ( *diffArea1 diffAreaFactor1* )  
( *diffArea2 diffAreaFactor2* ) ... )

Indicates that the *cut\_area* is multiplied by a *diffAreaFactor* computed from a piece-wise linear interpolation, based on the diffusion area attached to the cut.

The *diffArea* values are floats, specified in microns squared. The *diffArea* values should start with 0 and monotonically increase in value to the maximum size *diffArea* possible. The *diffAreaFactor* values are floats with no units. The *diffAreaFactor* values are normally between 0.0 and 1.0. If no statement rule is defined, the *diffMetalReduceFactor* value in the PAR(*m<sub>i</sub>*) equation defaults to 1.0.

For more information on the PAR(*m<sub>i</sub>*) equation and process antenna models, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAAREAFACTOR *value* [DIFFUSEONLY]

Specifies the multiply factor for the antenna metal area calculation. DIFFUSEONLY specifies that the current antenna factor should only be used when the corresponding layer is connected to the diffusion.

*Default:* 1.0

*Type:* Float

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

**Note:** If you specify a value that is greater than 1.0, the computed areas will be larger, and violations will occur more frequently.

ANTENNAAREAMINUSDIFF *minusDiffFactor*

Indicates that the antenna ratio *cut\_area* should subtract the diffusion area connected to it. This means that the ratio is calculated as:

$$\text{ratio} = (\text{cutFactor} \times \text{cut\_area} - \text{minusDiffFactor} \times \text{diff\_area}) / \text{gate\_area}$$

If the resulting value is less than 0, it should be truncated to 0. For example, if a *via2* shape has a final ratio that is less than 0 because it connects to a diffusion shape, then the cumulative check for *metal3* (or *via3*) above the *via2* shape adds a cumulative value of 0 from the *via2* layer. (See Example 1 in [Cut Layer Process Antenna Models](#), in [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#))

*Type:* Float

*Default:* 0.0

ANTENNAAREARATIO *value*

Specifies the maximum legal antenna ratio, using the area of the metal wire that is not connected to the diffusion diode. For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

*Type:* Integer

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

ANTENNACUMAREARATIO *value*

Specifies the cumulative antenna ratio, using the area of the metal wire that is not connected to the diffusion diode. For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

*Type:* Integer

ANTENNACUMDIFFAREARATIO {*value* | PWL ( ( *d1 r1* ) ( *d2 r2* ) ... ) }

Specifies the cumulative antenna ratio, using the area of the metal wire that is connected to the diffusion diode. You can supply an explicit ratio *value* or specify piece-wise linear format (PWL), in which case the cumulative ratio is calculated using linear interpolation of the diffusion area and ratio input values. The diffusion input values must be specified in ascending order.

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNACUMROUTINGPLUSCUT

Indicates that cumulative ratio rules (that is, ANTENNACUMAREARATIO, and ANTENNACUMDIFFAREARATIO) accumulate with the previous routing layer instead of the previous cut layer. Use this to combine metal and cut area ratios into one rule.

For more information on process antenna models, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNADIFFAREARATIO {*value* | PWL ( ( *d1 r1* ) ( *d2 r2* ) ... ) }

Specifies the antenna ratio, using the area of the metal wire connected to the diffusion diode. You can supply an explicit ratio *value* or specify piece-wise linear format (PWL), in which case the ratio is calculated using linear interpolation of the diffusion area and ratio input values. The diffusion input values must be specified in ascending order.

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAGATEPLUSDIFF *plusDiffFactor*

Indicates the antenna ratio gate area includes the diffusion area multiplied by *plusDiffFactor*. This means that the ratio is calculated as:

$$\text{ratio} = \text{cut\_area} / (\text{gate\_area} + \text{plusDiffFactor} \times \text{diff\_area})$$

The ratio rules without “DIFF” (the ANTENNAAREARATIO, ANTENNACUMAREARATIO, ANTENNASIDEAREARATIO, and ANTENNACUMSIDEAREARATIO statements), are unnecessary for this layer if ANTENNAGATEPLUSDIFF is defined because a zero diffusion area is already accounted for by the ANTENNADIFF\*RATIO statements.

*Type:* Float

*Default:* 0.0



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

For more information on process antenna models, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAMODEL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}

Specifies the oxide model for the layer. If you specify an ANTENNAMODEL statement, that value affects all ANTENNA\* statements for the layer that follow it until you specify another ANTENNAMODEL statement.

**Default:** OXIDE1, for a new LAYER statement

Because LEF is sometimes used incrementally, if an ANTENNA statement occurs twice for the same oxide model, the last value specified is used. For any given ANTENNA keyword, only one value or PWL table is stored for each oxide metal on a given layer.

For an example using the ANTENNAMODEL syntax, see [Example 1-10](#) on page 265.

ARRAYSPACING

Specifies array spacing rules to use on the cut layer. An array spacing rule is intended for large vias of size 3x3 or larger.

The ARRAYSPACING syntax is defined as follows:

```
[ARRAYSPACING [LONGARRAY]
  [WIDTH viaWidth] CUTSPACING cutSpacing
  {ARRAYCUTS arrayCuts
    SPACING arraySpacing} ... ;
]
```

CUTSPACING *cutSpacing*

Specifies the edge-of-cut to edge-of-cut spacing inside one cut array.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`ARRAYCUTS arrayCuts SPACING arraySpacing`

Indicates that a large via array with a size greater than or equal to  $\text{arrayCuts} \times \text{arrayCuts}$  in both dimensions must use  $N \times N$  cut arrays (where  $N = \text{arrayCuts}$ ) separated from other cut arrays by a distance of greater than or equal to *arraySpacing*.

For example, if *arrayCuts* = 3, then 2x3 and 2x4 arrays do not need to follow the array spacing rule. However, 3x3 and 3x4 arrays must follow the rule (3x4 is legal, if the `LONGARRAY` keyword is specified), while 4x4 or 4x5 arrays are violations, unless an *arrayCuts* = 4 rule is specified. (See [Array Spacing Rule Example 1](#)).

If you specify multiple `{ARRAYCUTS ...}` statements, the *arrayCuts* values must be specified in increasing order. (See [Array Spacing Rule Example 3](#).)

Specifying more than one `ARRAYCUTS` statement creates multiple choices for via array generation.

For example, you can define an *arrayCuts* = 4 rule with *arraySpacing* = 1.0, and an *arrayCuts* = 5 rule with *arraySpacing* = 1.5. Either rule is legal, and the application should choose which rule to use (presumably based on which rule produces the most via cuts in the given via area).

`LONGARRAY`

Indicates that the via can use  $N \times M$  cut arrays, where  $N = \text{arrayCuts}$ , and  $M$  can be any value, including one that is larger than  $N$ . (See [Array Spacing Rule Example 2](#).)

`WIDTH viaWidth`

Indicates that the array spacing rules only apply if the via metal width is greater than or equal to *viaWidth*. (See [Array Spacing Rule Example 1](#).)

## Example 1-2 Array Spacing Rules

### ■ Array Spacing Rule Example 1

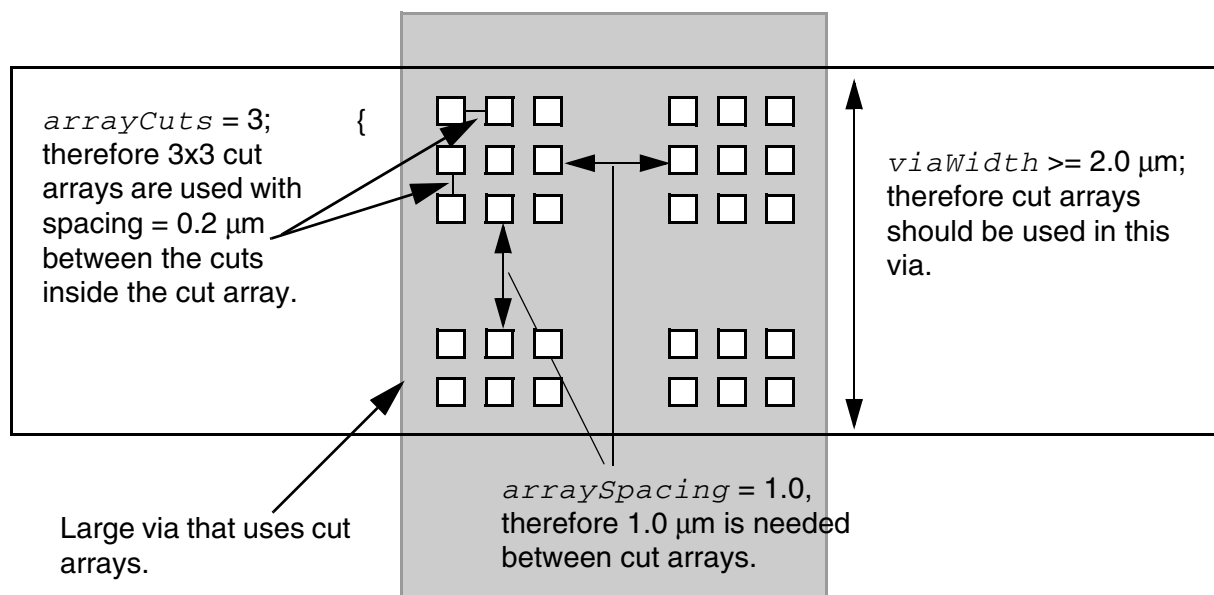
Assume the following array spacing rule exists:

```
ARRAYSPACING WIDTH 2.0 CUTSPACING 0.2 ARRAYCUTS 3 SPACING 1.0 ;
```

Any via with a metal width greater than or equal to  $2.0\ \mu\text{m}$  should use the cut spacing of  $0.2\ \mu\text{m}$  between cuts inside  $3\times 3$  cut arrays, and the cut arrays should be spaced apart by a distance of greater than or equal to  $1.0\ \mu\text{m}$  from other cut arrays. This creates the via shown in [Figure 1-1](#) on page 27.

An array of  $3\times 4$  or  $3\times 5$  cuts spaced  $0.2\ \mu\text{m}$  apart is a violation, unless the `LONGARRAY` keyword is specified. This is because the  $3\times 3$  sub-array, inside  $3\times 4$  or  $3\times 5$  cut array, does not meet  $1.0\ \mu\text{m}$  spacing from other cut arrays. Also, any larger array, such as  $4\times 4$  or  $4\times 5$  cuts, is a violation because the  $3\times 3$  sub-array inside  $4\times 4$  or  $4\times 5$  cut array requires  $1.0\ \mu\text{m}$  spacing from other cut arrays.

**Figure 1-1 Via Created With Array Spacing Width Rule**



### ■ Array Spacing Rule Example 2

The following array spacing rule is the same as Example 1, except the `LONGARRAY` keyword is present and the `WIDTH` keyword is not specified, so it creates the via shown in [Figure 1-2](#) on page 28:

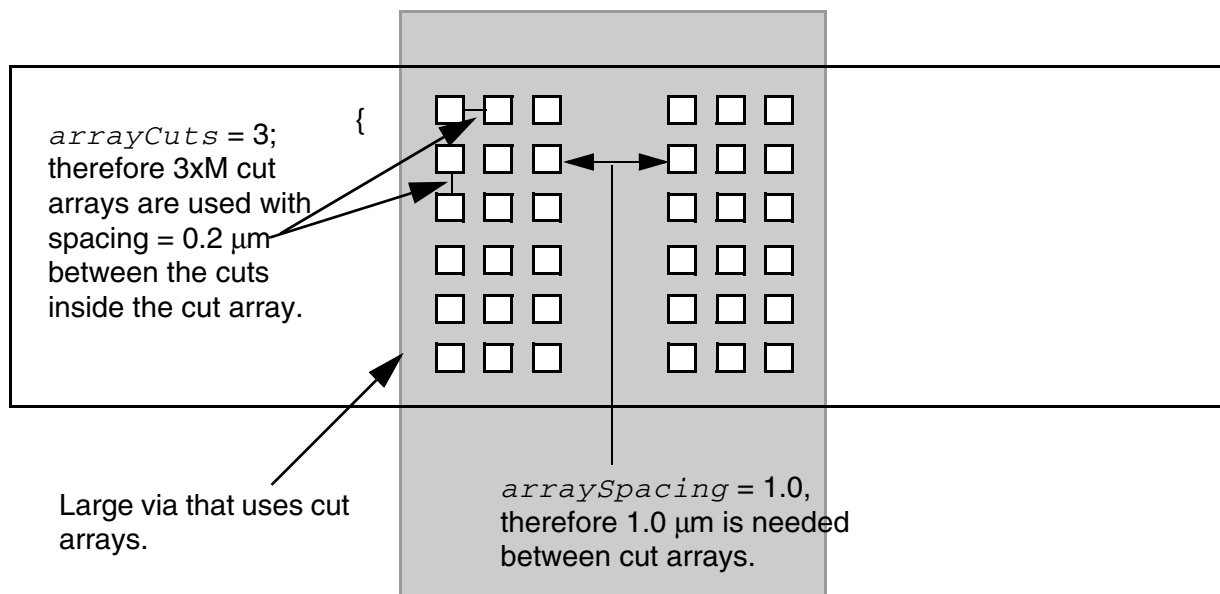
```
ARRAYSPACING LONGARRAY CUTSPACING 0.2 ARRAYCUTS 3 SPACING 1.0 ;
```

An array of  $2\times 2$ ,  $2\times 3$ , or  $2\times M$  cuts ignores this rule.

An array of 3x3 or 3xM must have 1.0  $\mu\text{m}$  spacing from other cut arrays and 0.2  $\mu\text{m}$  spacing between the cuts.

An array of 4x4 or 4xM is a violation because the array does not have 1.0  $\mu\text{m}$  space from the 3xM sub-array inside the 4xM array.

**Figure 1-2 Via Created With Array Spacing Long Array Rule**



### ■ Array Spacing Rule Example 3

Assume the following multiple array spacing rules exist:

```
ARRAYSPACING LONGARRAY CUTSPACING 0.2
  ARRAYCUTS 3 SPACING 1.0
  ARRAYCUTS 4 SPACING 1.5
  ARRAYCUTS 5 SPACING 2.0 ;
```

The application can choose between 3xM cut arrays with 1.0  $\mu\text{m}$  spacing, 4xM cut arrays with 1.5  $\mu\text{m}$  spacing, or 5xM cut arrays with 2.0  $\mu\text{m}$  spacing, using 0.2 cut-to-cut spacing inside each cut array. No `WIDTH` value indicates that any via with more than three via cuts in both dimensions (that is, 3x3 and 3x4, but not 2x4) must follow these rules.

### DCCURRENTDENSITY

Specifies how much DC current a via cut of a certain area can handle in units of milliamps per square micron ( $\text{mA}/\mu\text{m}^2$ ). For an example using the `DCCURRENTDENSITY` syntax, see [Example 1-11](#) on page 267.

The `DCCURRENTDENSITY` syntax is defined as follows:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
AVERAGE
{ value
| CUTAREA cutArea_1 cutArea_2 ... ;
  TABLEENTRIES value_1 value_2 ...
} ;
```

AVERAGE	Specifies the average limit for the layer.
<i>value</i>	Specifies a current limit for the layer in mA/μm <sup>2</sup> . <i>Type:</i> Float
CUTAREA	Specifies cut area values, in square microns. You can specify more than one cut area value. If you specify multiple cut area values, the values must be specified in ascending order. <i>Type:</i> Float
TABLEENTRIES	Specifies the maximum current density for each specified cut area, in mA/μm <sup>2</sup> . The final value for a specific cut area is computed from a linear interpolation of the table values. <i>Type:</i> Float

#### ENCLOSURE

Specifies an enclosure rule for the cut layer.

The ENCLOSURE syntax is described as follows:

```
[ENCLOSURE
 [ABOVE | BELOW] overhang1 overhang2
 [ WIDTH minWidth [EXCEPTEXTRACUT cutWithin]
 | LENGTH minLength]
;]
```

```
ENCLOSURE [ABOVE | BELOW] overhang1 overhang2
```

Indicates that any rectangle from this cut layer requires the routing layers to overhang by *overhang1* on two opposite sides, and by *overhang2* on the other two opposite sides. (See [Figure 1-3](#) on page 31.)

*Type:* Float, specified in microns

If you specify **BELOW**, the overhang is required on the routing layers below this cut layer. If you specify **ABOVE**, the overhang is required on the routing layers above this cut layer. If you specify neither, the rule applies to both adjacent routing layers.

```
WIDTH minWidth
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates that the enclosure rule only applies when the width of the routing layer is greater than or equal to *minWidth*. If you do not specify a minimum width, the enclosure rule applies to all widths (as if *minWidth* equaled 0).

*Type:* Float, specified in microns

If you specify multiple enclosure rules with the same width (or with no width), then there are several legal enclosure rules for this width, and the application only needs to meet one of the rules. If you specify multiple enclosure rules with different *minWidth* values, the largest *minWidth* rule that is still less than or equal to the wire width applies.

For example, if you specify enclosure rules for 0.0  $\mu\text{m}$ , 1.0  $\mu\text{m}$ , and 2.0  $\mu\text{m}$  widths, then a 0.5  $\mu\text{m}$  wire must meet a 0.0 rule, a 1.5  $\mu\text{m}$  wire must meet a 1.0 rule, and a 2.0  $\mu\text{m}$  wire must meet a 2.0 rule. (See [Example 1-3](#) on page 31.)

EXCEPTEXTRACUT *cutWithin*

Indicates that if there is another via cut having same metal shapes on both metal layers less than or equal to *cutWithin* distance away, this ENCLOSURE with WIDTH rule is ignored and the ENCLOSURE rules for minimum width wires (that is, no WIDTH keyword) are applied to the via cuts instead. (See [Example 1-4](#) on page 32.)

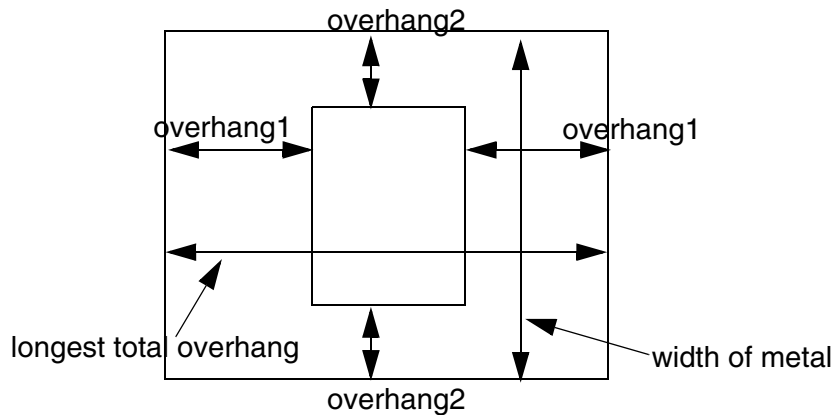
*Type:* Float, specified in microns

LENGTH *minLength*

Indicates that the enclosure rule only applies if the total length of the longest opposite-side overhangs is greater than or equal to *minLength*. The total length of the overhang is measured at the via cut center (see illustration F in [Figure 1-5](#) on page 35).

*Type:* Float, specified in microns

**Figure 1-3 Enclosure Rule**



**Example 1-3 Enclosure Rules**

- The following definition describes a cut layer that has different enclosure rules for *m1* below than for *m2* above.

```

LAYER via12
TYPE CUT ;
WIDTH 0.20 ;                               #cuts .20 x .20 squares
ENCLOSURE BELOW .03 .01 ;                   #m1: 0.03 on two opposite sides, 0.01 on other
ENCLOSURE ABOVE .05 .01 ;                   #m2: 0.05 on two opposite sides, 0.01 on other
RESISTANCE 10.0 ;                           #10.0 ohms per cut
...
END via12

```

- The following definition describes a cut layer that requires extra enclosure if the metal width is wider:

```

LAYER via23
TYPE CUT ;
WIDTH 0.20 ;                               #cuts .20 x .20 squares
SPACING 0.15                               #via23 edge-to-edge spacing is 0.15
ENCLOSURE .05 .01 ;                         #m2, m3: 0.05 on two opposite sides, 0.01 on
                                             #other sides

ENCLOSURE .02 .02 WIDTH 1.0 ;               #m2 needs 0.02 on all sides if m2 width >=1.0
                                             #m3 needs 0.02 on all sides if m3 width >=1.0
ENCLOSURE .05 .05 WIDTH 2.0 ;               #m2 needs 0.05 on all sides if m2 width >=2.0
                                             #m3 needs 0.05 on all sides if m3 width >=2.0
...
END via23

```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

- The following definition describes a cut layer that requires an overhang of .07  $\mu\text{m}$  on all sides of *metal3*, and an overhang of .09  $\mu\text{m}$  on all sides of *metal4*, if the widths of *metal3* and *metal4* are greater than or equal to 1.0  $\mu\text{m}$ :

```
LAYER via34
TYPE CUT ;
WIDTH 0.25 ;                               #cuts .25 x .25 squares
ENCLOSURE .05 .01 ;                         #minimum width enclosure rule
ENCLOSURE BELOW .07 .07 WIDTH 1.0 ; #m3 needs .07 on all sides if m3 width >=1.0
ENCLOSURE ABOVE .09 .09 WIDTH 1.0 ; #m4 needs .09 on all sides if m4 width >=1.0
...
END via34
```

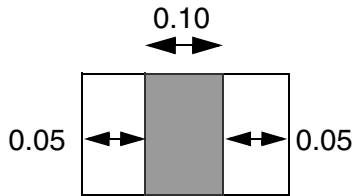
#### Example 1-4 Enclosure Rule With Width and ExceptExtraCut

The following definition describes a cut layer that requires an enclosure of either .05  $\mu\text{m}$  on opposite sides and 0.0  $\mu\text{m}$  on the other two sides, or 0.04  $\mu\text{m}$  on opposite sides and 0.01  $\mu\text{m}$  on the other two sides. It also requires an enclosure of 0.03  $\mu\text{m}$  in all directions if the wire width is greater than or equal to 0.03  $\mu\text{m}$ , unless there is an extra cut (redundant cut) within 0.2  $\mu\text{m}$ .

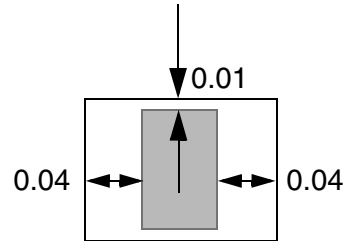
```
LAYER via34
TYPE CUT ;
WIDTH 0.10                                #cuts .10 x .10 squares
SPACING 0.10 ;                             #minimum edge-to-edge spacing is 0.10
ENCLOSURE 0.0 0.05 ;                       #overhang 0.0 0.05
ENCLOSURE 0.01 0.04 ;                      #or, overhang 0.01 0.04
#if width >= 0.3, need 0.03 0.03, unless extra cut across wire within 0.2 $\mu\text{m}$ 
ENCLOSURE 0.03 0.03 WIDTH 0.3 EXCEPTEXTRACUT 0.2 ;
...
END via34
```



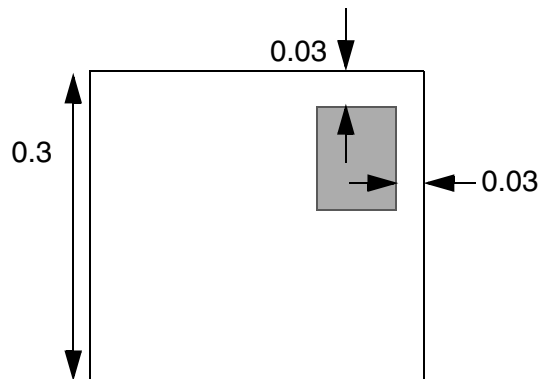
**Figure 1-4 Illustrations of Enclosure Rule With Width and ExceptExtraCut**



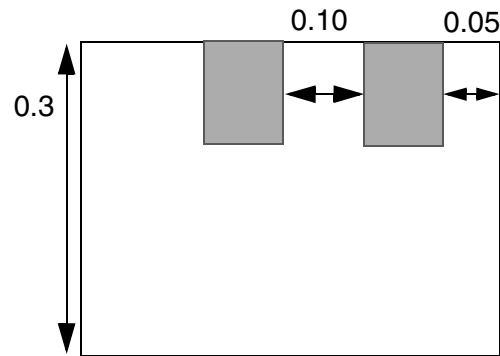
a) Okay; has 0.0 and 0.05 overhang.



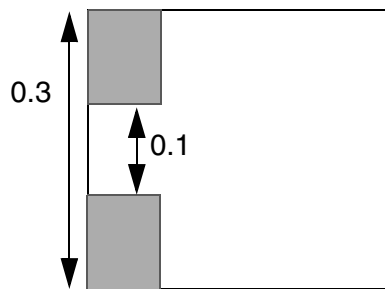
b) Okay; has 0.01 and 0.04 overhang.



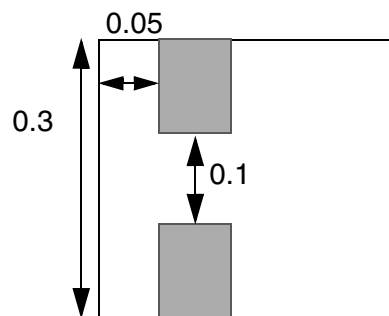
c) Okay; meets wide-wire enclosure rule of 0.03 0.03.



d) Okay; extra cut is  $\leq 0.2$  away; therefore, use min-width rule, and both cuts meet min-width enclosure rule of 0.0 and 0.5.



e) Violation. Extra cut is  $\leq 0.2$  away; therefore, use min-width rule, but cannot meet either 0.0 0.05 or 0.01 0.04 enclosure rules.



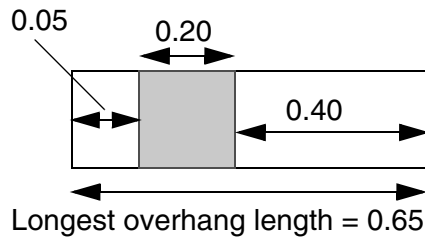
f) Okay. Extra cut is  $\leq 0.2$  away; therefore use min-width rule, and both cuts meet the min-width enclosure rule of 0.0 0.05.

### **Example 1-5 Enclosure Rule With Length and Width**

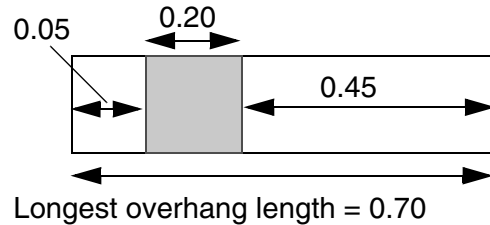
The following definition describes a cut layer that requires an enclosure of .05  $\mu\text{m}$  on opposite sides and 0.0  $\mu\text{m}$  on the other two sides, as long as the total length enclosure on any two opposite sides is greater than or equal to 0.7  $\mu\text{m}$ . Otherwise, it requires 0.05  $\mu\text{m}$  on all sides if the total enclosure length is less than or equal to 0.7  $\mu\text{m}$ . It also requires 0.10  $\mu\text{m}$  on all sides if the metal layer has a width that is greater than or equal to 1.0  $\mu\text{m}$ . ([Figure 1-5](#) on page 35 illustrates examples of violations and acceptable vias for the three ENCLOSURE rules.)

```
LAYER via34
TYPE CUT ;
WIDTH 0.20                               #cuts .20 x .20 squares
SPACING 0.20 ;                           #via34 edge-to-edge spacing is 0.20
ENCLOSURE 0.05 0.0 LENGTH 0.7 ;          #overhang 0.05 0.0 if total overhang >= 0.7
ENCLOSURE 0.05 0.05 ;                    #or, overhang 0.05 on all sides
ENCLOSURE 0.10 0.10 WIDTH 1.0 ;          #if width >= 1.0, always need 0.10
...
END via34
```

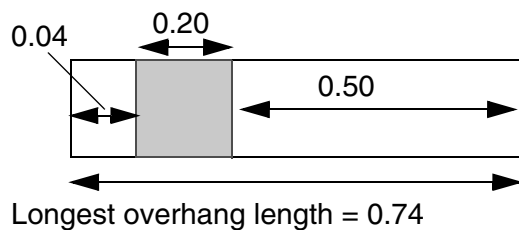
**Figure 1-5 Illustrations of Enclosure Rule With Length and Width**



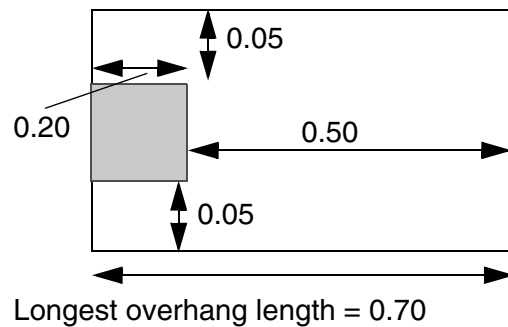
a) Violation. Longest overhang length  $< 0.70$ , and did not meet second rule of 0.05 on all sides.



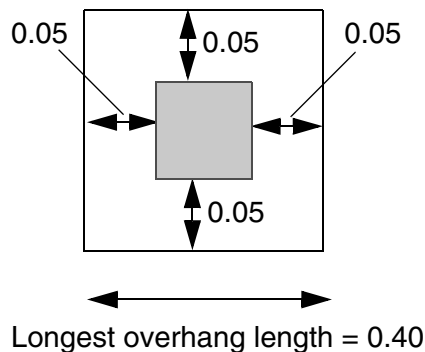
b) Okay. Longest overhang length  $\geq 0.70$ , and has 0.05 on opposite sides, and 0.0 on other sides.



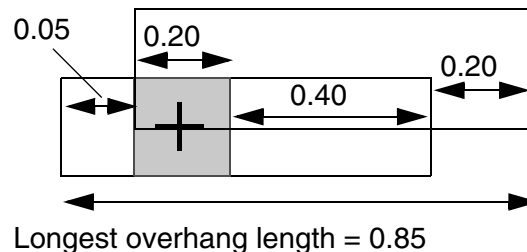
c) Violation. Longest overhang length  $\geq 0.70$ , but does not have 0.05 on opposite sides, and did not meet second rule of 0.05 on all sides.



d) Okay. Longest overhang length  $\geq 0.70$ , and has 0.05 on opposite sides, and 0.0 on other sides.



e) Okay. Total length  $< 0.7$ ; therefore first rule fails, but second rule for 0.05 on all sides is met.

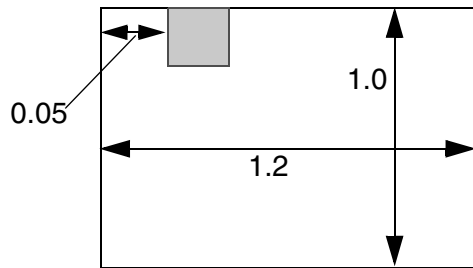


f) Okay. Overhang length  $\geq 0.70$ . (The center of the via cut is where the total overhang length is measured.)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



g) Violation. Meets first rule, but width  $\geq 1.0$ ; therefore must meet third rule: 0.10 on all sides.

LAYER *LayerName*

Specifies the name for the layer. This name is used in later references to the layer.

MASK *maskNum*

Specifies how many masks for double- or triple-patterning will be used for this layer. The *maskNum* variable must be an integer greater than or equal to 2. Most applications support values of 2 or 3 only.

PREFERENCLOSURE [ABOVE | BELOW] *overhang1 overhang2* [WIDTH *minWidth*]

Specifies preferred enclosure rules that can improve manufacturing yield, instead of enclosure rules that absolutely must be met (see the ENCLOSURE keyword). Applications should use the PREFERENCLOSURE rule when it has little or no impact on density and routability.

PROPERTY *propName propVal*

Specifies a numerical or string value for a layer property defined in the PROPERTYDEFINITIONS statement. The *propName* you specify must match the *propName* listed in the PROPERTYDEFINITIONS statement.

RESISTANCE *resistancePerCut*

Specifies the resistance per cut on this layer. LEF vias without their own specific resistance value, or DEF vias from a VIARULE without a resistance per cut value, can use this resistance value.

Via resistance is computed using *resistancePerCut* and Kirchoff's law for typical parallel resistance calculation. For example, if  $R = 10$  ohms per cut, and the via has one cut, then  $R = 10$  ohms. If the via has two cuts, then  $R = (1/2) * 10 = 5$  ohms.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### SPACING

Specifies the minimum spacing allowed between via cuts on the same net or different nets. For via cuts on the same net, this value can be overridden by a spacing with the SAMENET keyword. (See [Example 1-6](#) on page 39.)

The SPACING syntax is defined as follows:

```
[SPACING cutSpacing
  [CENTERTOCENTER]
  [SAMENET]
  [ LAYER secondLayerName [STACK]
  | ADJACENTCUTS {2 | 3 | 4} WITHIN cutWithin
    [EXCEPTSAMEPGNET]
  | PARALLELOVERLAP
  | AREA cutArea]
;] ...
```

*cutSpacing*      Specifies the default minimum spacing between via cuts, in microns.  
*Type:* Float

#### CENTERTOCENTER

Computes the *cutSpacing* or *cutWithin* distances from cut-center to cut-center, instead of from cut-edge to cut-edge (the default behavior). (See [Spacing Rule Example 4](#).)

#### SAMENET

Indicates that the *cutSpacing* value only applies to same-net cuts. The SAMENET *cutSpacing* value should be smaller than the normal SPACING *cutSpacing* value that applies to different-net cuts.

#### LAYER *secondLayerName*

Applies the spacing rule between objects on the cut layer and objects on *2ndLayerName*. The second layer must be a cut or routing layer already defined in the LEF file, or the next routing layer declared in the LEF file. This allows “one layer look ahead,” which is needed in some technologies. (See [Spacing Rule Example 1](#).)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**STACK** Indicates that same-net cuts on two different layers can be stacked if they are aligned. If the cuts are not the same size, the smaller cut must be completely covered by the larger cut, to be considered legal. If both cuts are the same size, the centers of the cuts must be aligned, to be legal; otherwise, the cuts must have *cutSpacing* between them. If *cutSpacing* is 0.0, the same-net cut vias can be placed anywhere legally, including slightly overlap case. (See [Spacing Rule Example 7.](#))

Most applications only allow spacing checks and **STACK** checking if *secondLayerName* is the cut layer below the current cut layer.

**ADJACENTCUTS** {2 | 3 | 4} WITHIN *cutWithin*

Applies the spacing rule only when the cut has two, three, or four via cuts that are less than *cutWithin* distance, in microns, from each other. You can specify only one **ADJACENTCUTS** statement per cut layer. For more information, see "[Adjacent Via Cuts.](#)"  
*Type:* Float (*distance*)

**EXCEPTSAMEPGNET**

Indicates that the **ADJACENTCUTS** rule does *not* apply between cuts, if they are on the same net, and are on a power or ground net. (See [Spacing Rule Example 5.](#))

**PARALLELOVERLAP**

Indicates that cuts on different metal shapes that have a parallel edge overlap greater than 0 require *cutSpacing* distance between them.

Only one **PARALLELOVERLAP** spacing value is allowed per cut layer. The rule does not apply to cuts that share the same metal shapes above or below that cover the overlap area between the cuts. (See [Spacing Rule Example 8.](#))

**AREA** *cutArea*

Indicates that any cut with an area greater than or equal to *cutArea* requires edge-to-edge spacing greater than or equal to *cutSpacing* to all other cuts. (See [Spacing Rule Example 6.](#))

A `SPACING` statement should already exist that applies to all cuts. Only cuts that have area greater than or equal to *cutArea* require extra spacing; therefore, *cutSpacing* for this keyword must be greater than the default spacing.

If you include `CENTERTOCENTER`, the *cutSpacing* values are computed from cut-center to cut-center, instead of from cut-edge to cut-edge.

*Type:* Float, specified in microns squared

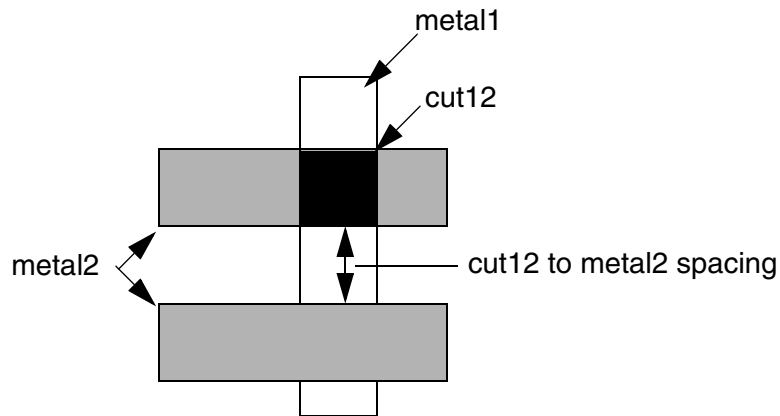
### Example 1-6 Spacing Rule Examples

#### ■ Spacing Rule Example 1

The following spacing rule defines the cut spacing required between a cut and the routing immediately above the cut. The spacing only applies to “outside edges” of the routing shape, and does not apply to a routing shape already overlapping the cut shape.

```
LAYER cut12
    SPACING 0.10 ;                #normal min cut-to-cut spacing
    SPACING 0.15 LAYER metal2 ;  #spacing from cut to routing edge above
    ...
END cut12
LAYER metal2
    ...
```

```
END metal2
```



The "SPACING 0.15 LAYER metal2 ;" rule only applies to outside edges; therefore, no violations between cut12 and the top metal2 shape will occur. Only the spacing to the bottom metal2 shape is checked.

## ■ Spacing Rule Example 2

The following spacing rule specifies that extra space is needed for any via with more than three adjacent cuts, which happens if one via has more than 2x2 cuts (see [Figure 1-6](#) on page 41). A cut that is within .25  $\mu\text{m}$  of three other cuts requires spacing that is greater than or equal to 0.22  $\mu\text{m}$ .

```
LAYER CUT12
    SPACING 0.20 ;                                #default cut spacing
    SPACING 0.22 ADJACENTCUTS 3 WITHIN 0.25 ;
    ...
END CUT12
```

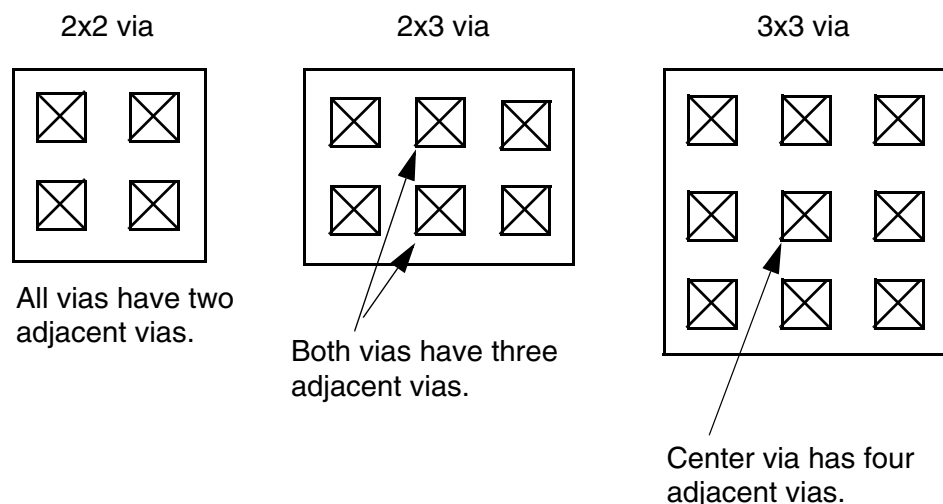
## Adjacent Via Cuts

A cut is considered adjacent if it is within *distance* of another cut in any direction (including a 45-degree angle). [Figure 1-6](#) on page 41 illustrates adjacent via cuts for 2x2, 2x3, and 3x3 vias, for typical spacing values (that is, the diagonal spacing is greater than the ADJACENTCUTS distance value). For three adjacent cuts, the ADJACENTCUTS rule allows tight cut spacing on 1x $n$  vias and 2x2 vias, but requires larger cut spacing on 2x3, 2x4 and 3x $n$  vias. For four adjacent cuts, the rule allows tight cut spacing on 2x $n$  vias, but it requires larger cut spacing on 3x $n$  vias.



The `ADJACENTCUTS` rule overrides the cut-to-cut spacing used in `VIARULE GENERATE` statements for large vias if the `ADJACENTCUTS` spacing value is larger than the `VIARULE` spacing value.

**Figure 1-6**



### ■ Spacing Rule Example 3

The following spacing rule specifies that extra space is required for any via with 3x3 cuts or more (that is, a cut with four or more adjacent cuts – see [Figure 1-6](#) on page 41). A cut that is within .25  $\mu\text{m}$  of four other cuts requires spacing that is greater than or equal to 0.22  $\mu\text{m}$ .

```
LAYER CUT12
    SPACING 0.20 ;                               #default cut spacing
    SPACING 0.22 ADJACENTCUTS 4 WITHIN 0.25 ;
    ...
END CUT12
```

### ■ Spacing Rule Example 4

The following spacing rule indicates that center-to-center spacing of greater than or equal to 0.30  $\mu\text{m}$  is required if the center-to-center spacing to three or more cuts is less than 0.30  $\mu\text{m}$ . This is equivalent to saying a cut can have only two other cuts with center-to-center spacing that is less than 0.30  $\mu\text{m}$ .

```
SPACING 0.30 CENTERTOCENTER ADJACENTCUTS 3 WITHIN 0.30 ;
```

### ■ Spacing Rule Example 5

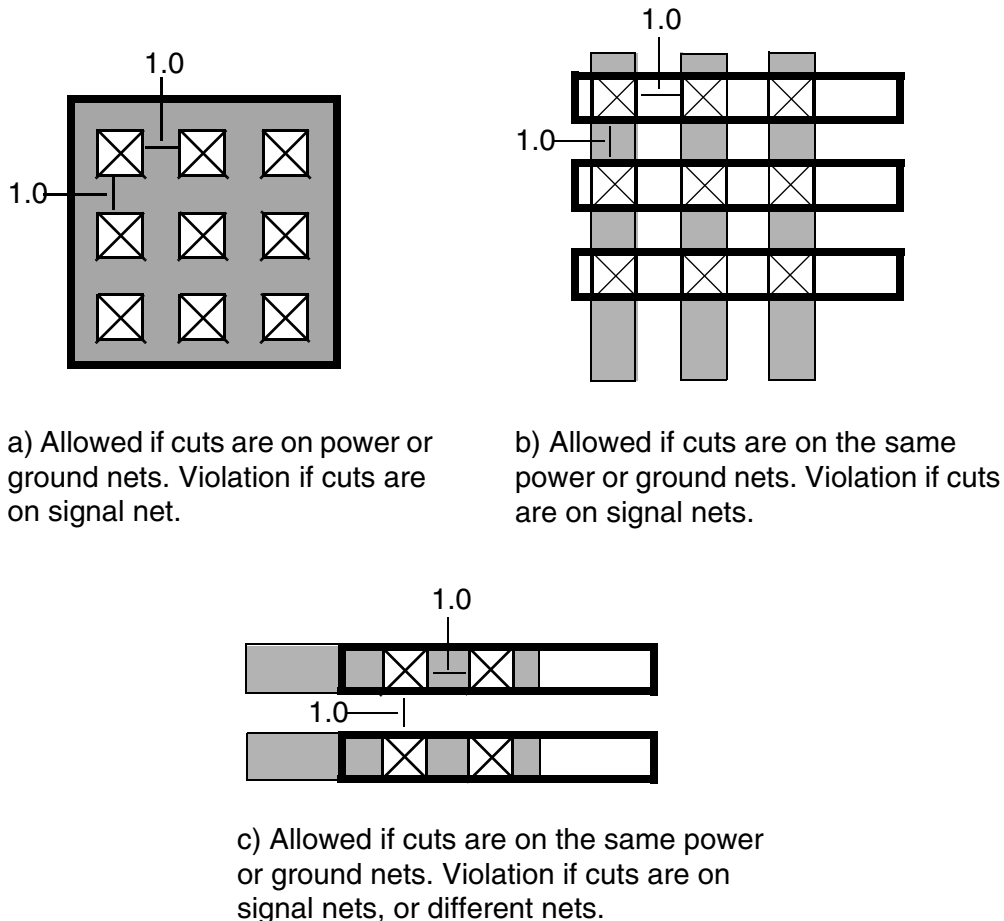
[Figure 1-7](#) on page 42 illustrates the following spacing rule:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

```
SPACING 1.0 ;  
SPACING 1.2 ADJACENTCUTS 2 WITHIN 1.5 EXCEPTSAMEPGNET ;
```

**Figure 1-7 Except Same PG Net Rule**



#### ■ Spacing Rule Example 6

The following spacing rule indicates that normal cuts require 0.10  $\mu\text{m}$  edge-to-edge spacing, and cuts with an area greater than or equal to 0.02  $\mu\text{m}^2$  require 0.12  $\mu\text{m}$  edge-to-edge spacing to all other cuts:

```
SPACING 1.0 ;  
SPACING 0.12 AREA 0.02 ;
```

#### ■ Spacing Rule Example 7

The following spacing rule indicates *cut23* cuts must be 0.20  $\mu\text{m}$  from *cut12* cuts unless they are exactly aligned:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

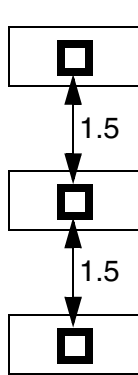
```
LAYER cut23 ;  
SPACING 0.20 SAMENET LAYER cut12 STACK ;
```

#### ■ Spacing Rule Example 8

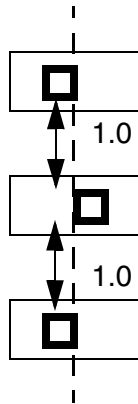
Figure 1-8 on page 44 illustrates the following spacing rule:

```
SPACING 1.0 ;  
SPACING 1.5 PARALLELOVERLAP ;
```

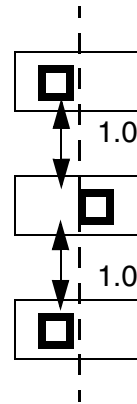
Figure 1-8 Parallel Overlap Rule



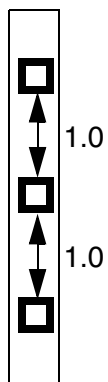
a) Okay. Cuts have parallel overlap  $> 0$ ; therefore `PARALLELOVERLAP` rule of 1.5 applies.



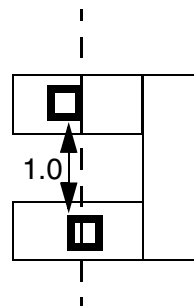
b) Okay. Cuts have parallel overlap  $= 0$ ; therefore `PARALLELOVERLAP` rule does not apply, and only 1.0 spacing is needed.



c) Okay. Cuts have no parallel overlap; therefore `PARALLELOVERLAP` rule does not apply, and only 1.0 spacing is needed.



d) Okay. Cuts overlap, but share the same metal above or below; therefore `PARALLELOVERLAP` rule does not apply, and only 1.0 spacing is needed.



e) Violation. Cuts must have above or below shared metal to cover the projected area between the cuts.

## SPACINGTABLE

Specifies spacing tables to use on the cut layer.

The `SPACINGTABLE` syntax is defined as follows:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
SPACINGTABLE ORTHOGONAL
  {WITHIN cutWithin SPACING orthoSpacing}...
;]
```

```
WITHIN cutWithin SPACING orthoSpacing
```

Indicates that if two cuts have parallel overlap that is greater than 0, and they are less than *cutWithin* distance from each other, any other cuts in an orthogonal direction must have greater than or equal to *orthoSpacing*. (See [Example 1-6](#) on page 39, and [Figure 1-9](#) on page 46.)

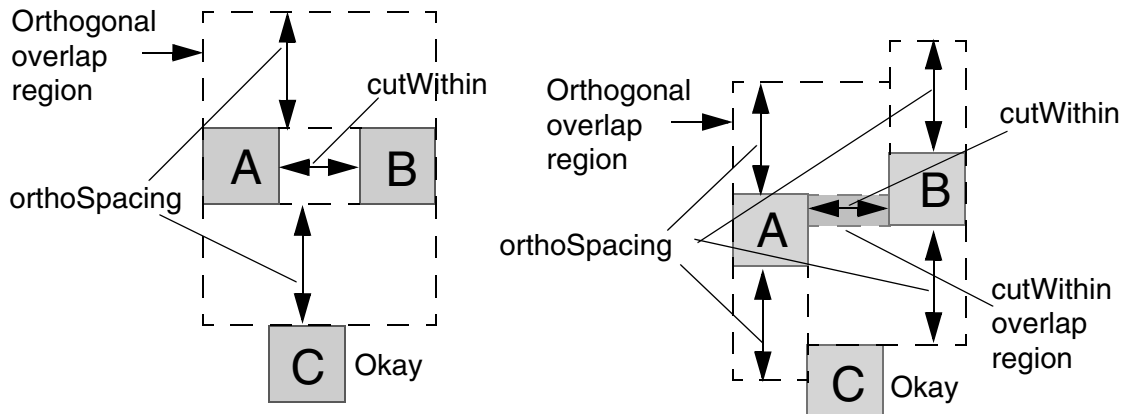
*Type:* Float, specified in microns (for both values)

#### Example 1-7 Spacing Table Orthogonal Rule

The following example shows how a spacing table orthogonal rule is defined:

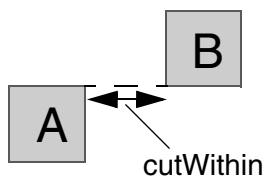
```
SPACING 0.10                                #min spacing for all cuts
SPACINGTABLE ORTHOGONAL
  WITHIN 0.15 SPACING 0.11
  WITHIN 0.13 SPACING 0.13
  WITHIN 0.11 SPACING 0.15 ;
```

**Figure 1-9 Spacing Table Orthogonal Overlap Regions**

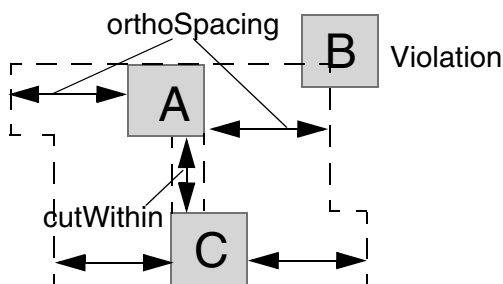


a) If two cuts < cutWithin apart and overlap, then no other cut is allowed inside the orthogonal overlap region.

b) Orthogonal overlap region is computed from the cutWithin overlap region and cuts extended out orthogonally by orthoSpacing.



c) Zero overlap of A and B; therefore no orthogonal overlap region exists.



d) The rule applies in both the X and Y directions. In this case, the horizontal overlap of A and C makes B a violation.

#### TYPE CUT

Specifies that the layer is for contact-cuts. The layer is later referenced in vias, and in rules for generating vias.

#### WIDTH *minWidth*

Specifies the minimum width of a cut. In most technologies, this is also the only legal size of a cut.

*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### Defining Cut Layer Properties to Create 32/28 nm and Smaller Nodes Rules

You can include cut layer properties in your LEF file to create 32/28 nm and smaller nodes rules that currently are not supported by existing LEF syntax. The properties are specified inside the `LAYER CUT` statements where they can be seen with other rules.

Before you can reference them, properties must be defined at the beginning of the LEF file in the `PROPERTYDEFINITIONS` statement, immediately before the first `LAYER` statement.

- Properties belong to the `LAYER` object and have a type of `STRING`.
- The property names used for these rules all start with `LEF58_`.

All properties use the following syntax within the LEF `PROPERTYDEFINITIONS` statement:

```
PROPERTYDEFINITIONS
    LAYER propName STRING ["stringValue"] ;
END PROPERTYDEFINITIONS
```

The property definitions for the cut layer properties are as follows:

```
PROPERTYDEFINITIONS
    LAYER LEF58_TYPE STRING ;
    LAYER LEF58_ADJACENTFOURCUTS STRING ;
    LAYER LEF58_ANTENNAGATEPWL
    LAYER LEF58_ANTENNAGATEPLUSDIFF
    LAYER LEF58_ARRAYSPACING STRING ;
    LAYER LEF58_BACKSIDE STRING ;
    LAYER LEF58_CUTCLASS STRING ;
    LAYER LEF58_CUTONCENTERLINE STRING ;
    LAYER LEF58_DIRECTIONALSPACING STRING ;
    LAYER LEF58_ENCLOSURE STRING ;
    LAYER LEF58_ENCLOSUREEDGE STRING ;
    LAYER LEF58_ENCLOSURETABLE STRING ;
    LAYER LEF58_ENCLOSURETOJOINT STRING ;
    LAYER LEF58_ENCLOSUREWIDTH STRING ;
    LAYER LEF58_EOLENCLOSURE STRING ;
    LAYER LEF58_EOLSPACING STRING ;
    LAYER LEF58_FORBIDDENSPACING STRING ;
    LAYER LEF58_KEEPOUTZONE STRING ;
    LAYER LEF58_MANUFACTURINGGRID STRING ;
    LAYER LEF58_MAXSPACING STRING ;
    LAYER LEF58_ONEDARRAY STRING ;
    LAYER LEF58_REGION STRING ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
LAYER LEF58_SAMEMETALALIGNEDCUTS STRING ;
LAYER LEF58_SPACING STRING ;
LAYER LEF58_SPACINGTABLE STRING ;
LAYER LEF58_VIACLUSTER STRING ;
LAYER LEF58_VIAGROUP STRING ;
LAYER LEF58_VIAGROUPSPACING STRING ;
END PROPERTYDEFINITIONS
```

### Type Rule

A type rule can be used to further classify a cut layer.

You can create a type rule by using the following property definition:

```
TYPE CUT;
  PROPERTY LEF58_TYPE
    "TYPE [TSV [LAYER bottomLayer topLayer] | PASSIVATION | MIMCAP | HIGHR]
    ;" ;
```

Where:

**HIGHR** Specifies that the layer is a special cut layer that is used to connect to a high resistance cut layer with type **HIGHR**.

**LAYER *bottomLayer topLayer***

Specifies a special TSV cut layer with cut vias that cross multiple layers from *bottomLayer* to *topLayer*. *bottomLayer* and *topLayer* must be routing layers, including the BACKSIDE routing layer.

*Type:* String

**MIMCAP** Indicates a mimcap layer, which is a cut layer that is not suitable for routing.

For example, the following rule indicates that layer VA is a mimcap cut layer:

```
LAYER VA
  TYPE CUT ;
  PROPERTY LEF58_TYPE "TYPE MIMCAP ;" ;
END VA
```

**PASSIVATION** Indicates that the cut layer is a passivation cut layer.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**TSV** Indicates that the cut layer is a through-silicon via (TSV) cut layer.

You can specify an `OVERLAP` layer name as the second layer to TSV cut layer `SPACING` rules, to indicate that the TSV (through-silicon-vias) cut must be some minimum spacing from any diffusion, poly, or well shapes. The syntax is as follows:

```
[SPACING spacing LAYER secondLayerName ;]
```

The *secondLayerName* can be a previously defined `OVERLAP` layer name, rather than a routing or cut layer name. However, if `LAYER` is specified in `TYPE` in a TSV layer, *secondLayerName* could be any routing or cut layer name between the given bottom and top layers.

Note that in many cases standard cells are designed with a small amount of diffusion, poly, or well shapes sticking outside the cell boundary (the `LEF SIZE` statement), so that must be considered.

For example, if 5  $\mu\text{m}$  spacing is needed from the TSV-cut to any poly, diffusion, or well shape, and the standard cells allow up to 0.1  $\mu\text{m}$  of diffusion, poly, or well shapes outside the cell boundary, then the rule should be specified as:

```
SPACING 5.1 LAYER OVERLAP ;
```

### **Adjacent Four Cuts Rule**

An adjacent four cuts rule can be used to describe a cut spacing rule for two sets of two cuts with `PRL` greater than zero when few other conditions are met.

You can create an adjacent four cuts rule by using the following property definition:

```
PROPERTY LEF58 ADJACENTFOURCUTS  
    "ADJACENTFOURCUTS cutSpacing CUTCLASS {className | ALL}  
      {ABOVE | BELOW}  
      PARALLEL parLength WITHIN parWithin [HORIZONTAL|VERTICAL]  
    ;" ;
```

Where:

```
ADJACENTFOURCUTS cutSpacing CUTCLASS {className | ALL}  
    {ABOVE | BELOW}  
    PARALLEL parLength WITHIN parWithin
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that if there are two sets of two cuts of cut class *className* with spacing less than *cutSpacing* and PRL greater than 0 of cuts within each sets, and each of the cuts has a neighbor cut of the other set within *parWithin* having parallel run length greater than *parLength*, it is a violation to have a wire that does not overlap or contain the cuts on the above or below metal layer in ABOVE or BELOW in a maximum bounding box formed by the facing cut edges. If ALL is specified in CUTCLASS, any cut combination of four cuts could trigger the rule.

*Type:* Float, specified in microns

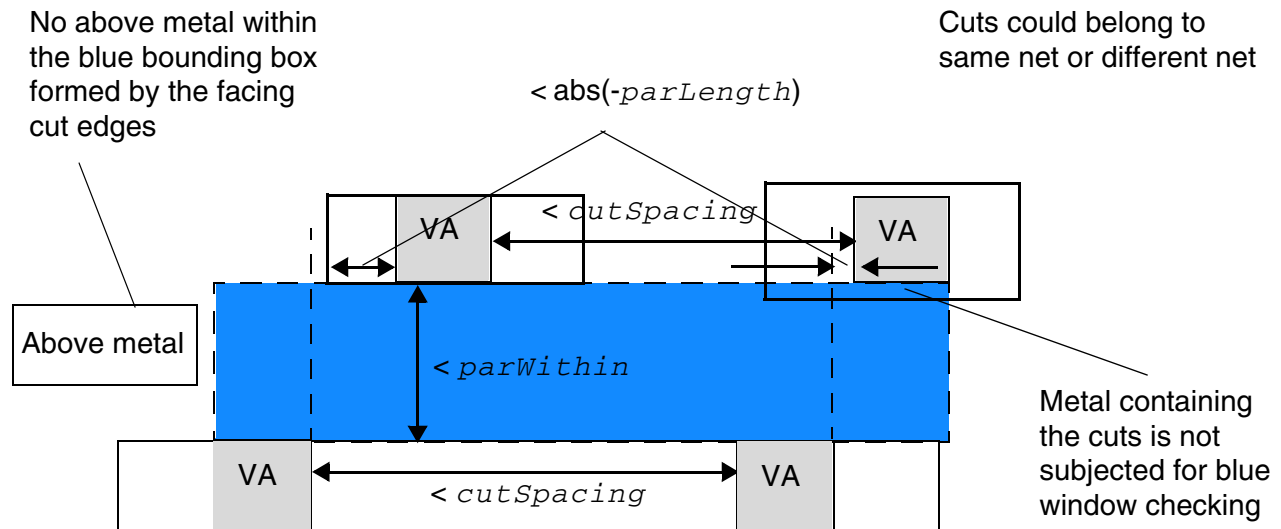
HORIZONTAL | VERTICAL Specifies that *cutSpacing* and *parLength* should be only along the given direction while *parWithin* should be only along the orthogonal direction when HORIZONTAL | VERTICAL is defined. For example, if HORIZONTAL is specified, *cutSpacing* and *parLength* should be only along the horizontal direction while *parWithin* should be only along the vertical direction.

### Adjacent Four Cuts Rule Example

- The next illustration depicts the following adjacent four cuts rule:

```
PROPERTY LEF58_ADJACENTFOURCUTS
  "ADJACENTFOURCUTS cutSpacing
    CUTCLASS VA ABOVE
    PARALLEL -parLength WITHIN parWithin HORIZONTAL
  ;";
```

**Figure 1-10 Illustration of Adjacent Four Cuts Rule**



If VERTICAL is defined, *cutSpacing* and *parLength* should be vertical while *parWithin* should be horizontal. If HORIZONTAL or VERTICAL is not defined, both cases should be checked.

### Antenna Gate PWL Rule

The antenna gate PWL rule can be used to define a PWL (piece-wise linear) table on the given antenna model that is indexed by the real gate area, and returns an “effective gate area” interpolated from the table.

You can create an antenna gate PWL rule by using the following property definition:

```
PROPERTY LEF58_ANTENNAGATEPWL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}
    "ANTENNAGATEPWL
        ((gateArea1 effectiveGateArea1) (gateArea2 effectiveGateArea2) ... ) ;" ;
```

*effectiveGateArea*

Indicates the effective gate area interpolated from the PWL table.  
If this table is not defined, the real gate area is used.  
*Type:* Float, specified in microns

*gateArea*

Indicates the real gate area.  
*Type:* Float, specified in microns

### ***Antenna Gate Plus Diffusion Rule***

The antenna gate plus diffusion rule can be used to represent the protection provided by the diffusion area that is added to the gate area value in the PAR (partial antenna ratio) equation on the given antenna model, as given below, which can be considered as the “additional effective gate-area”.

$$\text{PAR} = \{(\text{cutFactor} * \text{cut}) * \text{diffReducePWL}(\text{diff}) - (\text{minusDiffFactor} * \text{diff})\} / \{\text{gatePWL}(\text{gate}) + \text{plusDiffProtect}(\text{diff})\}$$

You can create an antenna gate plus diffusion rule by using the following property definition:

```
PROPERTY LEF58_ANTENNAGATEPLUSDIFF
  "ANTENNAGATEPLUSDIFF {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}
    {plusDiffFactor |
      PWL ((diffArea1 plusDiffProtect1) (diffArea1 plusDiffProtect2) ...)}
  ;" ;
```

Where:

All the other keywords are the same as the existing LEF cut layer ANTENNAGATEPLUSDIFF syntax.

*gatePWL(gate)* Indicates the gate area as defined in the PWL table.

*plusDiffProtect(diff)*

Indicates the diffusion area that is added to the gate area value in the PAR (partial antenna ratio) equation.

If *plusDiffFactor* is specified, then:

$$\text{plusDiffProtect} = \text{plusDiffFactor} * \text{diffArea}$$

If the PWL table is defined, then:

*plusDiffProtect* = interpolated PWL value indexed by *diffArea*.

If PAR is greater than *cutGateRatio* (from ANTENNAAREARATIO), and no diffusion is connected then there will be a violation.

If PAR is greater than *cutGateDiffRatio* (from ANTENNADIFFAREARATIO), then there will be a violation. This is checked even if the *diff\_area* = 0 (no diffusion is connected).

## **Antenna Gate Plus Diffusion Rule Examples**

The following example shows a rule for layer via1:

```
cut_area <= 100 * (funcGate(gate_area) + funcDiff(diff_area))  
funcGate = 1 for gate_area < 1, and equals 2 for gate_area >= 1  
funcDiff = 10 for diff_area < 2, and equals 50 for diff_area >= 2.
```

Then,

```
100 >= cut_area / (funcGate(gate) + funcDiff(diff))
```

This fits the PAR formula above using *gatePWL*(gate) and *plusDiffProtect*(diff) values.

Then the LEF file will have the following:

```
#Max-ratio = 100 for all diff-area values because the diff-protect value is  
#accounted for inside the PAR equation. Note, there is no need for ANTENNAAREARATIO  
#which is only checked for cut with no diff connected, while ANTENNADIFFAREARATIO  
#is checked for diff and no diff connected.  
ANTENNAMODEL OXIDE1 ;  
ANTENNADIFFAREARATIO ( 0 100 ) ( 10000.0 100 ) ;  
  
#This models funcGate(gate_area) above: 0 to 0.99 is 1, above 1.0 is 2  
PROPERTY LEF58_ANTENNAGATEPWL OXIDE1  
  "ANTENNAGATEPWL ( ( 0 1 ) ( 0.99 1 ) ( 1.0 2 ) ( 10000.0 2 ) ) ;" ;  
  
#This models funcDiff(diff_area) above: 0 to 1.99 is 10, above 1.0 is 50  
PROPERTY LEF58_ ANTENNAGATEPLUSDIFF  
  "ANTENNAGATEPLUSDIFF OXIDE1  
    (( 0 10 ) ( 1.99 10 ) ( 2.0 50 ) ( 10000.0 50 )) ;" ;
```

## **Array Spacing Rule**

You can use array spacing rules to require extra space between cut arrays, and between each cut array inside one large via. This rule only applies to large vias with many cuts; it does not apply to cuts for smaller vias.

When a cut layer has multiple cut classes defined, at the most one `ARRAYSPACING` property statement can be specified per cut class.

You can create an array spacing rule by using the following property definition:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_ARRAYSPACING
  "ARRAYSPACING [CUTCLASS className] [PARALLELOVERLAP]
    [LONGARRAY] [WIDTH viaWidth]
    [WITHIN within ARRAYWIDTH arrayWidth] CUTSPACING cutSpacing
    {ARRAYCUTS arrayCuts SPACING arraySpacing} ...
  ;..." ;
```

All other keywords are the same as the existing LEF cut layer ARRAYSPACING syntax.

Where :

CUTCLASS *className*

Defines the array spacing rule for a specific cut class (*className*). If CUTCLASS is defined in cut layer, then CUTCLASS must be specified in the ARRAYSPACING statement. Specify individual rules with the CUTCLASS keyword for each cut class, if needed.

PARALLELOVERLAP

Indicates that the array spacing rule applies only when there is a parallel edge overlap greater than 0.

WITHIN *within* ARRAYWIDTH *arrayWidth*

Indicates a specific way to check cut spacing of a via array. With WITHIN, only one ARRAYCUTS statement should be defined, and *cutSpacing* and *arraySpacing* should have the same value. Any same-net or different-net cuts within *within* would be considered as a via array. Then, a rectilinear bounding box of all of the cuts of the via array is formed. Any cuts completely inside a portion that has width less than or equal to *arrayWidth* is removed. The remaining cuts must have spacing of *cutSpacing* (or *arraySpacing*).

Type: Float, specified in microns

### Array Spacing Rule Examples

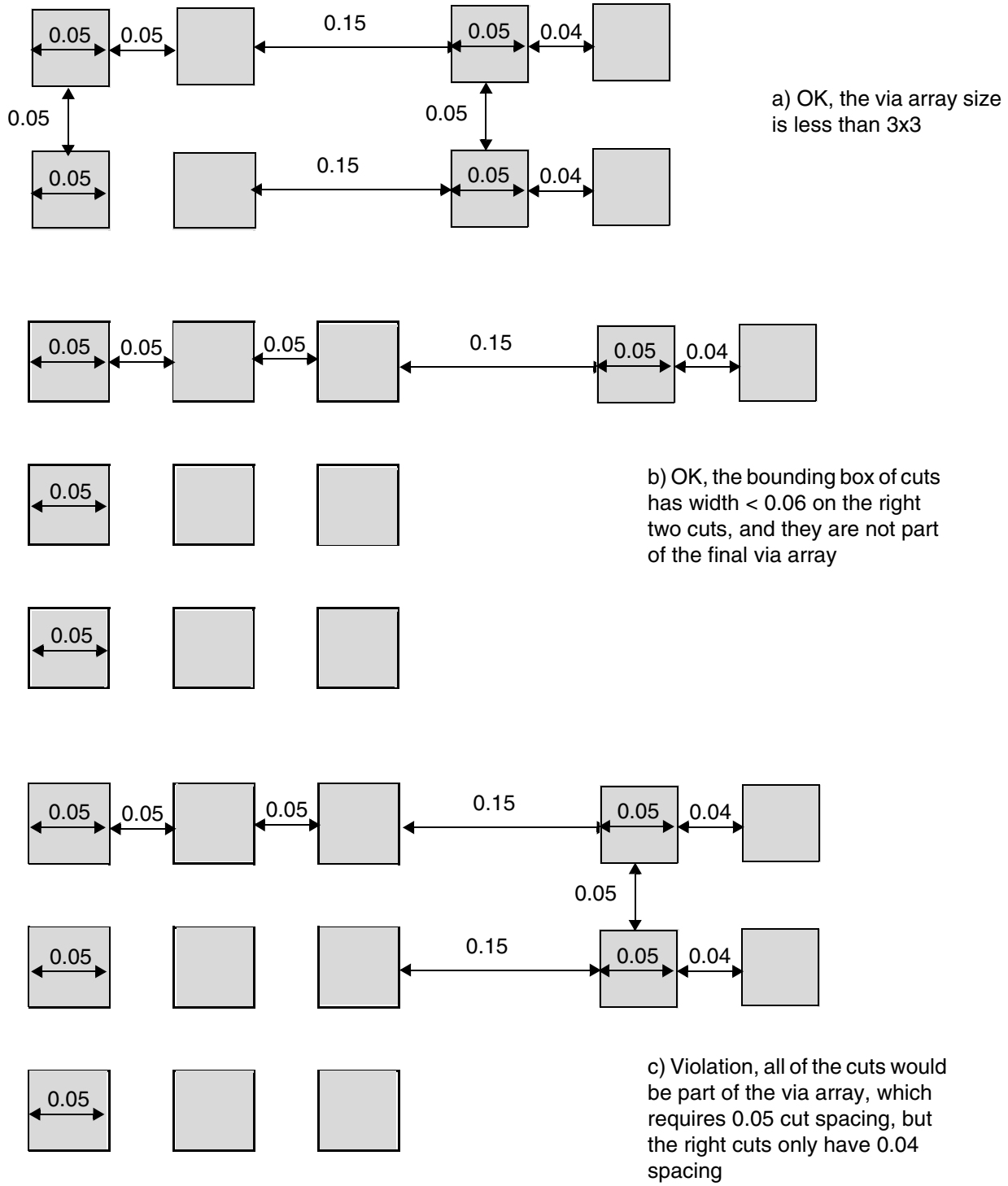
- The following array spacing rule indicates that any AY\_array via with a metal width greater than or equal to 1.0  $\mu\text{m}$  should use cut spacing of 0.10  $\mu\text{m}$  between cuts inside 3x3 cut arrays. The cut arrays should be at a distance greater than 0.30  $\mu\text{m}$  from other cut arrays with a parallel edge overlap greater than 0.

```
PROPERTY LEF58_ARRAYSPACING
  "ARRAYSPACING CUTCLASS AY_array PARALLELOVERLAP WIDTH 1.0 CUTSPACING 0.10
  ARRAYCUTS 3 SPACING 0.30 ;" ;
```

- The next example illustrates the following array spacing rule:

```
PROPERTY LEF58_ARRAYSPACING `
  ARRAYSPACING WITHIN 0.2 ARRAYWIDTH 0.06
  CUTSPACING 0.05
  ARRAYCUTS 3 SPACING 0.05 ; ` ;
```

**Figure 1-11 Illustration of Array Spacing Rule**



#### **Backside Rule**

A backside rule can be used to specify that cut layer is used on the underside of the die.

You can create a backside rule by using the following property definition:

```
PROPERTY LEF58_BACKSIDE
    "BACKSIDE ";" ;
```

Where:

BACKSIDE

Indicates that the cut layer is a backside cut layer. Only a regular cut layer or a passivation cut layer can be a backside layer; a TSV cut layer cannot be a backside layer.

#### **Cut Class Rule**

Cut class rules can be used to define the cut classes to which different types of vias can belong.

You can create a cut class rule by using the following property definition:

```
PROPERTY LEF58_CUTCLASS
    "CUTCLASS className WIDTH viaWidth [LENGTH viaLength] [CUTS numCut];..." ;
```

Where :

CUTCLASS *className*

Specifies the name of the cut class. This name is used in later references to the cut class.

WIDTH *viaWidth*

Specifies the cut width for this cut class. Any vias with cut widths of *viaWidth* belong to this class.

*Type:* Float, specified in microns

LENGTH *viaLength*

Specifies the cut length for this cut class. Any vias with cut lengths of *viaLength* belong to this class. LENGTH is only used for rectangular rather than square cuts, and *viaLength* must be greater than *viaWidth*.

*Type:* Float, specified in microns

CUTS *numCut*

Defines the equivalent number of cuts of the cut class for calculation of resistance value. Resistance value for this cut class vias would be  $(\text{resistance of the cut layer})/\text{numCut}$ .



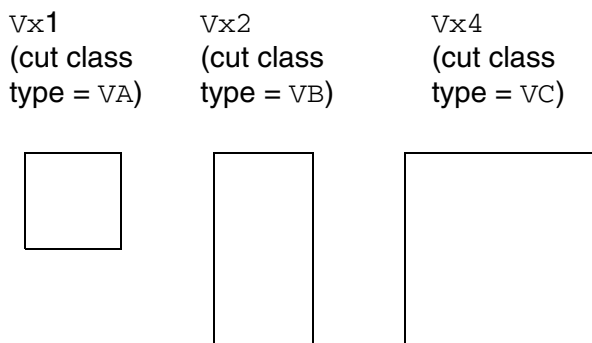
*Default: 1*

*Type: Integer*

Figure 1-12 on page 57 illustrates three via sizes. Via Vx1 contains one cut, via Vx2 contains 2 cuts, and via Vx4 contains four cuts. Using the above equation, Figure 1-12 shows how many cuts of each cut type are required to meet the listed minimum cut rule. The resistance value is determined using the following equation:

*cut layer resistance / numCut*

**Figure 1-12 Illustration of Via Sizes**



### Cut Class Rule Examples

- The following cut class rule indicates that any cut vias with a square dimension of 0.15 μm belong to cut class VA:

```
PROPERTY LEF58_CUTCLASS "CUTCLASS VA WIDTH 0.15 ;" ;
```

- The following cut class rule indicates that any cut vias with a rectangular dimension of 0.15 μm and 0.35 μm belong to cut class VB. This cut class uses 2 cuts; therefore, for a minimum cut rule requirement of two cuts, one via of this cut class is required to meet the rule. If the resistance value of a cut layer is 10 ohms, the resistance value of vias of this cut class is 1/2 x 10 = 5 ohms.

```
PROPERTY LEF58_CUTCLASS "CUTCLASS VB WIDTH 0.15 LENGTH 0.35 CUTS 2 ;" ;
```

- The following cut class rule indicates that any cut vias with a square dimension of 0.20 μm belong to cut class VC. This cut class uses 4 cuts; therefore, for a minimum cut rule requirement of 4 cuts, one via of this cut class is required to meet the rule.

```
PROPERTY LEF58_CUTCLASS "CUTCLASS VC WIDTH 0.20 CUTS 4 ;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### Cut On Center Line Rule

You can create a cut on center line rule to specify that the center of a via cut must be placed on the center lines of the wires on both metal layers if the wire width is less than the specified width.

You can create a cut on center line rule by using the following property definition:

```
PROPERTY LEF58_CUTONCENTERLINE
    "CUTONCENTERLINE width CUTCLASS className [ABOVE | BELOW]
        [EXTRACUT cutWithin WIDTH exactWidth]
        ;..." ;
```

Where:

**ABOVE | BELOW** Specifies that the cut on center line rule will be triggered based on the above or below metal wire width alone, if ABOVE or BELOW is specified.

**CUTCLASS className** Defines the cut on line center rule for a specific cut class, *className*. This CUTONLINECENTER keyword can only be defined on a cut layer with cut class definition.

**CUTONCENTERLINE width** Specifies that the center of a via cut must be placed on the centerlines of the wires on both metal layers if the wire width is less than *width*.  
*Type:* Float, specified in microns

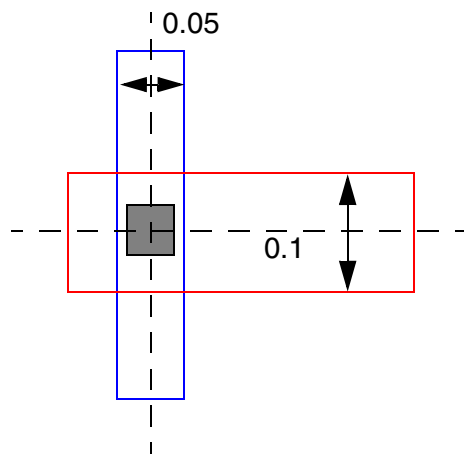
**EXTRACUT cutWithin WIDTH exactWidth** Specifies that for perfectly aligned via cuts within *cutWithin* in a wire with width equal to *exactWidth*, having the center of the bounding box of the cuts on the center lines of the wire can also meet the rule.  
*Type:* Float, specified in microns

#### Cut On Center Line Rule Examples

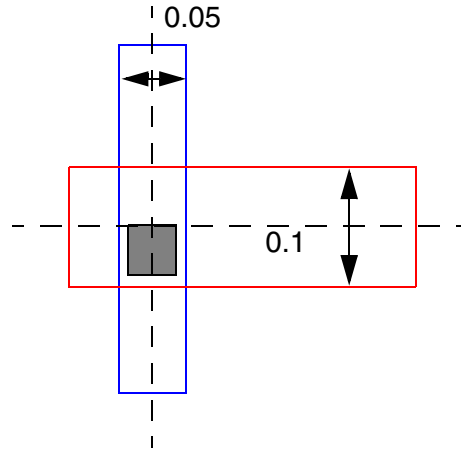
- The following cut on center line rule indicates that the center of a VA via cut must be on the centerlines of the wires on both metal layers with width less than 0.13  $\mu\text{m}$ :

```
PROPERTY LEF58_CUTONCENTERLINE
    "CUTONCENTERLINE 0.13 CUTCLASS VA ;" ;
```

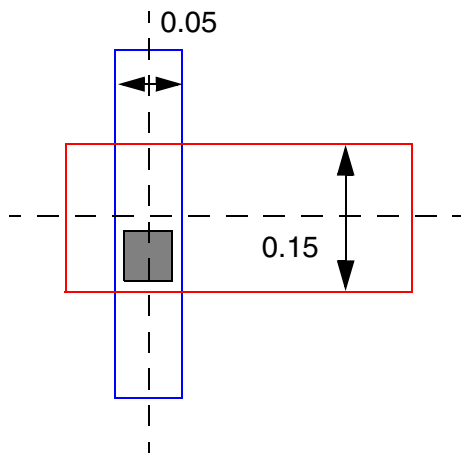
**Figure 1-13 Illustration of the Cut on Center Line Rule**



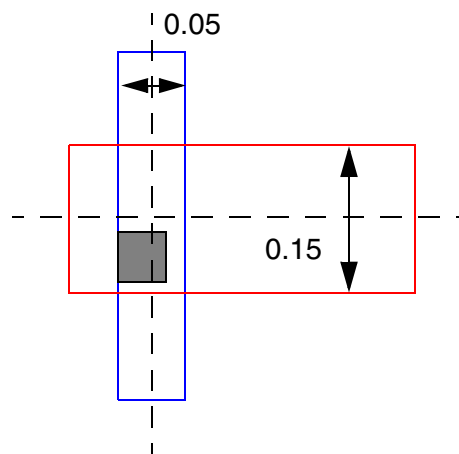
a) OK, the center of the via cut is on the centerlines of both wires



b) Violation, the center of the via cut is not on the centerline of the red wire



c) OK, the width of the red wire is 0.15 ( $\geq 0.13$ ), the centerline rule does not apply to that wire



d) Violation, the centerline rule still needs to be applied on the blue wire

### ***Directional Spacing Rule***

You can create a directional spacing rule by using the following property definition:

```
PROPERTY LEF58_DIRECTIONALSPACING
  "DIRECTIONALSPACING cutSpacing {HORIZONTAL | VERTICAL}
  PRL prl
  CUTCLASS className1 TO className2
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[PARALLEL parLength WITHIN parWithin
  [VIAGROUP groupName]]
;..." ;
```

Where:

```
DIRECTIONALSPACING cutSpacing {HORIZONTAL | VERTICAL}
  PRL pri
  CUTCLASS className1 TO className2
```

Specifies a horizontal or vertical spacing of *cutSpacing* between two cuts, one belonging to *className1* and the other belonging to *className2*, and the cuts have a parallel run length greater than *pri* vertically when HORIZONTAL is specified and horizontally when VERTICAL is specified.

*Type:* Float, specified in microns

```
PARALLEL parLength WITHIN parWithin
```

Specifies that the directional cut spacing rule applies only if a cut edge in the orthogonal direction of the spacing of a via of cut class *className1* has a neighbor cut of cut class *className2* within *parWithin* having parallel run length greater than *parLength* on the same side.

*Type:* Float, specified in microns

```
VIAGROUP groupName
```

Specifies that the directional cut spacing rule applies only if the neighbor via of cut class *className2* belongs to a via group *groupName*.

*Type:* Float, specified in microns

### Directional Spacing Rule Examples

- The following diagram illustrates the directional spacing rules given below:

```
PROPERTY LEF58_VIAGROUP "  
  VIAGROUP VGA CUTS 3 CUTCLASS VA  
    PRLTWSIDES -0.01 -0.03 -0.01 -0.03 WITHIN 0.02 0.04 ; "  
PROPERTY LEF58_DIRECTIONALSPACING "  
  DIRECTIONALSPACING 0.07 VERTICAL PRL -0.02  
    CUTCLASS VB TO VA PARALLEL -0.04 WITHIN 0.05 ;  
  DIRECTIONALSPACING 0.08 VERTICAL PRL -0.02
```

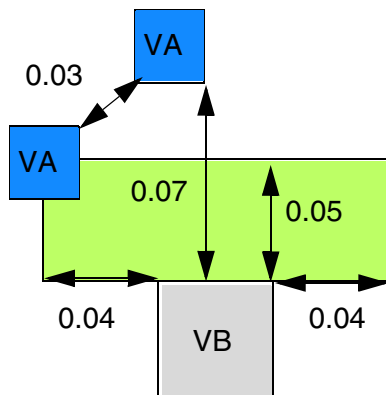
## LEF/DEF 5.8 Language Reference

### LEF Syntax

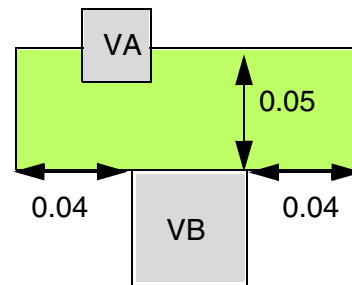
---

```
CUTCLASS VB TO VA
PARALLEL -0.04 WITHIN 0.05 VIAGROUP VGA ; " ;
```

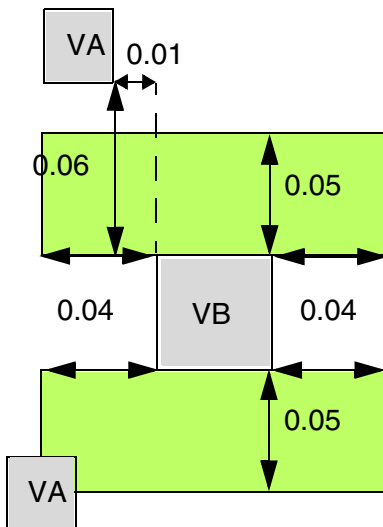
**Figure 1-14 Illustration of the Directional Spacing Rule**



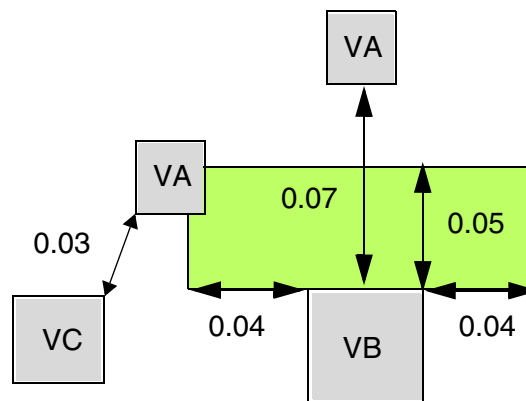
a) Violation. The first statement is met, but the second statement fails, assuming that the two blue VAs belong to via group VGA.



b) Violation. If the neighbor cut of VA overlaps with VB (within 0.02 in DIRECTIONWITHIN), cut spacing of 0.07 is applied and fails.



c) OK. The neighbor cut and the cut on which the spacing is applied must be on the same side.



d) OK. The bottom VA does not belong to the via group VGA, and the first statement is met.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### Enclosure Rule

An enclosure rule can be used to prohibit via cuts from sharing the same wire edge.

You can create an enclosure rule by using the following property definition:

```
PROPERTY LEF58_ENCLOSURE
  "ENCLOSURE [CUTCLASS className][ABOVE | BELOW]
  [MINCORNER]
  {EOL eolWidth [MINLENGTH minLength]
    [EOLONLY] [SHORTEDGEONEOL] eolOverhang otherOverhang
    [SIDESPACING spacing EXTENSION backwardExt forwardExt
    |ENDSPACING spacing EXTENSION extension
    ]
  |{overhang1 overhang2
    | [OFFCENTERLINE] END overhang1 SIDE overhang2}
    | HORIZONTAL overhang1 VERTICAL overhang2}
  [JOGLENGTHONLY length [INCLUDELSHAPE] ]
  [HOLLOW {HORIZONTAL|VERTICAL} length]
  [WIDTH minWidth
    [INCLUDEABUTTED]
    [EXCEPTEXTRACUT cutWithin [PRL | NOSHAREDEDGE | EXACTPRL prl]]
  |LENGTH minLength
  |EXTRACUT [EXTRAONLY [PRL prl]]
  |REDUNDANTCUT cutWithin
  |PARALLEL parLength [parLength2] WITHIN parWithin [parWithin2]
    [BELOWENCLOSURE belowEnclosure
      [ALLSIDES enclosure1 enclosure2]
    |ABOVEENCLOSURE aboveEnclosure]
  |CONCAVECORNERS numCorner]
  } ;..." ;
```

Where:

All other keywords are the same as the existing LEF cut layer ENCLOSURE syntax.

ABOVEENCLOSURE *aboveEnclosure*

Specifies that the enclosure rule applies only if the enclosure on the above metal layer is less than *aboveEnclosure* on the entire wire width or the edge containing the cut on either side perpendicular to the side having neighbors, or the wire direction containing the cut on the above metal layer. This means that BELOW must be defined along with it.

*Type:* Float, specified in microns

ALLSIDES *enclosure1* *enclosure2*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the rule is exempted if the below metal enclosure is greater than or equal to the minimum of the specified values, *enclosure1* and *enclosure2*, on all four sides including corners and greater than or equal to the maximum of *enclosure1* and *enclosure2* on any two opposite sides of the cut.

This exemption works independently from *belowEnclosure*. In other words, if the conditions on *ALLSIDES* are met or enclosure is greater than or equal to *belowEnclosure* on two opposite sides in the direction perpendicular to the cut side having neighbor, the rule is exempted.

*Type:* Float, specified in microns

BELOWENCLOSURE *belowEnclosure*

Specifies that the enclosure rule only applies if the enclosure on the below metal layer is less than *belowEnclosure* on the entire wire width or the edge containing the cut on either side perpendicular to the side having neighbors, or the wire direction containing the cut on the above metal layer. This means that the *ABOVE* keyword must be defined along with it.

*Type:* Float, specified in microns

CONCAVECORNERS *numCorner*

Specifies a search window by extending *overhang1* on four sides of the cut, *overhang1* must be equal to *overhang2*. If the search window finds the number of concave corners of the metal enclosing the cut to be greater than the *numCorner*, then it is a violation. If the metal corner collides with a corner of the search window, then it does not count. If the metal corner aligns with a side edge of the search window, then it will count.

*Type:* Integer

CUTCLASS *className*

Defines the enclosure rule for a specific cut class (*className*). If *CUTCLASS* is defined in cut layer, then *CUTCLASS* must be specified in *ENCLOSURE*.

Specify individual rules with the *CUTCLASS* keyword for each cut class, if needed.

END *overhang1* SIDE *overhang2*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that for rectangular cut vias, *overhang1* applies to the end edges, and *overhang2* applies to the side edges. You must use this syntax only with cut class having rectangular cut vias.

ENDSPACING *spacing* EXTENSION *extension*

Specifies the enclosure rule only applies if there is no different-net neighbors on the EOL edge with *spacing* away and extension of *extension*.

*Type:* Float, specified in microns

EOL *eolWidth* [MINLENGTH *minLength*] [EOLONLY]  
*eolOverhang otherOverhang*

Specifies the enclosure values for a cut near an end-of-line edge with length less than *eolWidth*. The first overhang, *eolOverhang*, must be applied to the EOL edge and it's opposite side while the second overhang, *otherOverhang*, must be applied to the other two sides. If EOLONLY is also specified, it should be specified for all ENCLOSURE EOL statements, and for an EOL edge, only one of the ENCLOSURE EOL statements should be observed while other ENCLOSURE statements without EOL keyword are ignored. Otherwise, fulfilling one of the ENCLOSURE with or without EOL keyword is allowed.

*Type:* Float, specified in microns

MINLENGTH specifies that an edge is EOL only if the end-of-line length is greater than or equal to *minLength* along both the sides. In the other words, if the end-of-line length is less than *minLength* along any one side, the rule does not apply.

*Type:* Float, specified in microns

EXCEPTEXTRACUT *cutWithin* [PRL | NOSHAREDEDGE | EXACTPRL *prl*]



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates that if there is another via cut having same metal shapes on both metal layers less than or equal to *cutWithin* distance away, then the ENCLOSURE with WIDTH rule is ignored, and the ENCLOSURE rules for minimum width wires (that is, no WIDTH keyword) are applied to the via cuts instead. If the NOSHAREDEDGE keyword is specified, the via cuts cannot share the same failing wire edge. (See [Figure 1-19](#) on page 74)

If the PRL keyword is used, the exemption will only be applied if there are neighbor cuts with common parallel run length greater than 0 on the opposite edges for all of the failing edges of a cut.

If the EXACTPRL keyword is used, the exemption will be applied only if the cuts have parallel run length exactly equal to *prl*.

If you have more than one ENCLOSURE statement for a given WIDTH, only one of the ENCLOSURE statements for that WIDTH needs to be met.

*Type:* Float, specified in microns (for all values)

EXTRACUT

Indicates that the enclosure rule only applies when there are two or more cuts having same metal shapes on both metal layers. If you have multiple ENCLOSURE statements (some with EXTRACUT) defined, only one of the rules need to be met.

EXTRAONLY

Specifies that vias with two or more cuts having the same metal shapes on both metal layers must follow this given enclosure statement, but not other ENCLOSURE without WIDTH statements. The ENCLOSURE WIDTH keyword will still be used on the corresponding wire width.

HOLLOW {HORIZONTAL|VERTICAL} *length*

Specifies that part of a via does not require any metal/enclosure. Take the center point of a cut and extend *length* horizontally or vertically, based on whether HORIZONTAL or VERTICAL is specified, and that portion does not require any metal/enclosure. See [Figure 1-33](#) on page 89

*Type:* Float, specified in microns

HORIZONTAL *overhang1* VERTICAL *overhang2*

Specifies that *overhang1* must be applied on the left/right opposite sides horizontally, and *overhang2* must be applied to bottom/top opposite sides vertically.

*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

INCLUDEABUTTED	Specifies that if a cut is abutted to the wide wire with width greater than or equal to <i>minWidth</i> , the corresponding overhang values will still be applied on the cut.
JOGLNGTHONLY <i>length</i> [INCLUDELSHAPE]	<p>Specifies that the enclosure rule applies only if a cut overlaps with a wrong way jog with minimum wrong way width and maximum edge length less than <i>length</i> sandwiched by two right way wires with projected PRL less than 0 in the wrong way direction in a 'Z' shape. If INCLUDELSHAPE is specified, the enclosure rule also applies if a cut overlaps with a wrong way jog with minimum wrong way width and length less than <i>length</i> connected to a right way wire in an 'L' shape.</p> <p><i>Type:</i> Float, specified in microns</p>
MINCORNER	Specifies that the enclosure in the corners only requires to cover the minimum enclosure value instead of MAXXY style of the two enclosure values.
OFFCENTERLINE	Specifies that the enclosure rule applies only for a via cut not colliding with the center line of the metal wires with width exactly equal to <i>width</i> . This keyword must be used along with CUTCLASS for rectangular cut vias and WIDTH for certain width.
PARALLEL <i>parLength</i> [ <i>parLength2</i> ] WITHIN <i>parWithin</i> [ <i>parWithin2</i> ]	

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the enclosure values must be fulfilled, but not other ENCLOSURE statements if a cut has a neighbor wire within (less than) *parWithin* having common parallel run length to the cut greater than or equal to *parLength* on one and only one side. The neighbor wire should have an edge in the preferred routing direction facing the cut to trigger the rule. In other words, that wire edge should not have a parallel run length greater than or equal to 0 with the orthogonal cut edge. If *parLength* is a negative value, the two edges will be checked in WITHIN style. In addition, the enclosure on a wide wire in ENCLOSURE WIDTH is usually larger, but this enclosure with a neighbor could be even larger. Hence, this enclosure rule should always be checked.

If a second set of length and within variables, *parLength2* and *parWithin2*, is specified, then they must be defined together. Here *parLength2* must be smaller than *parLength* and *parWithin2* must be larger than *parWithin*. Then, it will also be a violation if there is a neighbor that is greater than or equal to *parWithin* and less than *parWithin2* having a common parallel run length to the cut less than *parLength* and greater than or equal to *parLength2* on any one (or both) side(s). In other words, it is only legal if there are neighbors on both the sides based on *parWithin* and *parLength*, or there is no neighbor on either side based on *parWithin2* and *parLength2*.

*Type:* Float, specified in microns

PRL *prl*

Specifies that the enclosure rule must be applied for vias with two or more cuts if the cuts have parallel run length exactly equal to *prl*.

*Type:* Float, specified in microns

REDUNDANTCUT *cutWithin*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the enclosure on redundant cuts, which share the same metal shapes on both metal layers to a cut within *cutWithin* distance that fulfills a `ENCLOSURE` statement without `REDUNDANTCUT` keyword. This includes `ENCLOSURE` statement with `EXTRACUT` keyword. For example, when one of the cuts fulfills the `ENCLOSURE` statement using the `EXTRACUT` keyword, and is outside of a wide object, that triggers a separate `ENCLOSURE` statement with the `WIDTH` keyword, the neighbor cut within *cutWithin* distance away can merely fulfill the `ENCLOSURE` statement using the `REDUNDANTCUT` keyword, and it does not necessarily need to fulfill the `ENCLOSURE` statement with the `WIDTH` keyword.

If a separate `ENCLOSURE` statement with the `WIDTH` keyword is used, the cut in a wide object with width greater than or equal to *minWidth* should fulfill the corresponding overhang values in order for the redundant cuts to follow the overhang values in the `ENCLOSURE` statement with the `REDUNDANTCUT` keyword. You can specify only one `ENCLOSURE` statement with the `REDUNDANTCUT` keyword.

*Type:* Float, specified in microns

`SIDESPACING` *spacing* `EXTENSION` *backwardExt forwardExt*

Specifies the enclosure rule only applies if there is no different-net neighbors on the side (non-EOL) edges with spacing away and extensions from the EOL edge by *backwardExt* going backward and *forwardExt* going forward.

*Type:* Float, specified in microns

`SHORTEDGEONEOL`

Specifies that the *eolOverhang* value must be applied to the end/short edges of a rectangular cut via while the *otherOverhang* value must be applied to the other opposite sides. This keyword must be used along with `CUTCLASS` for rectangular cut vias.

### Enclosure Rule Examples

- The following enclosure rule specifies that VC vias should have 0.10 µm overhang on all four sides of the routing layers:

```
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE CUTCLASS VC 0.10 0.10 ;" ;
```

- The following enclosure rule specifies that the end/short edge must have 0.05 µm overhang to a EOL edge:

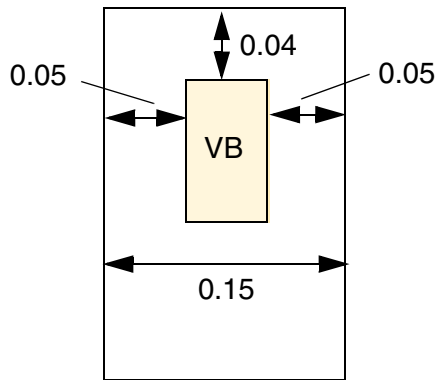
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_ENCLOSURE
  "ENCLOSURE CUTCLASS VB EOL 0.2
    SHORTEEDGEONEOL 0.05 0.04 ;" ;
```

**Figure 1-15 Illustration of Enclosure Rule With SHORTEEDGEONEOL**



a) Violation, the end/short edge must have 0.05 overhang to a EOL edge

## LEF/DEF 5.8 Language Reference

### LEF Syntax

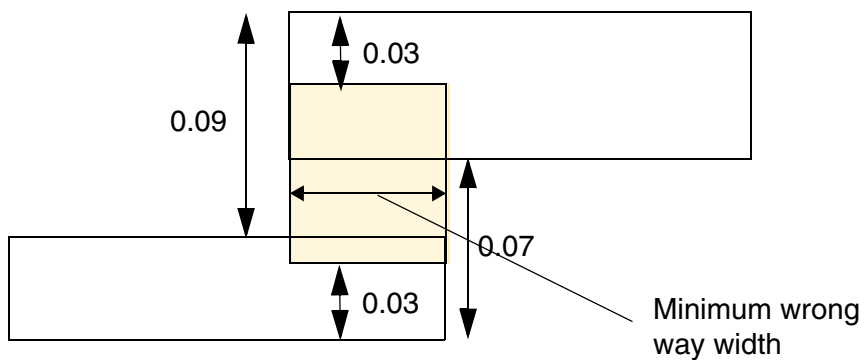
---

- The following enclosure rule illustrates JOGLENGTHONLY:

```
ENCLOSURE 0.00 0.04 ;  
PROPERTY LEF58_ENCLOSURE "  
    ENCLOSURE 0 0.03 JOGLENGTHONLY 0.1 ; "
```

**Figure 1-16 Illustration of the Enclosure Rule with JOGLENGTHONLY**

Horizontal is the preferred routing direction.



a) OK, the cut is inside a wrong way jog such that 0.03 enclosure is sufficient.

## LEF/DEF 5.8 Language Reference

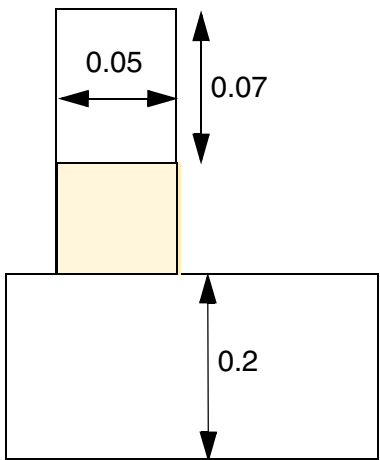
### LEF Syntax

---

- The following enclosure rule illustrates INCLUDEABUTTED:

```
ENCLOSURE 0.00 0.04 ;  
PROPERTY LEF58_ENCLOSURE  
    "ENCLOSURE 0.05 0.05  
        WIDTH 0.2 INCLUDEABUTTED ; " ;
```

**Figure 1-17 Illustration of the Enclosure Rule with INCLUDEABUTTED**



a) Violation, the cut is abutted to wide wire of 0.2. OK, if INCLUDEABUTTED is omitted.

## LEF/DEF 5.8 Language Reference

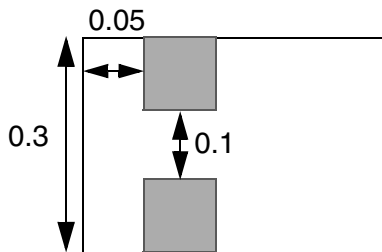
### LEF Syntax

---

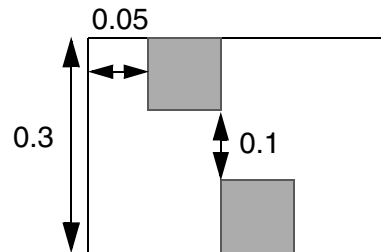
- **Figure 1-18** on page 72 illustrates the following enclosure rules:

```
ENCLOSURE 0.0 0.05 ;                #overhang 0.0 0.05
ENCLOSURE 0.02 0.02 ;                #or, overhang 0.02 0.02
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE 0.03 0.03 WIDTH 0.3 EXCEPTEXTRACUT 0.2 PRL ;" ;
```

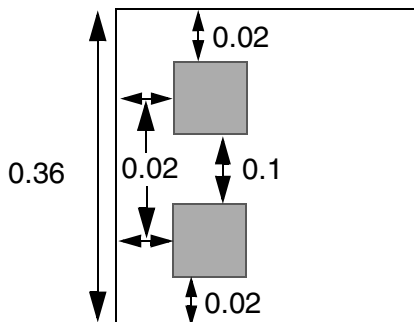
**Figure 1-18 Illustrations of the Enclosure Rule With PRL**



a) OK. Extra-cut is  $\leq 0.2$  away, with PRL  $> 0$ , so use min-width rule, both cuts meet the 0.0 0.05 enclosure rule of 0.0 0.05



b) Violation. Extra-cut is  $\leq 0.2$  away, but no PRL, so wide-wire enclosure of 0.03 0.03 is required



c) Violation, the left 0.02 failing edges do not have a neighbor cut on the opposite right edges



## LEF/DEF 5.8 Language Reference

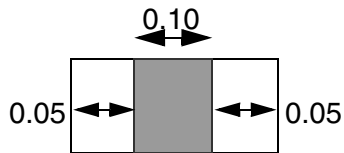
### LEF Syntax

---

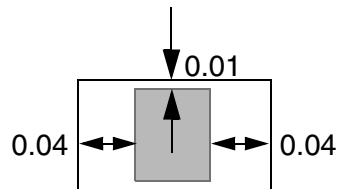
- Figure 1-19 on page 74 illustrates the following enclosure rules:

```
ENCLOSURE 0.0 0.05 ;           #overhang 0.0 0.05
ENCLOSURE 0.01 0.04 ;         #or overhang 0.01 0.04
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE 0.03 0.03 WIDTH 0.3 EXCEPTEXTRACUT 0.2 ;" ;
```

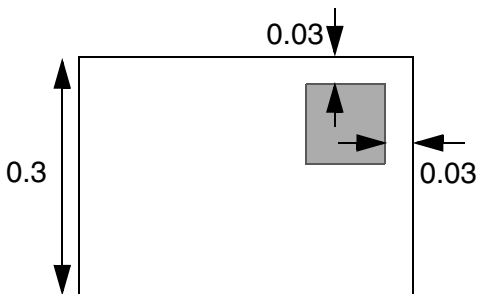
**Figure 1-19 Illustration of the Enclosure Rule With ExceptExtraCut and NoSharedEdge**



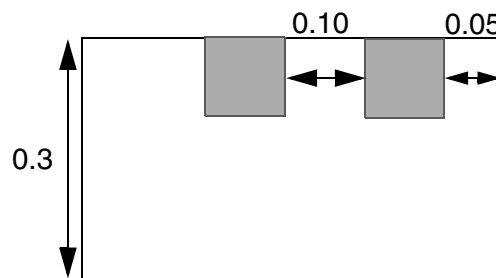
a) Okay; has 0.0 and 0.05 overhang.



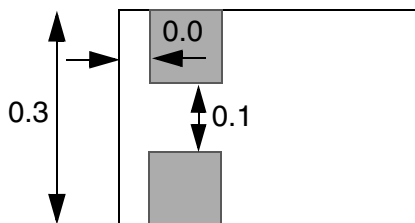
b) Okay; has 0.01 and 0.04 overhang.



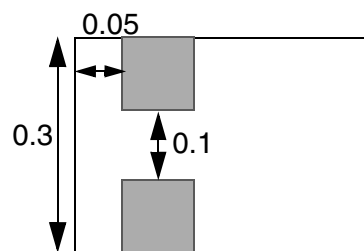
c) Okay; meets wide-wire enclosure rule of 0.03 0.03.



d) Okay. Extra cut is close enough so just need 0.0 0.5 enclosure. Violation if `NOSHAREDEDGE` is specified; extra cut has shared edge, so wide-wire enclosure of 0.03 0.03 is required.



e) Violation. Extra cut is  $\leq 0.2$  away; therefore, use minimum width rule, but cannot meet either 0.0 0.05 or 0.01 0.04 enclosure rules.



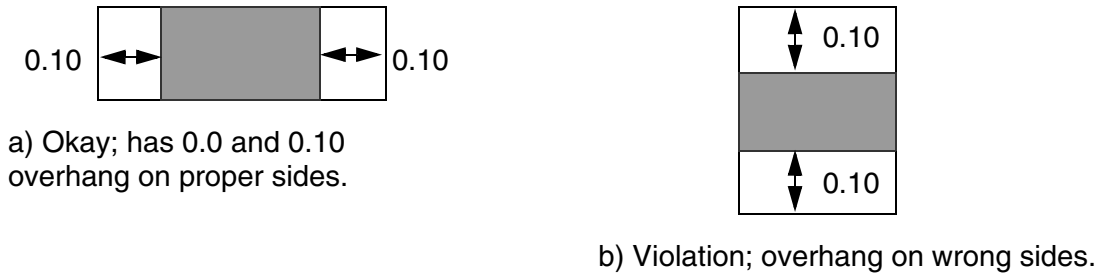
f) Okay. Okay for `NOSHAREDEDGE` also. Extra cut is  $\leq 0.2$  away, there is no shared edge; so use minimum width rule; both cuts meet the minimum width enclosure rule of 0.0 0.05.

- The following enclosure rule specifies that `VB` rectangular cut vias should have 0.10  $\mu\text{m}$  overhang on the end edges, and no overhang on the side edges on the routing layers. (See [Figure 1-20](#) on page 75.)

```
PROPERTY LEF58_ENCLOSURE
```

```
"ENCLOSURE CUTCLASS VB END 0.10 SIDE 0.000 ;" ;
```

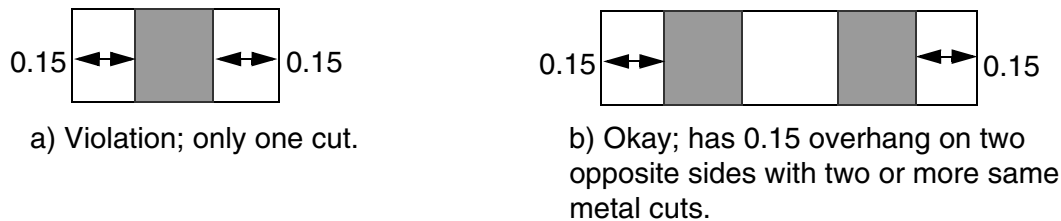
**Figure 1-20 Illustration of the Enclosure Rule With CutClass**



- The following enclosure rule specifies that `VA` vias with two or more same-metal cuts should have `0.15`  $\mu\text{m}$  overhang on any two opposite sides of the routing layers. (See [Figure 1-21](#) on page 75).

```
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE CUTCLASS VA 0.000 0.20 ;
    "ENCLOSURE CUTCLASS VA 0.000 0.15 EXTRACUT;" ;
```

**Figure 1-21 Illustration of the Enclosure Rule With CutClass**



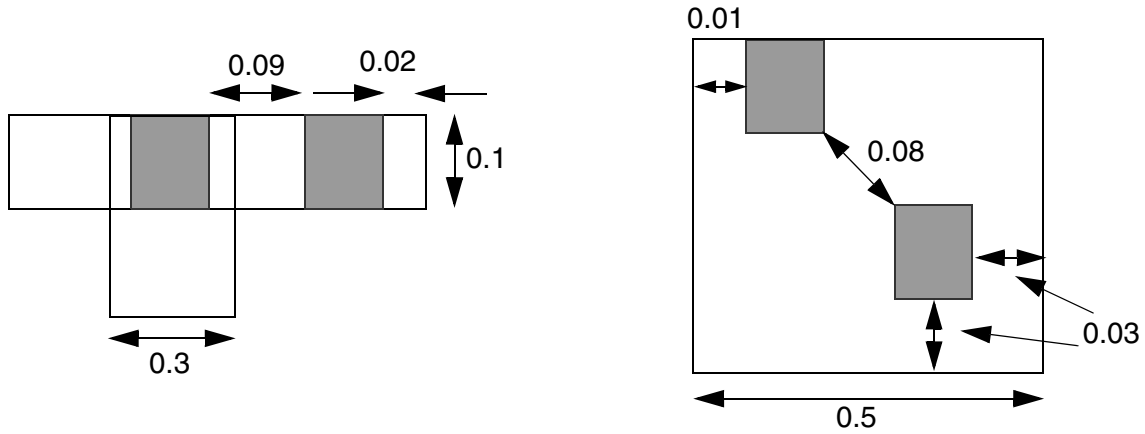
- The following enclosure rule indicates that a `VA` cut via must at least have either `0.10`  $\mu\text{m}$ , `0.12`  $\mu\text{m}$  or `0.0`  $\mu\text{m}$ , or `0.20`  $\mu\text{m}$  enclosures. For redundant cuts having same metal shapes on both metal layers, one of the `VA` cuts should at least have either `0.10`  $\mu\text{m}$ , `0.12`  $\mu\text{m}$  or `0.0`  $\mu\text{m}$ , or `0.20`  $\mu\text{m}$  enclosures, and the rest of the `VA` cuts within `0.15`  $\mu\text{m}$  distance from it, should have at least `0.0`  $\mu\text{m}$ , `0.05`  $\mu\text{m}$  enclosures.

```
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE CUTCLASS VA 0.10 0.12 ;
    "ENCLOSURE CUTCLASS VA 0 0 0.20 ;
    "ENCLOSURE CUTCLASS VA 0.0 0.05 REDUNDANTCUT 0.15 ;" ;
```

- [Figure 1-22](#) on page 76 illustrates `ENCLOSURE` rule with `REDUNDANTCUT`:

```
ENCLOSURE 0.03 0.03 WIDTH 0.3 ;
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE 0.0 0.02 EXTRACUT ;
    "ENCLOSURE 0.0 0.01 REDUNDANTCUT 0.1 ; " ;
```

**Figure 1-22 Illustrations of the Enclosure Rule With REDUNDANTCUT**



a) OK, the right (and left) cut(s) fulfills the EXTRACUT enclosure, which would exempt the left cut from the WIDTH enclosure, and can merely follow and meet the REDUNDANTCUT enclosure

b) OK, the bottom cut fulfills the WIDTH enclosure, which would exempt the top cut from the WIDTH enclosure, and can merely follow and meet the REDUNDANTCUT enclosure

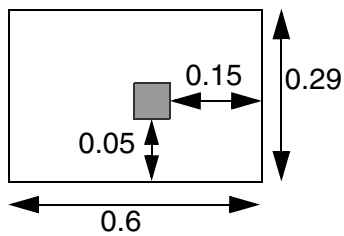
## LEF/DEF 5.8 Language Reference

### LEF Syntax

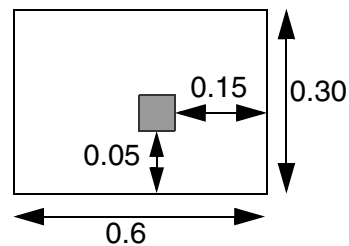
- The following enclosure rule indicates that the overhang of a via cut must be 0.15  $\mu\text{m}$  on an EOL edge with length less than 0.30  $\mu\text{m}$  and its opposite side, while the overhang must be 0.05 on the other two sides. If no EOL edge is involved, a via cut should have 0.10  $\mu\text{m}$  overhang on all four sides of the routing layers:

```
ENCLOSURE 0.10 0.10 ;  
PROPERTY LEF58_ENCLOSURE  
    "ENCLOSURE EOL 0.30 0.15 0.05 ; " ;
```

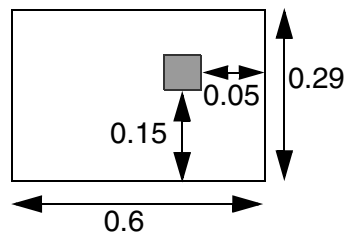
**Figure 1-23 Illustration of the Enclosure Rule With Enclosure EOL**



a) OK, ENCLOSURE EOL rule must be applied and met



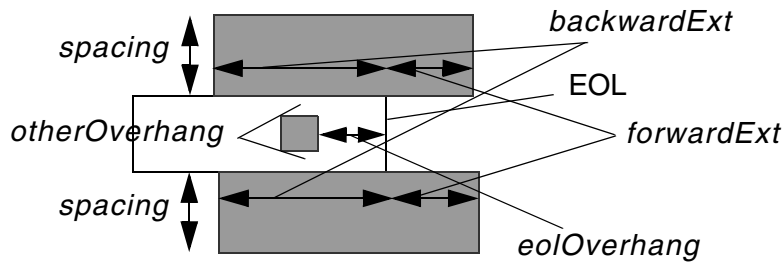
b) Violation, no EOL edge, and ENCLOSURE without EOL must be applied



c) Violation, at least, 0.15 overhang must be on the EOL edge

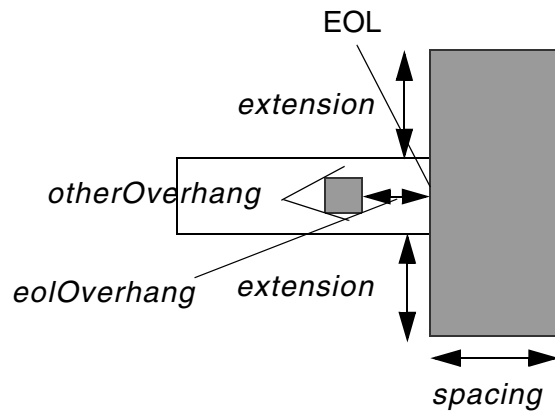
**Figure 1-24 Definition of SIDESPACING and ENDSPACING**

Definition of SIDESPACING *spacing* EXTENSION *backwardExt forwardExt*



There should have no neighbor wires on the 2 checking regions (in gray) in order to apply the given enclosure.

Definition of ENDSPACING *spacing* EXTENSION *extension*



There should have no neighbor wires on the checking region (in gray) in order to apply the given enclosure.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

- The following enclosure rule indicates that 0.15  $\mu\text{m}$  and 0.05  $\mu\text{m}$  can be the enclosure values of a cut on a EOL edge and the other sides if there is no neighbor wires within 0.04  $\mu\text{m}$  with extension of 0.03 on the side edges or within 0.08  $\mu\text{m}$  with extension of 0.06 on the EOL edge:

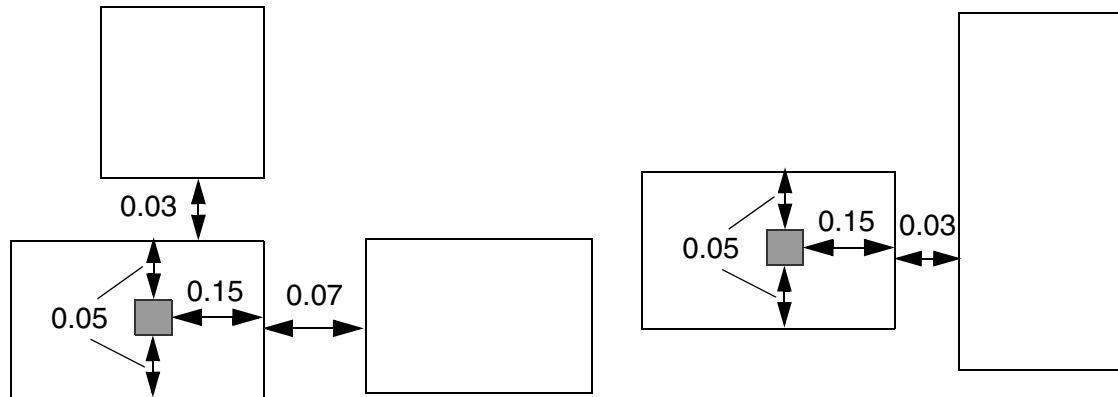
```
ENCLOSURE 0.1 0.1 ;
```

```
PROPERTY LEF58_ENCLOSURE
```

```
"ENCLOSURE EOL 0.2 0.15 0.05 SIDESPACING 0.04 EXTENSION 0.03 0.03 ;
```

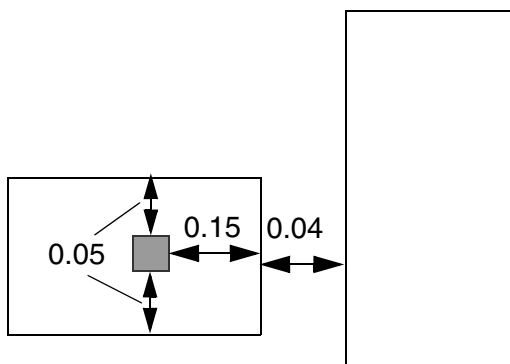
```
ENCLOSURE EOL 0.2 0.15 0.05 ENDSPACING 0.08 EXTENSION 0.06 ; " ;
```

**Figure 1-25 Illustration of the Enclosure Rule With SIDESPACING and ENDSPACING**

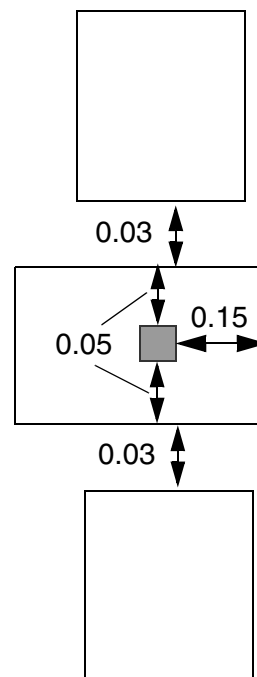


a) Violation, it has neighbor wires on orthogonal edges. 0.10 and 0.10 enclosure should be applied, and fails.

b) Violation, a neighbor wire with 0.03 ( $\leq 0.03$  extension) in the corner would fail both region checking. 0.10 and 0.10 enclosure should be applied, and fails.



c) OK, the wire is just outside 0.04 from the top/bottom edges. 0.15 and 0.05 enclosure should be applied, and is met.



d) OK, as long as the wires are not neighbors of orthogonal edges, the 0.15 and 0.05 enclosure can be applied.



## LEF/DEF 5.8 Language Reference

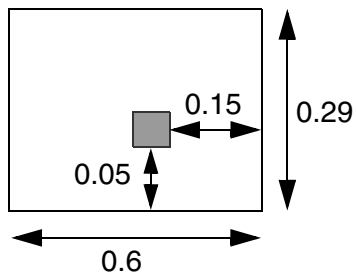
### LEF Syntax

- The following enclosure rule indicates that the overhang of a via cut must be 0.15  $\mu\text{m}$  on an EOL edge with length less than 0.30  $\mu\text{m}$  and the opposite side while the overhang must be 0.05  $\mu\text{m}$  on the other two sides. If no EOL edge is involved, a via cut should have 0.10  $\mu\text{m}$  overhang on all four sides of the routing layers:

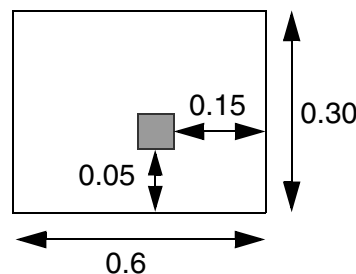
```
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE EOL 0.30 EOLONLY 0.15 0.05 ; " ;
```

**Figure 1-26 Illustration of the Enclosure Rule With EOLONLY**

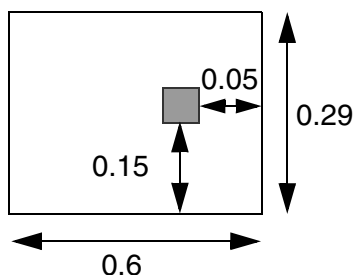
```
ENCLOSURE 0.10 0.10 ;
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE EOL 0.30 EOLONLY 0.15 0.05 ; " ;
```



a) OK, ENCLOSURE EOL rule should be applied and met



b) Violation, no EOL edge, and ENCLOSURE without EOL should be applied, but fails



c) Violation, at least, 0.15 overhang must on the EOL edge

- Figure 1-27 on page 82 illustrates the following enclosure rules:

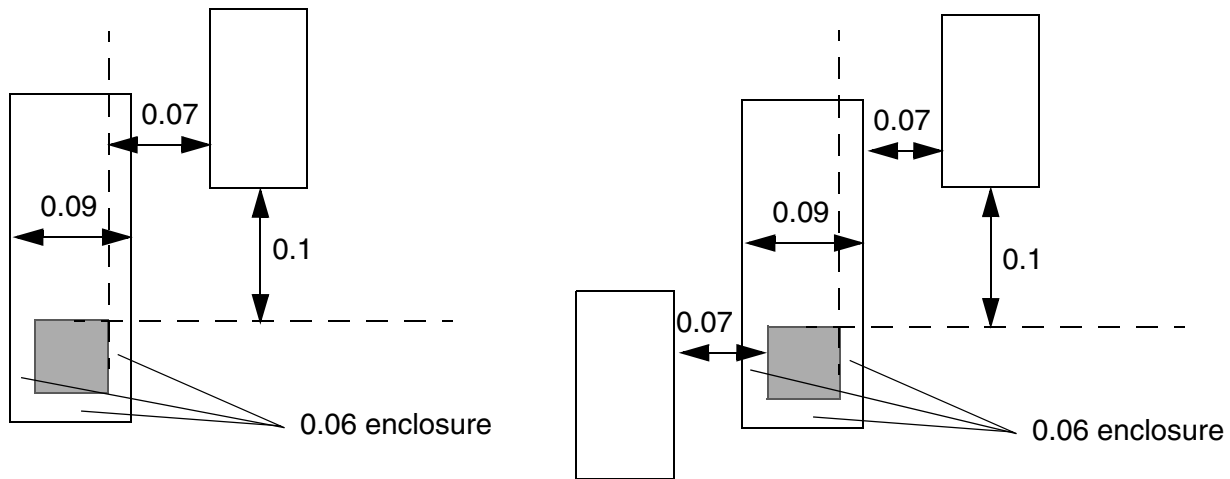
```
ENCLOSURE 0.05 0.00 ;
ENCLOSURE 0.06 0.00 WIDTH 0.1 ;
PROPERTY LEF58_ENCLOSURE
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

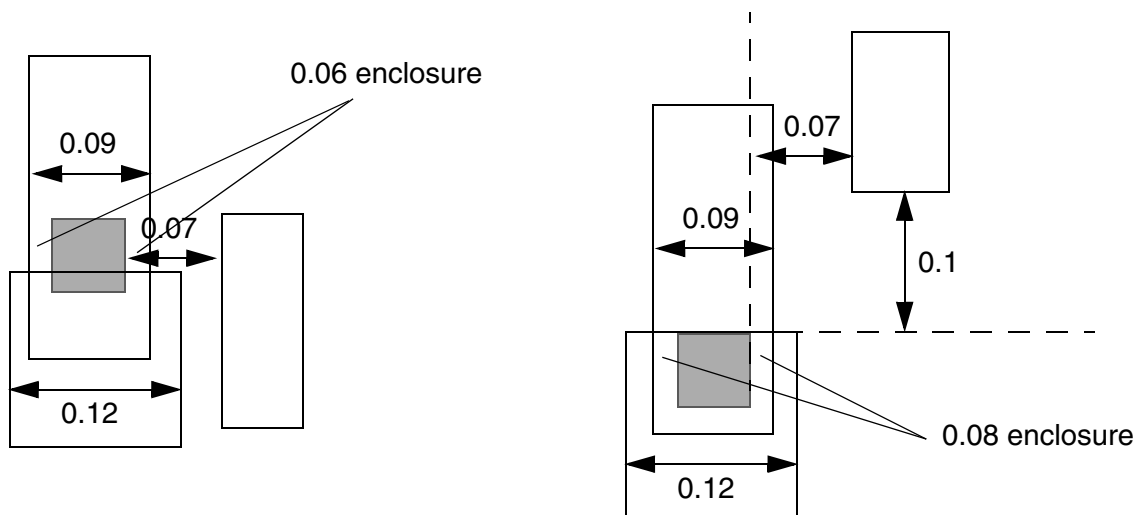
```
"ENCLOSURE 0.09 0.09 PARALLEL -0.11 WITHIN 0.08 ;" ;
```

**Figure 1-27 Illustration of the Enclosure Rule With PARALLEL and WITHIN**



a) Violation, the parallel and within conditions are fulfilled, 0.09 enclosure on all four sides is needed and fails.

b) OK, there are neighbors on both sides such that 0.09 enclosure rule does not apply, and 0.05, 0 enclosure is fulfilled.



c) Violation, the cut is not completely inside the wide wire of 0.12.

d) OK, the cut is completely inside the wide wire of 0.12, which fulfills the corresponding 0.06 0.00 enclosure

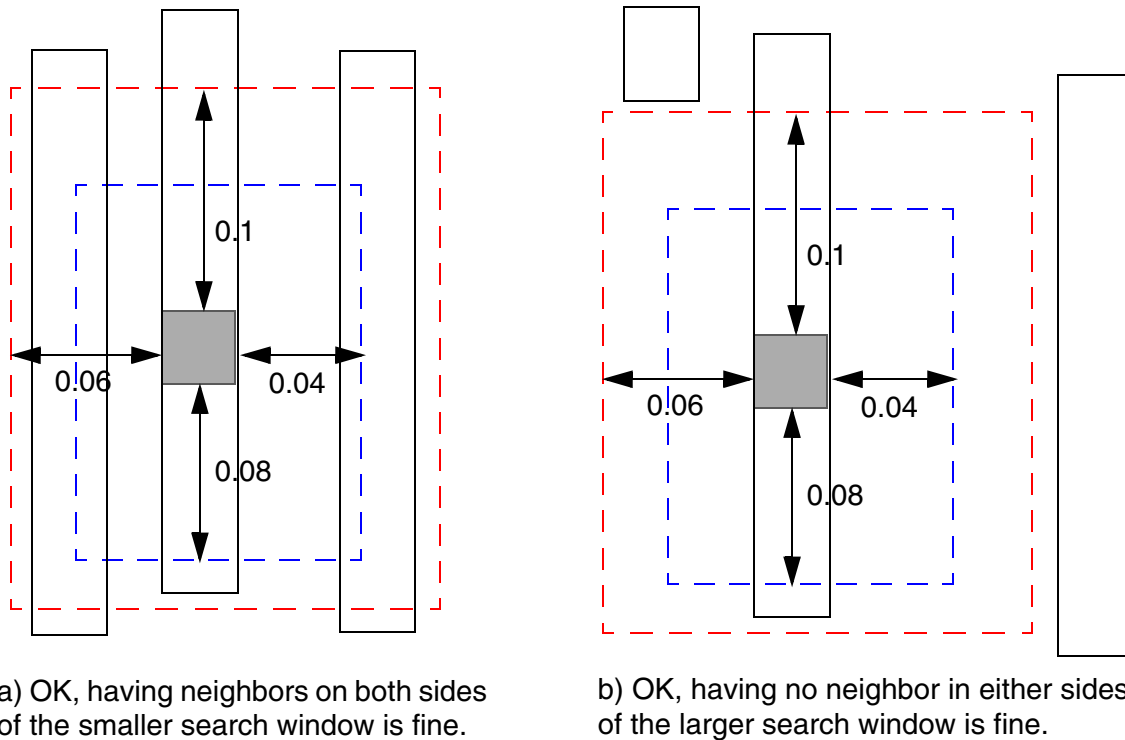
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- **Figure 1-28** on page 83 illustrates the following enclosure rules:

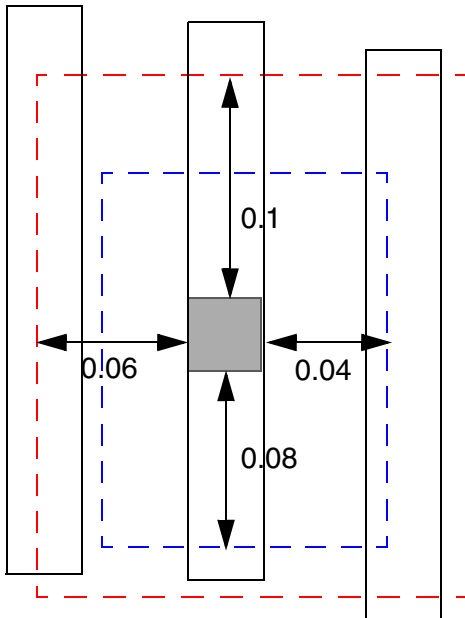
```
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE CUTCLASS VSINGLECUT ABOVE 0.020 0.000 ;
    ENCLOSURE CUTCLASS VSINGLECUT ABOVE 0.015 0.015
    PARALLEL -0.08 -0.1 WITHIN 0.04 0.06 ; "
```

**Figure 1-28 Illustration of the Enclosure Rule With PARALLEL and WITHIN**

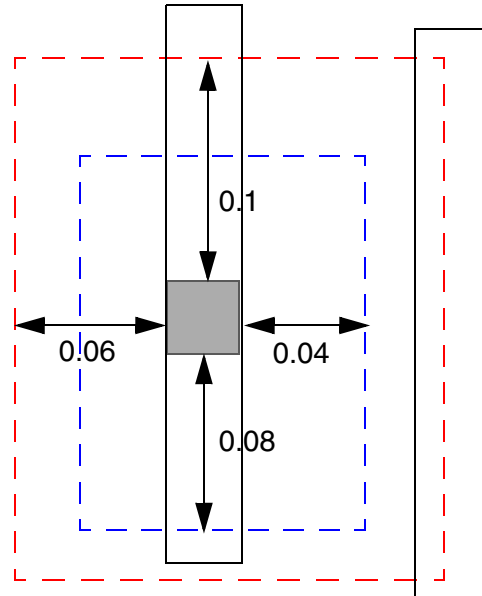


# LEF/DEF 5.8 Language Reference

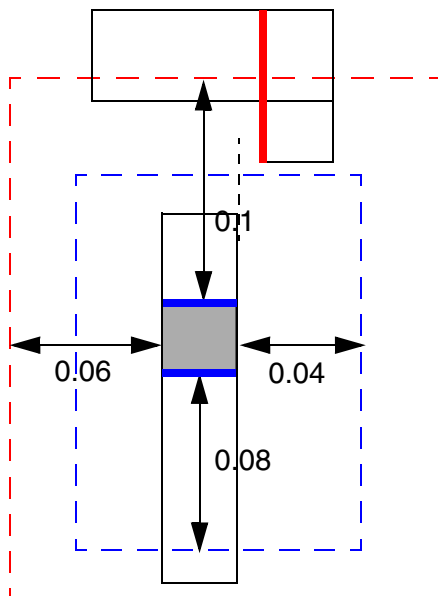
## LEF Syntax



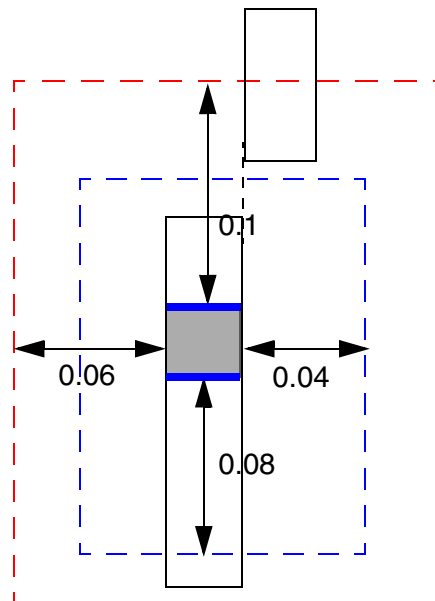
c) Violation, having a neighbor on right side alone of the smaller search window



d) Violation, having a neighbor between the smaller and large search windows.



e) Violation, the vertical red edge of the neighbor wire does not have PRL with the orthogonal blue cut edges; the horizontal edge of the neighbor wire is irrelevant



f) OK, the right neighbor wire has PRL=0 ( $\geq 0$ )

## LEF/DEF 5.8 Language Reference

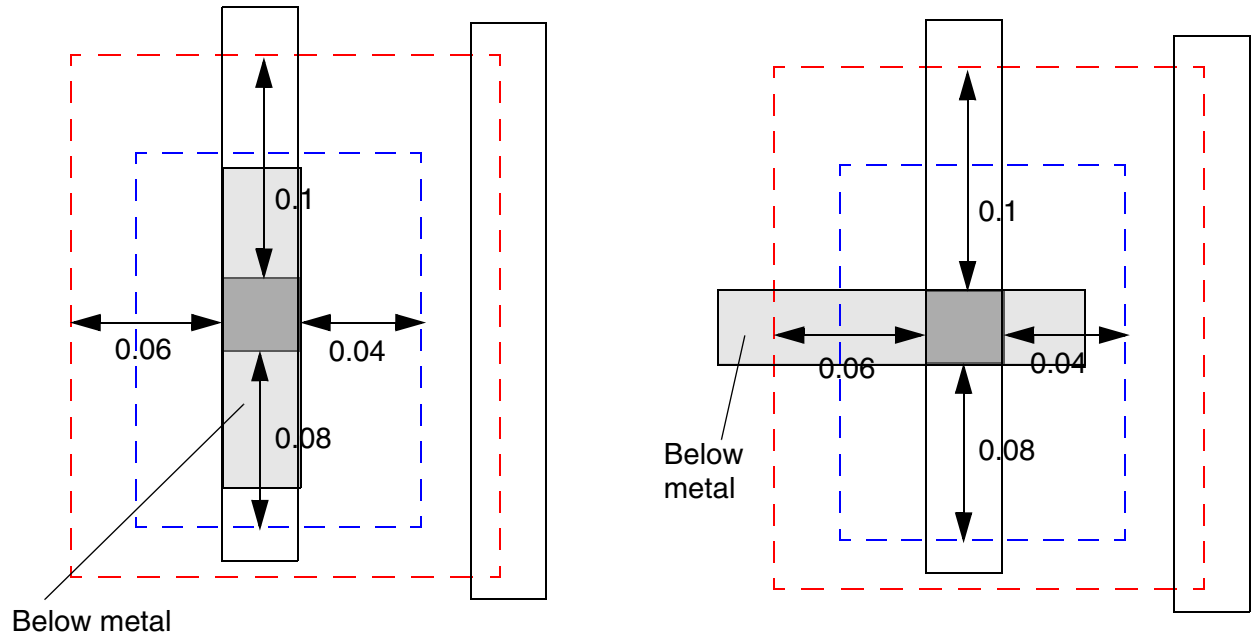
### LEF Syntax

---

- Figure 1-29 on page 86 illustrates the following enclosure rules:

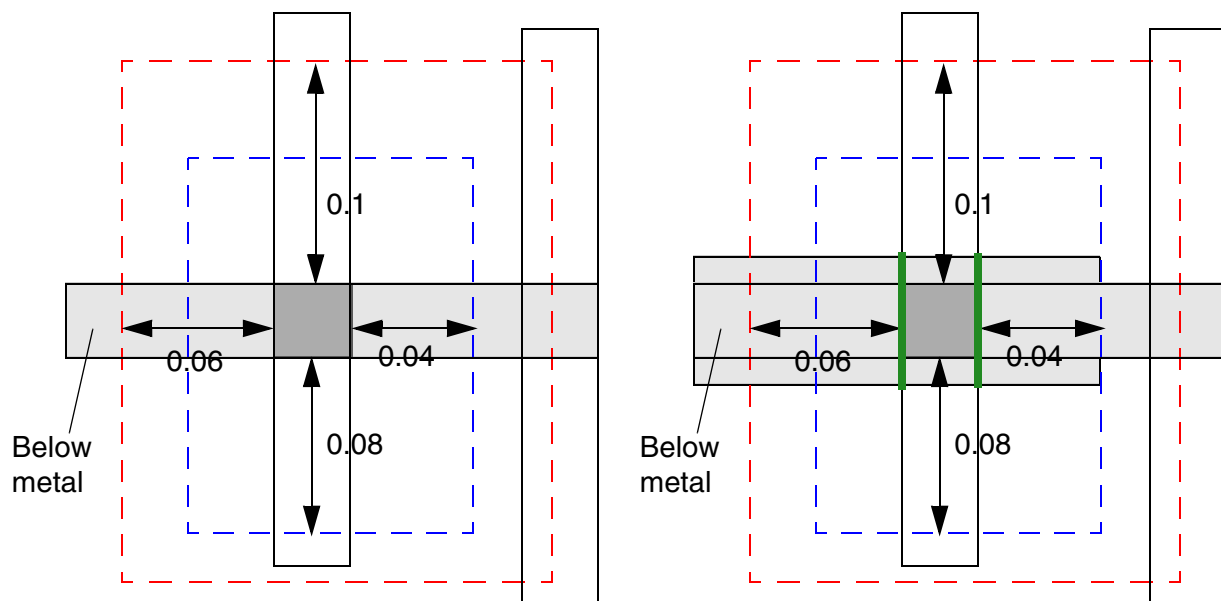
```
PROPERTY LEF58_ENCLOSURE
"ENCLOSURE CUTCLASS VSINGLECUT ABOVE 0.020 0.000 ;
ENCLOSURE CUTCLASS VSINGLECUT ABOVE 0.015 0.015
PARALLEL -0.08 -0.1 WITHIN 0.04 0.06
BELOWENCLOSURE 0.05 ; " ;
```

Figure 1-29 Illustration of the Enclosure Rule With PARALLEL and WITHIN



a) Violation, below metal enclosure is  $< 0.05$  on both sides perpendicular to the above metal wire direction

b) Violation, below metal enclosure is  $< 0.05$  on the right side



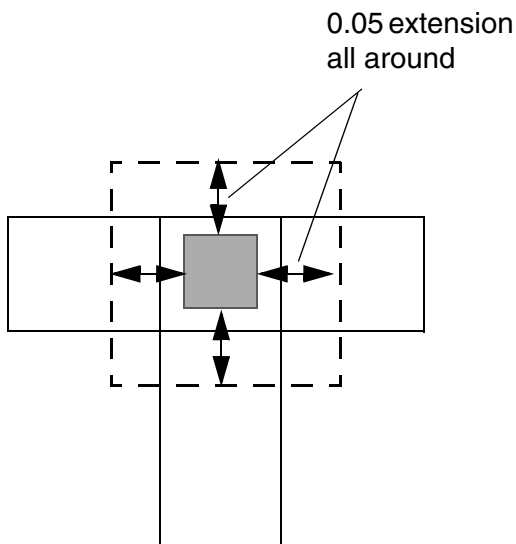
c) OK, below metal enclosure is  $\geq 0.05$  on both sides

d) Violation, the entire green wire edges require 0.05 enclosure and the enclosure on the right one is  $< 0.05$

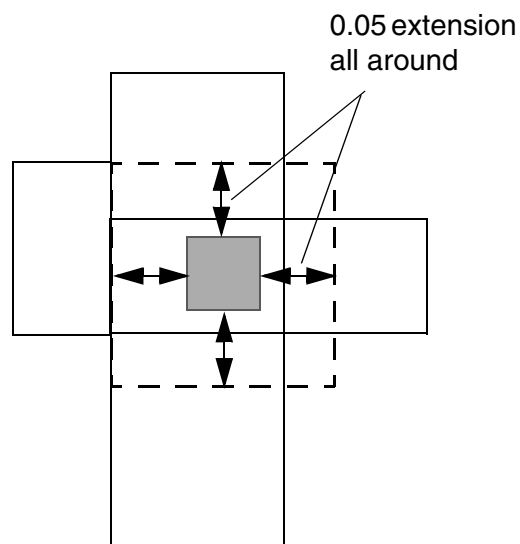
**Figure 1-30 Illustration of the Enclosure Rule With CONCAVECORNERS**

PROPERTY LEF58\_ENCLOSURE

"ENCLOSURE 0.05 0.05 CONCAVECORNERS 2 ; " ;



a) OK, only find two concave corners in the dotted search window



b) Violation, find 3 concave corners ( $> 2$ ), top left corner does not count when that corner collides with a corner of the search window while the bottom left corner counts for aligning to a side edge of the search window

## LEF/DEF 5.8 Language Reference

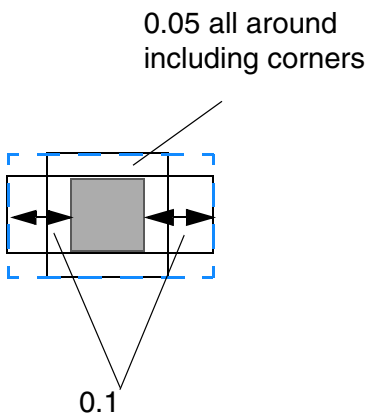
### LEF Syntax

---

- The following enclosure rule illustrates MINCORNER:

```
ENCLOSURE 0.00 0.04 ;  
PROPERTY LEF58_ENCLOSURE "  
ENCLOSURE MINCORNER 0.05 0.1 ; "
```

**Figure 1-31 Illustration of the Enclosure Rule With MINCORNER**



a) OK, only 0.05 enclosure is needed all around including corners. Violation if MINCORNER is omitted, which could require blue dotted line enclosure.

- The following diagram illustrates use of ALLSIDES:

**Figure 1-32 Illustration of the Enclosure Rule With ALLSIDES in BELOWENCLOSURE**

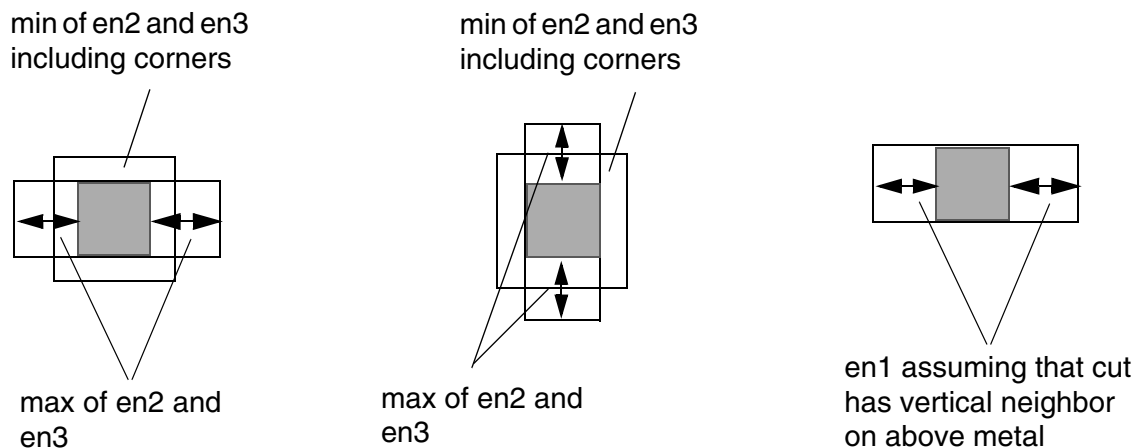
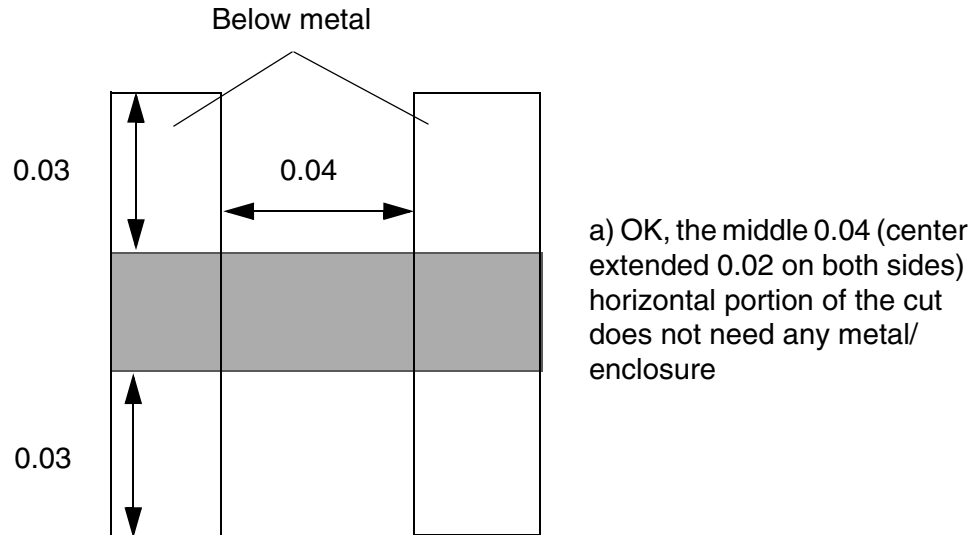


Illustration of minimum enclosure on below metal layer to exempt the rule with BELOWENCLOSURE en1 ALLSIDES en2 en3



**Figure 1-33 Illustration of the Enclosure Rule With HOLLOW**



Example of

```
ENCLOSURE 0.00 0.04 ;
PROPERTY LEF58_ENCLOSURE "
    ENCLOSURE BELOW 0 0.03 HOLLOW HORIZONTAL 0.02 ; " ;
```

### Enclosure Edge Rule

You can create a specific cut-edge enclosure rule that does not fit the normal enclosure rule semantics by using the following property definition:

```
PROPERTY LEF58_ENCLOSUREEDGE
    "ENCLOSUREEDGE [CUTCLASS className] [ABOVE | BELOW] overhang
    {OPPOSITE
        {[EXCEPTEOL eolWidth] [NOCONCAVECORNER within]
        [CUTTOBELOWSPACING spacing
        [ABOVEMETAL extension]]
        | WRONGDIRECTION
        }
    | [INCLUDECORNER]
    {WIDTH [BOTHWIRE] minWidth [maxWidth]
    | SPANLENGTH minSpanLength [maxSpanLength]}
    PARALLEL parLength
    {WITHIN parWithin | WITHIN minWithin maxWithin}
    [EXCEPTEXTRACUT [cutWithin]]
    [EXCEPTTWOEDGES [exceptWithin]]
    | CONVEXCORNERS convexLength adjacentLength
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
        PARALLEL parWithin LENGTH length
    }
;]..." ;
```

Where:

All other keywords are the same as the existing LEF cut layer ENCLOSUREEDGE syntax.

ABOVE | BELOW

If you specify ABOVE, the overhang is required on the routing layers above this cut layer. If you specify BELOW, the overhang is required on the routing layers below this cut layer. If you specify neither, the rule applies to both adjacent routing layers.

ABOVEMETAL *extension*

Specifies that the cut to below metal different-net spacing of a cut edge fulfilling the overhang requirement is measured from the projected above metal edge with extension along the cut edge direction. See [Figure 1-35](#) on page 94.

*Type:* Float, specified in microns

BOTHWIRE

Specifies that the rule only applies if the width of the neighbor wire is greater than or equal to *minWidth*.

CONVEXCORNERS *convexLength* *adjacentLength*  
PARALLEL *parWithin* LENGTH *length*

Specifies the overhang on an edge with length less than or equal to *convexLength* between two convex corners to be *overhang*, if the following conditions are met:

The edge has adjacent edge with length less than or equal to *adjacentLength* and is also between two convex corners. The cut has neighbor wires with parallel run length greater than 0 and within (less than) *parWithin* on both of that edge and another adjacent edge with length greater than or equal to *length*.

*Type:* Float, specified in microns

CUTCLASS *className*

Defines the enclosure edge rule for a specific cut class (*className*). If CUTCLASS is defined in cut layer, then CUTCLASS must be specified in the ENCLOSUREEDGE statement.

Specify individual rules with the CUTCLASS keyword for each cut class, if needed.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

CUTTOBELOWSPACING *spacing*

Defines the spacing of a cut edge fulfilling the overhang requirement to a different-net wire on the below metal layer.

*Type:* Float, specified in microns

ENCLOSUREEDGE *overhang* WIDTH *minWidth* PARALLEL *parLength* WITHIN *parWithin*

Indicates that any edge from this cut layer that is enclosed by metal that is greater than or equal to *minWidth* wide, and the enclosing metal edge is parallel to another metal edge greater than *parLength* in length and less than *parWithin* distance away, requires *overhang* enclosure.

*Type:* Float, specified in microns (for all values)

EXCEPTEXTRACUT [*cutWithin*]

Indicates that if there is another via cut in the same metal intersection in both the above and below layers, this rule is not checked. If you specify *cutWithin*, the other via cut should be less than and equal to *cutWithin* distance away in order to ignore this rule.

EXCEPTTWOEDGES [*exceptWithin*]

Specifies that if the enclosing metal edges have parallel metal edges greater than *parLength* that are less than *exceptWithin*, if specified, or *parWithin* away on opposite sides that covers the same PRL between the cut, then the rule does not apply.

INCLUDECORNER

Defines the enclosure requirement applied to the corners of the cut in Euclidean measurement. This corner check is essential when the cut is slightly outside the projection of the neighbor wire.

NOCONCAVECORNER *within*

Specifies that there should be no concave corner on the above metal layer that is a *within* distance away after the extending *overhang* on the orthogonal edges of the edges that fulfill the *overhang* requirement.

*Type:* Float, specified in microns

OPPOSITE [EXCEPTEOL *eolWidth*]

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that if overhang of a via cut is less than or equal to *overhang* on the above metal layer (`ABOVE` must be specified when `OPPOSITE` is used), the entire opposite edge must also have overhang less than or equal to *overhang*.

The `EXCEPTEOL` keyword indicates that EOL edges with width less than to *eolWidth* are exempted from the rules.

*Type:* Float, specified in microns

`SPANLENGTH minSpanLength [maxSpanLength]`

Specifies that the enclosure edge rule is triggered by the span length of the wire containing the cut via in the direction perpendicular to the parallel run length is greater than or equal to *minSpanLength*, instead of the width of the wire. An optional *maxSpanLength* has a similar definition as *maxWidth* in `WIDTH`.

*Type:* Float, specified in microns

`WIDTH minWidth [maxWidth]`

Specifies an optional *maxWidth*, such that if the width of the wire containing the cut via is greater than or equal to *minWidth* and less than *maxWidth*, then all the corresponding enclosure edge rules that meet the parallel and within conditions will be applied to the cut.

If any one of the `ENCLOSUREEDGE` statements use *maxWidth*, then all the `ENCLOSUREEDGE` statements must have *maxWidth*. If `SPANLENGTH` is specified then all of them must have *maxSpanLength*, and all of the enclosure edge rules that meet the width condition in `WIDTH` statement or span length condition in `SPANLENGTH` construct (parallel run length is greater than or equal to *minSpanLength* and less than *maxSpanLength*) and the parallel and within conditions will be applied to the cut.

*Type:* Float, specified in microns

`WITHIN minWithin maxWithin`

Specifies that the rule only applies if parallel neighbor metal edge is greater than or equal to *minWithin* and less than *maxWithin* distance away.

*Type:* Float, specified in microns

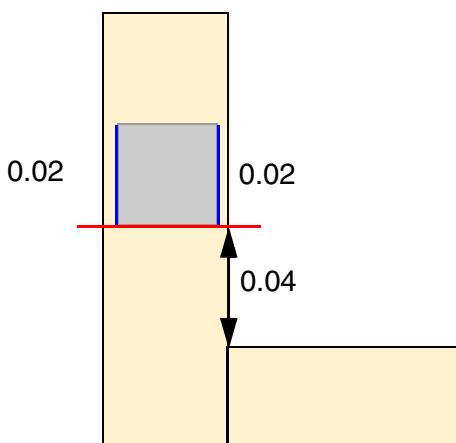
**WRONGDIRECTION** Specifies that overhang in the non-preferred direction on the above metal layer (you must specify the **ABOVE** keyword when **OPPOSITE** is defined) of a via cut must be exactly equal to the *overhang*.

### Enclosure Edge Rule Examples

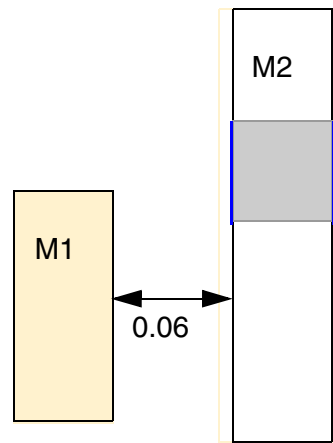
- **Figure 1-34** on page 93 illustrates enclosure edge rule with **NOCONCAVECORNER** and **CUTTOBELOWSPACING**:

```
PROPERTY LEF58_ENCLOSUREEDGE
  "ENCLOSUREEDGE ABOVE 0.02 OPPOSITE
    NOCONCAVECORNER 0.05
    CUTTOBELOWSPACING 0.07 ; " ;
```

**Figure 1-34 Illustration of Enclosure Edge Rules**



a) Violation, the blue edges fulfill 0.02 overhang requirement, and by extending 0.02 (in red) on the orthogonal bottom edge, the concave corner is within 0.05 away

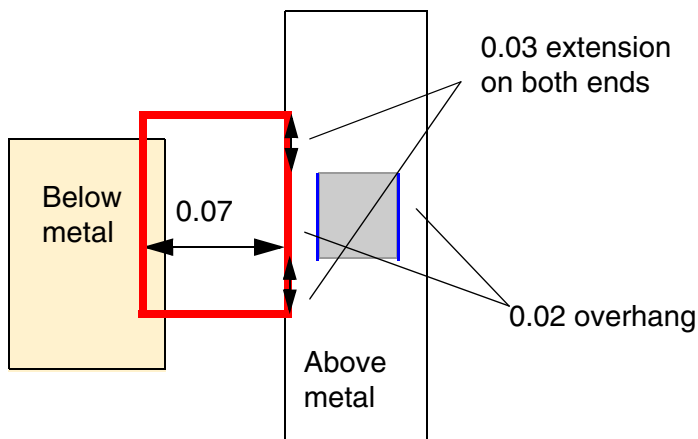


b) Violation, the blue edges fulfilling 0.02 overhang requirement need 0.07 spacing to a below metal object

- The following figure illustrates enclosure edge rule with ABOVEMETAL:

```
PROPERTY LEF58_ENCLOSUREEDGE
  "ENCLOSUREEDGE ABOVE 0.02 OPPOSITE
    CUTTOBELOWSPACING 0.07
    ABOVEMETAL 0.03 ; " ;
```

**Figure 1-35 Illustration of Enclosure Edge Rules**



a) Violation, the blue edges fulfilling 0.02 overhang requirement need 0.07 spacing to a below metal object from the projected above metal with 0.03 extension, which means no neighbor object in the red region

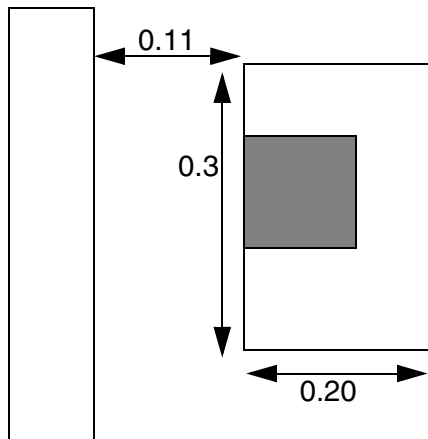
## LEF/DEF 5.8 Language Reference

### LEF Syntax

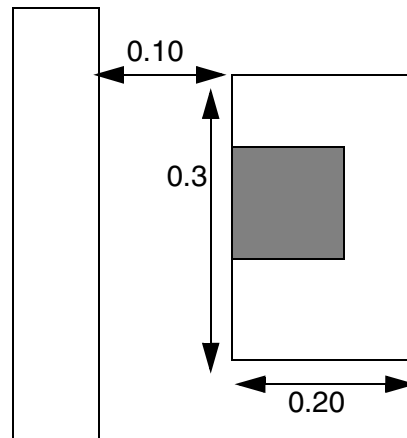
- **Figure 1-36** on page 95 illustrates the following enclosure edge rule:

```
ENCLOSURE 0.0 0.05 ;           #normal enclosure rule
PROPERTY LEF58_ENCLOSUREEDGE
    "ENCLOSUREEDGE 0.02 WIDTH 0.2 PARALLEL 0.25 WITHIN 0.11 ;" ;
```

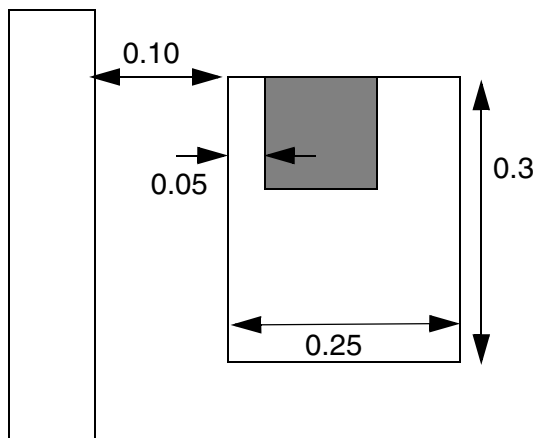
**Figure 1-36 Illustration of Enclosure Edge Rules**



a) Okay. Width  $\geq 0.2$ , but spacing to neighbor is  $\geq 0.11$ , so edge enclosure rule is not required. Cut meets the normal 0.00 0.05 enclosure rule.



b) Violation. Width  $\geq 0.2$ , spacing to neighbor is  $< 0.11$ , parallel length is  $> 0.25$ , so edge enclosure rule of 0.02 is required but not met.



c) Okay. Width  $\geq 0.2$ , spacing to neighbor is  $< 0.11$ , parallel length is  $> 0.25$ , so edge enclosure rule of 0.02 is required and met. Cut also meets the normal 0.00 0.05 enclosure rule.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

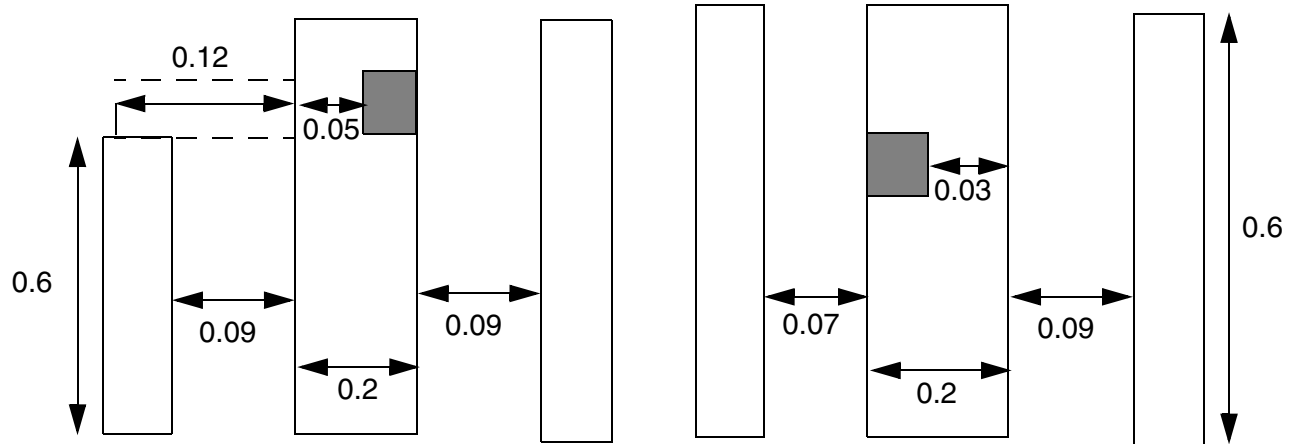
---

- Figure 1-37 on page 97 illustrates the following enclosure edge rule:

```
PROPERTY LEF58_ENCLOSUREEDGE
    "ENCLOSUREEDGE 0.05 WIDTH 0.20
      PARALLEL 0.50 WITHIN 0.08 0.10
      EXCEPTTWOEDGES 0.12 ; " ;
```

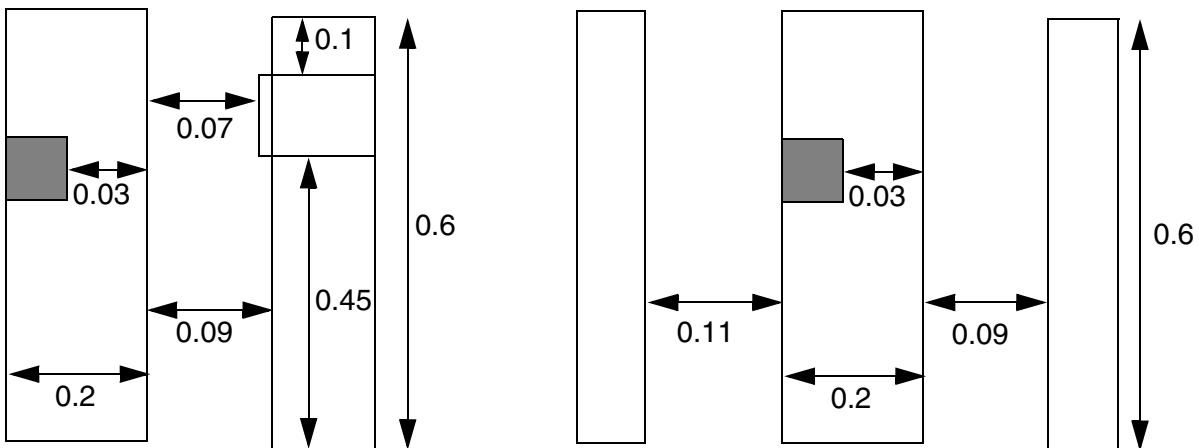


**Figure 1-37 Illustration of the Enclosure Edge Rule with EXCEPTTWOEDGES**



a) Violation, the right neighbor does not have corresponding left neighbor on the dotted region, and EXCEPTTWOEDGES is not triggered on that region to be a violation

b) OK, the lower bound of WITHIN of 0.08 is irrelevant on EXCEPTTWOEDGES, and the left neighbor is  $< 0.12$  to exempt the rule.



c) OK, the portion with 0.07 ( $< 0.08$ ) spacing away is ignored, the rest having 0.45 and 0.1 ( $< 0.50$ ) parallel run length should be treated individually, and the rule is not triggered.

d) OK, there are two neighbor wires  $< 0.12$ . Violation if 0.12 is not specified in EXCEPTTWOEDGES.

## LEF/DEF 5.8 Language Reference

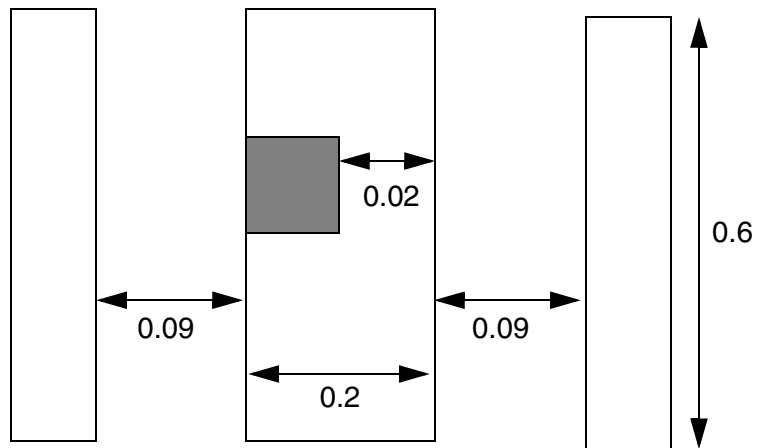
### LEF Syntax

---

- Figure 1-38 on page 98 illustrates the following enclosure edge rule:

```
PROPERTY LEF58_ENCLOSUREEDGE  
  "ENCLOSUREEDGE 0.05 WIDTH 0.20  
  PARALLEL 0.50 WITHIN 0.10 EXCEPTTWOEDGES ;" ;
```

**Figure 1-38 Illustration of Enclosure Edge Rules**



a) Okay, since it has 2 parallel neighbors, the rule is ignored.

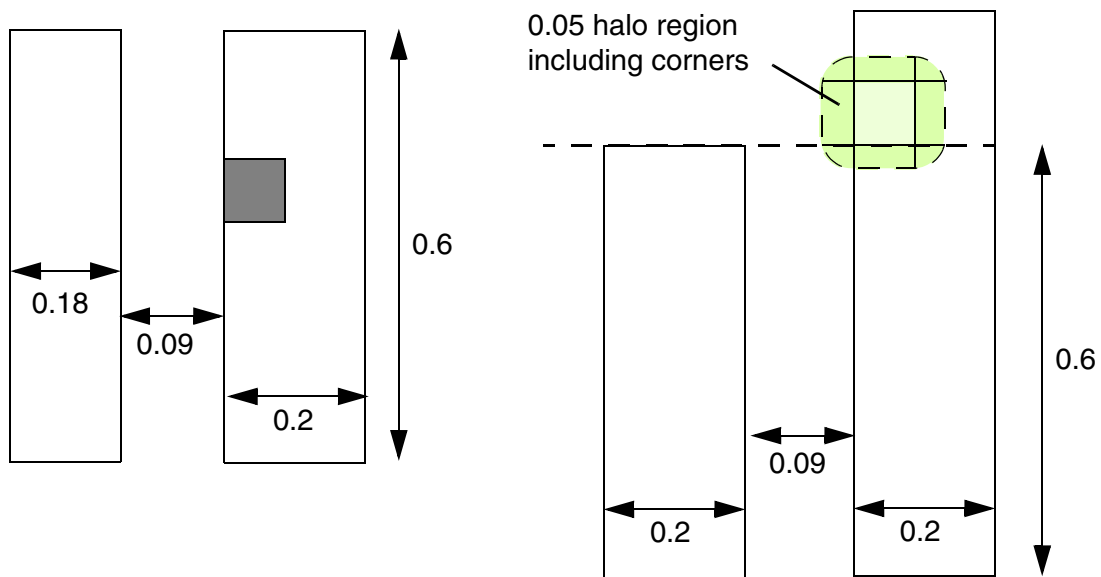
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- **Figure 1-39** on page 99 illustrates the following enclosure edge rule:

```
PROPERTY LEF58_ENCLOSUREEDGE
  "ENCLOSUREEDGE 0.05 INCLUDECORNER
    WIDTH BOTHWIRE 0.20
    PARALLEL 0.5 WITHIN 0.10 ; " ;
```

**Figure 1-39 Illustration of the Enclosure Edge Rule with BOTHWIRE and INCLUDECORNER**



a) OK, the width of the left neighbor wire is  $0.18 < 0.20$ , which will not trigger the rule. Violation, if BOTHWIRE is not specified.

b) Violation, the bottom left corner overlaps with the projection of the neighbor wire. OK, if INCLUDECORNER is not specified.

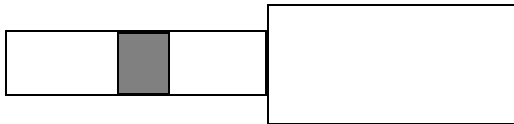
## LEF/DEF 5.8 Language Reference

### LEF Syntax

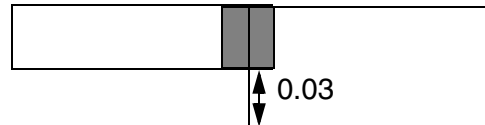
---

**Figure 1-40 Illustration of the Enclosure Edge Rule with OPPOSITE and EXCEPTEOL**

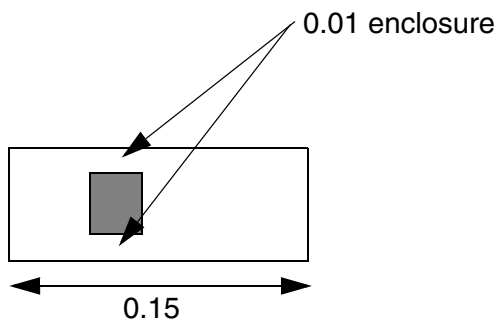
```
PROPERTY LEF58_ENCLOSUREEDGE  
  "ENCLOSUREEDGE ABOVE 0.02 OPPOSITE  
    EXCEPTEOL 0.15 ; " ;
```



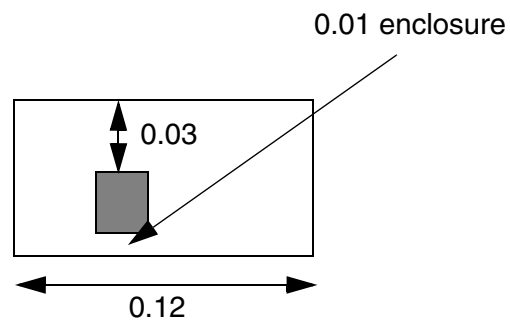
a) OK, 0 enclosure on 2 entire opposite edges



b) Violation, the enclosure of the bottom cut edge is  $\geq 0.02$



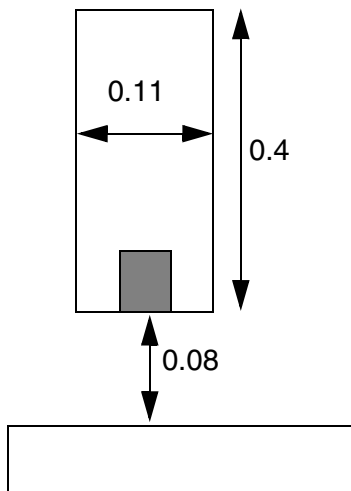
c) OK, 0.01 enclosure on 2 entire opposite edges



d) OK, EOL edges are exempted from the rule

**Figure 1-41 Illustration of the Enclosure Edge Rule with SPANLENGTH**

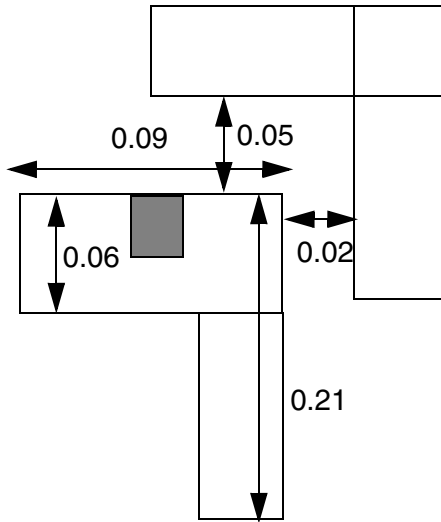
```
PROPERTY LEF58_ENCLOSUREEDGE  
  "ENCLOSUREEDGE 0.02 SPANLENGTH 0.2  
    PARALLEL 0.1 WITHIN 0.09 ; " ;
```



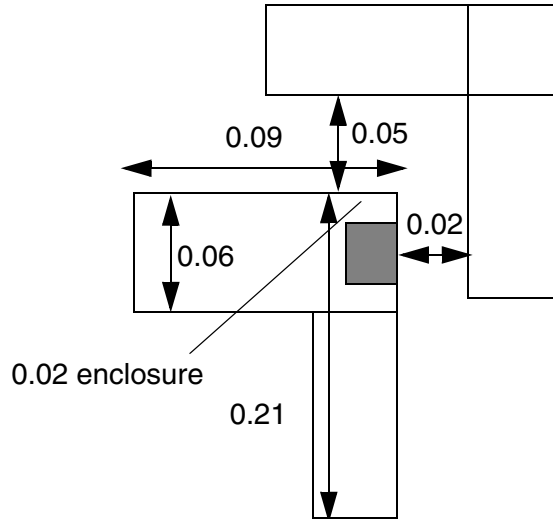
a) Violation, although the width is merely 0.11, the span length in the direction perpendicular to PRL is 0.4 ( $\geq 0.2$ ), with PRL of 0.11 ( $> 0.1$ ), the rule is triggered.

**Figure 1-42 Illustration of the Enclosure Edge Rule with CONVEXCORNERS**

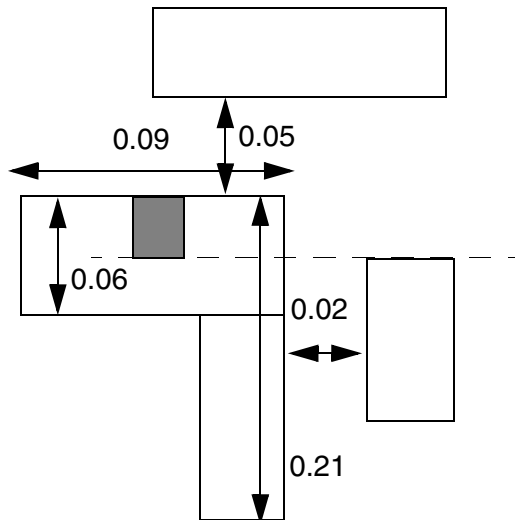
```
PROPERTY LEF58_ENCLOSUREEDGE
  "ENCLOSUREEDGE 0.02 CONVEXCORNERS 0.1 0.08
    PARALLEL 0.06 LENGTH 0.2 ; " ;
```



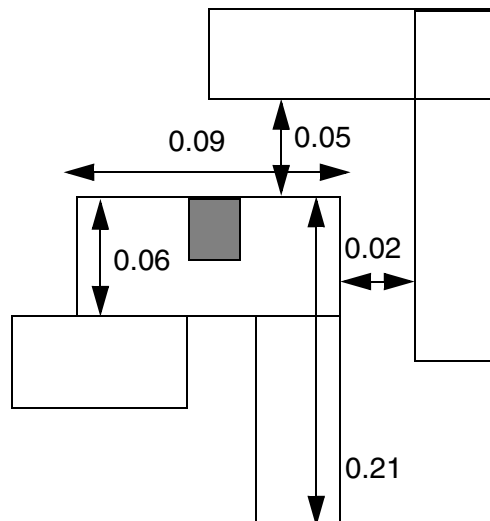
a) Violation, all of the conditions are met, but enclosure is 0 on the top edge



b) OK, enclosure requirement is only on top edge, the enclosure on the left edge is irrelevant



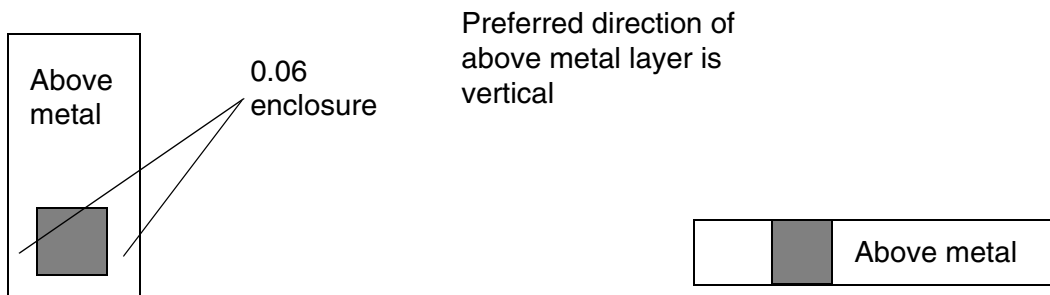
c) OK, no parallel run length with the cut on the right wire



d) OK, the 0.06 adjacent edge is not between 2 convex corners

**Figure 1-43 Illustration of the Enclosure Edge Rule with WRONGDIRECTION**

```
PROPERTY LEF58_ENCLOSUREEDGE
  "ENCLOSUREEDGE ABOVE 0.05 OPPOSITE
    WRONGDIRECTION ; " ;
```



a) Violation, the enclosure in the non-preferred direction (horizontal) must be exactly 0.05. Larger than that is still a violation.

b) Violation, the enclosure requirement in the non-preferred direction also applies to non-preferred wires.

### **Enclosure Table Rule**

You can create an enclosure table rule to specify the enclosure requirement based on the wire width in a table format by using the following property definition:

```
PROPERTY LEF58_ENCLOSURETABLE
  "ENCLOSURETABLE [CUTCLASS className] [MASK maskNum]
    [USEMAXWIDTH | WITHINFIRSTWIDTH]
    [DEFAULT {[ABOVE | BELOW]
      overhang1 overhang2 overhang3 overhang4}...]
    {WIDTH width {[ABOVE | BELOW]
      overhang1 overhang2 overhang3 overhang4 [MINSUM]
      [MAXLENGTH length]}
      [LAYER trimLayer OVERLAP {1|2}
        [TRIMLENGTHOUTERMETAL trimLength]]}...
    | OTHERWIDTH otherWidth [PARALLEL parLength WITHIN parWithin]
      {[ABOVE | BELOW]
        overhang1 overhang2 overhang3 overhang4 [MINSUM]}...}...
  ; " ;
```

Where:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

All the other keywords are the same as the existing LEF cut layer ENCLOSURETABLE syntax.

CUTCLASS *className*

Defines enclosure rules for a specific cut class *className*. If CUTCLASS is defined in cut layer, then CUTCLASS must be specified in the ENCLOSURETABLE statement.

DEFAULT {[ABOVE | BELOW] *overhang1 overhang2 overhang3 overhang4*}...

Specifies *overhang1* and *overhang2* to be overhangs of a cut on two opposite sides and *overhang3* and *overhang4* on the other two opposite sides in a wire of any width. For a rectangular cut, the first two overhang values, *overhang1* and *overhang2*, would be applied to the end or short cut edges while the last two overhang values, *overhang3* and *overhang4*, would be applied to the side or long cut edges.

If you specify BELOW, the overhang is required on the routing layers below this cut layer. If you specify ABOVE, the overhang is required on the routing layers above this cut layer. If you specify neither, the rule applies to both adjacent routing layers.

*Type:* Float, specified in microns

LAYER *trimLayer* OVERLAP {1|2}

Specifies a special enclosure requirement on rectangular pins depending on which pin end overlaps with or touches shapes in *trimLayer*, which must be a layer with TYPE TRIMMETAL. If 1 is given in OVERLAP, the corresponding enclosure rule could be applied to the pin end that overlaps with the shapes in *trimLayer*. If 2 is given in OVERLAP, the corresponding enclosure rule could be applied to both ends of a pin only if both the ends overlap with the shapes in *trimLayer*.

MASK *maskNum*

Specifies which mask the metal of this enclosure table rules will be applied on. With ABOVE/BELOW, the mask is referred on above/below metal layers. Without ABOVE or BELOW, the mask is referred to both above and below metal layers. The *maskNum* must be a positive integer, and most applications only support values of 1, 2, or 3. If MASK is used on one mask, separate tables should be defined for other masks.

*Type:* Integer



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

MAXLENGTH <i>length</i>	<p>Specifies the overhang requirements only applies if the span length of the wire containing the cut is less than or equal to the <i>length</i>.</p> <p><i>Type:</i> Float, specified in microns</p>
MINSUM	<p>Indicates that it is legal for sum of overhang values on two opposite sides greater than or equal to the sum of the specified overhangs, and the smaller overhang value is greater than or equal to the smaller specified overhangs. This keyword should only be defined for asymmetrical overhang values.</p>
{OTHERWIDTH <i>otherWidth</i> {{ABOVE  BELOW} <i>overhang1 overhang2 overhang3 overhang4</i> [MINSUM]}...}...	<p>Specifies the overhang requirements on the above/below metal layer corresponding to the specified ABOVE/BELOW will depend on the width of the other (below/above) metal layer. The meaning of overhang and MINSUM is the same as in WIDTH. See <a href="#">Figure 1-45</a> on page 108.</p> <p><i>Type:</i> Float, specified in microns</p>
TRIMLENGTHOUTERMETAL <i>trimLength</i>	<p>Specifies that the enclosure with a shape on the layer <i>trimLayer</i> overlapping with or touching a wire end applies only if the <i>trimLayer</i> shape has a length greater than or equal to <i>trimLength</i>, and only the outermost wires overlapping or touching the <i>trimLayer</i> shape are subjected to this rule.</p> <p><i>Type:</i> Float, specified in microns</p>
USEMAXWIDTH	<p>Specifies that the maximum wire width with which a cut overlaps is used to look up the proper width rows for the overhang requirements. It is similar to the width interpretation in ENCLOSURE.</p>
PARALLEL <i>parLength</i> WITHIN <i>parWithin</i>	<p>Specifies that the enclosure rule applies only if there is a metal neighbor at least on one side of the cut within (less than) <i>parWithin</i> having common parallel run length greater than or equal to <i>parLength</i> on the layer on which the enclosure is applied. If <i>parLength</i> is a negative value, the two edges will be checked in WITHIN style. The metal containing the cut is not considered a neighbor.</p> <p><i>Type:</i> Float, specified in microns</p>

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
{WIDTH width {[ABOVE| BELOW]
overhang1 overhang2 overhang3 overhang4}...}...
```

Specifies the overhang requirements for a cut in a wire having a certain width. The width values must be the same or in increasing order (from top to bottom) in the table. If any of the overhang values on a routing layer are asymmetrical, that is, *overhang1* is not equal to *overhang2* or *overhang3* is not equal to *overhang4*, then the minimum wire width that the cut touches or overlaps with is determined. The last row with wire width greater than *width* and any one of the corresponding specified overhangs or overhangs defined in DEFAULT must be fulfilled.

In addition, the minimum overhang value on all the four sides, including corners, is applied. The other overhang values per cut edges, without corner consideration, are applied. If all of the overhang values are symmetrical, the maximum wire width that the cut overlaps with or touches and the traditional rectangle enclosure by using one overhang value on the two opposite sides and another overhang value on the other two opposite sides must be used.

The definition of overhangs ABOVE and BELOW have the same meaning as those in DEFAULT.

For a rectangular cut, the first two overhang values, *overhang1* and *overhang2*, would be applied to the end or short cut edges while the last two overhang values, *overhang3* and *overhang4*, would be applied to the side or long cut edges.

*Type:* Float, specified in microns

WITHINFIRSTWIDTH

Specifies that the enclosure of the first row width applies only if a cut is completely within the width applicable to the first width. Otherwise, the maximum wire width that a cut touches or overlaps with is still used to look up the enclosure on the rest of the rows.

### Enclosure Table Rule Examples

- The following enclosure table rule indicates that:
  - For a cut in a wire of any width, 0.01  $\mu\text{m}$  on two opposite sides and 0.02  $\mu\text{m}$  on the other two opposite sides can be an overhang choice.

## LEF/DEF 5.8 Language Reference

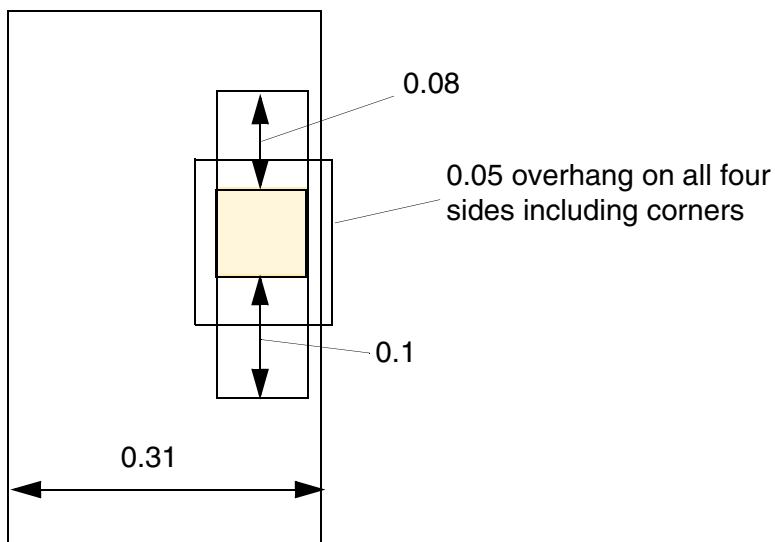
### LEF Syntax

- ❑ For a via cut in a wire with width greater than 0.1  $\mu\text{m}$  and less than or equal to 0.2  $\mu\text{m}$ , 0.015  $\mu\text{m}$  overhang on four sides can be applied.
- ❑ For a via cut in a wire with width greater than 0.2  $\mu\text{m}$  and less than or equal to 0.3  $\mu\text{m}$ , 0.015  $\mu\text{m}$  and 0.016  $\mu\text{m}$  overhangs on two opposite sides, and 0.017  $\mu\text{m}$  and 0.018  $\mu\text{m}$  overhangs on the other two opposite sides, or 0.025  $\mu\text{m}$  and 0.03  $\mu\text{m}$  or 0.026  $\mu\text{m}$  and 0.029  $\mu\text{m}$  or 0.027  $\mu\text{m}$  and 0.28  $\mu\text{m}$  on two opposite sides, 0.00  $\mu\text{m}$  on the other two opposite sides can be applied.
- ❑ For a via cut in a wire with width greater than 0.3  $\mu\text{m}$ , 0.019  $\mu\text{m}$  overhang on four sides can be applied.

```
PROPERTY LEF58_ENCLOSURETABLE
"ENCLOSURETABLE
    DEFAULT 0.01 0.01 0.02 0.02
    WIDTH 0.1 0.015 0.015 0.015 0.015
    WIDTH 0.2 0.015 0.016 0.017 0.018
           0.025 0.03 0.00 0.00 MINSUM
    WIDTH 0.3 0.019 0.019 0.019 0.019 ; "
```

**Figure 1-44 Illustration of Enclosure Table Rules**

```
PROPERTY LEF58_ENCLOSUREWIDTH
"ENCLOSURETABLE
    WIDTH 0.3 0.05 0.05 0.08 0.1; "
```



a) OK, 0.05 all around the cut, and 0.08 and 0.1 based on the cut edges

## LEF/DEF 5.8 Language Reference

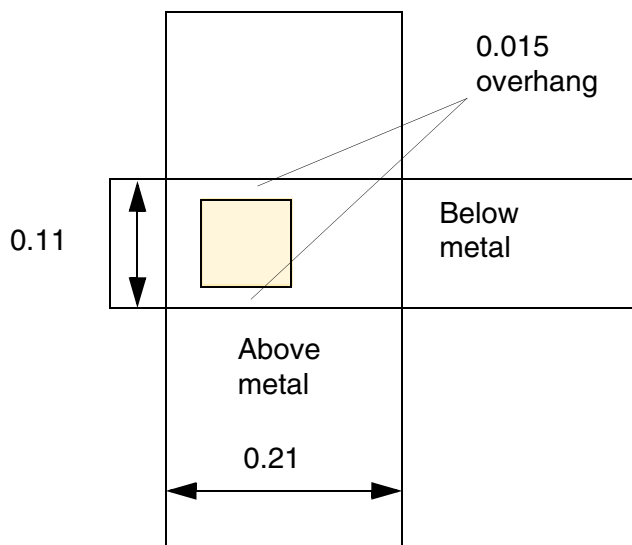
### LEF Syntax

- The following rule indicates that for a via cut in a wire with width  $> 0.1\ \mu\text{m}$ , 0.015 overhang on four sides can be applied, for a via cut in a wire with width on the above metal layer  $> 0.2\ \mu\text{m}$ , 0.019 overhang on the below metal layer on four sides can be applied.

```
PROPERTY LEF58_ENCLOSURETABLE
"ENCLOSURETABLE
  WIDTH 0.1 0.015 0.015 0.015 0.015
  OTHERWIDTH 0.2 BELOW 0.019 0.019 0.019 0.019 ; " ;
```

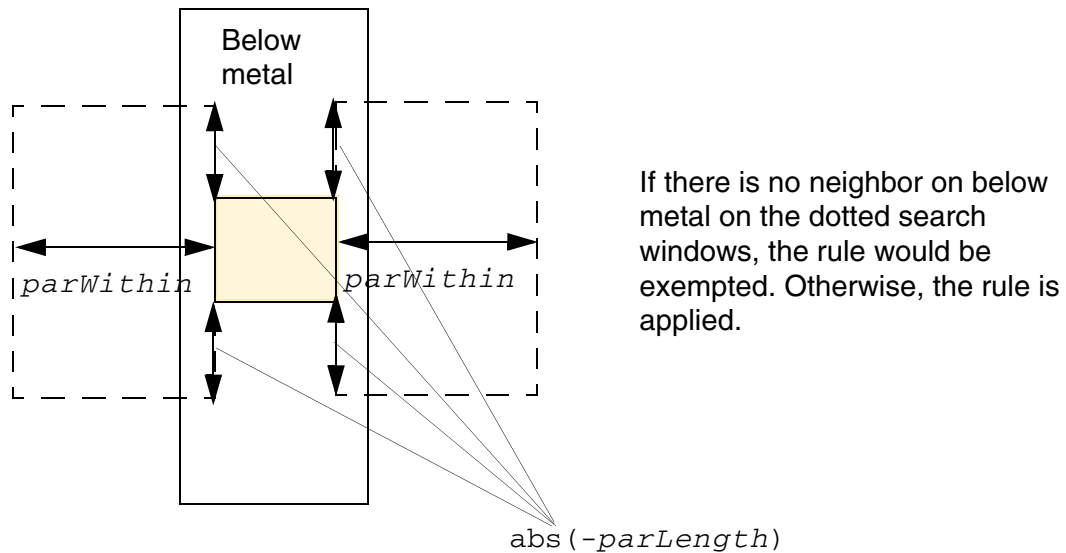
**Figure 1-45 Illustration of Enclosure Table Rules with OTHERWIDTH**

```
PROPERTY LEF58_ENCLOSURETABLE
"ENCLOSURETABLE
  WIDTH 0.1 0.015 0.015 0.015 0.015
  OTHERWIDTH 0.2 BELOW 0.019 0.019 0.019 0.019 ; " ;
```



a) Violation, the 0.015 overhang on the below metal would fulfill the WIDTH 0.1 requirement, but would fail OTHERWIDTH 0.2 based on the width of the above metal

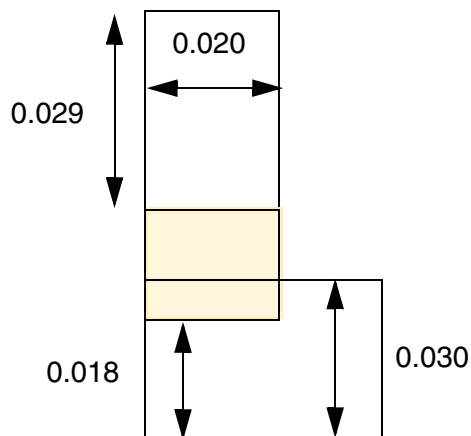
**Figure 1-46 Illustration of Enclosure Table Rules with OTHERWIDTH**



Illustrations of OTHERWIDTH ...  
PARALLEL -parLength WITHIN parWithin  
BELOW ...

**Figure 1-47 Illustration of Enclosure Table Rule with WITHINFIRSTWIDTH**

```
PROPERTY LEF58_ENCLOSURETABLE
  "ENCLOSURETABLE WITHINFIRSTWIDTH
    WIDTH 0.000 0.020 0.015 0.000 0.000
    WIDTH 0.024 0.020 0.015 0.001 0.001; "
```



a) Violation, the enclosure of the first row applies only if the cut is completely within the wire portion with width of 0.020. OK if WITHINFIRSTWIDTH is omitted.

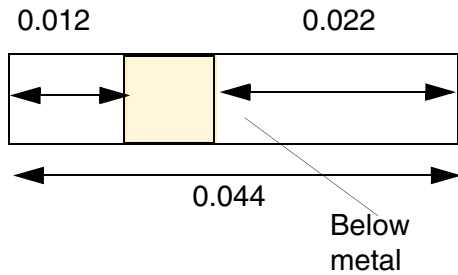
## LEF/DEF 5.8 Language Reference

### LEF Syntax

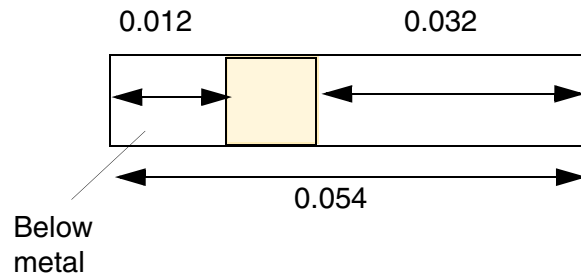
---

**Figure 1-48 Illustration of Enclosure Table Rules**

```
PROPERTY LEF58_ENCLOSURETABLE
  "ENCLOSURETABLE
    WIDTH 0.0 BELOW 0.0 0.0 0.015 0.015
    BELOW 0.0 0.0 0.012 0.012 MAXLENGTH 0.05 ; " ;
```



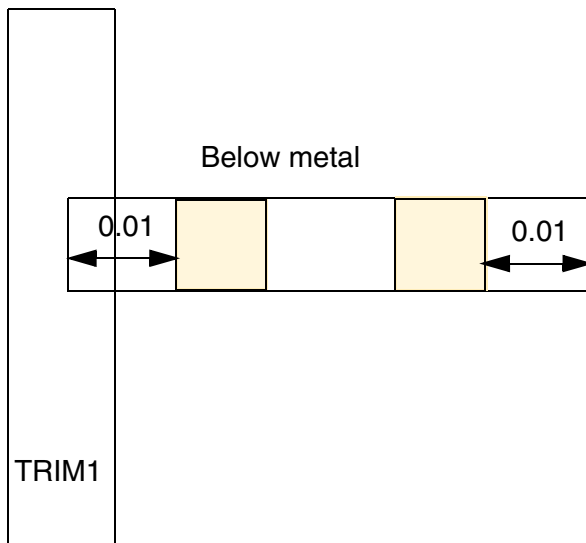
a) OK, the span length of 0.044 ( $\leq 0.05$ ), and enclosure of 0.012 is sufficient



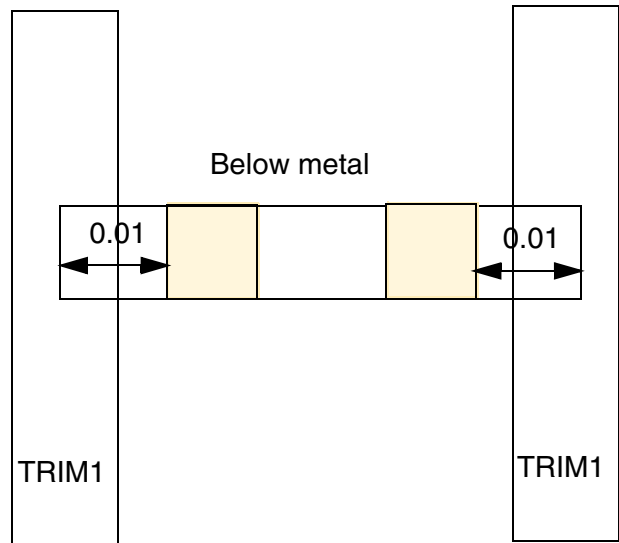
b) Violation, the span length of 0.054 requires 0.015 enclosure

**Figure 1-49 Illustration of Enclosure Table Rules with Layer**

```
PROPERTY LEF58_ENCLOSURETABLE
"ENCLOSURETABLE
  WIDTH 0.0 BELOW 0.02 0.02 0.0 0.0
  WIDTH 0.0 BELOW 0.01 0.01 0.0 0.0
  LAYER TRIM1 OVERLAP 1 ; " ;
```



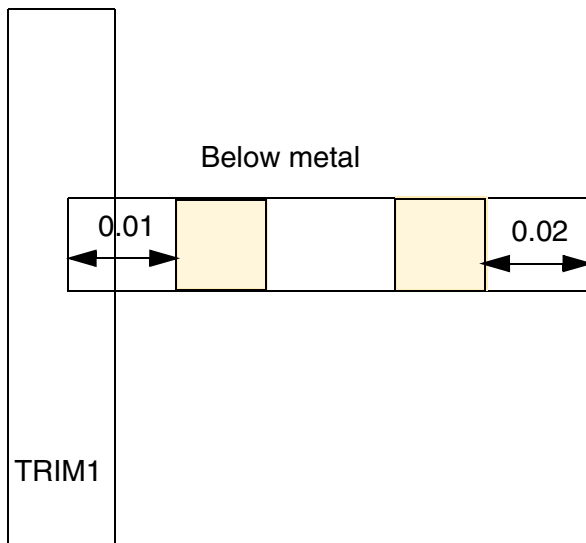
a) Violation, the left cut overlaps with TRIM1 shape, and 0.01 enclosure is sufficient, but the right cut needs 0.02 & fails



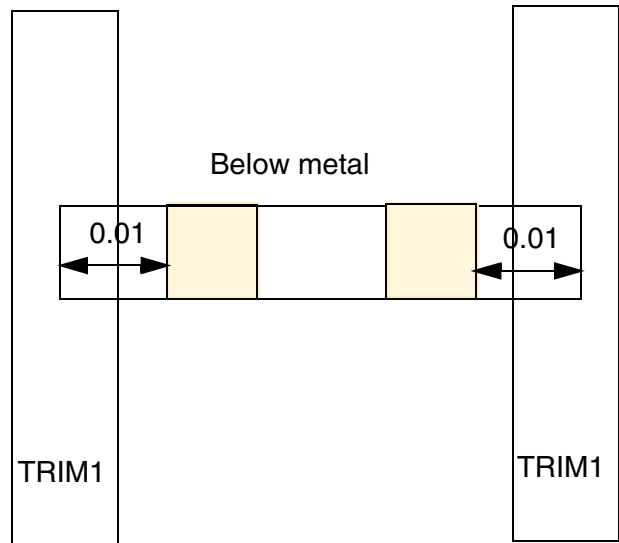
b) OK, both ends overlap with TRIM1 shape, 0.01 enclosure could be used on both of them.

**Figure 1-50 Illustration of Enclosure Table Rules with Layer**

```
PROPERTY LEF58_ENCLOSURETABLE
  "ENCLOSURETABLE
    WIDTH 0.0 BELOW 0.02 0.02 0.0 0.0
    WIDTH 0.0 BELOW 0.01 0.01 0.0 0.0
    LAYER TRIM1 OVERLAP 2 ; " ;
```



a) Violation, only left end overlaps with TRIM1, 0.02 enclosure must be used on both ends. If OVERLAP 1 is given, it would be fine.

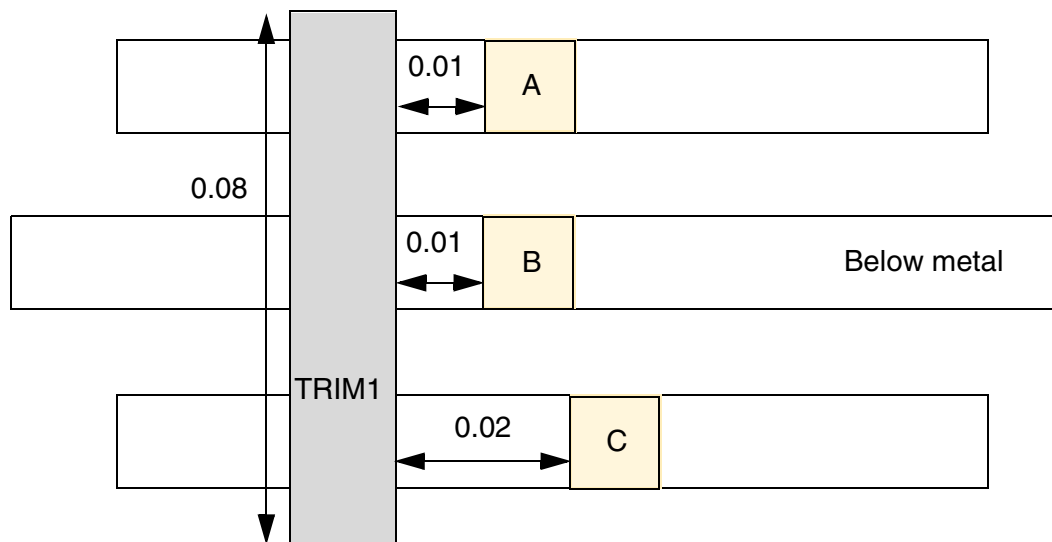


b) OK, both ends overlap with TRIM1 shape, 0.01 enclosure could be used on both of them



**Figure 1-51 Illustration of Enclosure Table Rules with TRIMLENGTHOUTERMETAL**

```
PROPERTY LEF58_ENCLOSURETABLE
  "ENCLOSURETABLE
    WIDTH 0.0 BELOW 0.01 0.01 0.0 0.0
    WIDTH 0.0 BELOW 0.02 0.02 0.0 0.0
    LAYER TRIM1 OVERLAP 1
    TRIMLENGTHOUTERMETAL 0.08 ; " ;
```



a) Violation, via A fails the second `LAYER` enclosure statement. Via B is not subjected to the second statement and passes the first enclosure statement. Via C passes the second enclosure statement.

### **Enclosure Width Rule**

You can create a specific cut-width enclosure rule that does not fit the normal enclosure rule semantics by using the following property definition:

```
PROPERTY LEF58_ENCLOSUREWIDTH
  "ENCLOSUREWIDTH {VIAOVERLAPONLY | USEMINWIDTH}
  ; " ;
```

Where:

```
ENCLOSUREWIDTH VIAOVERLAPONLY
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates a method to check the enclosure width requirements. If a metal shape of a cut via protrudes on a certain side of the wide wire, the enclosure is not checked on that side (or the enclosure should be checked on the sides overlapped with the wide wire). In addition, the enclosure based on the width of the protrusion must also be checked against the cut.

When `ENCLOSUREWIDTH VIAOVERLAPONLY` is defined in a cut layer, for a given width, if only one `ENCLOSURE` statement with `WIDTH` construct having the same overhang values is found, the special edge based enclosure checking is applied. Otherwise, the traditional enclosure method is applied.

Cuts within a wide wire not only need to fulfill the corresponding `ENCLOSURE WIDTH` statements, but also need to fulfill one of the `ENCLOSURE` statements without `WIDTH`.

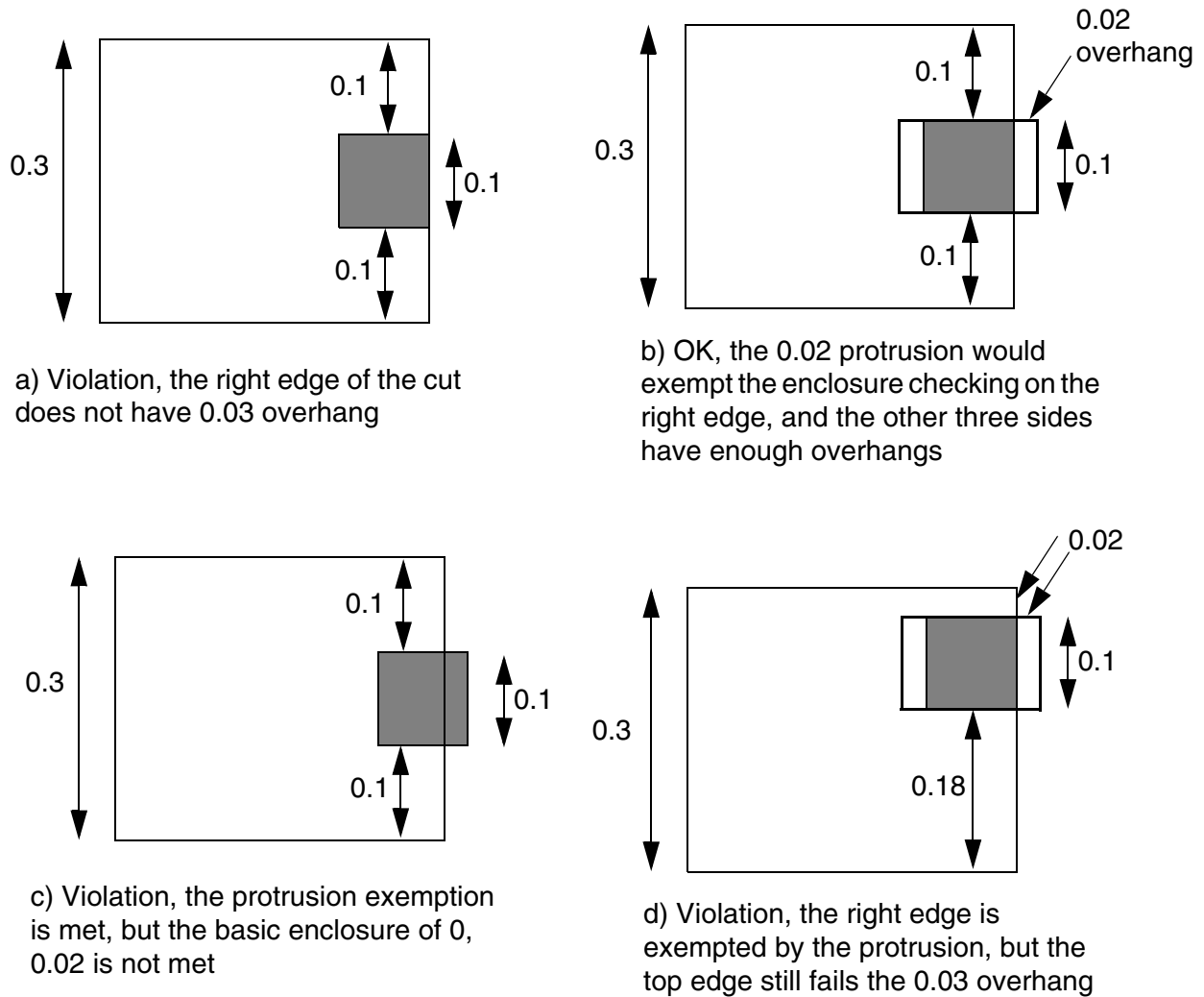
`USEMINWIDTH`

Specifies that the minimum width of the wires that overlap with a cut should be used to look up for the corresponding width-based enclosure requirements. This affects only `ENCLOSURE` statements and has no impact on `ENCLOSURETABLE`. When `INCLUDEABUTTED` is specified on an `ENCLOSURE` statement, it has precedence over `USEMINWIDTH`. This means that a cut overlapping or touching a wire with width greater than or equal to *minWidth* needs to follow only the specified enclosure requirements.

### Enclosure Width Rule Examples

**Figure 1-52 Illustration of Enclosure Width Rules**

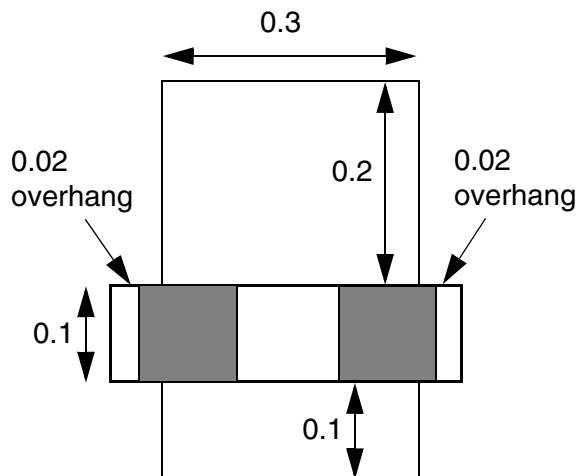
```
PROPERTY LEF58_ENCLOSUREWIDTH
  "ENCLOSUREWIDTH VIAOVERLAPONLY ;" ;
ENCLOSURE 0.0 0.02 ;
ENCLOSURE 0.02 0.02 WIDTH 0.15 ;
ENCLOSURE 0.03 0.03 WIDTH 0.3 ;
```



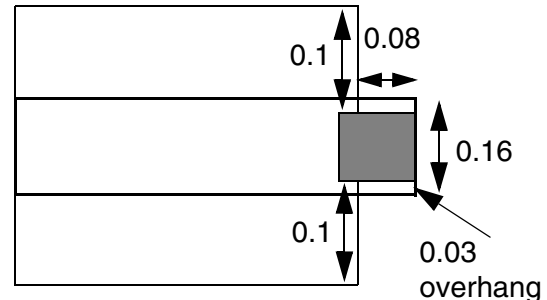
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



e) OK, the protrusion on left and right edges could exempt the corresponding checking on those edges of the cuts, and the rest of the edges have enough overhangs. In addition, the basic enclosure of 0, 0.02 is met



f) Violation, the width of the protrusion is considered as the width of the max fractured rectangle of 0.16, and the corresponding enclosure 0.02, 0.02 is not met.

### Enclosure Joint Rule

You can create enclosure joint rule to specify the overhang between a non-redundant cut to two consecutive joints in either above or both above and below routing layer.

You can create a enclosure joint rule by using the following property definition:

```
PROPERTY LEF58_ENCLOSURETOJOINT
    "ENCLOSURETOJOINT [CUTCLASS className] [ABOVE | BELOW]
        toOneJointOverhang [toBothJointOverhang]
        [EOLMINLENGTH minLength]
        JOINTWIDTH jointWidth JOINTLENGTH spanLength
    ; " ;
```

Where:

ABOVE | BELOW

Specifies that the rule only applies if the via cut is in two consecutive joints on the above or below routing layer.

CUTCLASS *className*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines the enclosure to joint rule for a specific cut class *className*. If CUTCLASS is defined in cut layer, then CUTCLASS must be specified in ENCLOSURETOJOINT statement.

For each cut class, one individual rule with the CUTCLASS keyword should be specified, if needed.

ENCLOSURETOJOINT *toOneJointOverhang toBothJointOverhang*  
JOINTWIDTH *jointWidth* JOINTLENGTH *spanLength*

Specifies the overhang between a non-redundant cut to two consecutive joints in either above or both above and below routing layer, with span greater than *spanLength* and not a EOL edge with length equal to the wire width, which is less than *jointWidth*, to be either *toOneJointOverhang* to one of the joints or *toBothJointOverhang* to both of the joints.

For redundant cuts sharing the same metal on the above and below routing layer, one of the cuts fulfilling the rule will be sufficient.

*Type:* Float, specified in microns

EOLMINLENGTH *minLength*

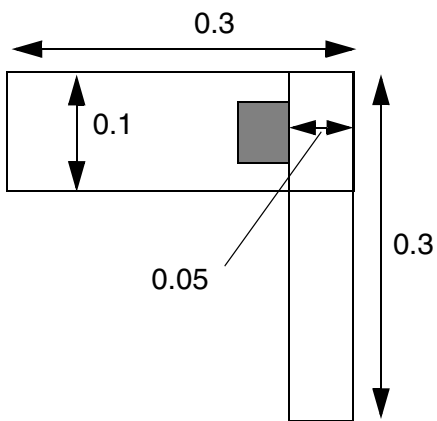
Indicates that the joint must not be a EOL edge with length greater than or equal to the specified *minLength* along both sides and the length of the EOL edge is no longer necessarily equal to the wire width.

*Type:* Float, specified in microns

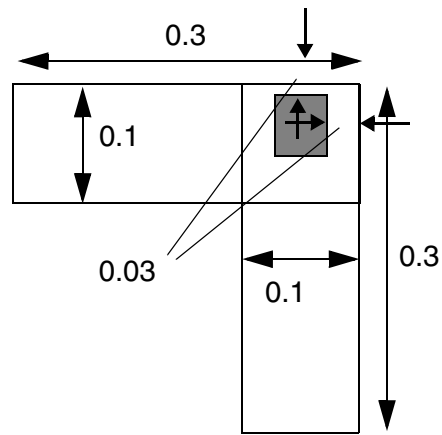
## Enclosure Joint Rule Examples

**Figure 1-53 Illustration of Enclosure Joint Rule**

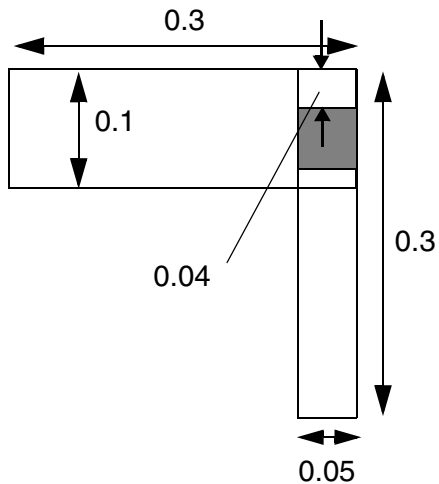
```
PROPERTY LEF58_ENCLOSURETOJOINT  
  "ENCLOSURETOJOINT 0.05 0.03  
    JOINTWIDTH 0.15 JOINTLENGTH 0.1  ;" ;
```



a) OK, having 0.05 overhang to one of the joint edges



b) OK, having 0.03 overhang to both of the joint edges



c) Violation, having 0.04 and 0.0 overhang to the joints will not meet either of the requirements

## EOL Spacing Rule

You can create EOL spacing rule to specify the cut spacing on certain edges of a cut on a EOL edge above routing layer with width less than the specified EOL width.

You can create a EOL spacing rule by using the following property definition:

```
PROPERTY LEF58_EOLSPACING
    "EOLSPACING cutSpacing1 cutSpacing2
        [CUTCLASS className1 [{TO className2 cutSpacing1 cutSpacing2}...]]
        ENDWIDTH eolWidth PRL prl
        ENCLOSURE smallerOverhang equalOverhang
        EXTENSION sideExt backwardExt SPANLENGTH spanLength
    ; " ;
```

Where:

```
[CUTCLASS className1 [{TO className2 cutSpacing1
cutSpacing2}...]]
```

Defines the EOL cut spacing rule for a specific cut class *className* to any other cuts, including those belonging to a different cut class. If TO is defined, the corresponding cut spacings will be applied between *className1* and *className2*. If CUTCLASS is defined in cut layer, then CUTCLASS must be specified in the EOLSPACING statement.

For each cut class, at the most, one individual rule with CUTCLASS keyword should be specified, if required.

```
ENCLOSURE smallerOverhang equalOverhang
```

Specifies that the EOL cut spacing rule only applies if overhang of the cut on one of the edges is less than *smallerOverhang* and the overhang on one of the orthogonal edges exactly equal to *equalOverhang* on above routing layer.

*Type:* Float, specified in microns

```
EOLSPACING cutSpacing1 cutSpacing2 ENDWIDTH eolWidth PRL prl
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the cut spacing on certain edges of a cut on a EOL edge on above routing layer with width less than *eolWidth*, if additional conditions (described later) are met. If the cut has a common parallel run length less than *prl* to a neighbor cut, *cutSpacing1* is applied. Otherwise, *cutSpacing2* is applied in a maximum projection style (when the cuts have common parallel run length greater than or equal to *prl*). If *prl* is negative, it means that a neighbor cut within  $\text{abs}(\text{prl})$  distance, including exactly equal to  $\text{abs}(\text{prl})$ , beyond the edge of a cut will need to be *cutSpacing2* distance from it.

*Type:* Float, specified in microns

EXTENSION *sideExt backwardExt SPANLENGTH spanLength*

Specifies that the EOL cut spacing rule only applies if certain neighbor wire condition is met. If the wire containing the cut has a span length greater than or equal to *spanLength* (having a EOL edge is not a necessary condition), and a neighbor wire with edge parallel to *backwardExt* direction must overlap with a search window by extending *sideExt* along the edge with *smallerOverhang* and *backwardExt* along the orthogonal edge that the cut has overhang exactly equal to *equalOverhang*. Otherwise (the wire containing the cut has a span length less than *spanLength*), a similar search window on the opposite side must not overlap with any neighbor wires with edge parallel to *backwardExt* direction as an additional condition. Then, the cut spacing requirement is applied to the opposite edges of the edge having enclosure less than *smallerOverhang* and the orthogonal edge that the cut has overhang exactly equal to *equalOverhang* with a neighbor.

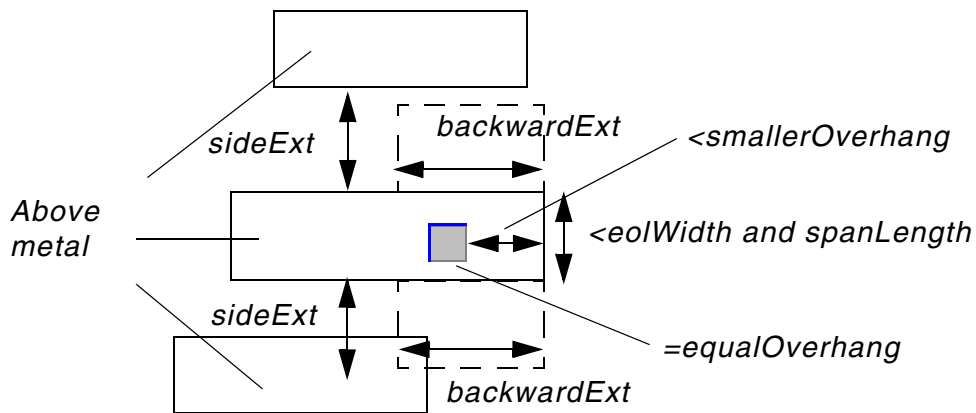
*Type:* Float, specified in microns



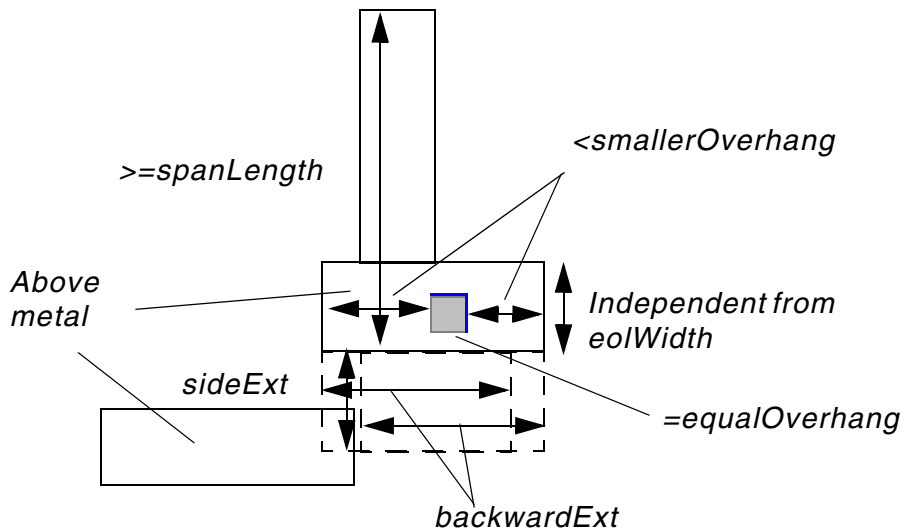
## EOL Spacing Rule Examples

**Figure 1-54 Illustration of EOLSPACING Rule**

Illustration of EOLSPACING ... ENDWIDTH *eolWidth* ...  
ENCLOSURE *smallerOverhang equalOverhang*  
EXTENSION *sideExt backwardExt* SPANLENGTH *spanLength*



The blue edges are subject to the cut spacing rule



There are two EOL edges and two search window at the bottom. Since there is a bottom neighbor wire, the blue edges are subject to the cut spacing rule. Particularly, the left edge is opposite to a EOL edge that fulfills the neighboring condition.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### EOL Enclosure Rule

You can create EOL enclosure rule to specify the enclosure requirement on a EOL edge with width less than the specified EOL width.

You can create a EOL enclosure rule by using the following property definition:

```
PROPERTY LEF58_EOLENCLOSURE
    "EOLENCLOSURE eolWidth [MINEOLWIDTH minEolWidth]
        [EQUALRECTWIDTH] [CUTCLASS className] [ABOVE | BELOW]
        {{LONGEDGEONLY | SHORTEDEGEONLY} overhang
        | overhang
        | exactOverhang
        | PARALLELEDGE parSpace EXTENSION backwardExt forwardExt
        [MINLENGTH minLength]
        | MINLENGTH minLength
        ]
    }
; "
```

Where:

All other keywords are the same as the existing LEF cut layer EOLENCLOSURE syntax.

ABOVE | BELOW

Specifies that the EOL enclosure rule only applies on the above or below routing layer.

CUTCLASS *className*

Defines the EOL enclosure rule for a specific cut class *className*. If CUTCLASS is defined in cut layer, then CUTCLASS must be specified in the EOLENCLOSURE statement.

For each cut class, one individual rule with CUTCLASS keyword must be specified, if required.

EOLENCLOSURE *eolWidth*

Specifies the enclosure requirement on a EOL edge with width less than *eolWidth*. No overhang requirement is specified for a cut on a non-EOL edge in this statement, which is a supplement ENCLOSURE requirement on a EOL edge. The ENCLOSURE requirement on a non-EOL edge is still governed by other ENLCOSURE statements.

*Type:* Float, specified in microns

EQUALRECTWIDTH

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the EOL enclosure rule only applies to EOL edges with length equal to the wire width.

LONGEDGEONLY | SHORTEDEGEONLY *overhang*

Specifies the ENCLOSURE EOL requirement of *overhang* is only applied to the side/long edges in LONGEDGEONLY or end/short edges in SHORTEDEGEONLY of the rectangular cut, which means that a rectangular class name must be defined in CUTCLASS.

MINEOLWIDTH *minEolWidth*

Specifies that the rule only applies when *minEolWidth* is less than or equal to EOL width or length, which should be less than *eolWidth* and the EOL edge must touch a wire with width greater than or equal to *minEolWidth*.

*Type:* Float, specified in microns

MINLENGTH *minLength*

Indicates that the EOL enclosure only applies if EOL edge have length greater than or equal to the *minLength* along both sides. In other words, if the EOL length is less than the *minLength* along any one side, the edge is not a EOL edge to trigger the rule.

*Type:* Float, specified in microns

*overhang* [*exactOverhang*]

Specifies that the enclosure EOL requirement is either exactly equal to *exactOverhang* or greater than or equal to *overhang*.

*Type:* Float, specified in microns

PARALLELEDGE *parSpace* EXTENSION *backwardExt forwardExt*  
[MINLENGTH *minLength*]

Indicates that the EOLenclosure rule only applies if there is a parallel edge on one side that is less than the *parSpace* subtracting the width of the EOL edge away and by extending *backwardExt* going backward and *forwardExt* going forward in the direction orthogonal to the EOL edge.

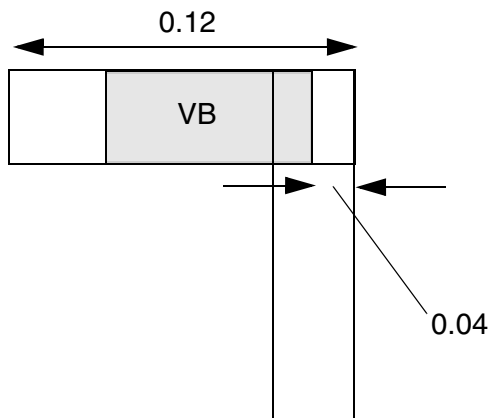
The MINLENGTH keyword indicates that if the EOL length is less than *minLength* along the side, then any parallel edge on that side is ignored, and the rule may not apply.

*Type:* Float, specified in microns

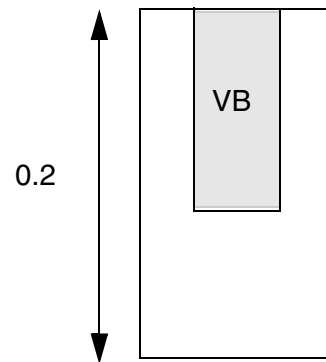
## EOL Enclosure Rule Examples

Figure 1-55 Illustration of EOL Enclosure Rule

```
PROPERTY LEF58_EOLENCLOSURE  
  "EOLENCLOSURE 0.15  
    CUTCLASS VB  
    LONGEDGEONLY 0.04 ; " ;
```



a) Violation, 0.04 overhang must be on the side/long edge when the cut is on a EOL edge



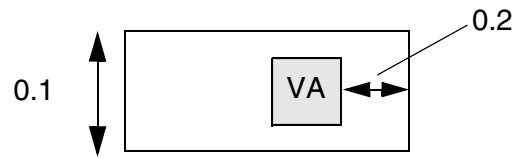
b) OK, the side/long edge of the cut is not on a EOL edge

**Figure 1-56 Illustration of EOL Enclosure Rule with EQUALRECTWIDTH**

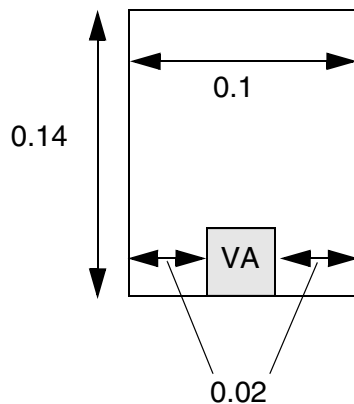
```
PROPERTY LEF58_EOLENCLOSURE
  "EOLENCLOSURE 0.15 EQUALRECTWIDTH
    CUTCLASS VA 0.05 0.0 ; " ;
```



a) OK, zero overhang is allowed



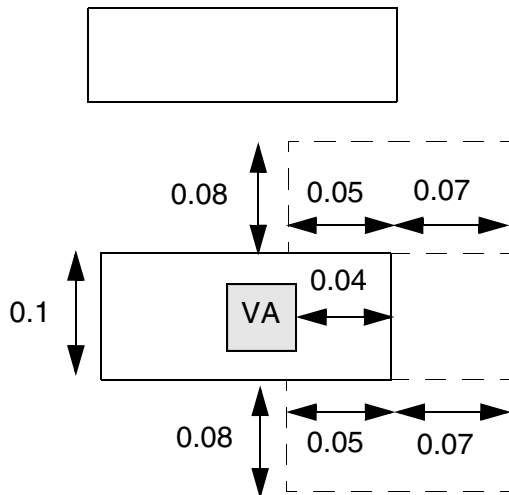
b) Violation, overhang should be either zero or  $\geq 0.05$



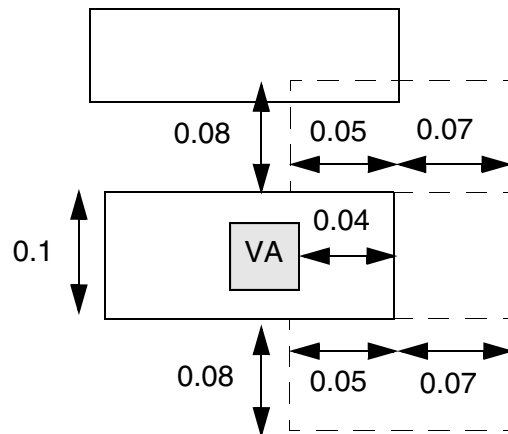
c) OK, the vertical edges are not EOL edges with length equal to wire width, and the rule does not apply on them. The bottom edge has zero overhang, which is allowed. It is a violation if EQUALRECTWIDTH is not specified.

**Figure 1-57 Illustration of EOL Enclosure Rule with PARALLELEDGE and EXTENSION**

```
PROPERTY LEF58_EOLENCLOSURE
  "EOLENCLOSURE 0.15
    CUTCLASS VA 0.05 PARALLELEDGE 0.18
    EXTENSION 0.05 0.07 ; " ;
```



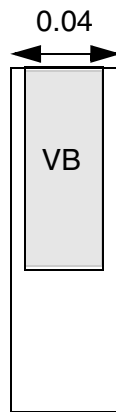
a) OK, the dotted line neighbor searching window is formed by extending 0.08 (0.18 - width of wire of 0.1) along the EOL edge, 0.05 & 0.07 backward & forward perpendicular to the EOL edge, and it could not find a neighbor, which would not trigger the EOL enclosure rule



b) Violation, having a neighbor would trigger the rule, and 0.04 ( $\leq 0.05$ ) enclosure is not enough

**Figure 1-58 Illustration of EOL Enclosure Rule with MINEOLWIDTH**

```
PROPERTY LEF58_EOLENCLOSURE
  "EOLENCLOSURE 0.1 MINEOLWIDTH 0.05
    CUTCLASS VB SHORTEDEGEONLY 0.02 ; " ;
```



b) OK, the rule does not apply on EOL width of 0.04 ( $< 0.05$ ).

## Forbidden Spacing Rule

You can create a forbidden spacing rule to specify forbidden spacing between two cuts.

You can create a forbidden spacing rule by using the following property definition:

```
PROPERTY LEF58_FORBIDDENSPACING
  "FORBIDDENSPACING CUTCLASS className minSpacing maxSpacing
    [SAMEMASK]
    [SHORTEDEGEONLY | LONGEDGEONLY]
    [PRL {prl TO prlClassName}...]
  ; " ;
```

Where:

FORBIDDENSPACING CUTCLASS *className* *minSpacing* *maxSpacing*

Specifies that spacing between two cuts on a given layer belonging to the same *className* cut class cannot be in the range  $\text{minSpacing} \leq \text{spacing} \leq \text{maxSpacing}$ .  
*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

LONGEDGEONLY	Specifies that the forbidden spacing rule only applies to the side/long edge of cuts with a common parallel run length greater than 0. This means that the <i>className</i> must be a rectangular cut class.
SAMEMASK	Specifies that the forbidden spacing rule only applies to same-mask cuts.
SHORTEDGEONLY	Specifies that the forbidden spacing rule only applies to the end/short edge of <i>className</i> cuts with a common parallel run length greater than 0. This means that <i>className</i> must be a rectangular cut class.
PRL { <i>prl</i> TO <i>prlClassName</i> }...	<p>Specifies that forbidden spacing is between a <i>className</i> cut to <i>prlClassName</i> cut with common parallel run length greater than <i>prl</i>. If <i>prl</i> is negative, it is similar to extending the end or short edge by <math>\text{abs}(\text{prl})</math> in a WITHIN style. If some of the cut classes are not defined in <i>prlClassName</i>, this means that there is no constraint between <i>className</i> to those undefined cut classes. See <a href="#">Figure 1-60</a> on page 130.</p> <p><i>Type:</i> Float, specified in microns</p>

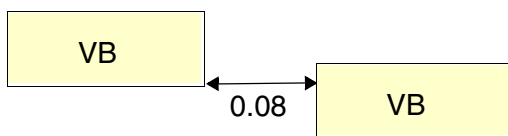


### Forbidden Spacing Rule Examples

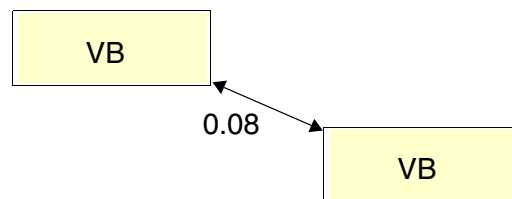
- The following forbidden spacing table rule illustrates forbidden spacing between two VB cuts:

```
PROPERTY LEF58_FORBIDDENSPACING
  "FORBIDDENSPACING CUTCLASS VB 0.07 0.09
  SHORTEGEONLY ; " ;
```

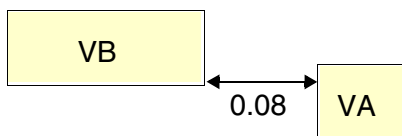
**Figure 1-59 Illustration of Forbidden Spacing Rule**



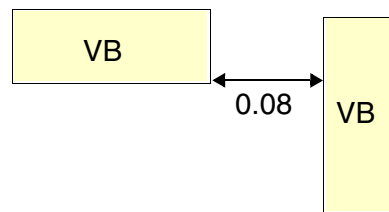
a) Violation, the spacing between the two short/end edges is inside the forbidden spacing range.



b) OK, the cuts do not have common parallel run length > 0. Violation, if SHORTEGEONLY is omitted.



c) OK, the forbidden spacing is only between two VB cuts.



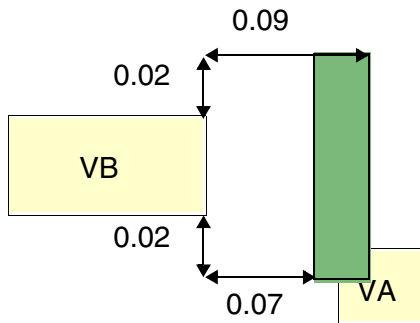
d) OK, the forbidden spacing is only between two short/end edges. Violation if SHORTEGEONLY is omitted.

**Figure 1-60 Illustration of Forbidden Spacing Rule**

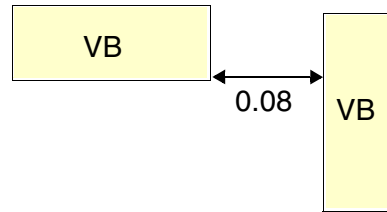
```
PROPERTY LEF58_FORBIDDENSPACING
```

```
  "FORBIDDENSPACING CUTCLASS VB 0.07 0.09
```

```
    SHORTEDEGEONLY PRL -0.02 TO VA -0.03 TO VB ; " ;
```



a) Violation, the forbidden spacing is between a short/end edge of VB to VA within  $\text{abs}(-0.02)$ , and the green region shows an forbidden area for any VA



b) Violation, the forbidden spacing is between a short/end edge of VB to any edges of VB. OK if PRL is omitted.

## Keep-out Zone Rule

The keep-out zone rule can be used to demarcate a cut spacing zone in which cuts of other classes cannot overlap.

You can create a keep-out zone rule by using the following property definition:

```
PROPERTY LEF58_KEEPOUTZONE
```

```
  "KEEPOUTZONE CUTCLASS className1 [TO className2]
```

```
    [SAMEMASK]
```

```
    [EXCEPTEXACTALIGNED [SIDE | END] spacing]
```

```
    {EXTENSION sideExtension forwardExtension |
```

```
    ENDEXTENSION endSideExtension endForwardExtension
```

```
    SIDEEXTENSION sideSideExtension sideForwardExtension |
```

```
    HORIZONTALEXTENSION horzSideExtension horzForwardExtension
```

```
    VERTICALEXTENSION vertSideExtension vertForwardExtension}
```

```
    SPIRALEXTENSION extension [CORNERONLY]
```

```
  ;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

**CORNERONLY** Specifies that the SPIRALEXTENSION keep-out zone applies only on the corners that are outside the zones of EXTENSION, ENDEXTENSION and SIDEEXTENSION, or HORIZONTALEXTENSION and VERTICALEXTENSION.

**EXCEPTEXACTALIGNED** [SIDE | END] *spacing*

Specifies that if cuts are exactly aligned, they just need to be *spacing* apart. These cuts are not subjected to the keepout zone checking. For rectangular cuts, SIDE and END would mean that side/long edges and end/short edges are exactly aligned correspondingly. If neither of them is specified for rectangular cuts, exactly aligned cuts on any edge would only be spacing apart.

**HORIZONTALEXTENSION** *horzSideExtension horzForwardExtension*  
**VERTICALEXTENSION** *vertSideExtension vertForwardExtension*

Specifies the extensions of the keep-out zone on the neighbor cuts based on the horizontal or vertical cut edge. This is particularly useful in case of a square cut.

*Type:* Float, specified in microns

```
KEEPOUTZONE CUTCLASS className1 [TO className2]  
{EXTENSION sideExtension forwardExtension |  
ENDEXTENSION endSideExtension endForwardExtension  
SIDEEXTENSION sideSideExtension sideForwardExtension}  
SPIRALEXTENSION extension
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies a keep-out zone for cut class *className1* to cut class *className2*, if specified, or any cut classes.

ENDEXTENSION and SIDEEXTENSION must be used only on rectangular cuts.

EXTENSION forms a keep-out zone by extending *forwardExtension* on both sides on any cut edges and *sideExtension* in the orthogonal direction toward outside the cut.

SPIRALEXTENSION forms another keep-out zone by extending *extension* on all cut edges in Euclidean measurement. It is a violation if a *className2* cut, if specified, or any cut is inside any of the two keep-out zones. Similarly, ENDEXTENSION and SIDEEXTENSION specify the extensions on the end or short edges and the side or long cut edges of *className1*, respectively.

*Type:* Float, specified in microns

Note that this keep-out zone rule is an alternative cut spacing definition of CUTCLASS SPACINGTABLE. If any defined cut class does not appear in CUTCLASS SPACINGTABLE, then one or more KEEPOUTZONE statements must be specified for that cut class to define its cut spacing to the other cut classes. For example, suppose there are three cut classes: VA, VB, and VC. If VA is not defined in CUTCLASS SPACINGTABLE, then VA must be defined as *className1* in a KEEPOUTZONE statement. There can either be one KEEPOUTZONE statement without a TO cut class (so that it is applied to both VB and VC) or there can be two separate KEEPOUTZONE statements, one with TO VB and another with TO VC.

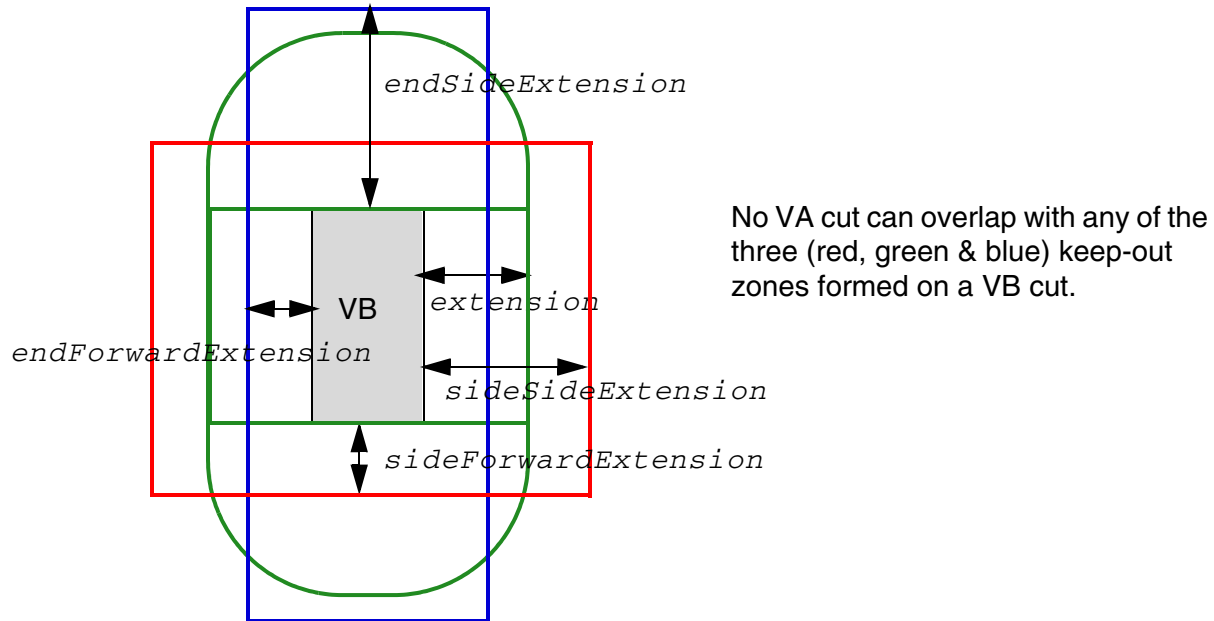
SAMEMASK Specifies the keep-out zone rule applies only to same-mask cuts.

### Keep-out Zone Rule Examples

- The following example illustrates the keep-out zone rule:

```
KEEPOUTZONE CUTCLASS VB TO VA
  ENDEXTENSION endSideExtension endForwardExtension
  SIDEEXTENSION sideSideExtension sideForwardExtension
  SPIRALEXTENSION extension
```

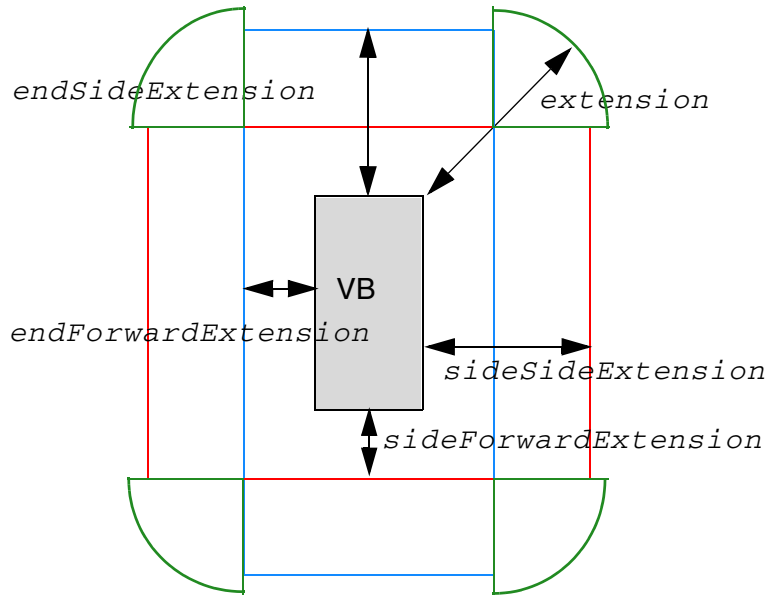
**Figure 1-61 Illustration of the Keep-out Zone Rule**



- The following example illustrates the keep-out zone rule with CORNERONLY:

```
KEEPOUTZONE CUTCLASS VB TO VA
  ENDEXTENSION endSideExtension endForwardExtension
  SIDEEXTENSION sideSideExtension sideForwardExtension
  SPIRIALEXTENSION extension CORNERONLY
```

**Figure 1-62 Illustration of the Keep-out Zone Rule with CORNERONLY**

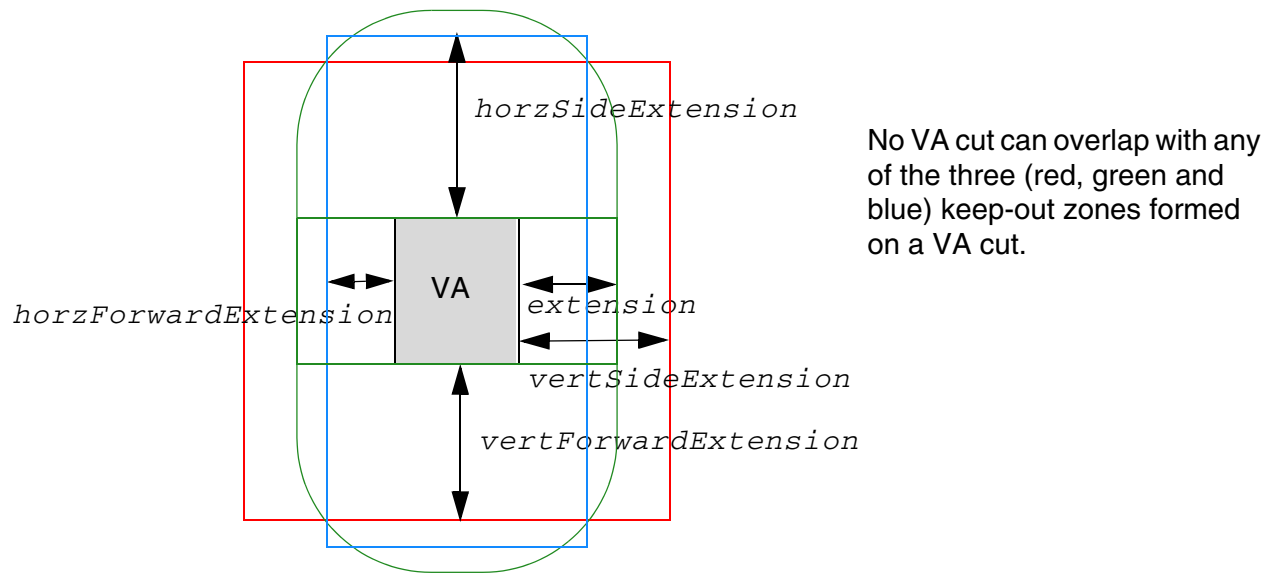


No VA cut can overlap with any of the three (red, green and blue) keep-out zones formed on a VB cut. The green keep-out zone exists only in the corners that are not covered by the red and blue keep-out zones.

- The following example illustrates the keep-out zone rule with HORIZONTALEXTENSION and VERTICALEXTENSION:

```
KEEPOUTZONE CUTCLASS VA TO VA
HORIZONTALEXTENSION horzSideExtension horzForwardExtension
VERTICALEXTENSION vertSideExtension vertForwardExtension
SPIRIALEXTENSION extension
```

**Figure 1-63 Illustration of the Keep-out Zone Rule with HORIZONTAL EXTENSION and VERTICAL EXTENSION**



## Manufacturing Grid Rule

You can create a manufacturing grid rule to specify the manufacturing grid on a given layer to override the library manufacturing grid value.

You can create a manufacturing grid rule by using the following property definition:

```
PROPERTY LEF58_MANUFACTURINGGRID
    "MANUFACTURINGGRID value
    ; " ;
```

Where:

MANUFACTURINGGRID *value*

Specifies the manufacturing grid on the given layer that overrides the library manufacturing grid value.

*Type:* Float, specified in microns

## Maximum Spacing Rule

You can create a maximum spacing rule to specify the maximum spacing between any two cuts.

You can create a maximum spacing rule by using the following property definition:

```
PROPERTY LEF58_MAXSPACING
    "MAXSPACING spacing [CUTCLASS className]
    ;" ;
```

Where:

*CUTCLASS className*

Specifies that the maximum spacing between cuts of the given *className* to any cuts. If *CUTCLASS* is defined in cut layer, then *CUTCLASS* must be specified in the *MAXSPACING* statement.

You can specify individual rules with the *CUTCLASS* keyword for each cut class, if needed.

*MAXSPACING spacing*

Specifies the maximum spacing between any two cuts.  
*Type:* Float, specified in microns

## Spacing with Same Metal Rule

You can use spacing with same metal rules to:

- Require extra space between cuts on different nets that overlap orthogonally, in order to avoid stress migration between the cuts. This rule does not apply if the cuts have the same metal above or below.
- Require extra spacing between different cut layers unless they are connected by a single metal shape.

You can create a spacing with same metal rule by using the following property definition:

```
PROPERTY LEF58_SPACING
    "SPACING cutSpacing
    [SAMEMASK
    |MAXXY
    | [CENTERTOCENTER]
    [SAMENET | SAMEMETAL | SAMEVIA]
    [ LAYER secondLayerName
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[STACK
| ORTHOGONALSPACING orthogonalSpacing]
| CUTCLASS className
    [SHORTEDGEONLY [PRL prl]
    | CONCAVECORNER
        [WIDTH width ENCLOSURE enclosure
        EDGELENGTH edgeLength
        | PARALLEL parLength WITHIN parWithin
        ENCLOSURE enclosure
        | EDGELENGTH edgeLength
        ENCLOSURE edgeEnclosure adjEnclosure]
    | EXTENSION extension
    | NONEOLCONVEXCORNER eolWidth
        [MINLENGTH minLength]
    | ABOVEWIDTH width [ENCLOSURE enclosure]
    | MASKOVERLAP
    | WRONGDIRECTION]]]
| ADJACENTCUTS {2 | 3 | 4}
    [EXACTALIGNED exactAlignedCut]
    [TWO CUTS twoCuts [TWO CUT SPACING twoCutsSpacing]
    [SAME CUT]]
    WITHIN cutWithin | cutWithin1 cutWithin2
    [EXCEPT SAME PGNET]
    [EXCEPT ALL WITHIN exceptAllWithin]
    [ENCLOSURE [ABOVE|BELOW] enclosure]
    [CUTCLASS className [TO ALL]]
    [NOPRL | SIDE PARALLEL OVERLAP]
    [SAME MASK]
| PARALLEL OVERLAP [EXCEPT SAME NET | EXCEPT SAME METAL
| EXCEPT SAME METAL OVERLAP | EXCEPT SAME VIA]
| PARALLEL WITHIN within [EXCEPT SAME NET]
    [CUTCLASS className
    [LONG EDGE ONLY
    | ENCLOSURE enclosure {ABOVE | BELOW}
    PARALLEL parLength WITHIN parWithin]]
| SAME METAL SHARED EDGE parwithin [ABOVE] [CUTCLASS className]
    [EXCEPT TWO EDGES] [EXCEPT SAME VIA numCut]
| AREA cutArea ; ... " ;
```

Where:

All other keywords are the same as the existing LEF cut layer SPACING syntax.

ABOVEWIDTH *width* [ENCLOSURE *enclosure*]

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the cut to metal spacing applies only on the above metal layer wire with width greater than or equal to the *width*. This means that the second layer must be a metal layer. In addition, the spacing applies only to different-net objects.

When you define the `ENCLOSURE` keyword, the spacing applies only if the enclosure on layer *secondLayerName* of the cut is less than the specified *enclosure*, including the enclosure at the corners.

*Type:* Float, specified in microns

ADJACENTCUTS ... SAMEMASK

Specifies that the adjacent cut rule only applies to same mask cuts. The presence of different mask cuts is irrelevant for this adjacent cut rule checking.

ADJACENTCUTS... WITHIN {*cutWithin* | *cutWithin1 cutWithin2*}

Specifies that the adjacent cut rule only counts neighbor cut greater than or equal to *cutWithin1* and less than *cutWithin2* when two within values are specified. See [Figure 1-81](#) on page 162.

ADJACENTCUTS {2 | 3 | 4} [EXACTALIGNED *exactAlignedCut*] WITHIN *cutWithin* [NOPRL | SIDEPARALLELOVERLAP]

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

The `EXACTALIGNED` keyword specifies that the adjacent cut spacing rule applies when the cut has at least *exactAlignedCut* via cuts that are perfectly aligned horizontally and/or vertically and are less than *cutWithin* distance from each other. Otherwise, the adjacent cut spacing rule applies when the number of adjacent cuts is equal to or greater than the specified number of cuts after the `ADJACENTCUTS` keyword, which must be smaller than *exactAlignedCut*, and at least one adjacent cut is not exactly aligned.

The `NOPRL` keyword specifies that the adjacent cut rule only applies among cuts with no common parallel run length. The `EXACTALIGNED` keyword cannot be specified along with `NOPRL`.

The `SIDEPARALLELOVERLAP` keyword indicates that the adjacent cut spacing rule applies only when there is a parallel edge overlap greater than 0 side by side between two rectangular cut vias.

**Note:** Do not use the `SIDEPARALLELOVERLAP` for square cut classes.

`CUTCLASS className [TO ALL]`

Defines the adjacent cut spacing rule for a specific cut class (*className*). If `CUTCLASS` is defined in cut layer, then `CUTCLASS` must be specified in the `SPACING` statement.

You can specify individual rules with the `CUTCLASS` keyword for each cut class, if needed.

The `TO ALL` keyword specifies that the adjacent cut rule applies between cuts of *className* to cuts of any cut classes.

When both `TO ALL` and `TWOCUTS` are specified, the two cuts having multiple neighbor cuts to violate `TWOCUTS` must still belong to *className* to be a violation.

`CUTCLASS className [LONGEDGEONLY]`

Specifies that the parallel within cut spacing only applies to cuts belonging to the same *className*. If *className* is a rectangular cut class, then you can specify the `LONGEDGEONLY` keyword to limit the cut spacing between two long/side edges having a neighbor wire on the above metal between the cuts.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

CUTCLASS *className* [SHORTEDGEONLY [PRL *prl*] | CONCAVECORNER]

Defines the inter-layer cut spacing of *className* to a metal in *secondLayerName*, which must be a routing layer. If *className* is a rectangular cut class, the SHORTEDGEONLY statement will limit the spacing from an end/short edge of the cut to a metal when there is common parallel run length greater than 0 or greater than *prl* if PRL is defined between them. This cut to metal spacing is ignored for same-net.

The CONCAVECORNER keyword specifies the spacing of the cut to the concave corner of a wire that contains the cut.

EDGELENGTH *edgeLength* ENCLOSURE *edgeEnclosure* *adjEnclosure*

Specifies that an edge of a concave corner of a wire containing a cut has length less than *edgeLength*, the other end of that edge is a convex corner and the cut has parallel run length greater than or equal to 0 to that edge, the spacing between that edge and the facing cut edge must be greater than or equal to *cutSpacing*. In addition, if that spacing/enclosure is less than *edgeEnclosure* (and greater than or equal to *cutSpacing*, which must be less than *edgeEnclosure*), the enclosure of the adjacent two opposite sides must be greater than or equal to *adjEnclosure*. Otherwise, it is a violation.

*Type:* Float, specified in microns

ENCLOSURE [ABOVE | BELOW] *enclosure*

Specifies that the adjacent cut rule applies only if the checking cut has enclosure less than *enclosure* on any one side in the preferred routing direction on the above and below metal layers. If ABOVE or BELOW is specified, the enclosure requirement would be applied only on the specified above or below metal layer. See [Figure 1-83](#) on page 163 for an example.

*Type:* Float, specified in microns

ENCLOSURE *enclosure* {ABOVE | BELOW}  
PARALLEL *parLength* WITHIN *parWithin*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the cut spacing only applies if the enclosure on either above or below metal layer (depending on whether ABOVE or BELOW is defined) is less than *enclosure* on a side having a neighbor wire within less than *parWithin* and common parallel run length to the cut greater than or equal to *parLength* on one and only one side. The cut spacing is only applied to the side without a neighbor wire and on the facing edge of the neighbor cut. If *parLength* is a negative value, the two edges would be checked in a WITHIN style. In addition, the rule is applied between a cut of *className* to cuts of any cut classes. See [Figure 1-93](#) on page 172.

*Type:* Float, specified in microns

EXCEPTALLWITHIN *exceptAllWithin*

Specifies the exception that if a cut has all neighbors within *cutWithin* or range of *cutWithin1* and *cutWithin2*, which are also within *exceptAllWithin*, it is allowed. *exceptAllWithin* should be less than or equal to *cutSpacing*. See [Figure 1-76](#) on page 158.

*Type:* Float, specified in microns

EXCEPTSAMENET Indicates that the parallel overlap rule does not apply to same-net cuts.

EXCEPTSAMEMETAL Indicates that the parallel overlap rule does not apply to cuts that share the same metal shapes above or below metal layers of the cuts.

EXCEPTSAMEMETALOVERLAP

Indicates that the parallel overlap rule does not apply to cuts that share the same metal shapes above or below metal layers that cover the projected overlap area between the cuts. This is the default.

EXCEPTSAMENET Indicates that the parallel within rule does not apply to same-net cuts.

EXCEPTSAMEVIA Indicates that the parallel overlap rule does not apply to cuts that share the same metal shapes both above and below the metal layers that cover the overlap area between the cuts.

EXTENSION *extension*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

	<p>Specifies that the <i>extension</i> should be extended on the cut edges that do not fulfill the overhang defined in <code>ENCLOSUREEDGE OPPOSITE</code> before <i>cutSpacing</i> is applied between the extended edges to the metal in <i>secondLayerName</i>.</p> <p><i>Type:</i> Float, specified in microns</p>
MASKOVERLAP	<p>Specifies the cut to metal, containing the cut, spacing on the overlap area of two different masks. This means that the second layer must be a metal layer.</p>
MAXXY	<p>Indicates that the <i>cutSpacing</i> value is used as the largest x or y distance for spacing between objects. This keyword can be applied only when <code>EUCLIDEAN</code> is specified in <code>CLEARANCEMEASURE</code>.</p>
NONEOLCONVEXCORNER	<p><i>eolWidth</i> [<code>MINLENGTH</code> <i>minLength</i>]</p> <p>Specifies the spacing of a cut to a convex corner that does not touch a EOL edge with width less than <i>eolWidth</i> of a metal shape containing the cut in form of a triangle formed by the smaller edge length or <i>cutSpacing</i>. In other words, the triangle is a keepout region that the cut could not overlap.</p> <p>The <code>MINLENGTH</code> keyword indicates that the EOL edge must have length greater than or equal to the <i>minLength</i> along both sides. In other words, if the EOL length is less than <i>minLength</i> along any one side, then the edge is not a EOL edge.</p> <p><i>Type:</i> Float, specified in microns</p>
ORTHOGONALSPACING	<p><i>orthogonalSpacing</i></p>

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the different-net spacing between objects on the cut layer and objects on the *secondLayerName*, which must be a routing layer. The rule does not apply to same-net objects.

The spacing check is done by using the following method:

The difference between *cutSpacing* and *orthogonalSpacing* is calculated, and half of that difference is shrunk on two sides of the cut and grown on the other two sides. Then, a Euclidean spacing of the average of *cutSpacing* and *orthogonalSpacing* is applied to the newly formed rectangle geometry. The shrink and grow operations are done on the alternated sides to form another checking region. A violation occurs if neighbor wires on the *secondLayerName* are found in both the regions.

*Type:* Float, specified in microns

**Note:** You cannot use ORTHOGONALSPACING along with CENTERTOCENTER, SAMENET, SAMEMETAL, or SAMEVIA.

PARALLELOVERLAP

Indicates that the cuts that have a parallel edge overlap greater than 0 require *cutSpacing* distance between them. Only one PARALLELOVERLAP spacing value is allowed per cut layer.

**Note:** You should not use PARALLELOVERLAP along with SAMENET, SAMEMETAL, or SAMEVIA.

PARALLEL *parLength* WITHIN *parWithin* ENCLOSURE *enclosure*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the cut to concave corner spacing only applies for a wire with default wire width containing the cut and all the conditions as described below hold true. The cut has zero enclosure on two opposite sides, and the enclosure is less than *enclosure* on one of the other two opposite sides on the specified *secondLayerName*. There are neighbor wires on both metal layers of the cut on opposite sides. The spacing to the below metal must be exactly equal to  $parWithin - 0.001$  (due to `WITHIN` nature, the given value is offset by  $0.001$ ) and the above metal neighbor is within *parWithin* of the cut edge with zero enclosure having parallel run length greater than *parLength*. If *parLength* is a negative value, the cut edges would be extended by  $abs(parLength)$  along both the sides to search for the neighbors. See [Figure 1-75](#) on page 157. There is a corner case on the zero enclosure requirement. If the cut edge having a neighbor not on layer *secondLayerName* touches or collides with the concave corner, the rule will be applicable.

*Type:* Float, specified in microns

`PARALLELWITHIN` *within*

Specifies that if the edge of a neighbor cut is within *within* distance beyond the edge of another cut, *cutSpacing* is required between them. If a cut layer has more than one cut class, the parallel within rule could be applied among all cut classes, including via cuts belonging to different cut classes

**Note:** You should not use `PARALLELWITHIN` along with `SAMENET`, `SAMEMETAL`, or `SAMEVIA`. In addition, you can either specify `PARALLELOVERLAP` or `PARALLELWITHIN`.

`SAMECUT`

Specifies that the adjacent cut rule only applies between two cuts, each having the same cut neighbors, besides potential neighbor of each other. See [Figure 1-83](#) on page 163.

`SAMEMASK`

Indicates that the minimum spacing rule only applies between objects on the same mask.

`SAMEMETAL`

Indicates that the *cutSpacing* value only applies to cuts that are overlapped with the same metal shape. The `SAMEMETAL` *cutSpacing* value should be smaller than the normal `SPACING` *cutSpacing* value that applies to different-net cuts.

See [Figure 1-65](#) on page 148 for an example of the difference between `SAMEMETAL` and `SAMENET`.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

SAMEMETALSHAREDEDGE *parWithin*

[ABOVE] [EXCEPTTWOEDGES] [EXCEPTSAMEVIA *numCut*]

Specifies the spacing greater than and equal to *cutSpacing* between two via cuts that have a common parallel run length greater than 0, have common above and /or below metal shapes covering the entire length of common projection between them and have neighbor wire(s) within *parWithin* distance from them on the same edge.

ABOVE specifies that the rule only applies if the via cut must have a common metal routing layer. This means that having a common below metal routing layer of the cuts is irrelevant.

EXCEPTTWOEDGES specifies that if the cuts have two neighbor wires within *parWithin* distance on the opposite sides, the spacing rule is ignored.

EXCEPTSAMEVIA *numCut* specifies that if there are greater than and equal to *numCut* of cuts having common same metal on both the above and below layers, the spacing rule is ignored.  
*Type: Integer*

SAMEVIA

Indicates that the *cutSpacing* value only applies to cuts that share the same metal shapes above and below that cover the overlap area between the cuts. The SAMEVIA *cutSpacing* value should be smaller than the normal SPACING *cutSpacing* value that applies to different-metal cuts.

TWOCUTS *twoCuts*

Specifies the cut spacing between two cuts, each having a specified number of neighbor cuts, defined using *twoCuts*, that are within (or less than) a *cutWithin* distance away, that is the *cutSpacing*. The value of *twoCuts* must be smaller than the specified number of cuts defined with the ADJACENTCUTS keyword. See [Figure 1-80](#) on page 162 for an example.  
*Type: Integer*

TWOCUTSSPACING *twoCutsSpacing*

Specifies that the adjacent cut rule applies only if all of the cuts fulfilling the TWOCUTS condition have a spacing less than *twoCutsSpacing*. See [Figure 1-81](#) on page 162 for an example.

WIDTH *width* ENCLOSURE *enclosure* EDGELENGTH *edgeLength*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the cut to concave corner spacing applies if the width of an edge forming the concave corner is less than or equal to the *width*, the length of the adjacent edge is greater than or equal to the *edgeLength*, and forms a search window for cuts by extending *cutSpacing* on the concave corner along both sides of the edge with width less than or equal to the *width*. Any cut edge parallel to the metal edge with width less than or equal to the *width* inside that search window must have enclosure greater than or equal to *enclosure*. Otherwise, it will be a violation.

*Type:* Float, specified in microns

WRONGDIRECTION

Specifies the spacing for a cut to a non-preferred direction metal, when the cut is contained in a preferred direction same-metal wire. This indicates that the second layer must be a metal layer.

### Spacing Rule Examples

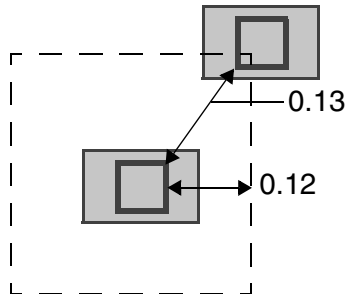
- The following spacing rule indicates that the minimum spacing of objects on the same mask in the routing layer is 0.15  $\mu\text{m}$  while the minimum spacing of objects on different masks is 0.10  $\mu\text{m}$ :

```
SPACING 0.10 ;  
PROPERTY LEF58_SPACING  
    "SPACING 0.15 SAMEMASK ; " ;
```

- The following spacing rule indicates that cuts should have x or y distance of maximum 0.12  $\mu\text{m}$  between them:

```
PROPERTY LEF58_SPACING  
    "SPACING 0.12 MAXXY ;" ;
```

**Figure 1-64 Illustration of Spacing Rule with MAXXY**



a) Violation, a neighbor cut edge is within 0.02 of a cut edge, 0.1 spacing is required between them.

- The following figure illustrates the SAMEMETAL rules for the following examples.

If the *via3* layer has the following spacing rules:

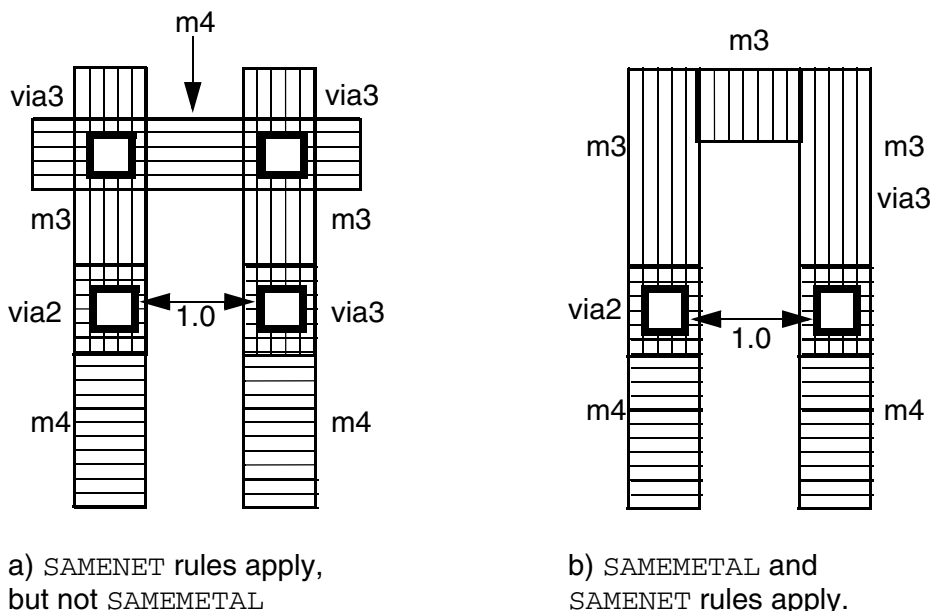
```
SPACING 1.5 ;                #via3 to via3 spacing
SPACING 1.5 LAYER via2 ;     #via3 to via2 spacing
```

Then both a) and b) are violations.

- If the *via3* layer has the following spacing rules, then a) is a violation, but b) is allowed:

```
SPACING 1.5 ;
SPACING 1.5 LAYER via2 ;
PROPERTY LEF58_SPACING
    "SPACING 0 SAMEMETAL LAYER via2 STACK ;" ;
```

**Figure 1-65 Illustration of SAMEMETAL Rules**



- The following spacing rule specifies that a VA via can have at most one neighboring VA via within 0.30  $\mu\text{m}$  center-to-center distance away:

```
PROPERTY LEF58_SPACING
```

```
"SPACING 0.30 CENTERTOCENTER ADJACENTCUTS 2 WITHIN 0.30 CUTCLASS VA ;" ;
```

- The following spacing rule specifies that 0.1  $\mu\text{m}$  spacing is required between two cuts if one cut is within 0.02  $\mu\text{m}$  beyond the edge of the other cut:

```
PROPERTY LEF58_SPACING
```

```
"SPACING 0.09 ;" ;
```

```
"SPACING 0.1 PARALLELWITHIN 0.02 ;" ;
```

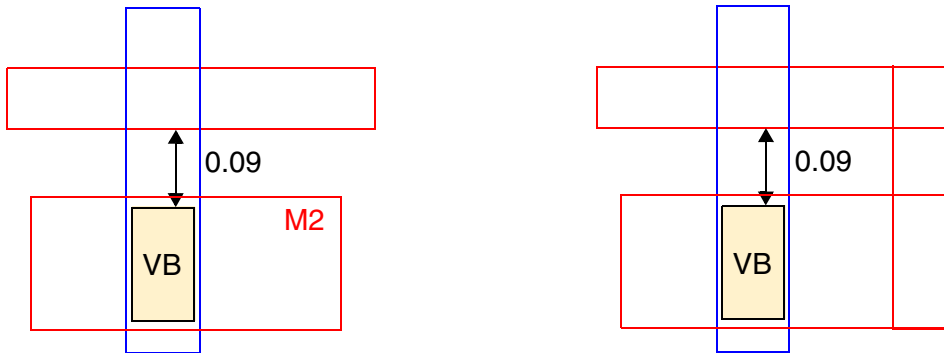
- The following spacing rule indicates that same-net is exempted for the cut to metal rule:

```
PROPERTY LEF58_SPACING
```

```
"SPACING 0.1 LAYER M2
```

```
CUTCLASS VB SHORTEDEGEONLY ;" ;
```

**Figure 1-66 Illustration of CUTCLASS with SHORTEDEGEONLY**

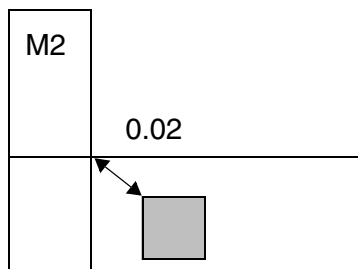


a) Violation, the spacing between the short/end edge of a VXBAR to an edge of the above metal wire is 0.09 ( $< 0.1$ )

b) OK, same-net is exempted for the cut to metal rule

**Figure 1-67 Illustration of CUTCLASS with CONCAVECORNER**

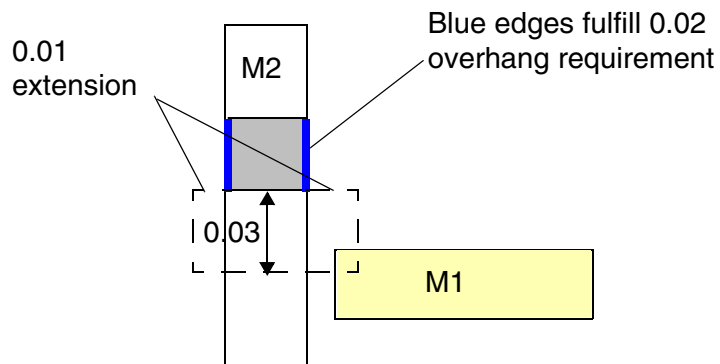
```
PROPERTY LEF58_SPACING
  "SPACING 0.03 LAYER M2 CUTCLASS VA
    CONCAVECORNER ; " ;
```



a) Violation, the cut needs to be 0.03 away from the concave corner

**Figure 1-68 Illustration of CUTCLASS with Extension**

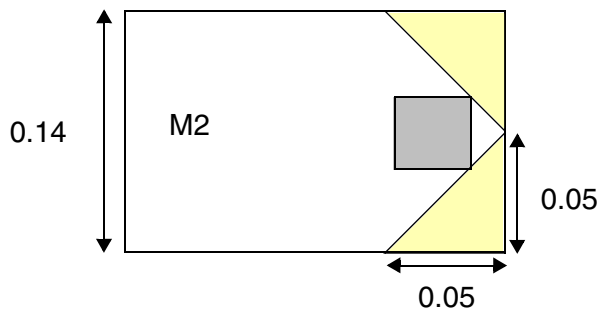
```
PROPERTY LEF58_ENCLOSUREEDGE
    "ENCLOSUREEDGE ABOVE 0.02 OPPOSITE ; " ;
PROPERTY LEF58_SPACING
    "SPACING 0.03 LAYER M1 CUTCLASS VA
        EXTENSION 0.01 ; " ;
```



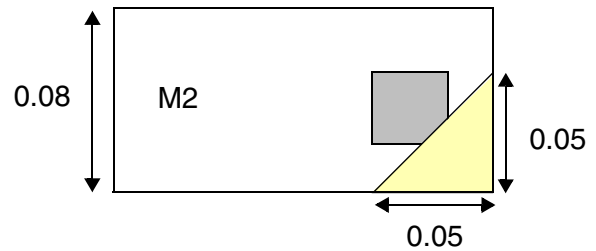
a) Violation, apply extension of 0.01 on the non-blue edges, and find M1 metal within 0.03 search window

**Figure 1-69 Illustration of CUTCLASS with NONEOLCONVEXCORNER**

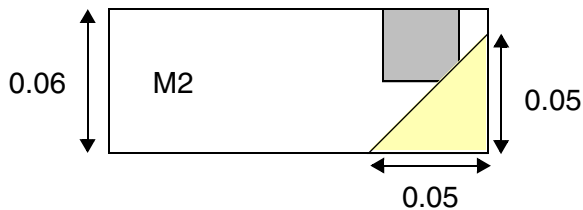
```
PROPERTY LEF58_SPACING
  "SPACING 0.05 LAYER M2 CUTCLASS VA
    NONEOLCONVEXCORNER 0.07 ; " ;
```



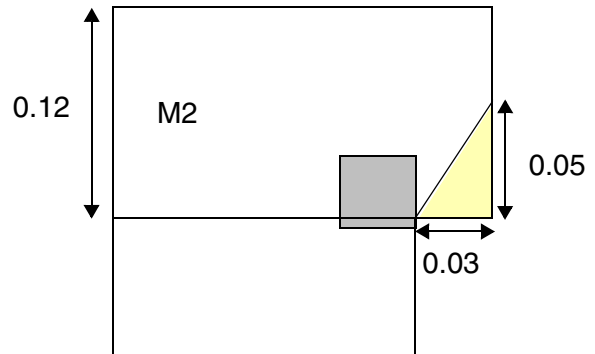
a) OK, the cut is outside of the 0.05 triangle



b) Violation, the cut overlaps with the triangle keepout.



c) OK, the edge is a EOL.

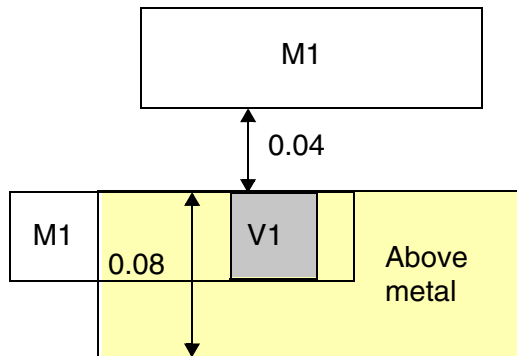


d) OK, the bottom edge of the triangle becomes edge length when it is < 0.05, and touching the keepout is fine

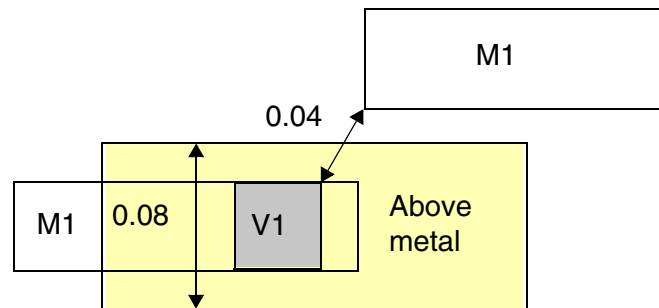
**Figure 1-70 Illustration of CUTCLASS with ABOVEWIDTH and ENCLOSURE**

On layer V1,

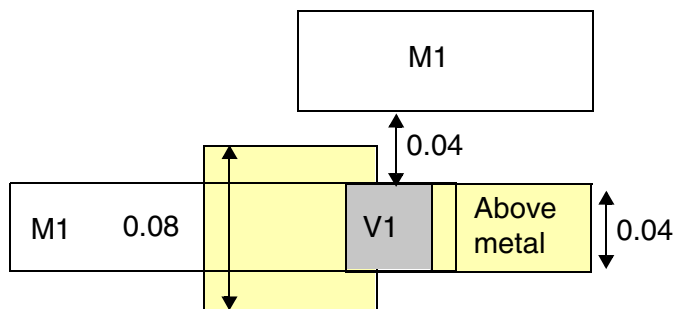
```
PROPERTY LEF58_SPACING
  "SPACING 0.05 LAYER M1 CUTCLASS VA
    ABOVEWIDTH 0.07 ENCLOSURE 0.01 ; "
```



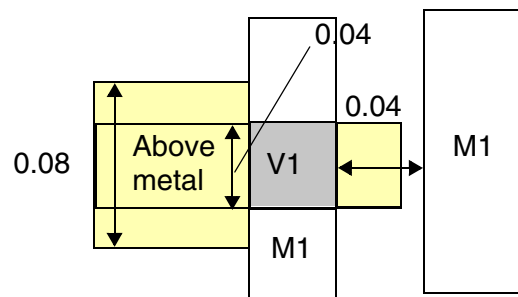
a) Violation, cut to metal spacing is 0.04 ( $\leq 0.05$ ).



b) Violation, cut to metal spacing is measured in Euclidean style.

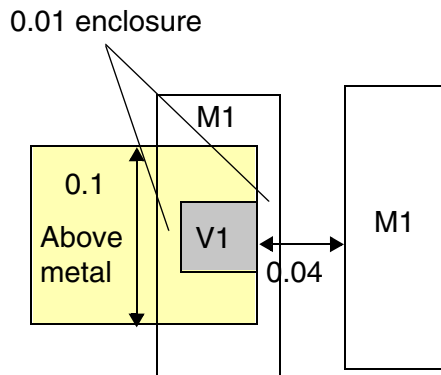


c) Violation, as long as part of the cut is inside a wide wire, the entire cut edge is subjected for the spacing check.



d) Violation, the left edge of the cut touches a wire with width of 0.08 ( $\geq 0.07$ ).

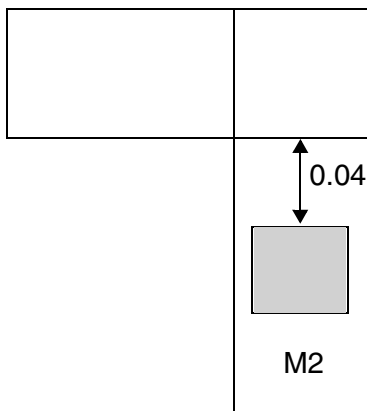




e) OK, both set of opposite cut sides  
have enclosure  $\geq 0.01$

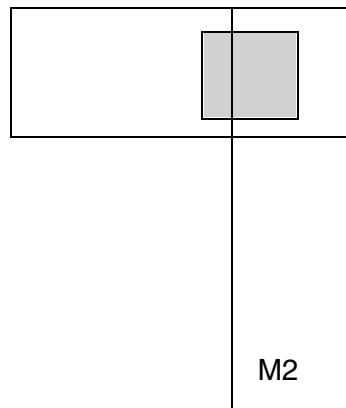
**Figure 1-71 Illustration of CUTCLASS with WRONGDIRECTION**

```
PROPERTY LEF58_SPACING
  "SPACING 0.05 LAYER M2 CUTCLASS VA
    WRONGDIRECTION ; " ;
```



a) Violation, the cut needs 0.05 spacing  
to non-preferred wires.

Preferred  
direction of  
M2 is vertical

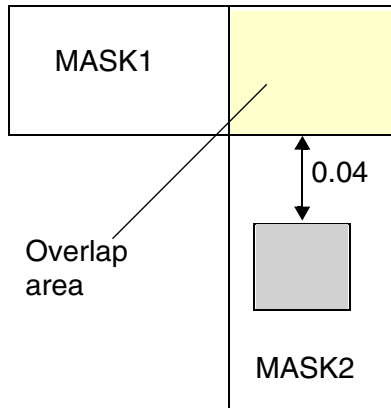


b) Violation, the cut could not be  
inside or touch non-preferred wires.

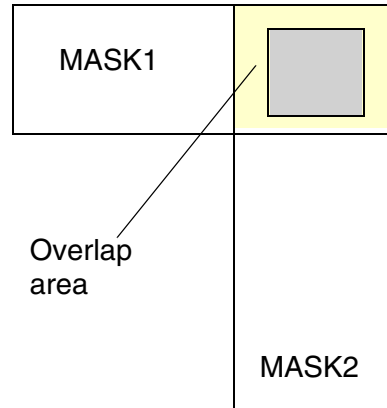
**Figure 1-72 Illustration of CUTCLASS with MASKOVERLAP**

On layer V1,

```
PROPERTY LEF58_SPACING  
  "SPACING 0.05 LAYER M2 CUTCLASS VA  
    MASKOVERLAP ; " ;
```



a) Violation, the cut needs 0.05 spacing to the overlap area of the 2 masks.



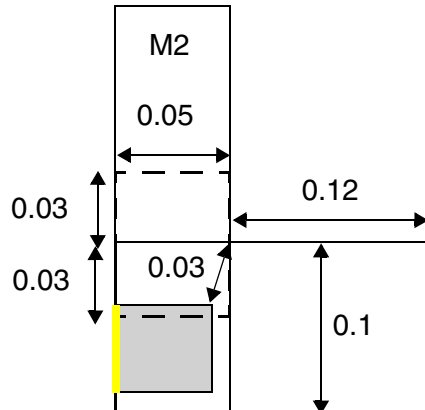
b) Violation, the cut cannot be inside the overlap area.

# LEF/DEF 5.8 Language Reference

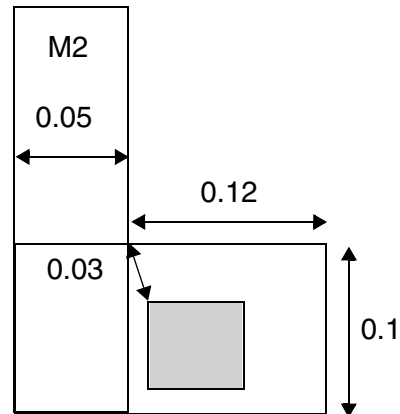
## LEF Syntax

**Figure 1-73 Illustration of CONCAVECORNER with WIDTH**

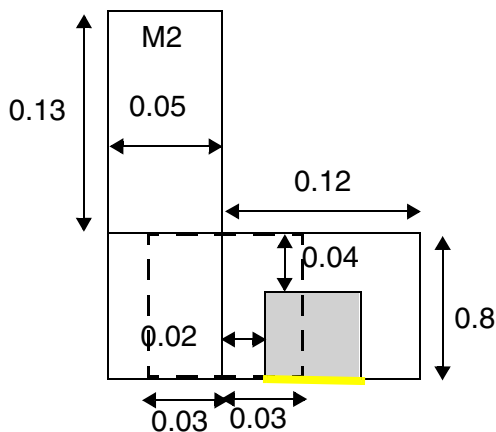
```
PROPERTY LEF58_SPACING
  "SPACING 0.03 LAYER M2 CUTCLASS VA
    CONCAVECORNER
      WIDTH 0.08 ENCLOSURE 0.02 EDGELENGTH 0.05 ; " ;
```



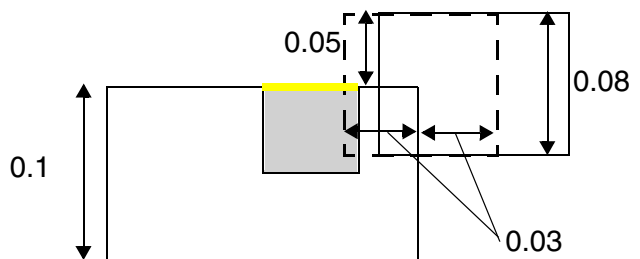
a) Violation, the vertical wire fulfills the width condition, the search window finds a cut, and the yellow cut edge does not have enclosure of 0.02.



b) OK, the cut is on the other side of the edge fulfilling the width condition.



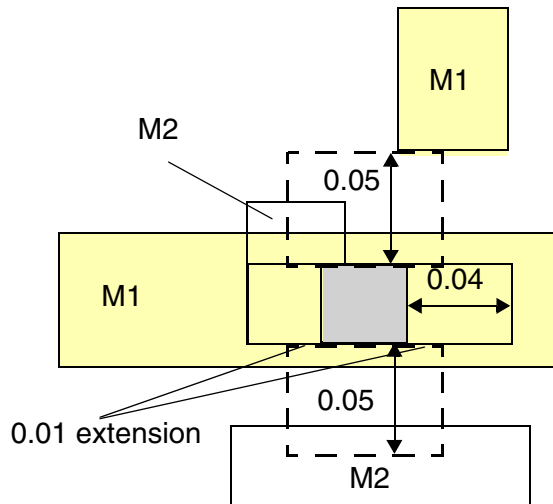
c) Violation, both the wires fulfill the width and length conditions, and the yellow cut edge does not have enclosure of 0.02.



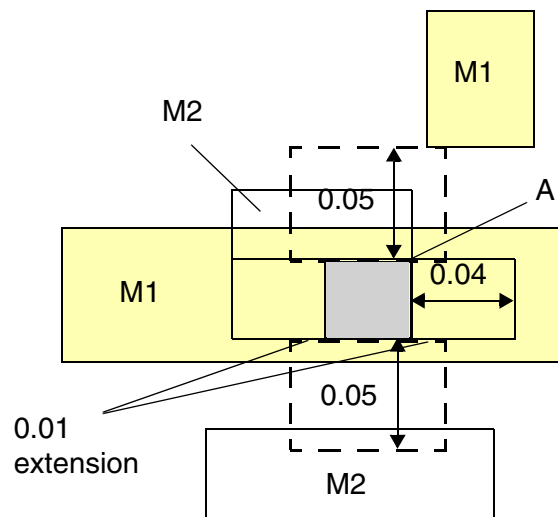
d) Violation, the search will be extended independent of whether there is a jog or not, and the yellow cut edge does not have enclosure of 0.02.

**Figure 1-74 Illustration of CONCAVECORNER with WITHIN in PARALLEL**

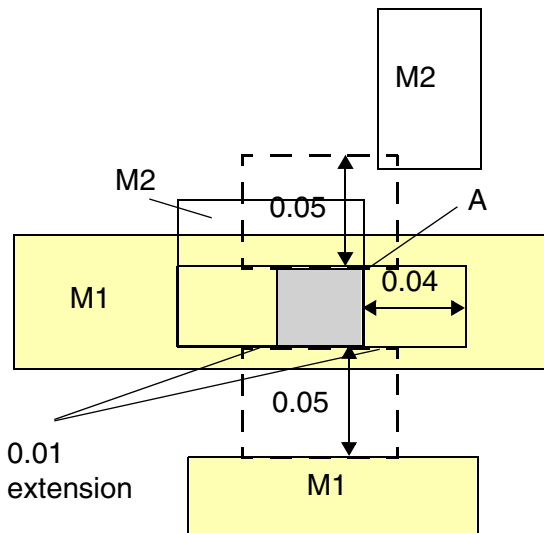
```
PROPERTY LEF58_SPACING
  "SPACING 0.04 LAYER M2 CUTCLASS VA
    CONCAVECORNER
      PARALLEL -0.01 WITHIN 0.051 ENCLOSURE 0.06 ; " ;
```



a) Violation, as long as part of two opposite cut edges having zero enclosure, it is still a violation.



b) Violation, cut corner A having a neighbor on M1 collides with the concave corner is still a violation



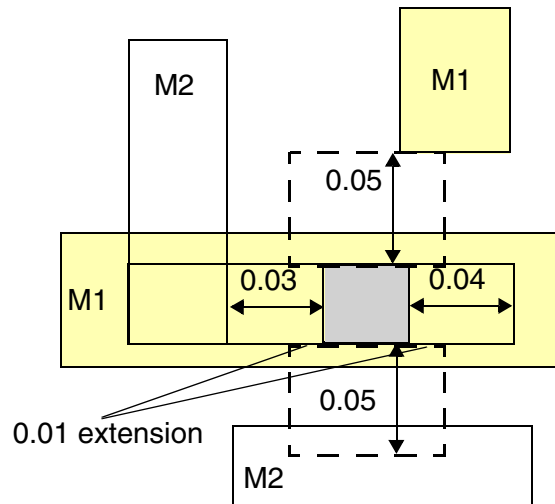
c) OK, cut corner A having a neighbor on M2 collides with the concave corner would not trigger the rule

# LEF/DEF 5.8 Language Reference

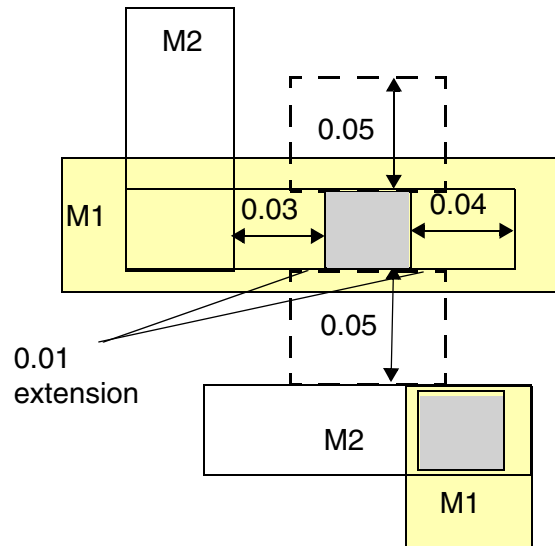
## LEF Syntax

**Figure 1-75 Illustration of CONCAVECORNER with WITHIN in PARALLEL**

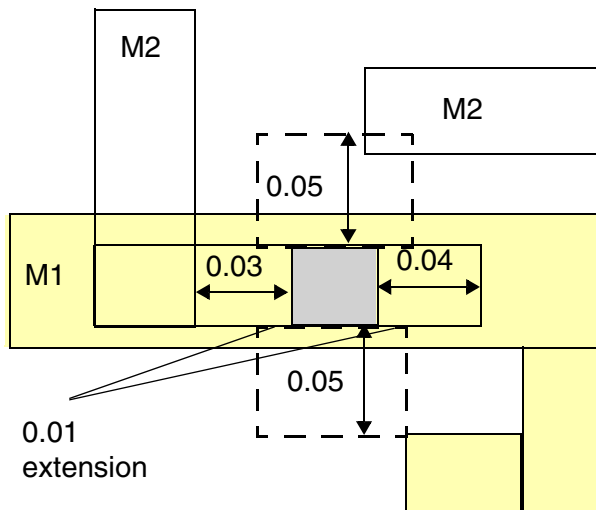
```
PROPERTY LEF58_SPACING
  "SPACING 0.04 LAYER M2 CUTCLASS VA
    CONCAVECORNER
      PARALLEL -0.01 WITHIN 0.051 ENCLOSURE 0.06 ; "
```



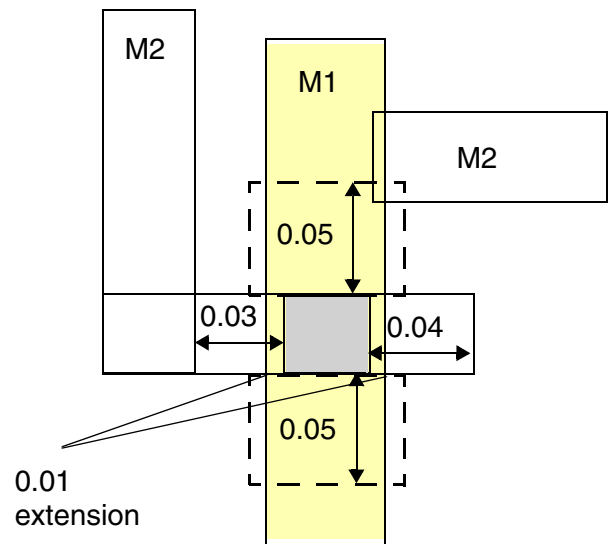
a) Violation, all of the conditions are met, particularly the M1 neighbor is exactly 0.05 apart.



b) OK, neighbor wires are not on opposite side



c) Violation, the same-metal/same-net wire could be a valid neighbor.

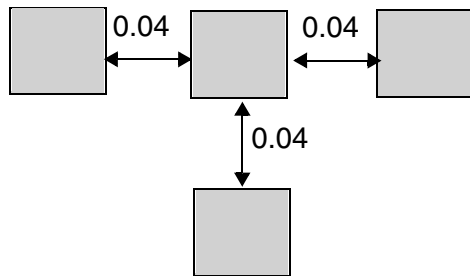


d) OK, the wire directly connected to the cut is not a valid neighbor

**Figure 1-76 Illustration of ADJACENTCUTS with EXCEPTALLWITHIN**

Example of

```
PROPERTY LEF58_SPACING "  
    SPACING 0.06 ADJACENTCUTS 2 WITHIN 0.07  
    EXCEPTALLWITHIN 0.05 ; "
```

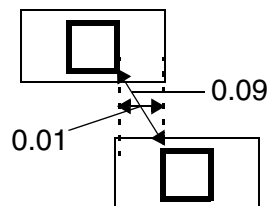


a) OK, it does not matter how many neighbors the middle cut has; as long as they are all within 0.05, it is allowed

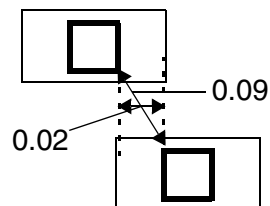


b) Violation, the middle cut has two neighbor within 0.07, and the left cut has spacing of 0.04 (< 0.06)

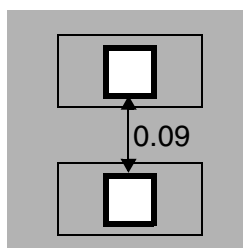
**Figure 1-77 Illustration of PARALLELWITHIN Rules**



a) Violation, a neighbor cut edge is within 0.02 of a cut edge, 0.1 spacing is required among them.



b) Okay, the neighbor cut is not within 0.02 beyond the other cut edge.



c) Violation, the 0.1 parallel within spacing applies to same-metal as well

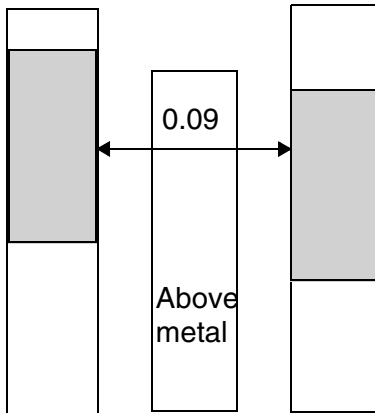
## LEF/DEF 5.8 Language Reference

### LEF Syntax

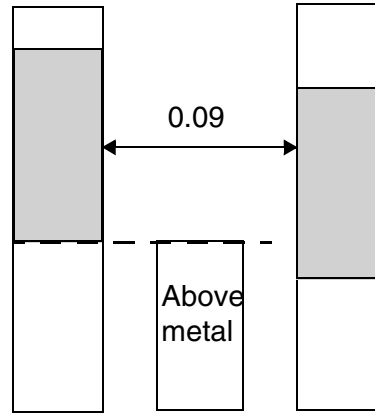
---

**Figure 1-78 Illustration of PARALLELWITHIN with LONGEDGEONLY**

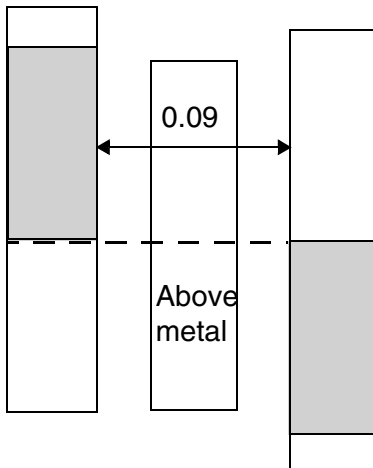
```
PROPERTY LEF58_SPACING  
  "SPACING 0.1 PARALLELWITHIN 0.0 CUTCLASS VB  
    LONGEDGEONLY ; " ;
```



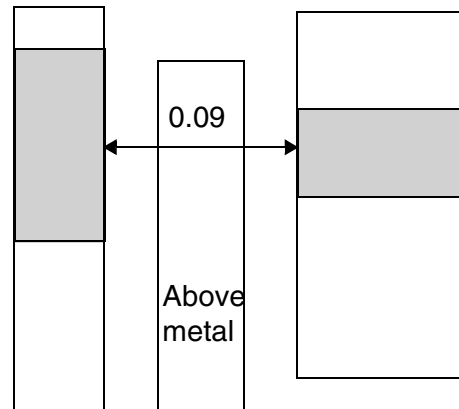
a) Violation, the long/side cut edges has a middle wire with spacing 0.09 ( $< 0.1$ )



b) OK, no middle neighbor wire in the common projected run length of the cuts



c) OK, no common projected run length between the cuts



d) OK, the spacing only applies to two long/side edges. Violation if LONGEDGEONLY is omitted.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

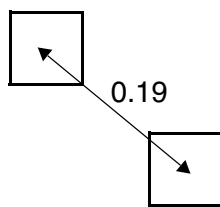
- The following spacing rule indicates that cuts that share the same metal shapes on both above and below metal layers should be 0.12  $\mu\text{m}$  apart:

```
PROPERTY LEF58_SPACING  
    "SPACING 0.12 SAMEVIA ;" ;
```

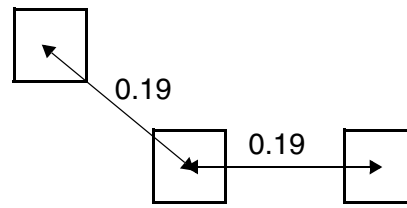
- The following spacing rule indicates that the center-to-center spacing between cuts must be at least 0.2 if a cut has at least three exactly aligned neighbor cuts, horizontally and/or vertically, less than or equal to 0.2  $\mu\text{m}$  away, or a cut has at least two neighbor cuts less than 0.2  $\mu\text{m}$  away and one of those neighbor cuts is not exactly aligned:

```
PROPERTY LEF58_SPACING  
    "SPACING 0.2 CENTERTOCENTER ADJACENTCUTS 2 EXACTALIGNED 3 WITHIN 0.2 ;" ;
```

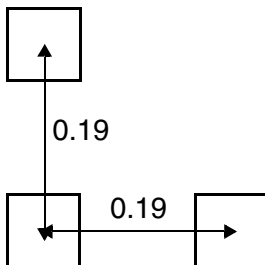
**Figure 1-79 Illustration of the ADJACENTCUTS Rule with EXACTALIGNED**



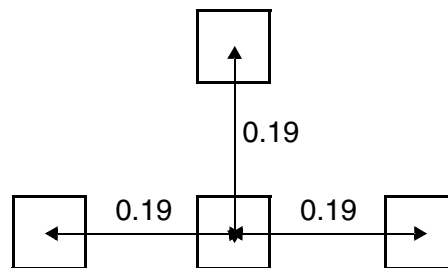
a) Okay. Only 1 neighbor cut within 0.2



b) Violation. When there is 1 non-perfectly aligned neighbor cut (on the left), any other neighbor cut, perfectly aligned or not (on the right) would trigger the rule spacing of 0.2



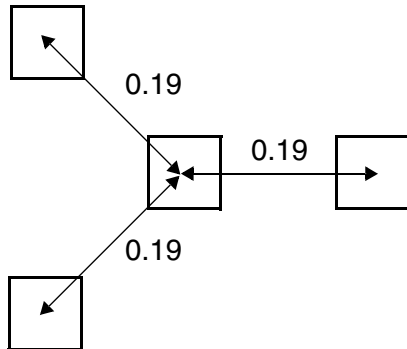
c) Okay. Two perfectly aligned neighbor cuts in 'L' or a straight line are fine



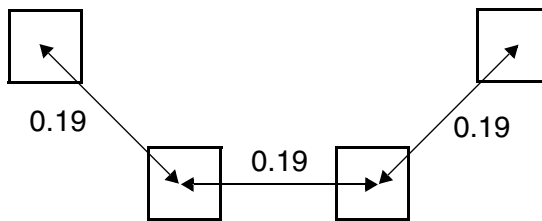
d) Violation. Three perfectly aligned neighbor cuts would trigger the rule spacing of 0.2

**Figure 1-80 Illustration of the ADJACENTCUTS Rule with TWOCUTS**

```
PROPERTY LEF58_SPACING
  "SPACING 0.2 CENTERTOCENTER ADJACENTCUTS 3 ;
    TWOCUTS 2 WITHIN 0.2 ;" ;
```



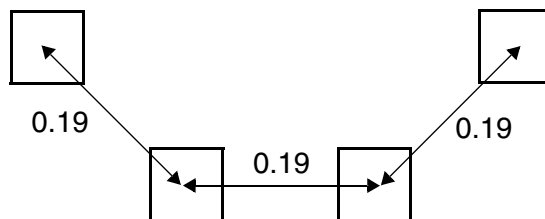
a) Violation, the middle cut has 3 neighbor cuts within 0.2



b) Violation, each of the middle 2 cuts have 2 neighbor cuts within 0.2, and the spacing between them is < 0.2. OK if TWOCUTS is not used.

**Figure 1-81 Illustration of the ADJACENTCUTS Rule with TWOCUTSSPACING**

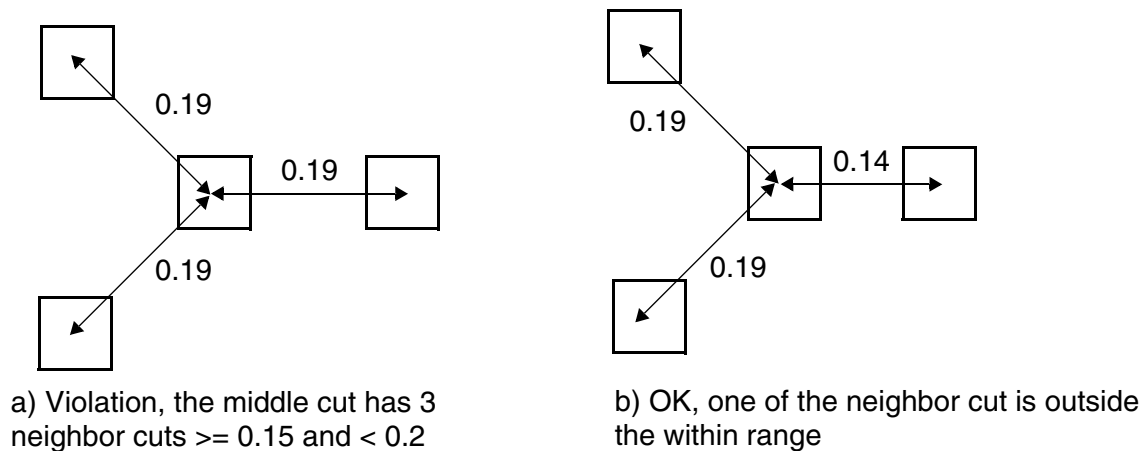
```
PROPERTY LEF58_SPACING
  "SPACING 0.2 CENTERTOCENTER ADJACENTCUTS 3
    TWOCUTS 2 TWOCUTSSPACING 0.18 WITHIN 0.2 ;" ;
```



a) OK, the middle two cuts fulfilling the TWOCUTS condition do not have a spacing < 0.18. Violation if TWOCUTSSPACING is omitted.

**Figure 1-82 Illustration of the ADJACENCUTS Rule with WITHIN**

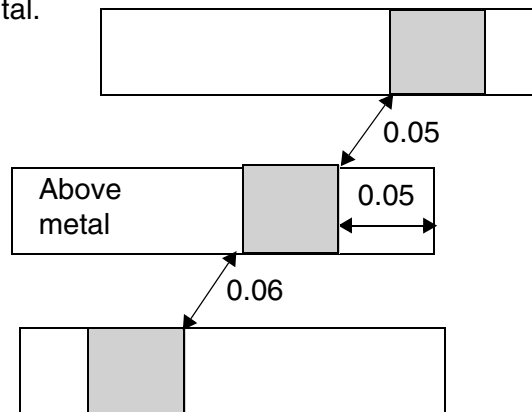
```
PROPERTY LEF58_SPACING
  "SPACING 0.2 CENTERTOCENTER ADJACENCUTS 3 ;
    WITHIN 0.15 0.2 ;"
```



**Figure 1-83 Illustration of the ADJACENCUTS Rule with ENCLOSURE**

```
PROPERTY LEF58_SPACING
  "SPACING 0.07 ADJACENCUTS 2 WITHIN 0.07
    ENCLOSURE ABOVE 0.05 ;"
```

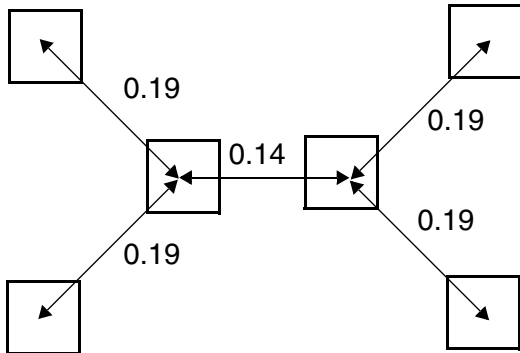
Preferred routing direction on above metal is horizontal.



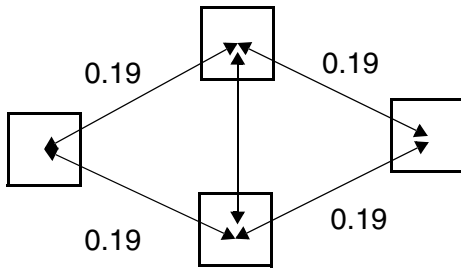
a) OK, the middle has enclosure of 0.05 in the preferred routing direction on the above metal layer. It fails if ENCLOSURE is omitted.

**Figure 1-84 Illustration of ADJACENTCUTS Rule with SAMECUT**

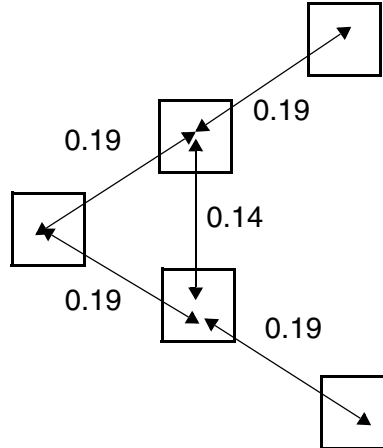
```
PROPERTY LEF58_SPACING
  "SPACING 0.2 CENTERTOCENTER ADJACENTCUTS 4
    TWOCUTS 3 SAMECUT WITHIN 0.2 ;" ;
```



a) OK, the middle two cuts do not have common neighbors. Violation if SAMECUT is omitted



b) Violation, the middle cuts have common neighbor besides neighboring to each other



c) OK, all of the neighbors must be common.

## LEF/DEF 5.8 Language Reference

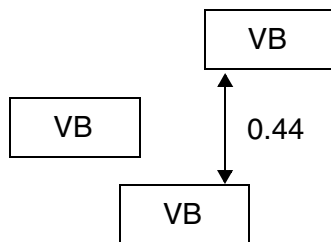
### LEF Syntax

- The following spacing rule indicates that a VB via can at most have one neighbor VB via within 0.45  $\mu\text{m}$  distance if they have a side to side parallel edge overlap greater than 0:

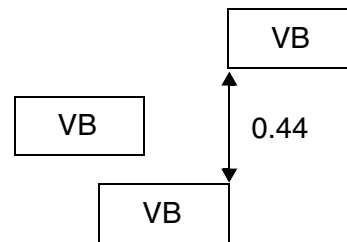
```
PROPERTY LEF58_SPACING
```

```
"SPACING 0.45 ADJACENTCUTS 2 WITHIN 0.45 CUTCLASS VB SIDEPARALLELOVERLAP ;" ;
```

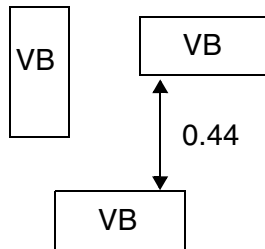
**Figure 1-85 Illustration of ADJACENTCUTS Rule with SIDEPARALLELOVERLAP**



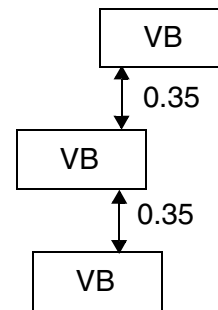
a) Violation, 2 VB < 0.2 with side by side parallel edge overlap > 0



b) OK, the right VB does not have side by side parallel edge overlap > 0



c) OK, the left VB only has a side to end parallel edge overlap > 0

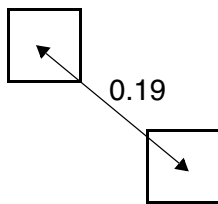


d) Violation, the side by side parallel edge overlap > 0 does not need to be on the same edge

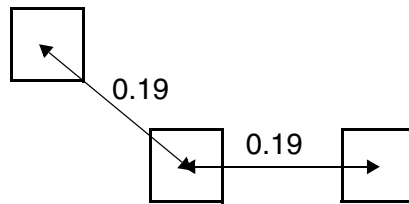
- The following example shows spacing rule with NOPRL:

**Figure 1-86 Illustration of ADJACENTCUTS Rule with NOPRL**

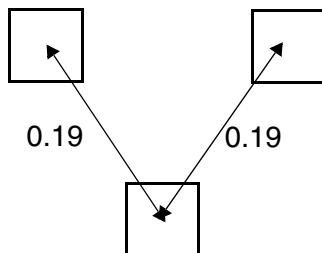
```
PROPERTY LEF58_SPACING
  "SPACING 0.2 CENTERTOCENTER ;
    ADJACENTCUTS 2 NOPRL ;"
```



a) Okay. Only 1 neighbor cut within 0.2



b) OK, the right cut has parallel run length > 0 with the middle cut, and is not counted as a neighbor



c) Violation, has 2 neighbor cuts without parallel run length

## LEF/DEF 5.8 Language Reference

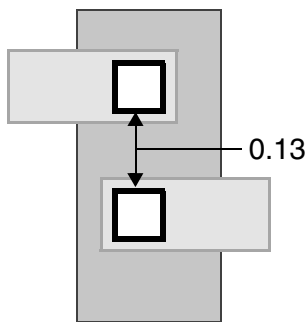
### LEF Syntax

---

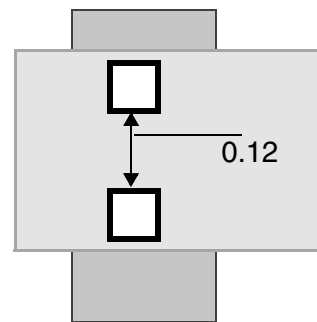
- The following spacing rule indicates that cuts that do not share the same metal shapes both above and below metal layers and have a parallel edge overlap greater than 0 must have 0.15  $\mu\text{m}$  distance between them:

```
PROPERTY LEF58_SPACING
  "SPACING 0.12 SAMEVIA ;" ;
  "SPACING 0.13 ;" ;
  "SPACING 0.15 PARALLELOVERLAP EXCEPTSAMEVIA ;" ;
```

**Figure 1-87 Illustration of PARALLELOVERLAP Rule with EXCEPTSAMEVIA**



a) Violation, the cuts only share metal shape on one metal layer, and 0.15 parallel overlap spacing is needed. Okay, if EXCEPTSAMEVIA is not used.



b) OK, the cuts share metal shapes on both metal layers, and 0.12 SAMEVIA spacing is needed and met

## LEF/DEF 5.8 Language Reference

### LEF Syntax

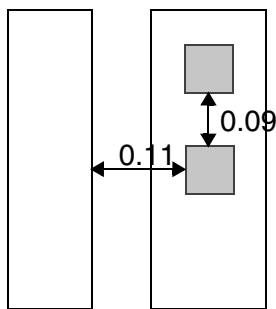
---

- The following spacing rule indicates that two via cuts having common parallel run length greater than 0, having common above and /or below metal shapes covering the entire length of common projection between them and having one neighbor wire within 0.12  $\mu\text{m}$  distance of them on the same edge must be, at least 0.1  $\mu\text{m}$  spacing apart:

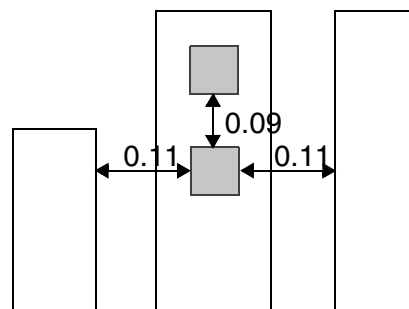
```
PROPERTY LEF58_SPACING
```

```
"SPACING 0.1 SAMEMETALSHAREDEDGE 0.12 EXCEPTTWOEDGES ;" ;
```

**Figure 1-88 Illustration of SAMEMETALSHAREDEDGE rule**



a) Violation, there is a neighbor within 0.12, and 0.1 spacing needed between the cuts

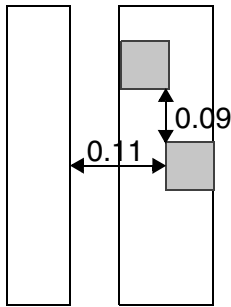


b) Okay, at least 1 cut having 2 neighbors within 0.12 on opposite sides will exempt the rule

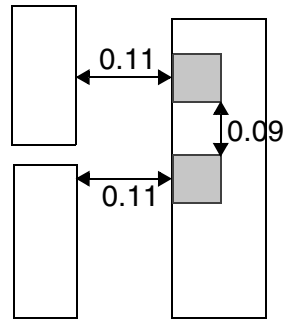


**Figure 1-89 Illustration of SAMEMETALSHAREDEDGE Rule**

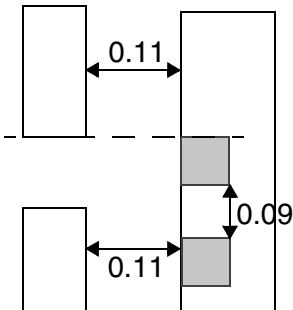
PROPERTY LEF58\_SPACING "SPACING 0.1 SAMEMETALSHAREDEDGE 0.12 ;" ;



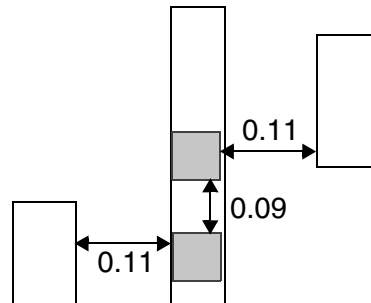
a) Okay, the cuts do not have common parallel run length greater than 0



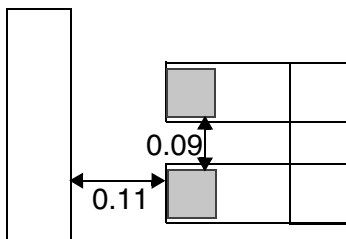
b) Violation, the neighbors could be different wires



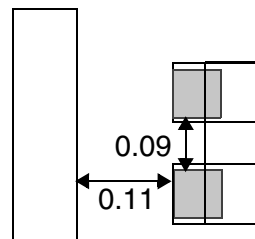
c) Okay, the top cut does not have a neighbor



d) Violation, parallel opposite neighbor edges, one neighbor per cuts, triggers the rule

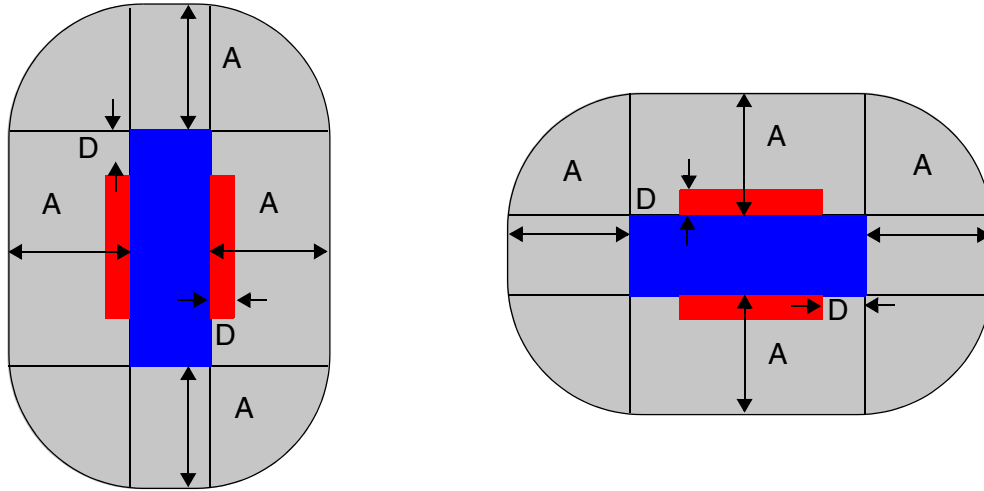


e) Okay, the cuts do not have common metal shapes



f) Violation, the common metal shapes covering the entire length of the projection between the cuts

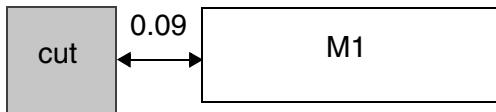
**Figure 1-90 Definition of Spacing Rule with ORTHOGONALSPACING**



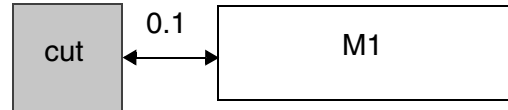
Start with a square red cut, shrink by  $D$ ,  $\text{abs}(\text{cutSpacing} - \text{orthogonalSpacing})/2$  on two sides of the cut and grow by  $D$  on the other two sides to result the blue rectangle. Then, spacing of  $A$ ,  $(\text{cutSpacing} + \text{orthogonalSpacing}) / 2$ , is applied in a Euclidean fashion. Violation, when wires on layer *secondLayerName* are found in both the gray regions.

**Figure 1-91 Illustration of Spacing Rule with ORTHOGONALSPACING**

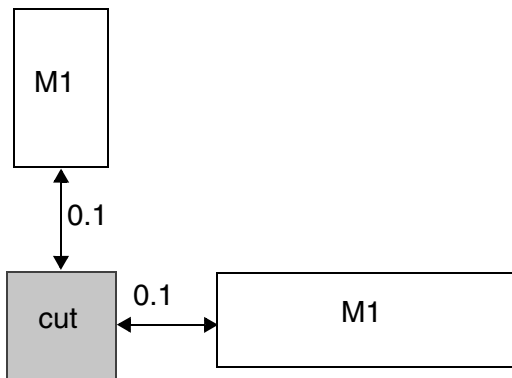
```
PROPERTY LEF58_SPACING  
"SPACING 0.1 LAYER M1 ORTHOGONALSPACING 0.2 ;" ;
```



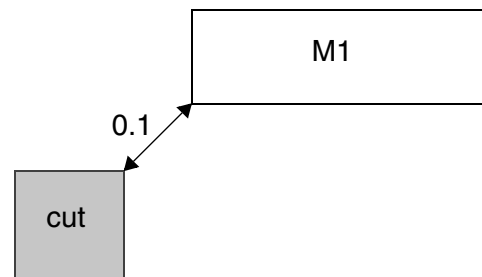
a) Violation since 0.09 is less than the smallest spacing of 0.1



b) OK, only one M1 is less than 0.2 away on one side, but greater than or equal to 0.1



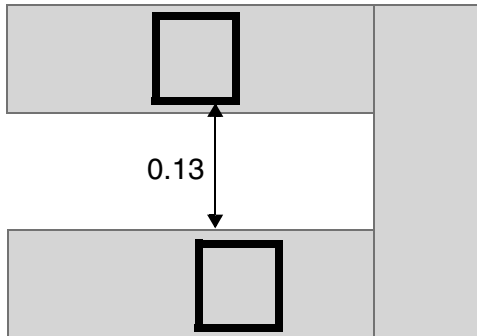
c) Violation, both the sides have wires less than 0.2 distance away



d) Violation, the M1 wire will be inside both the checking regions

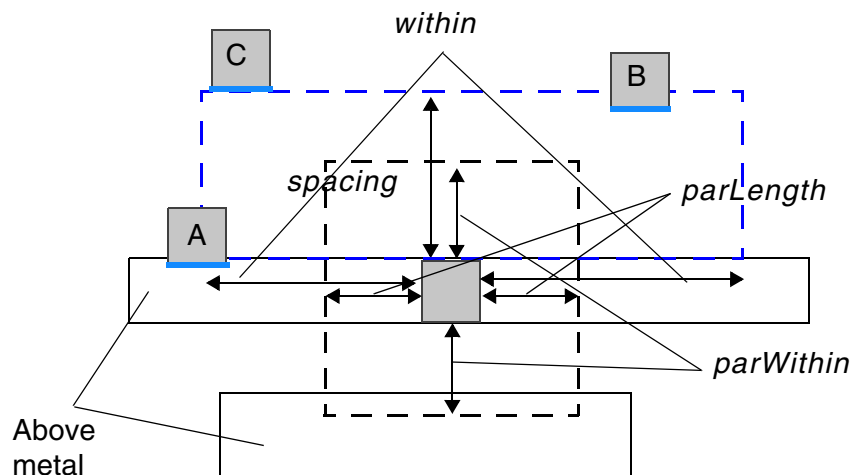
**Figure 1-92 Illustration of Spacing Rule with EXCEPTSAMEMETAL**

```
SPACING 0.13 ;
PROPERTY LEF58_SPACING
"SPACING 0.15 PARALLELOVERLAP EXCEPTSAMEMETAL ;" ;
```



a) OK, but with the default of EXCEPTSAMEMETALOVERLAP, it will be a violation since the same-metal must cover the projected overlap area of the cuts.

**Figure 1-93 Illustration of ENCLOSURE with ABOVE**



The bottom cut edge having a neighbor wire has enclosure less than *enclosure* on the above metal layer. Hence, the facing blue edge of any neighbor cuts must be outside of the blue dotted window. Hence, via A is ok since its blue edge is outside, but via B will be a violation. The via C is ok since the blue edge touches the window.

Illustration of SPACING *spacing* PARALLELWITHIN *within* CUTCLASS *cutClass*  
ENCLOSURE *enclosure* ABOVE PARALLEL -*parLength* WITHIN *parWithin*

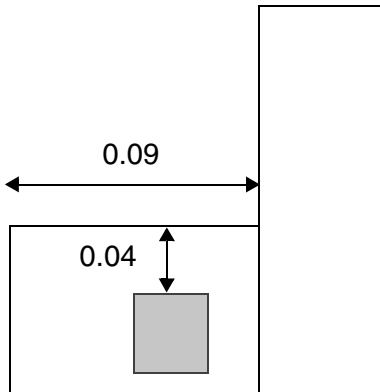
## LEF/DEF 5.8 Language Reference

### LEF Syntax

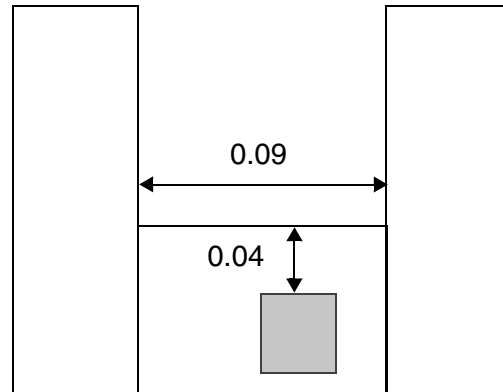
---

**Figure 1-94 Illustration of EDGELENGTH with ENCLOSURE**

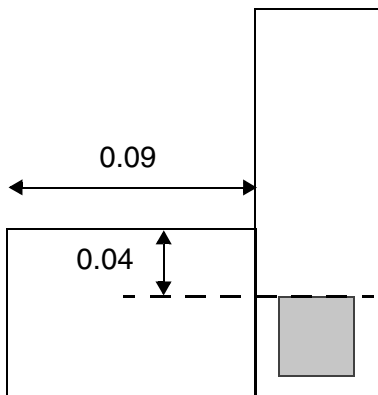
```
PROPERTY LEF58_SPACING "  
  SPACING 0.05 LAYER M2 CUTCLASS VA CONCAVECORNER  
  EDGELENGTH 0.1 ENCLOSURE 0.06 0.08 ; "
```



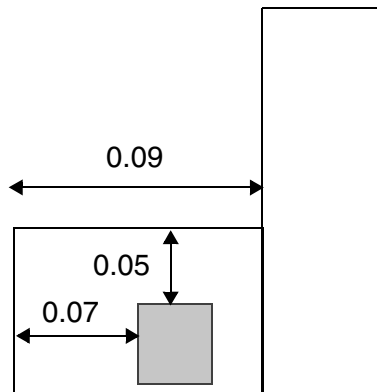
a) Violation, the spacing of the cut to an edge connected to a concave corner with length of 0.09 ( $< 0.1$ ) is 0.04 ( $< 0.05$ ).



b) OK, the 0.09 edge does not contain to a convex corner on the other end.



c) OK, the cut does not have PRL  $\geq 0$  to the 0.09 edge.



d) Violation, the enclosure of the top cut edge is 0.05 ( $\geq 0.05$ , but  $< 0.06$ ), the left adjacent cut edge has enclosure of 0.07 ( $< 0.08$ ).

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### Spacing Table Rule

The spacing table rule can be used to define cut spacing between different cut classes.

You can create a spacing table rule by using the following property definition:

```
[PROPERTY LEF58_SPACINGTABLE
  "SPACINGTABLE
    [ORTHOGONAL
      {WITHIN cutWithin SPACING orthoSpacing} ... ;
    | [DEFAULT defaultCutSpacing]
      [SAMEMASK]
      [SAMENET | SAMEMETAL | SAMEVIA]
      [LAYER secondLayerName
        [NOSTACK]
        [NONZEROENCLOSURE
          | PRLFORALIGNEDCUT
            {{className1 | ALL} TO {className2 | ALL} }...
          | EXCEPTENCLOSURE exceptEnclosure]]
      [CENTERTOCENTER
        {{className1 | ALL} | TO {className2 | ALL}}...]
      [CENTERANDEDGE [NOPRL]
        {{className1 | ALL} | TO {className2 | ALL}}...]
      [PRL prl [ HORIZONTAL| VERTICAL] [MAXXY]
        [{{className1 | ALL} TO {className2 | ALL} ccPrl
          [ HORIZONTAL| VERTICAL] }...] ]
      [PRLTWO SIDES
        {prl1 prl2 prl3 prl4 [WITHIN within]
          className1 TO className2 spacing}...]
      [ENDEXTENSION extension [{TO className classExtension}...]
      [SIDEEXTENSION {TO className classExtension}... ]
      [EXACTALIGNEDSPACING [HORIZONTAL | VERTICAL]
        {className exactAlignedSpacing}...]
      [NONOPPOSITEENCLOSURESPACING
        {className nonOppositeEnclosureSpacing}...]
      [OPPOSITEENCLOSURERESIZESPACING
        {className resize1 resize2
          oppositeEnclosureResizeSpacing}...]
      CUTCLASS { {className1 | ALL} [SIDE | END]}...
        {{className2 | ALL} [SIDE | END] {- | cutSpacing1}
          {- | cutSpacing2}...}...;
    ]
  ;..." ;
```

Where:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

All other keywords are the same as the existing LEF cut layer SPACINGTABLE syntax.

#### CENTERTOCENTER

```
{{className1 | ALL} TO {className2 | ALL}}...
```

Computes the *cutSpacing1* and *cutSpacing2* distance from cut-center to cut-center, instead of cut-edge to cut-edge (the default behavior), for the given list of class name pairs. The *className1* is one of the cut classes in the first row of the table and *className2* is one of the cut classes in the first column of the table. The `ALL` keyword applies to all the vias on a cut layer that has only one cut class without an explicit `CUTCLASS` definition. The keyword should be specified only if one of the layers does not have a cut class.

#### CENTERANDEDGE [NOPRL]

```
{{className1 | ALL} TO {className2 | ALL}}...
```

Indicates that center-to-center measurement applies to the larger of the *cutSpacing1* and *cutSpacing2*, which must be different values, while edge-to-edge measurement applies to the smaller of the two cut spacings.

The `NOPRL` keyword indicates that both center to center and edge to edge spacing must be met, where the required center to center spacing is the maximum of *cutSpacing1* and *cutSpacing2*, and the required edge to edge spacing is the minimum of *cutSpacing1* and *cutSpacing2*.

**Note:** The `CENTERTOCENTER` and `CENTERANDEDGE` keywords can be defined simultaneously, but must be on different cut classes.

```
{{className1 | ALL} TO {className2 | ALL} ccPr1  
[HORIZONTAL | VERTICAL]}...
```

Specifies a specific *ccPr1* value between the given cut classes. If `HORIZONTAL` or `VERTICAL` is specified, the common parallel run length is measured horizontally or vertically, respectively. Any cut class combinations that is not specified would still follow *pr1* defined in `PRL`.

*Type:* Float, specified in microns

```
CUTCLASS { {className1 | ALL} [SIDE | END|]}
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies a list of cut classes, which may not necessarily be a complete list of all cut classes in the cut layer. In this case, the cut spacing requirements are not needed for any missing cut classes. The `ALL` keyword applies to all vias on a cut layer which has only one cut class without an explicit `CUTCLASS` definition.

If an intercut layer is specified (using the `LAYER` keyword), the cut classes in *className1* are defined for the layer for which the spacing table is defined. When intercut layer spacing is needed between a cut layer with multiple cut classes and a cut layer without a cut class, this cut class spacing table should be specified, and the "`SPACING ... LAYER ... ;`" statement cannot be used.

If the cut class has a rectangular cut shape, the `SIDE` and `END` keywords can be used to specify cut spacing on a certain edge - side/long or end/short (see [Figure 1-95](#) on page 183). The diagram indicates the regions that the other cut via should be fully contained on certain edges to be applied.

```
{{className2 | ALL} [SIDE | END] {- | cutSpacing1} {- |  
cutSpacing2}
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates that *cutSpacing1* and *cutSpacing2* is applied between *className2* and *className1* in the first row of the table.

There are two sets of cut spacing values for each table entry. The first *cutSpacing1* value applies if there is no parallel edge overlap between the via cuts. The second *cutSpacing2* value applies if there is a parallel edge overlap greater than 0. If **CENTERTOCENTER** keyword is used for two cut classes, then *cutSpacing1* and *cutSpacing2* should be identical. If - is specified, the *defaultCutSpacing* value (specified with the **DEFAULT** keyword) is used.

If interlayer cut spacing is specified with the **LAYER** keyword, the cut class in *className2* is defined for the *secondLayerName* (specified with the **LAYER** keyword). If **LAYER** is not specified, then the table must be a NxN symmetrical table with spacing values in any (i, j) and (j, i) locations in the table being the same.

The **ALL**, **SIDE**, and **END** keywords are the same as described earlier.

*Type:* Float, specified in microns

**DEFAULT** *defaultCutSpacing*

Indicates the default cut spacing between cut classes.

*Type:* Float, specified in microns

**Note:** If a table entry contains -, the *defaultCutSpacing* value applies. In this case the **DEFAULT** keyword must be specified.

An undefined cut class will have the spacing of **DEFAULT** *defaultCutSpacing*, and the spacing values in any (i, j) and (j, i) location in the table are the same.

**ENDEXTENSION** *extension* [{**TO** *className* *classExtension*}...]

Defines extension values applied to the end/short edge of all rectangular cuts, before *cutSpacing1* and *cutSpacing2*, are measured to the cuts with extensions. If the **TO** keyword is specified, then *classExtension* is applied to the end/short edge of all rectangular cuts to measure the cut spacing to cuts in *className*. Otherwise, *extension* is applied to the end/short edge of all rectangular cuts to measure the cut spacing to cuts for any non-specified cut classes.

*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

EXACTALIGNEDSPACING [HORIZONTAL | VERTICAL]  
{*className exactAlignedSpacing*}

Specifies the spacing between two perfectly aligned, horizontally or vertically, *className* cuts must be greater than or equal to the *exactAlignedSpacing*. If HORIZONTAL or VERTICAL is also specified, the spacing only applies to the corresponding direction. The *className* must be a square cut class.

*Type:* Float, specified in microns

**Note:** The definition of EXACTALIGNEDSPACING remains edge-to-edge even when CENTERTOCENTER keyword is specified.

EXCEPTENCLOSURE *exceptEnclosure*

Specifies that the inter-layer cut spacing between any cuts in the current layer to any cuts in layer *secondLayerName* applies only for the cuts in the current layer with enclosure on the above metal layer on all four sides greater than *exceptEnclosure* and parallel run length greater than 0 with the cuts in layer *secondLayerName*. In other words, if the above metal layer of the cuts in the current layer is less than or equal to *exceptEnclosure* on any cut edges, the inter-layer cut spacing is not applied. In addition, *cutSpacing1* should be zero while *cutSpacing2* is the required cut spacing between cuts for the two specified cut classes.

*Type:* Float, specified in microns

LAYER *secondLayerName*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines the inter-cut-layer spacing between a *className1* via in the first row of the table on the layer that this spacing table is being defined to another *className2* via in the first column on *secondLayerName* cut layer. This cut spacing is ignored for same-net. This second layer must be a previously defined cut layer immediately below the layer that this spacing table is being defined, that is, “one layer look ahead” is not supported.

If an inter-cut-layer spacing table is defined for same-net cuts using the `SAMENET` keyword, the cuts on two different layers can always be stacked if they are exactly aligned (that is, the centers of the cuts are aligned) for same sized cuts. For different sized cuts, it is legal if the smaller cut is completely covered by the bigger cut. Otherwise, the cuts must have *cutSpacing* between them.

**Note:** When *secondLayerName* is not defined in a same-layer cut spacing table, then the cut classes in the columns must have the same order as the rows.

NONOPPOSITEENCLOSURESPACING

*{className nonOppositeEnclosureSpacing}...*

Specifies that the center-to-center spacing between a cut of *className* that does not fulfill the overhang defined in `ENCLOSUREEDGE OPPOSITE` to any cut of *className* is *nonOppositeEnclosureSpacing* independent of parallel run length.

*Type:* Float, specified in microns

NONZEROENCLOSURE

Specifies that the inter-layer cut spacing between cuts in the current layer to cuts in the specified *secondLayerName* only applies for cut edges with enclosure on the above metal layer of the cuts in the current layer greater than 0 and has parallel run length greater than 0 with the cuts in *secondLayerName* layer. In other words, the *cutSpacing1* should be 0, while *cutSpacing2* is the required cut spacing between cuts for the specified two cut classes.

NOSTACK

Specifies that stacked vias between any cuts in the current layer to any cuts in layer *secondLayerName* are not allowed. This construct must be specified with either `SAMENET` or `SAMEMETAL`.

OPPOSITEENCLOSURERESIZESPACING *{className resize1 resize2 oppositeEnclosureResizeSpacing}...*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the center-to-center spacing between two cuts of *className* that fulfill the overhang defined in

ENCLOSUREEDGE OPPOSITE is

*oppositeEnclosureResizeSpacing* independent of parallel run length if the corners of two windows barely touch by expanding the two opposite cut edges fulfilling the overhang defined in ENCLOSUREEDGE OPPOSITE by *resize1* and the other two opposite cut edges by *resize2*. See [Figure 1-97](#) on page 185.

*Type:* Float, specified in microns

PRLFORALIGNEDCUT {{*className1* | ALL} TO {*className2* | ALL} } ...

Specifies the second *cutSpacing2* value only applies to cut edges of *className2* on *secondLayerName* that fulfill the overhang defined in ENCLOSUREEDGE OPPOSITE to the cut of *className1* when parallel edge overlap is greater than 0 or *prl*, if specified. In other words, even when parallel edge overlap condition is satisfied, the *cutSpacing1* value could be used instead if the cut edges do not entirely align to the above metal. The spacing is measured from the projection on the metal edge, and negative *prl*, if specified, is treated as an extension of the metal edge.

PRL *prl* [ HORIZONTAL | VERTICAL ] [MAXXY]

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines the condition when to apply *cutSpacing1* and *cutSpacing2* between cuts. When cuts have a common parallel run length greater than *prl*, *cutSpacing2* is used between the cuts. Otherwise, *cutSpacing1* is used. If *prl* is negative, it indicates that a neighbor cut within  $\text{abs}(\text{prl})$  distance beyond the edge of a cut will need to be *cutSpacing2* distance away from it. This is a global definition that applies to cuts in all cut classes.

*Type:* Float, specified in microns

HORIZONTAL specifies that *cutSpacing2* is used only when the common parallel run length measured horizontally is greater than *prl*. Similarly, VERTICAL specifies that *cutSpacing2* is used only when the common parallel run length measured vertically is greater than *prl*. Otherwise, *cutSpacing1* is used.

The MAXXY keyword specifies that the spacing is measured as the maximum projection style when common parallel run length is larger than the given *prl*, particularly for a negative *prl* value. Otherwise, the spacing is measured from edge-to-edge.

```
PRLTWO SIDES {prl1 prl2 prl3 prl4 [WITHIN within] className1 TO  
className2 spacing}...
```

Specifies that the center-to-center spacing between cuts of *className1* and *className2* that are greater than or equal to *prl1* and less than or equal to *prl2* on one side and greater than or equal to *prl3* and less than or equal to *prl4* on the other side to be *spacing*. If WITHIN is specified, the cuts must be within *within*, and *spacing* is measured as corner to corner. If the cuts do not fulfill these PRL and WITHIN conditions, it would follow the specified cut class spacing as usual.

*Type:* Float, specified in microns

SAMEMASK

Specifies that the cut spacing applies between cuts on the same mask. When a cut layer has more than one mask, you can specify up to two cut class SPACINGTABLE rules, one with SAMENET or SAMEMETAL, and one with neither of them for the same cut layer spacing for same-mask cuts (with SAMEMASK on all of those tables). Then you can specify (without the SAMEMASK keyword) up to another two tables for different-mask cuts.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

SAMENET	Indicates that <i>cutSpacing1</i> and <i>cutSpacing2</i> values only apply to same-net cuts. The SAMENET cut spacing values should be smaller than the normal SPACINGTABLE <i>cutSpacing</i> values that apply to different-net cuts.
SAMEMETAL	Indicates that <i>cutSpacing1</i> and <i>cutSpacing2</i> values only apply to cuts that are overlapped with the same metal shape. The SAMEMETAL cut spacing values should be smaller than the normal SPACINGTABLE <i>cutSpacing</i> values that apply to different-metal cuts.
SAMEVIA	Indicates that <i>cutSpacing1</i> and <i>cutSpacing2</i> values only apply to cuts that are overlapped with the same metal shape on both the above and below metal. The SAMEVIA cut spacing values should be smaller than the normal SPACINGTABLE <i>cutSpacing</i> values that apply to different-metal cuts.
SIDEEXTENSION {TO <i>className</i> <i>classExtension</i> }	<p>Defines extension value, <i>classExtension</i>, applied to the side/long edge of all rectangular cuts before <i>cutSpacing1</i> and <i>cutSpacing2</i> are measured to the cuts in <i>className</i> with extensions. When side extension is applied between certain cut classes, the end extension is not applied. Hence, you cannot apply the same <i>className</i> after the TO statement.</p> <p><i>Type:</i> Float, specified in microns</p>

You can specify up to two cut class SPACINGTABLE rules, one with the SAMENET or SAMEMETAL rule, and one with neither of them for the same cut layer spacing. You can specify up to another three tables for intercut layer spacing - one with SAMENET, one with SAMEMETAL, and one with neither of them. You cannot mix with any other cut layer spacing statements, except for ADJACENTCUTS, PARALLELWITHIN, and SAMEMETALSHAREDEDGE.

Multiple spacing among different cut classes is possible. For cut layer shapes in PIN or OBS statement that belong to a macro of class CORE, the cut size of the via should match one of the sizes of the cut classes so that the tools can determine the proper spacing for the cut. If not, specific spacing should be defined for the cut in the macro definition using the SPACING keyword that is part of the layer geometry specification in MACRO. If the SPACING keyword is not specified, minimum spacing in the cut layer is used. If the cut size of the abstracted cuts does not match one of the cut class sizes, a single spacing value is applied to all four sides of the cut. This may cause DRC violations.

As with any OBS shapes, a cut layer OBS shape is always considered to belong to a net that is different from any pin, even if it is overlapping with a pin geometry on the adjacent metal

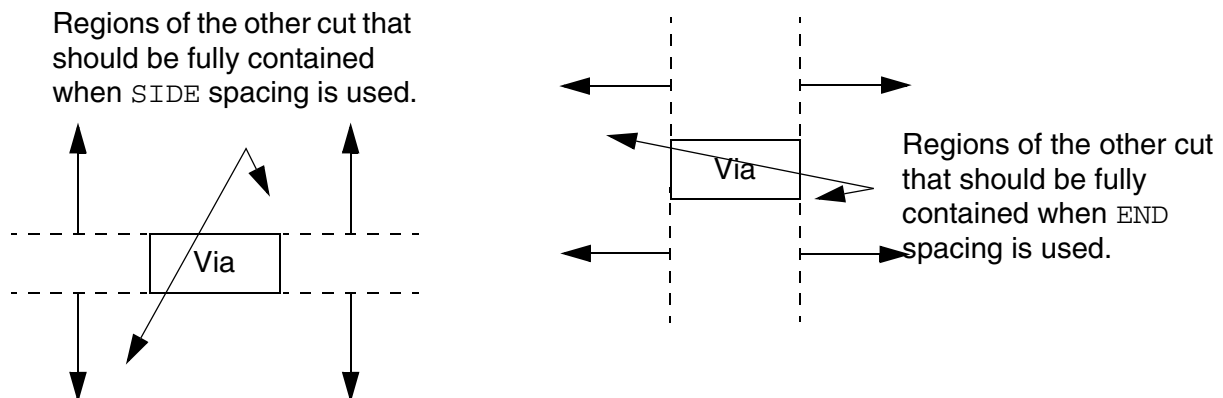
layer. In this case, different-net cut-to-cut spacing is used to compute the cut-to-cut distance between the OBS and any cut that is connected to the corresponding pin.

**Note:** It is recommended not to define a large cut layer OBS shape abstracting cut shapes, even in a macro for a non-standard cell. If defined, minimum cut spacing is applied to prevent blocking via access of nearby pins. This may, however, cause DRC violations. The cut layer blockage shapes (defined using the `BLOCKAGES` keyword) will use minimum cut spacing around them, similar to OBS.

### Spacing Table Rule Examples

- The following illustration shows the regions that the cut via should be overlapped with when `SIDE` or `END` keywords are used.

**Figure 1-95 Illustration of Spacing Table Rule With Side and End**



- The following spacing table rules specify the spacing requirements between `VA` via and end edge of `VB` via, and end edge of `VB` via and `VA` via:

Cut-to-cut spacing between two center-to-center <code>VA</code> vias	0.20 $\mu\text{m}$
Cut-to-cut spacing between two center-to-center <code>VC</code> vias	0.50 $\mu\text{m}$
Edge-to-edge spacing between the end edge of <code>VB</code> and <code>VA</code> or <code>VC</code> via with parallel edge overlap greater than 0	0.30 $\mu\text{m}$
Edge-to-edge spacing between the end edge of <code>VB</code> via and an edge of another <code>VB</code> via with a parallel edge overlap greater than 0	0.40 $\mu\text{m}$
Edge-to-edge spacing for the rest of the combinations	0.15 $\mu\text{m}$

The rules translate into the following `SPACINGTABLE` property definition:

```
PROPERTY LEF58_SPACINGTABLE
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```

DEFAULT 0.15 CENTERTOCENTER VA TO VA VC TO VC
CUTCLASS    VA          VB SIDE      VB END      VC
VA           0.20  0.20  -            -  0.30  -      -
VB SIDE      -      -      -      -      -  0.40  -      -
VB END      -      0.30  -      0.40  -  0.40  -      0.30
VC           -      -      -      -      -  0.30  0.50  0.50

```

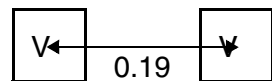
- The following spacing table rule indicates that center-to-center spacing between two square cut VA vias must be greater than or equal to 0.20  $\mu\text{m}$ :

```

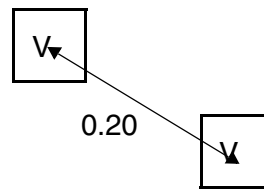
PROPERTY LEF58_SPACINGTABLE CENTERTOCENTER VA TO VA
CUTCLASS    VA
VA           0.20  0.20

```

**Figure 1-96 Illustration of Spacing Table Rule With CenterToCenter**



a) Violation. Center-to-center spacing < 0.20.

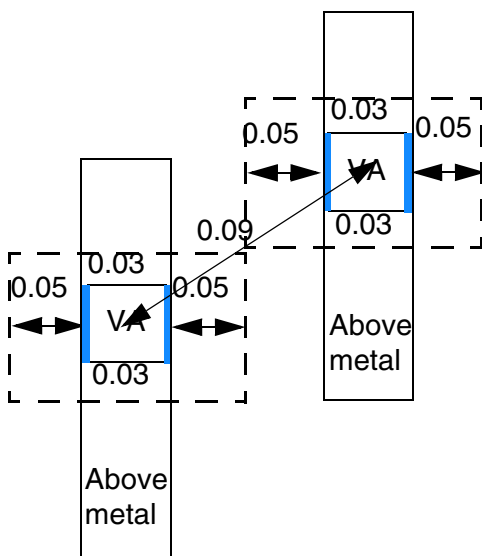


b) Okay. Center-to-center spacing  $\geq$  0.20.



**Figure 1-97 Illustration of Spacing Table Rule With Opposite Enclosure Resize Spacing**

```
PROPERTY LEF58_ENCLOSUREEDGE
    "ENCLOSUREEDGE ABOVE 0.02 OPPOSITE ; " ;
PROPERTY LEF58_SPACINGTABLE
    "SPACINGTABLE CENTERTOCENTER VA TO VA
    OPPOSITEENCLOSURERESIZESPACING VA 0.05 0.03 0.07
    CUTCLASS VA ...
    VA 0.1 0.1 ... ; " ;
```

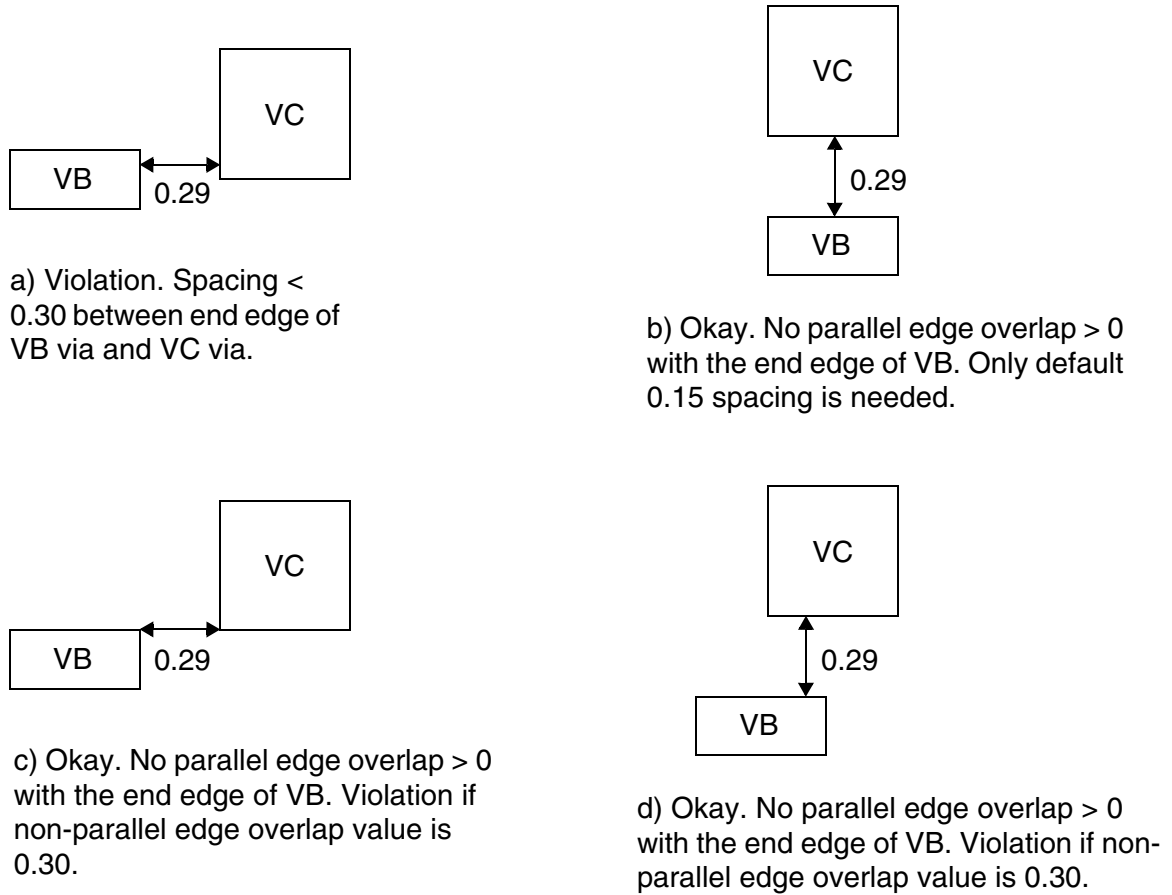


a) OK, the blue edges fulfill ENCLOSUREEDGE OPPOSITE, the dotted windows after expanding 0.05 on blue edges & 0.03 on the other 2 opposite edges touch at the corners, and cut spacing of 0.09 ( $\geq 0.07$ ) is good

- The following spacing table rule indicates that a default spacing of  $0.15 \mu\text{m}$  is required between via VC and end edge of via VB when parallel edge overlap is less than or equal to 0:

```
PROPERTY LEF58_SPACINGTABLE DEFAULT 0.15
CUTCLASS    VB SIDE    VB END    VC
VB SIDE      -      -      -    0.40    -      -
VB END       -    0.40    -    0.40    -    0.30
VC           -      -      -    0.30    0.50  0.50
```

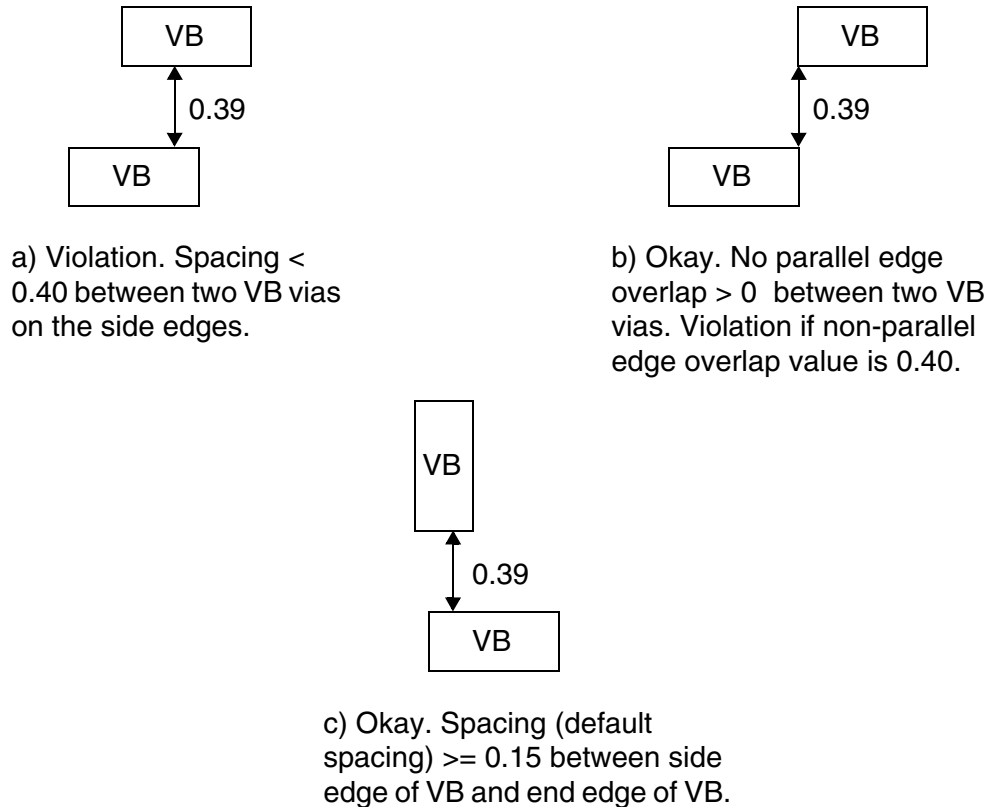
**Figure 1-98 Illustration of Spacing Table Rule With Default**



- The following spacing table rule indicates the spacing requirements between two VB vias when default spacing of 0.15  $\mu\text{m}$  is specified:

```
PROPERTY LEF58_SPACINGTABLE DEFAULT 0.15
CUTCLASS    VB SIDE    VB END
VB SIDE     -    0.40   -    -
VB END      -    -      -    -
```

**Figure 1-99 Illustration of Spacing Table Rule With Default**



- The following spacing table rules specify the intercut layer spacing of different metals between vias:

Intercut layer center-to-center spacing between two VA vias, when one of the vias is on the current cut layer and the other is on V1 cut layer	0.10 $\mu\text{m}$
Intercut layer spacing between the side edge of VB to VA via, when one of the vias is on the current cut layer and the other is on V1 cut layer	0.20 $\mu\text{m}$
Intercut layer spacing between the side edge of two VB vias, when one of the vias is on the current cut layer and the other is on V1 cut layer.	0.30 $\mu\text{m}$

**Note:** All the intercut layer spacings are excluded for same metal cuts.

The rules translate into the following SPACINGTABLE property definition:

## LEF/DEF 5.8 Language Reference

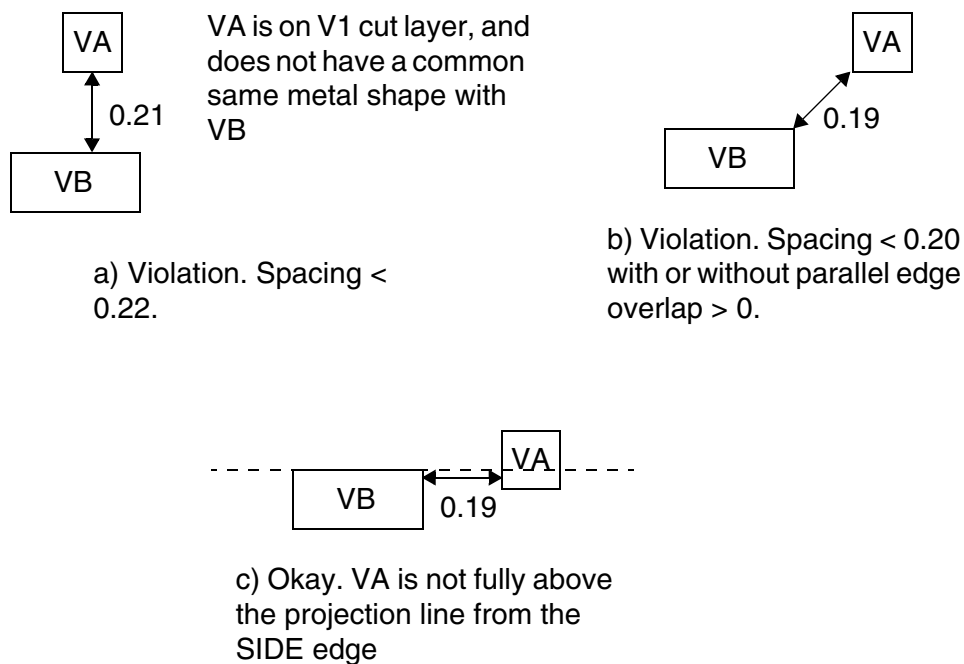
### LEF Syntax

```
PROPERTY LEF58_SPACINGTABLE LAYER V1 CENTERTOCENTER VA TO VA
CUTCLASS      VA              VB SIDE
VA            0.10  0.10    0.20  0.20
VB SIDE      0.20  0.20    0.30  0.30
```

- The following spacing table rule indicates that via VA should not overlap with the projection line from the side edge of via VB:

```
PROPERTY LEF58_SPACINGTABLE DEFAULT 0.0 LAYER V1
CUTCLASS      VA              VB SIDE
VA            -      -      0.20  0.22
VB SIDE      0.20  0.22  -      -
```

**Figure 1-100 Illustration of Spacing Table Rule With Layer**



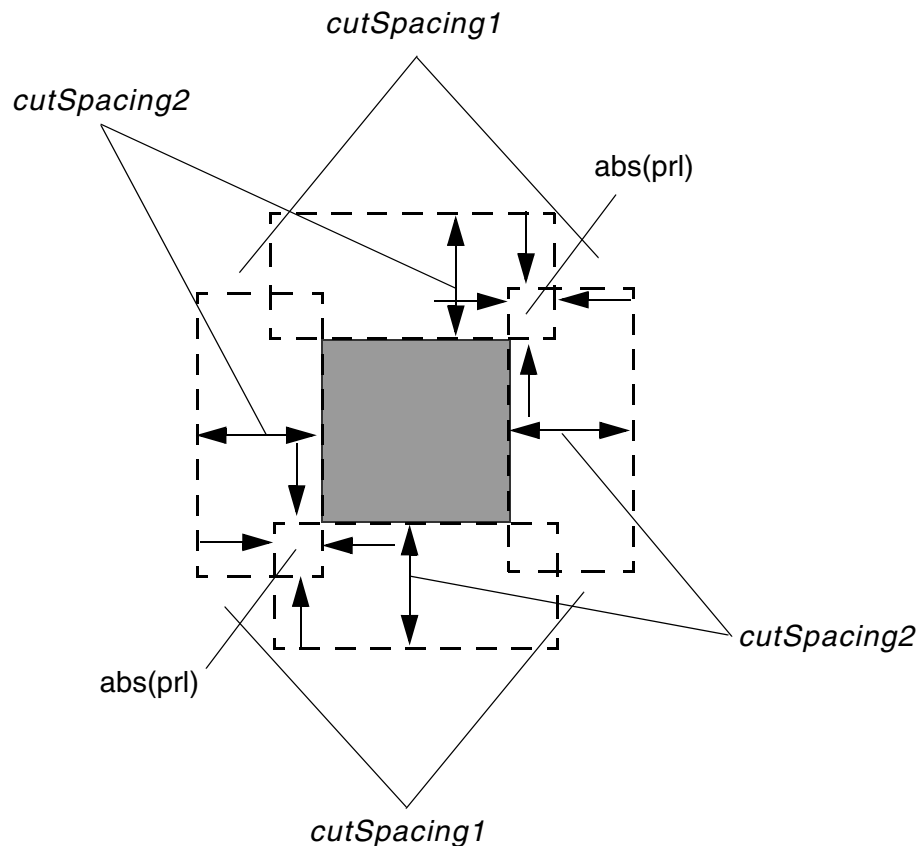
- The following spacing table rule specifies that the intercut layer spacing between VA via on the current cut layer to any center-to-center vias on C1 cut layer, when the cuts do not share a common same metal shape, must be 0.15  $\mu\text{m}$ :

```
PROPERTY LEF58_SPACINGTABLE LAYER C1 CENTERTOCENTER VA TO ALL
CUTCLASS      VA
ALL          0.15  0.15
```

- The following illustration defines PRL with MAXXY:

**Figure 1-101 Definition of Spacing Table Rule With PRL**

Definition of PRL *prl* [MAXXY] on a square cut when the *prl* value is negative.

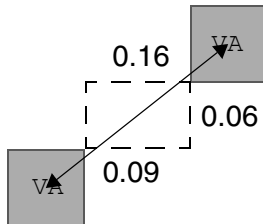


The four dotted line regions with *cutSpacing2* from the edge of a cut and *abs(prl)* extended beyond the edges are forbidden from having a neighbor cut if MAXXY is specified. Otherwise, *cutSpacing2* is measured as edge-to-edge style in those regions. If a neighbor cut is in the four corners, *cutSpacing1* should be observed.

- The following spacing table rule illustrates the use of PRLTWO SIDES:

```
PROPERTY LEF58_SPACINGTABLE "
  SPACINGTABLE CENTERTOCENTER VA TO VA
    PRLTWO SIDES -0.07 -0.05 -0.1 -0.08 VA TO VA 0.15
    CUTCLASS VA ...
    VA      0.2 0.2 ... ; " ;
```

**Figure 1-102 Definition of Spacing Table Rule With PRL**

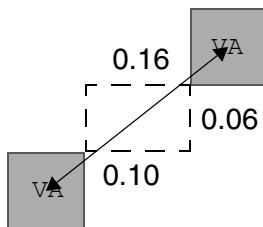


a) OK, the PRL conditions of PRLTWOSIDES is fulfilled, 0.16 ( $\geq 0.15$ ) is OK

- The following spacing table rule illustrates the use of PRLTWOSIDES with WITHIN:

```
PROPERTY LEF58_SPACINGTABLE "  
    SPACINGTABLE CENTERTOCENTER VA TO VA  
        PRLTWOSIDES -0.07 -0.05 -0.1 -0.08 WITHIN 0.11  
        VA TO VA 0.09  
    CUTCLASS VA ...  
    VA      0.2 0.2 ... ; " ;
```

**Figure 1-103 Definition of Spacing Table Rule With WITHIN in PRLTWOSIDES**



a) Violation, corner to corner spacing is 0.117 ( $> 0.11$ ). The WITHIN condition is not fulfilled, 0.2 center to center is needed & failed.

- The following spacing table rule indicates that 0.20  $\mu\text{m}$  is the cut to cut spacing between two VA vias measuring center-to-center when the cuts have parallel edge overlap greater than 0 while 0.10  $\mu\text{m}$  is the cut to cut spacing between two VA vias measuring edge-to-edge when the cuts have no parallel edge overlap:

```
PROPERTY LEF58_SPACINGTABLE  
    "SPACINGTABLE  
        CENTERANDEDGE VA TO VA  
        CUTCLASS      VA  
        VA            0.10 0.20 ; " ;
```

- The following spacing table rule indicates that the cut to cut spacing between two VA vias is the maximum of 0.20  $\mu\text{m}$  measuring center-to-center and 0.10  $\mu\text{m}$  measuring edge-to-edge:

```
PROPERTY LEF58_SPACINGTABLE
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
"SPACINGTABLE
  CENTERANDEDGE NOPRL VA TO VA
  CUTCLASS      VA
  VA            0.10  0.20 ; " ;
```

- The following spacing table rule indicates that two perfectly aligned VA cuts must have spacing greater than or equal to 0.1  $\mu\text{m}$ , two cuts with parallel edge overlap greater than 0 must have spacing greater than or equal to 0.12  $\mu\text{m}$ , and two cuts without parallel edge overlap must have spacing greater than or equal to 0.11  $\mu\text{m}$ :

```
PROPERTY LEF58_SPACINGTABLE
  "EXACTALIGNEDSPACING VA 0.1
  CUTCLASS  VA ...
  VA 0.11  0.12 ...; " ;
```

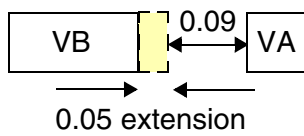
- The following spacing table rule indicates that 0.15  $\mu\text{m}$  extension is applied to the end/short edge of rectangular VB cuts before 0.10  $\mu\text{m}$  spacing is required to square VC cuts, -0.05  $\mu\text{m}$  extension (pulled in by 0.05  $\mu\text{m}$ ) is applied to the end/short edge of rectangular VB cuts before 0.10  $\mu\text{m}$  spacing is required to square VA cuts, and 0.03  $\mu\text{m}$  extension is applied to the side/long edge of rectangular VB cuts before 0.10  $\mu\text{m}$  spacing is required between two VB cuts.

```
PROPERTY LEF58_SPACINGTABLE
  "SPACINGTABLE
  DEFAULT 0.10
  ENDEXTENSION -0.05 TO VC 0.15 SIDEEXTENSION TO VB 0.03
  CUTCLASS      VA      VB END      VB SIDE      VC
  VA            -      -      -      -      -      -      -
  VB END        -      -      -      -      -      -      -
  VB SIDE       -      -      -      -      -      -      -
  VC            -      -      -      -      -      -      - ; " ;
```

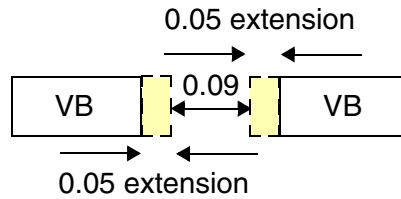
- The following example shows spacing table rule with ENDEXTENSION:

```
PROPERTY LEF58_SPACINGTABLE
  "SPACINGTABLE
  DEFAULT 0.10
  ENDEXTENSION 0.05 TO VC 0.15
  CUTCLASS      VA      VB END      VB SIDE      VC
  VA            -      -      -      -      -      -      -
  VB END        -      -      -      -      -      -      -
  VA SIDE       -      -      -      -      -      -      -
  VC            -      -      -      -      -      -      - ; " ;
```

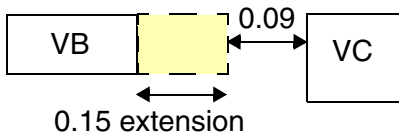
**Figure 1-104 Illustration of Spacing Table Rule with ENDEXTENSION**



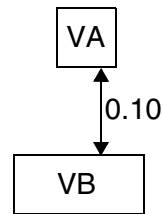
a) Violation, 0.1 spacing is needed after 0.05 extension on the end/short edge of a VB.



b) Violation, 0.05 extension is applied to the end/short edge of both VBs



c) Violation, 0.15 extension is applied when the neighbor cut belongs to VC



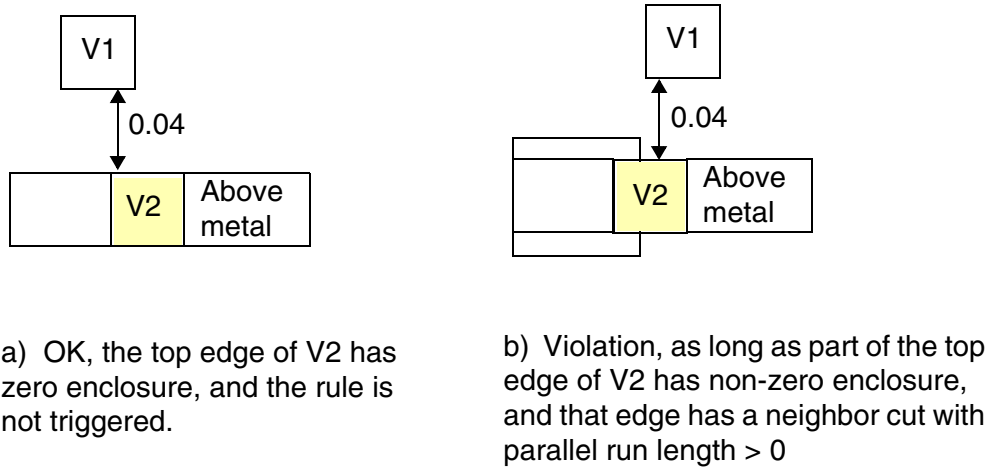
d) OK, no extension is applied to the side/long edge of a VB

- The following spacing table rule illustrates use of NONZEROENCLOSURE on layer V2:

```
PROPERTY LEF58_SPACINGTABLE
"SPACINGTABLE
  LAYER V1
    NONZEROENCLOSURE
  CUTCLASS V2....
  V1      0.000  0.05...; " ;
```



**Figure 1-105 Illustration of Spacing Table Rule with NONZEROENCLOSURE**

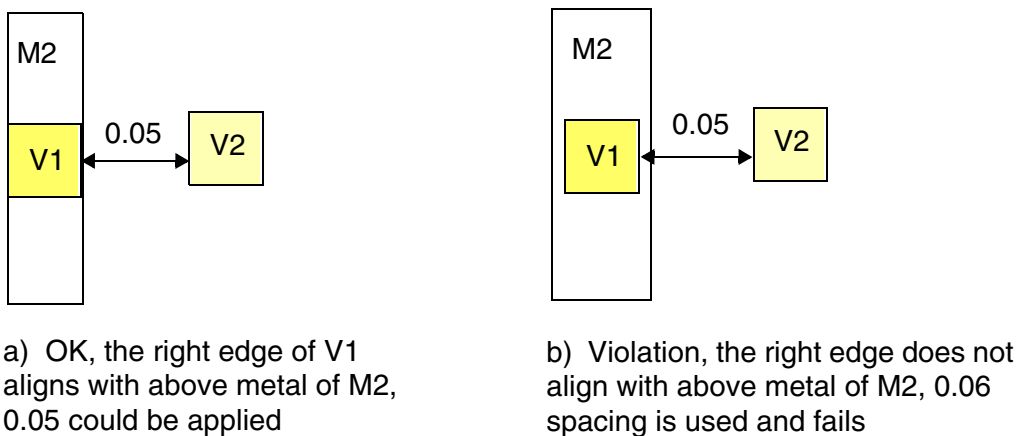


- The following example shows parallel run length for aligned cuts:

On layer V2,

```
PROPERTY LEF58_SPACINGTABLE
  "SPACINGTABLE
    LAYER V1
      PRLFORALIGNEDCUT V2 TO V1
    CUTCLASS      V2 ...
    V1             0.06   0.05 ... ; " ;
```

**Figure 1-106 Example of Spacing Table Rule with PRLFORALIGNEDCUT**

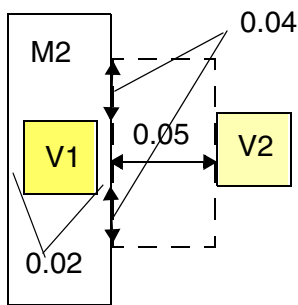


- The following example shows parallel run length for aligned cuts:

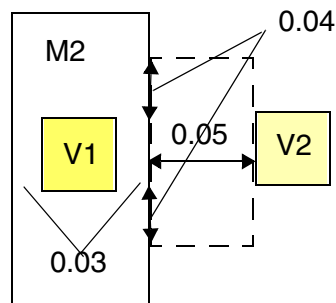
On layer V2,

```
PROPERTY LEF58_ENCLOSUREEDGE `
    ENCLOSUREEDGE ABOVE 0.02 OPPOSITE ; ` ;
PROPERTY LEF58_SPACINGTABLE
    "SPACINGTABLE
        LAYER V1
            PRLFORALIGNEDCUT V2 TO V1
            PRL -0.04
        CUTCLASS          V2 ...
        V1                0.09    0.05 ... ; " ;
```

**Figure 1-107 Example of Spacing Table Rule with PRLFORALIGNEDCUT**



a) OK, the right edge of V1 fulfills ENCLOSUREEDGE OPPOSITE, and V2 is barely outside the dotted search window on the projected metal edge by extending 0.04 of prl vertically and 0.05 of spacing horizontally.



b) Violation, the right edge does not does not fulfill ENCLOSUREEDGE OPPOSITE, 0.09 spacing is used and fails

## Center Spacing Rule

You can create a center spacing rule to specify the intercut layer spacing among cuts in center-to-center style.

You can use the following property definition:

```
PROPERTY LEF58_SPACINGTABLE
    "SPACINGTABLE CENTERSPACING
        LAYER secondLayerName
        CUTCLASS {{className1 | ALL} }...
                {{className2 | ALL} {cutSpacing}...}...
    ; " ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

LAYER *secondLayerName*

The *secondLayerName* variable defines the intercut layer spacing between a *className1* via in the first row of the table on the layer that this spacing table is being defined to another *className2* via in the first column on the *secondLayerName* cut layer. This cut spacing is ignored for same-net. This second layer must be a previously defined cut layer immediately below the layer that this spacing table is being defined, that is., "one layer look ahead" is not supported.

CUTCLASS {{*className1* | ALL}}...

Specifies a list of cut classes, which may not necessarily be a complete list of all cut classes in the cut layer. Cut spacing requirements are not needed for any missing cut classes.

The ALL keyword applies to all vias on a cut layer which has only one cut class without an explicit CUTCLASS definition.

{{*className2* | ALL} {*cutSpacing*}...}...

The *className2* variable indicates that the center-to-center *cutSpacing* is applied between itself on *secondLayerName* to the first row *className1*.

The ALL keyword has the same meaning as defined previously.  
*Type:* Float, specified in microns

For each cut layer, there can be at the most one such inter-layer CUTCLASS CENTERSPACING SPACINGTABLE. If such a spacing table is specified, then any other inter-layer spacing table cannot be defined.

### Center Spacing Rule Examples

- The following example indicates that 0.1  $\mu\text{m}$  is the center-to-center spacing between two VA via cuts, 0.20  $\mu\text{m}$  is the center-to-center spacing between two VB via cuts, and 0.15  $\mu\text{m}$  is the center-to-center spacing between a VA via cut and a VB via cut:

```
PROPERTY LEF58_SPACINGTABLE
"SPACINGTABLE CENTERSPACING
  LAYER V1
    CUTCLASS      VA      VB
    VA             0.10    0.15
    VB             0.15    0.20 ; " ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

One of the via is on the current layer with the spacing table definition while the other via is on layer V1.

### One Array Rule

You can create a one array rule to define a 1xN array via by using the following property definition:

```
PROPERTY LEF58_ONEDARRAY
    "ONEDARRAY CUTCLASS className CUTSPACING cutSpacing
      ARRAYCUTS arrayCuts SPACING spacing
      LAYER secondLayerName SPACING interLayerSpacing
      ENCLOSURE overhang1 overhang2
    ; " ;
```

Where:

ARRAYCUTS *arrayCuts*

Specifies the N, which is greater than or equal to *arrayCuts* in the 1xN array via.

*Type:* Integer

ENCLOSURE *overhang1* *overhang2*

Specifies that the via cuts in this 1XN array must have *overhang1* on two opposite sides and *overhang2* on the other two opposite sides on the above metal layer.

*Type:* Float, specified in microns

LAYER *secondLayerName* SPACING *interLayerSpacing*

Specifies the edge-to-edge cut spacing to be *interLayerSpacing* between this 1xN array via to any other different-net via cuts of any cut classes in layer *secondLayerName*.

Note: Forward look-up reference is allowed for LAYER *secondLayerName*.

*Type:* Float, specified in microns

ONEDARRAY CUTCLASS *className* CUTSPACING *cutSpacing*

Defines a 1xN array via of *className*, which cuts, with edge-to-edge spacing of *cutSpacing*, must be perfectly aligned in the preferred direction of the routing layer below this cut layer.

*Type:* Float, specified in microns

SPACING *spacing*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the edge-to-edge cut spacing of this 1xN array via to any other via cuts of any cut classes or another 1xN array via to be *spacing* when the cuts have common parallel run length are greater than 0. The 1xN array via will be merged together as one rectangular shape before the spacing check is applied. that will override other cut spacing rules (for common parallel run length greater than 0).

*Type:* Float, specified in microns

### Orthogonal Spacing Rule

You can create an orthogonal spacing rule to define additional spacing constraints for two exactly aligned cuts.

You can use the following property definition:

```
PROPERTY LEF58_ORTHOGONALSPACING
    "ORTHOGONALSPACING cutSpacing
        [CUTCLASS className] {ABOVE | BELOW}
        PARALLEL parLength WITHIN parWithin WIDTH width
        [METALPRL metalPrl METALWITHIN metalWithin]
        [HORIZONTAL | VERTICAL]
    ; " ;
```

Where:

```
METALPRL metalPrl METALWITHIN metalWithin
```

Specifies that the search window for metal neighbor wire is formed by *metalPrl* and *metalWithin* on the cut having a neighbor cut.

*Type:* Float, specified in microns

```
ORTHOGONALSPACING cutSpacing
    [CUTCLASS className] {ABOVE | BELOW}
    PARALLEL parLength WITHIN parWithin WIDTH width
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

The orthogonal spacing rule specifies that if there are two exactly aligned cuts of cut class *className*, if specified, or of any cut classes with spacing less than *cutSpacing*, and there is another cut of cut class *className*, if specified, or of any cut classes within *parWithin* having parallel run length greater than *parLength* in the orthogonal direction, it is a violation to have a wire with width less than or equal to *width* on the above or below metal layer in ABOVE or BELOW overlapping with, or even touching, a search window formed by extended *parLength* on both sides on the cut edge having a neighbor to the neighbor cut edge. Metal containing the cuts is excluded.

*Type*: Float, specified in microns

HORIZONTAL | VERTICAL

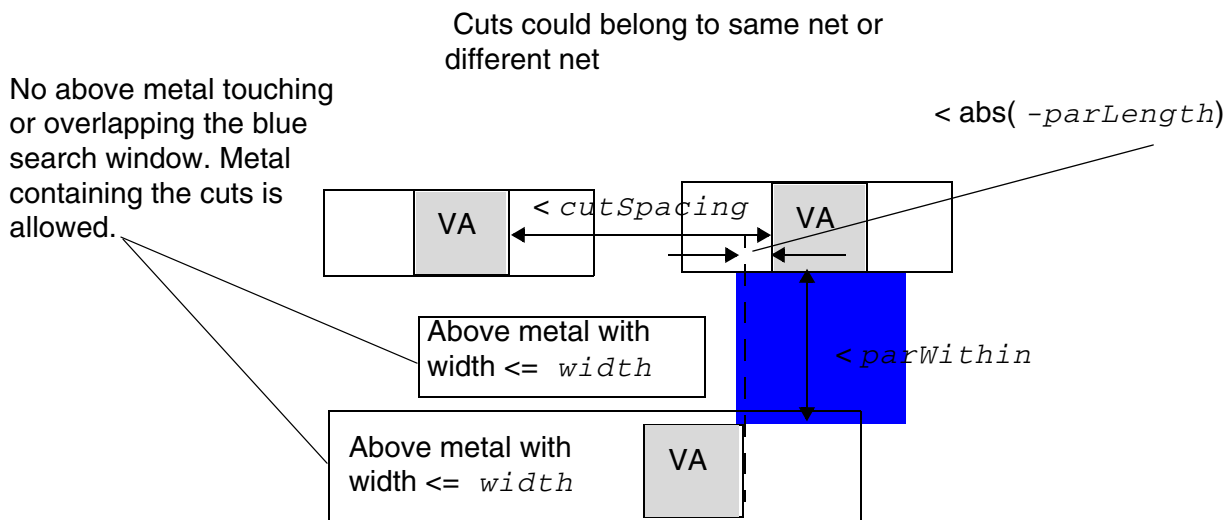
Specifies that *cutSpacing* applies only in the given direction.

### Orthogonal Spacing Rule Examples

- The following example is an illustration of the orthogonal spacing rule:

```
ORTHOGONALSPACING cutSpacing
CUTCLASS VA ABOVE
PARALLEL -parLength WITHIN parWithin
WIDTH width
```

**Figure 1-108 Illustration of Orthogonal Spacing Rule**



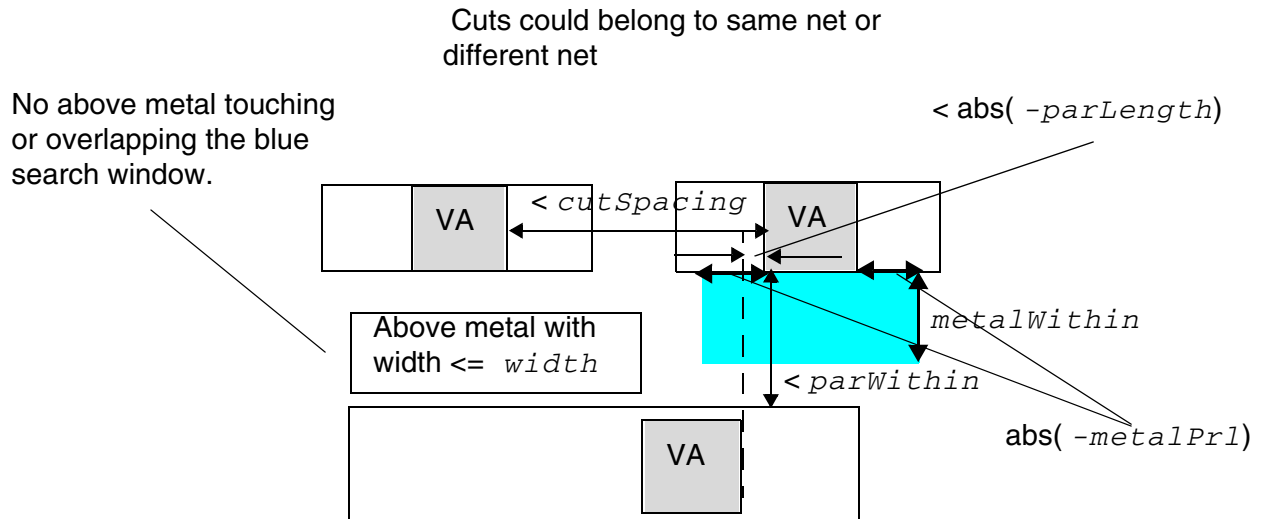
- The following example is an illustration of the orthogonal spacing rule:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

```
ORTHOGONALSPACING cutSpacing
CUTCLASS VA ABOVE
PARALLEL -parLength WITHIN parWithin
WIDTH width
METALPRL metalPr1 METALWITHIN metalWithin
```

**Figure 1-109 Illustration of the Orthogonal Spacing Rule with METALPRL and METALWITHIN**



### PRL Two Sides Spacing Rule

You can create a PRL two sides spacing rule to define additional spacing constraints for cut spacing.

You can use the following property definition:

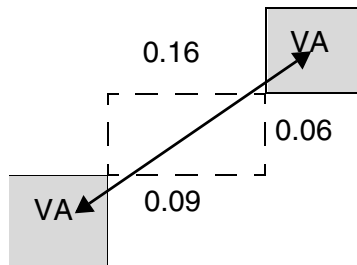
```
PROPERTY LEF58_PRLTWO SIDESSPACING
    "PRLTWO SIDESSPACING prlSpacing nonPrlSpacing
      CUTCLASS className1 TO className2
      PRL prl1 prl2 prl3 prl4
      ; " ;
```

The PRL two sides spacing rule specifies that the center-to-center spacing between a cut of *className1* and another cut of *className2* that are greater than or equal to *prl1* and less than or equal to *prl2* on one side, and greater than or equal to *prl3* and less than or equal to *prl4* on the other side to be *prlSpacing*. Otherwise, the center-to-center spacing of *nonPrlSpacing* should be applied on them. This spacing is checked in addition to the spacing defined in CUTCLASS SPACINGTABLE. This means that it is a violation if cut spacing fails in PRLTWO SIDESSPACING or in the spacing table.

**Figure 1-110 Illustration of the PRL Two Sides Spacing Rule**

Example of

```
PROPERTY LEF58_PRLTWO SIDESSPACING  "
    PRLTWO SIDESSPACING 0.15 0.2 CUTCLASS VA TO VA
    PRLTWO SIDES -0.07 -0.05 -0.1 -0.08 ; " ;
```



a) OK, the PRL conditions of PRLTWO SIDES is fulfilled, 0.16 ( $\geq$  0.15) is OK

### Region Rule

You can create a region rule to define area-based rules on a cut layer.

You can use the following property definition:

```
PROPERTY LEF58_REGION
    "REGION regionLayerName BASEDLAYER cutLayerName
    ; " ;
```

Where:

```
REGION regionLayerName BASEDLAYER cutLayerName
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies a set of region- or area-based rules on a cut layer *cutLayerName*. *regionLayerName* is a masterslice layer with type REGION. Users can create a blockage on that masterslice layer on a design to indicate the areas of the region. In addition, MACRO may also have OBS on that masterslice layer to indicate that the areas of its instance should be subjected to these region-based rules. All the rules specified on this cut layer with REGION are applied to the corresponding areas of *regionLayerName* on *cutLayerName*. The areas outside *cutLayerName* honor the set of rules on the given cut layer without type REGION. If a rule check crosses the boundaries of a region, it is subjected to the rules for both inside and outside the region, with the tighter rule values of the two sets dominating.

The set of rules defined in the layer with type REGION should be very small as compared to the layer without type REGION. The rules defined in this layer with type REGION should be a replacement of the corresponding rule (having the same set of the constructs of a LEF statement or a complete replacement on a table, like SPACINGTABLE) on the layer without type REGION. All the other rules defined on the layer without type REGION are also applicable on the layer with type REGION.

### Region Rule Example

The following example means that a cut overlapping region R1 should have spacing of 0.7 and a cut spacing check outside region R1 should have spacing of 0.5. Adjacent cut and any other rules on layer V1 (without REGION) would also be applied on region R1:

```
LAYER R1
    TYPE MASTERSLICE ;
    PROPERTY LEF58_TYPE "TYPE REGION ; " ;
END R1
...
LAYER V1
    TYPE CUT ;
    ...
    SPACING 0.5 ;
    SPACING 0.8 ADJACENTCUTS 2 WITHIN 0.8 ;
    ...
END V1

LAYER V1R1
```

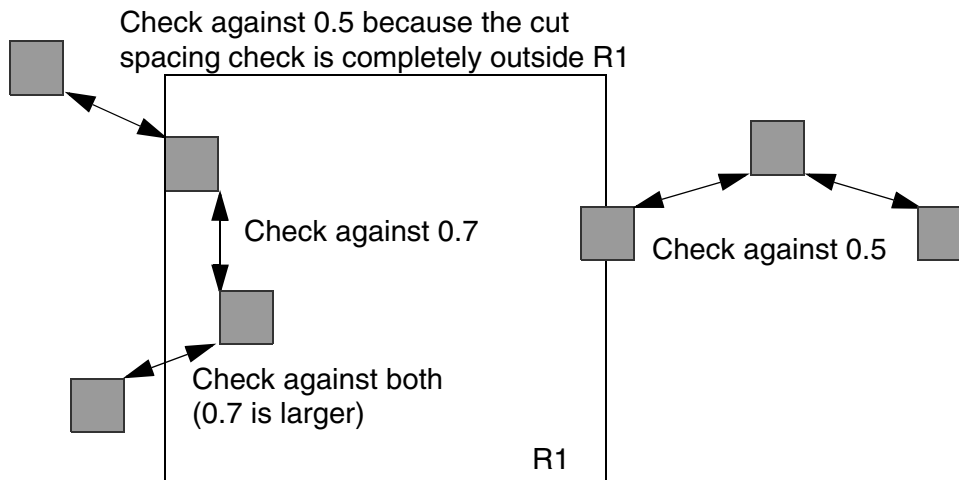
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
TYPE CUT ;
PROPERTY LEF58_REGION "REGION R1 BASEDLAYER V1 ; " ;
...
SPACING 0.7 ;
...
END V1R1
```

**Figure 1-111 Illustration of the Region Rule**



### Same-Metal Aligned Cuts Rule

You can create a same-metal aligned cuts rule to define the number of center-aligned cuts that share the same metal on a cut layer.

You can use the following property definition:

```
PROPERTY LEF58_SAMEMETALALIGNEDCUTS
    "SAMEMETALALIGNEDCUTS numCuts CUTCLASS {className | ALL}
    [ABOVE | BELOW] WIDTH width SPACING spacing
    ; " ;
```

Where:

ABOVE | BELOW      Specifies that the rule applies only if the cuts share the same metal on the given above or below metal layer.

CUTCLASS {className | ALL}      Specifies that the rule applies only if the cuts belong to the given cut class of *className* or any cut class if ALL is specified.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`SAMEMETALALIGNEDCUTS numCuts WIDTH width SPACING spacing`

Specifies the number of center-aligned cuts to be less than or equal to *numCuts* if the cuts share the same metal on the above or below metal layer with width less than or equal to *width* and center-to-center spacing less than *spacing*.

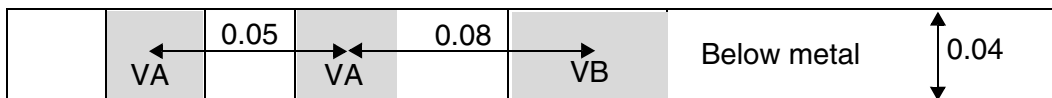
*Type:* Float, specified in microns

### Same Metal Aligned Cuts Rule Example

- The following example is an illustration of the same metal aligned cuts rule:

```
PROPERTY LEF58_SAMEMETALALIGNEDCUTS "  
    SAMEMETALALIGNEDCUTS 2 CUTCLASS ALL  
    BELOW WIDTH 0.04 SPACING 0.1 ; "
```

**Figure 1-112 Illustration of the Same Metal Aligned Cuts Rule**



a) Violation, the number of center-aligned cuts of any cut class sharing the same metal on the below metal layer with center-to-center spacing  $< 0.1$  is 3 ( $> 2$ ).

### Via Cluster Rule

You can create a via cluster rule to define a list of critiques to form a via cluster for via cuts belonging to a cut class, and the spacing requirements on them.

You can use the following property definition:

```
PROPERTY LEF58_VIACLUSTER  
    "VIACLUSTER CUTCLASS className  
        CUTS perpendicularNumCut  
        [DIAGONAL diagonalNumCut]  
        [NOPRLSPACING diagonalSpacing bendSpacing WITHIN within  
            EXTENSION sideExtension edgeExtension]]  
    WITHIN minWithin maxWithin  
    ; "
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

CUTS *perpendicularNumCut* [DIAGONAL *diagonalNumCut*]

Specifies that the maximum consecutive number of via cuts of a cluster perfectly aligned in a direction perpendicular to the cut edges that fulfill the overhang defined in ENCLOSUREEDGE OPPOSITE to be *perpendicularNumCut* or diagonally aligned without bend/notch to be *diagonalNumCut*, if defined. In other words, there is no limit to the consecutive number of via cuts perfectly aligned in a direction parallel to the cut edges that fulfill the overhang defined using ENCLOSUREEDGE OPPOSITE or diagonally if DIAGONAL keyword is not defined.

*Type:* Integer

NOPRLSPACING *diagonalSpacing* *bendSpacing* WITHIN *within*  
EXTENSION *sideExtension* *edgeExtension*

Specifies the cluster criteria when *diagonalNumCut* is set to 0, which implies that there is no limit to the number of cuts that can be inserted diagonally and bend/notch is allowed. If the cuts are inserted diagonally without bend/notch, at the most two neighbor cuts can be found within *within* having spacing greater than or equal to *diagonalSpacing* in center-to-center measurements (on both *within* and *diagonalSpacing*). Otherwise (with bend/notch), at the most two non-perfectly aligned neighbor cuts can be found within *within* having spacing greater than or equal to the *bendSpacing* in center-to-center measurements (on both *within* and *bendSpacing*).

In addition to forming a via cluster, one neighbor cut inserted diagonally without any bend/notch or two neighbor cuts having a bend/notch must be found touching a search window formed by extending *sideExtension* along both sides on the cut edges, that fulfill the overhang defined in ENCLOSUREEDGE OPPOSITE rule, and with the *edgeExtension* perpendicular to those edges.

*Type:* Float, specified in microns

VIACLUSTER CUTCLASS *className*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines a list of critiques to form a via cluster for via cuts belonging to cut class *className*, and the spacing requirements on them.

WITHIN *minWithin* *maxWithin*

Specifies that the via cuts must be larger than or equal to *minWithin* from each other in square corner (maximum x and y) style. For diagonal cluster, the via cuts must be *minWithin* apart in both x and y direction in corner to corner style. To form a via cluster, the via cuts must be within *maxWithin* in square corner style.

In other words, the cut to cut spacing among via cuts in the same cluster will be *minWithin* while a via cut of a cluster to a via cut of another cluster or not belonging to a cluster will be subject to the regular cut spacing rules.

*Type:* Float, specified in microns

**Figure 1-113 Illustration of the Via Cluster Rule with WITHIN**

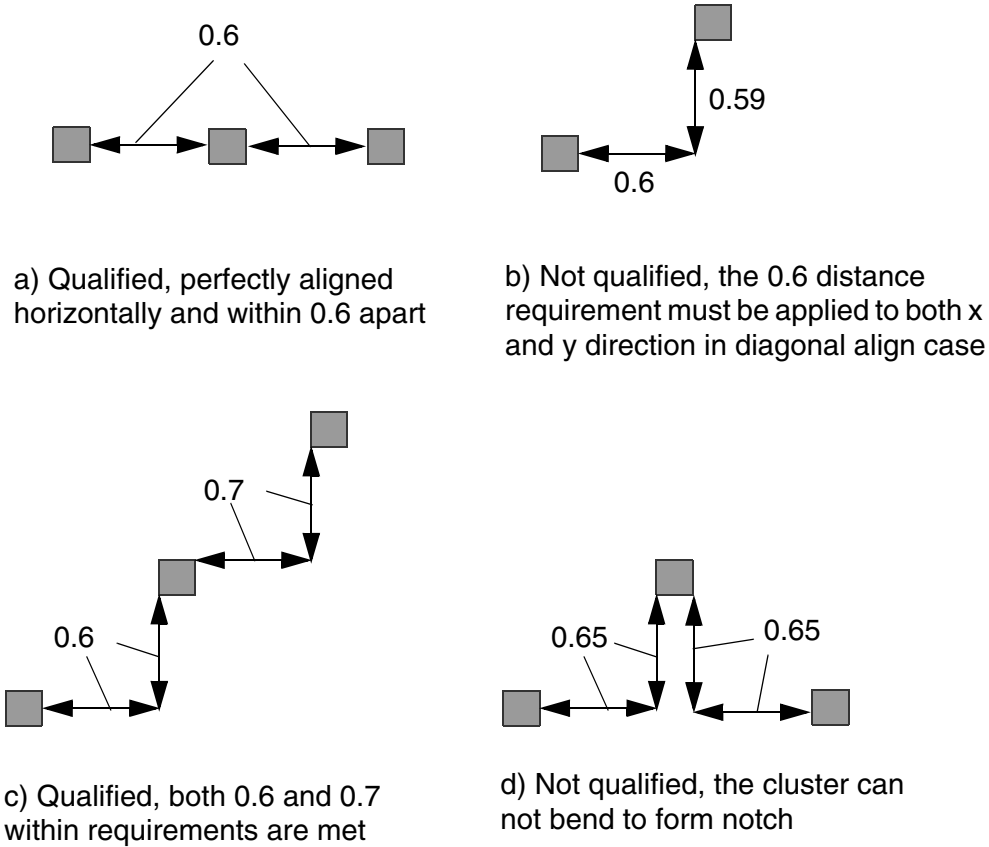
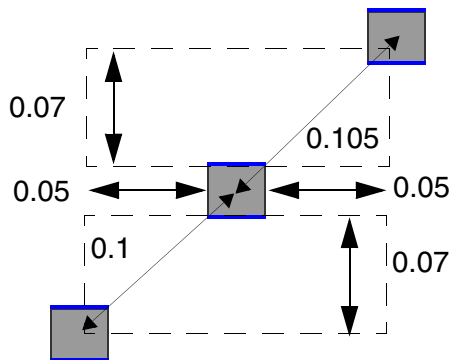


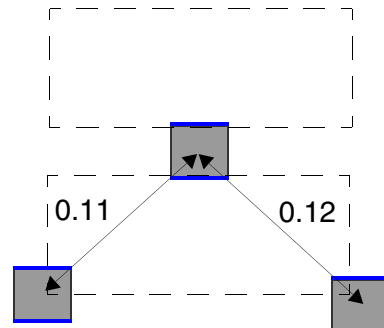
Illustration of CUTS 3 DIAGONAL 3 WITHIN 0.6 0.7

**Figure 1-114 Illustration of the Via Cluster Rule with NOPRLSPACING**

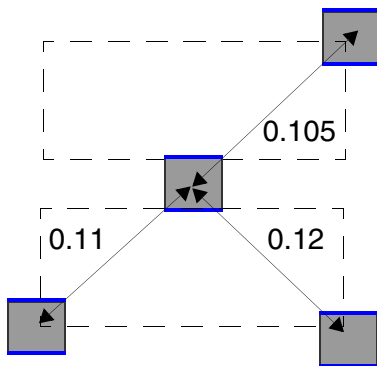
The blue edges fulfill the ENCLOSUREEDGE OPPOSITE rule



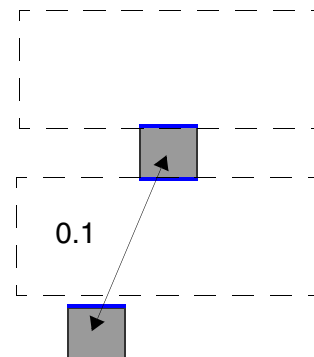
a) Qualified, 2 neighbor cuts diagonally with center-to-center spacing > 0.1



b) Qualified, 2 neighbor cuts in bend/notch style



c) Not qualified, too many neighbor cuts within 0.13 distance



d) Not qualified, for non perfectly aligned cuts with parallel run length > 0, the neighbor cut would be outside the search window

Illustration of CUTS 3 DIAGONAL 0  
NOPRLSPACING 0.1 0.11 WITHIN 0.13  
EXTENSION 0.05 0.07

## Via Group Rule

You can create a via group rule by using the following property definition:

```
PROPERTY LEF58_VIAGROUP
    "VIAGROUP groupName CUTS numCut CUTCLASS className
    PRLTWSIDES prl1 prl2 prl3 prl4
    SPACING minSpacing maxSpacing
    ; " ;]
```

Where:

```
VIAGROUP groupName CUTS numCut CUTCLASS className
    PRLTWSIDES prl1 prl2 prl3 prl4
    SPACING minSpacing maxSpacing
```

Specifies the maximum number of cuts within a via group of cuts belonging to cut class *className* to be *numCut*. The via group has the name of *groupName*. Any two cuts of a via group must have the following:

- PRL greater than *prl1* and less than *prl2* on one side
- PRL greater than *prl3* and less than *prl4* on the other side
- Spacing greater than *minSpacing* and less than *maxSpacing*

In addition, all the cuts must be diagonal without any bend/notch.

*Type:* Float, specified in microns

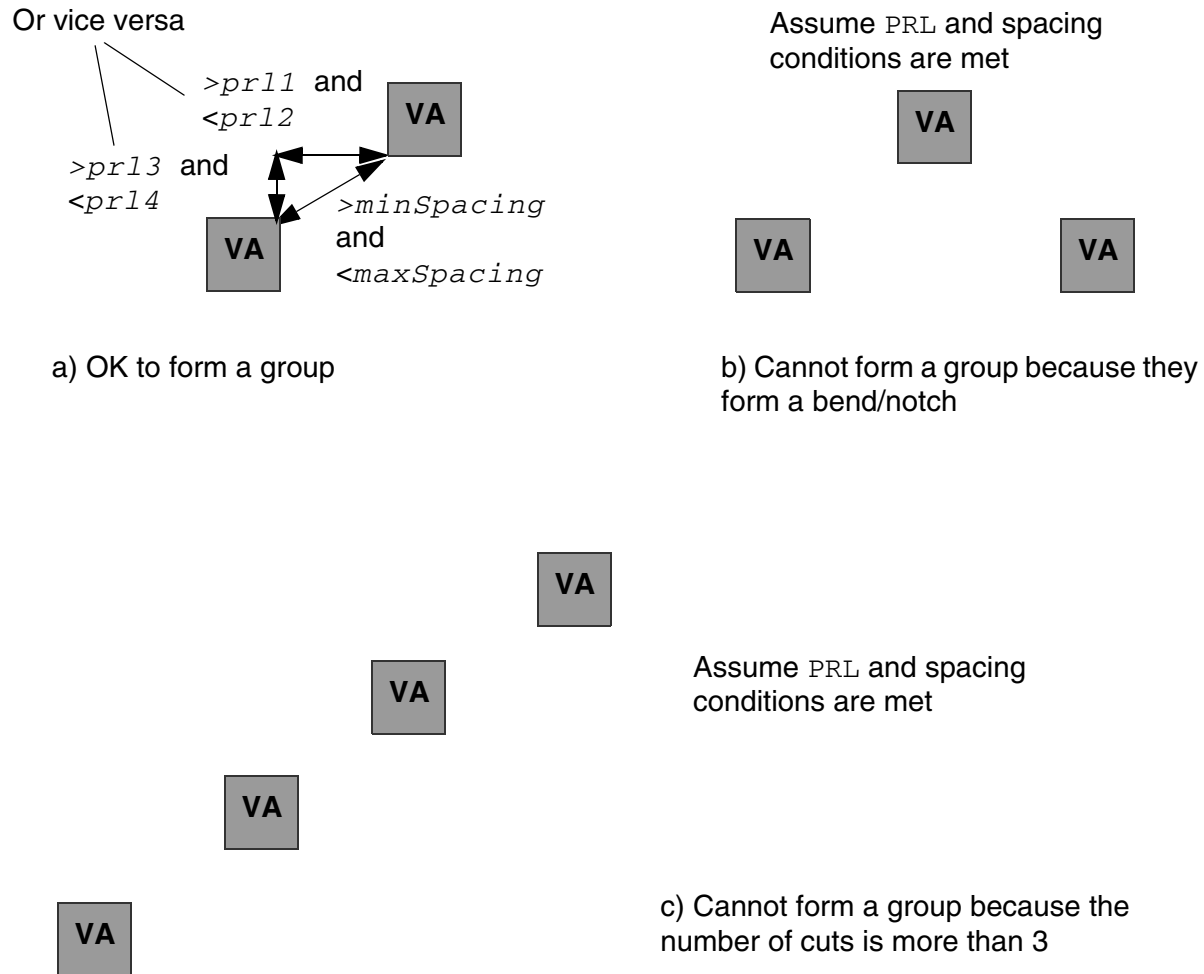
## Via Group Rule Example

- The following example is an illustration of the via group rule:

```
PROPERTY LEF58_VIAGROUP "  
    VIAGROUP groupName CUTS 3 CUTCLASS VA  
    PRLTWSIDES prl1 prl2 prl3 prl4  
    SPACING minSpacing maxSpacing "  
    ;
```



**Figure 1-115 Illustration of the Via Group Rule**



## Via Group Spacing Rule

You can create a via group spacing rule by using the following property definition:

```
PROPERTY LEF58_VIAGROUPSPACING
    "VIAGROUPSPACING spacing VIAGROUP groupName CUTS numCut
    CUTCLASS className
    [PARALLEL parLength WITHIN parWithin
    OTHERCUTCLASS otherClassName]
    ; " ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

```
PARALLEL parLength WITHIN parWithin  
    OTHERCUTCLASS otherClassName
```

Specifies the spacing rule applies only if any cuts belonging to via group *groupName* is a neighbor cut of a top or bottom edge of a cut with cut class *otherClassName* within *parWithin* having a parallel run length greater than *parLength*.

*Type:* Float, specified in microns

```
VIAGROUPSPACING spacing VIAGROUP groupName CUTS numCut  
    CUTCLASS className
```

Specifies the spacing of cuts belonging to via group *groupName* with number of cuts *numCut* to cuts belonging to cut class *className* to be *spacing*.

*Type:* Float, specified in microns

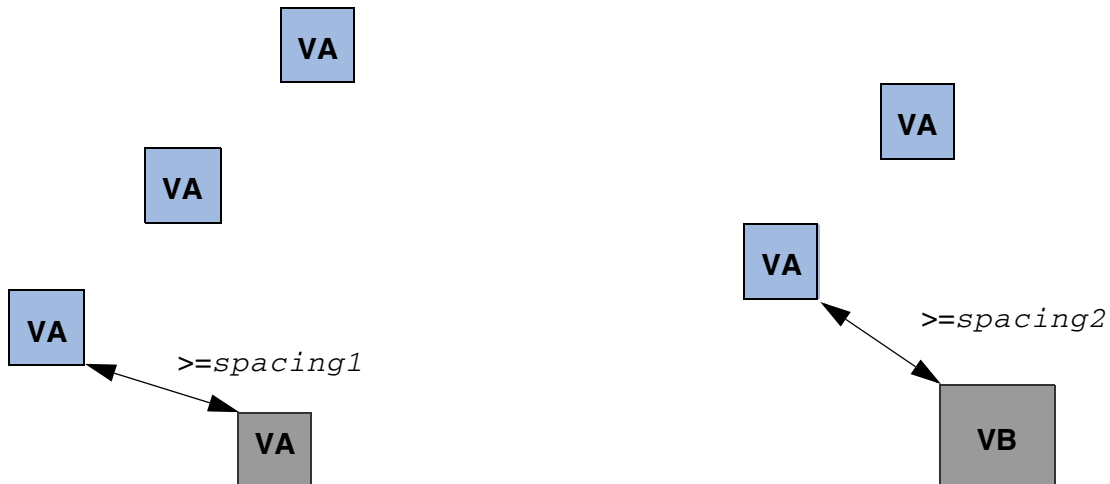
### Via Group Spacing Rule Examples

- This example illustrates the following via group spacing rules:

```
VIAGROUP VGA CUTS 3 CUTCLASS VA  
    PRLTWSIDES prl1 prl2 prl3 prl4  
    SPACING minSpacing maxSpacing  
VIAGROUPSPACING spacing1  
    VIAGROUP VGA CUTS 3 CUTCLASS VA  
VIAGROUPSPACING spacing2  
    VIAGROUP VGA CUTS 2 CUTCLASS VB
```

**Figure 1-116 Illustration of the Via Group Spacing Rule**

Assume the blue VA cuts meet the PRL and spacing conditions to be part of the via group VGA.



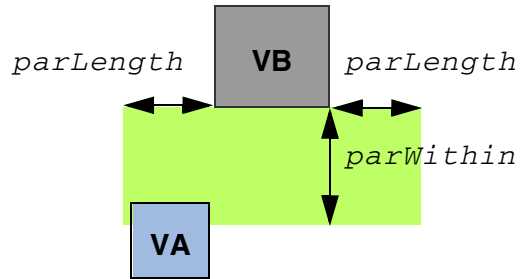
- This example illustrates the following via group spacing rule:

```

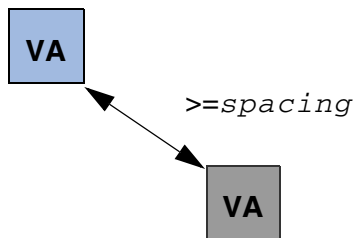
VIAGROUP VGA CUTS 3 CUTCLASS VA
    PRLTWO SIDES prl1 prl2 prl3 prl4
    SPACING minSpacing maxSpacing
VIAGROUPSPACING spacing
    VIAGROUP VGA CUTS 2 CUTCLASS VA
    PARALLEL parLength WITHIN parWithin
    OTHERCUTCLASS VB
    
```

**Figure 1-117 Illustration of the Via Group Spacing Rule with OTHERCUTCLASS**

Assume the blue VA cuts meet the `PRL` and spacing conditions to be part of the via group VGA.



A VA of group VGA overlaps with the green search window formed by a VB cut.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### Layer (Implant)

```
LAYER layerName
    TYPE IMPLANT ;
    [MASK maskNum ;]
    [WIDTH minWidth ;]
    [SPACING minSpacing [LAYER layerName2]    ;] ...
    [PROPERTY propName propName ;] ...
    [PROPERTY LEF58 AREA
        "AREA minArea
    ]; " ;
    [PROPERTY LEF58 COREEDGELENGTH
        "COREEDGELENGTH minLength
        [EXCEPTADJACENTLENGTH
            {exactEdgeLength adjLength [EXACTADJACENTLENGTH]}...]
    ]; " ;
    [PROPERTY LEF58 MINENCLOSEDAREA
        "MINENCLOSEDAREA area
    ]; " ;
    [PROPERTY LEF58 SPACING
        "SPACING minSpacing [LAYER layerName2]
        [HORIZONTAL|VERTICAL PRL prl ] [EXCEPTABUTTED]
        [EXCEPTCORNERTOUCH]
    ]; " ;
    [PROPERTY LEF58 WIDTH
        "WIDTH minWidth [ZEROPRL] [EXCEPTCORNERTOUCH]
    ]; " ;

END layerName
```

Defines implant layers in the design. Each layer is defined by assigning it a name and simple spacing and width rules. These spacing and width rules only affect the legal cell placements. These rules interact with the library methodology, detailed placement, and filler cell support. You must define implant layers separately, with their own layer statements.

LAYER <i>layerName</i>	Specifies the name for the layer. This name is used in later references to the layer.
LAYER <i>layerName2</i>	Specifies the name of another implant layer that requires extra spacing that is greater than or equal to <i>minSpacing</i> from this implant layer.
MASK <i>maskNum</i>	Specifies how many masks for double- or triple-patterning will be used for this layer. The <i>maskNum</i> variable must be an integer greater than or equal to 2. Most applications only support values of 2 or 3.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

PROPERTY *propName propVal*

Specifies a numerical or string value for a layer property defined in the `PROPERTYDEFINITIONS` statement. The *propName* you specify must match the *propName* listed in the `PROPERTYDEFINITIONS` statement.

SPACING *minSpacing*

Specifies the minimum spacing for the layer. This value affects the legal cell placement.

*Type:* Float, specified in microns

TYPE IMPLANT

Identifies the layer as an implant layer.

WIDTH *minWidth*

Specifies the minimum width for this layer. This value affects the legal cell placement.

*Type:* Float, specified in microns

#### Example 1-8 Implant Layer

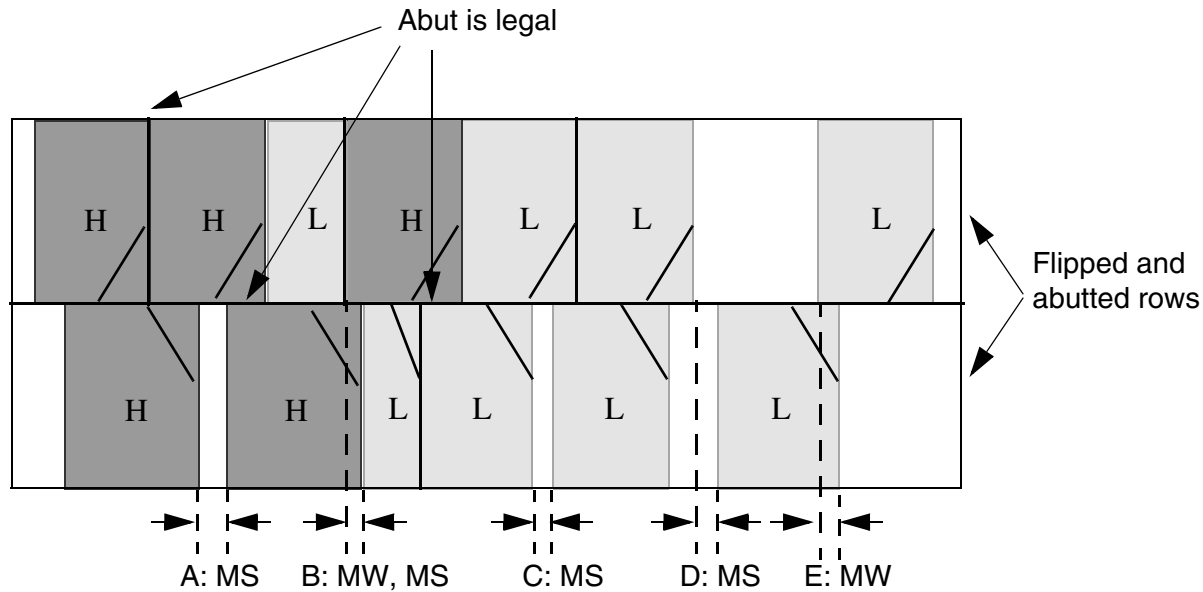
Typically, you define high-drive cells on one implant layer and low-drive cells on another implant layer. The following example defines high-drive cells on *implant1* and low-drive cells on *implant2*. Both implant layers cover the entire cell. The placer and filler cell creation attempt to legalize the cell overlaps in abutting rows to ensure that the minimum width and spacing values are met.

```
LAYER implant1          #high-drive implant layer
    TYPE IMPLANT ;
    WIDTH 0.50 ;         #implant rectangles must be >=0.50 microns wide
    SPACING 0.50 ;       #implant rectangles must be >=0.50 microns apart
END implant1

LAYER implant2          #low-drive implant layer
    TYPE IMPLANT ;
    WIDTH 0.50 ;         #implant rectangles must be >=0.50 microns wide
    SPACING 0.50 ;       #implant rectangles must be >=0.50 microns apart
END implant2
```

Assume that the high-drive cells and low-drive cells are completely covered by their respective implant layers. Because there is no spacing between *implant1* and *implant2* specified, you might see a placement like that illustrated in [Figure 1-118](#) on page 215.

**Figure 1-118**



MS = Minimum spacing error  
MW = Minimum width error

Note that you can correct A, C, D, and E by putting in filler cells with the appropriate implant type. However, B cannot be corrected by a filler cell—either the placer must avoid it, or you must allow the filler cell or post-process command to move cells or modify the implant layer to correct the error.

### **Area Rule**

An area rule can be used to specify the minimum area on the implant layer.

You can use the following property definition:

```
PROPERTY LEF58_AREA
    "AREA minArea
    ; " ;
```

Where:

*AREA minArea* Specifies the minimum area on the implant layer.  
*Type:* Float, specified in microns.

### Core Edge Length Rule

A core edge length rule can be used to indicate that the edge length of an implant shape of a CORE (standard) cell must be greater than or equal to the specified minimum length.

You can create a core edge length rule by using the following property definition:

```
PROPERTY LEF58_COREEDGELENGTH
    "COREEDGELENGTH minLength
        [EXCEPTADJACENTLENGTH
            {exactEdgeLength adjLength [EXACTADJACENTLENGTH]}...]
    ; " ;
```

Where:

`COREEDGELENGTH minLength`

Specifies that the edge length of the implant shape of a CORE (standard) cell, in the direction of the cell row, must be greater than or equal to the *minLength*.

*Type:* Float, specified in microns

`EXCEPTADJACENTLENGTH`

`{exactEdgeLength adjLength [EXACTADJACENTLENGTH]}...`

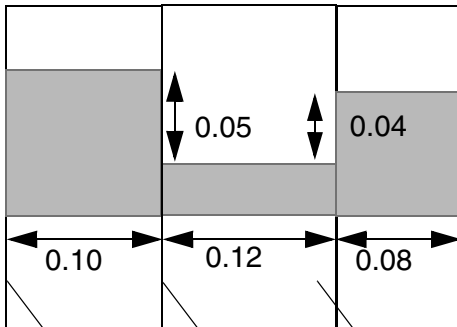
Specifies that the edge length is exempted if it's length is exactly equal to *exactEdgeLength*, with an adjacent length less than or equal to *adjLength*, or exactly equal to *adjLength* if EXACTADJACENTLENGTH is defined.

*Type:* Float, specified in microns



**Figure 1-119 Example of Core Edge Length Rule**

```
PROPERTY LEF58_COREEDGELENGTH "  
COREEDGELENGTH 0.12 EXCEPTADJACENTLENGTH  
0.08 0.04 EXACTADJACENTLENGTH 0.10 0.06 ; "
```



a) OK, the left edge has length of 0.10 with an adjacent length of 0.05 ( $\leq 0.06$ ), the middle edge has length of 0.12 ( $\geq 0.12$ ) and the right edge has length of 0.08 with an adjacent length of 0.04, which the left & right edges fulfill the exemption

### **Min Enclosed Area Rule**

A min enclosed area rule can be used to specify the minimum area that is enclosed by shapes on the current implant layer.

You can create a min enclosed area rule by using the following property definition:

```
PROPERTY LEF58_MINENCLOSEDAREA  
"MINENCLOSEDAREA area  
; "
```

Where:

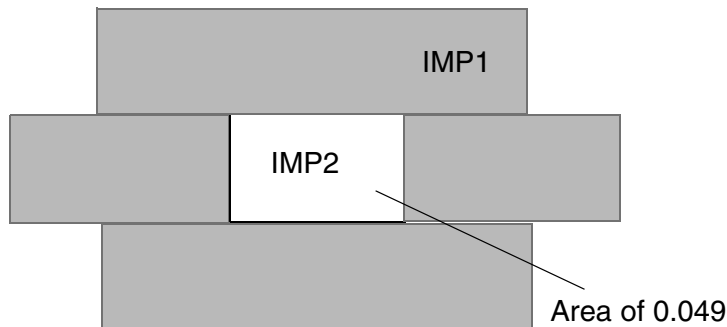
**MINENCLOSEDAREA area**

Specifies the minimum area that is enclosed by shapes on the current implant layer.

*Type:* Float, specified in microns square

**Figure 1-120 Example of the Min Enclosed Area Rule**

```
LAYER IMP1
  TYPE IMPLANT
  PROPERTY LEF58_MINENCLOSEDAREA "
    MINENCLOSEDAREA 0.5 ; " ;
END IMP1
```



a) Violation, the area doughnut hole of IMP1 < 0.5

### ***Spacing Rule***

A spacing rule can be used to specify the spacing requirements for the current implant layer. You can create a spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING
  "SPACING minSpacing [LAYER layerName2]
    [HORIZONTAL|VERTICAL PRL prl] [EXCEPTABUTTED]
    [EXCEPTCORNERTOUCH]
  ; " ;
```

Where:

HORIZONTAL|VERTICAL PRL *prl*

Specifies that the implant spacing rule applies only in the given direction. By default, the spacing is applied horizontally for implant shapes on the same row. In the VERTICAL case, the rule applies only if the horizontal projected PRL is greater than *prl*. If *prl* is negative, the rule is checked for two implant shapes within  $\text{abs}(\textit{prl})$ .

*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**EXCEPTABUTTED** Specifies that the implant spacing rule is exempted if the two implant shapes are abutted. PRL in the orthogonal direction of the spacing (that is, vertical PRL in HORIZONTAL or horizontal PRL in VERTICAL) must be greater than 0 to trigger the exemption. In other words, the corner or point touch in the VERTICAL case for horizontal cell rows is not exempted.

**EXCEPTCORNERTOUCH**

Specifies that the implant spacing rule is exempted if the two implant shapes are corner or point touch in the VERTICAL case for horizontal cell rows.

**LAYER** *layerName2*

Specifies the name of another implant layer that requires extra spacing that is greater than or equal to *minSpacing* from this implant layer.

**SPACING** *minSpacing*

Specifies the minimum spacing for the layer. This value affects the legal cell placement.

*Type:* Float, specified in microns

### Width Rule

A width rule can be used to specify the width requirements for the current implant layer. You can create a spacing rule by using the following property definition:

```
[PROPERTY LEF58 WIDTH
  "WIDTH minWidth [ZEROPRL] [EXCEPTCORNERTOUCH]
]; "
```

Where:

**EXCEPTCORNERTOUCH**

Specifies that the width requirement does not apply if corners touch.

**WIDTH** *minWidth*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

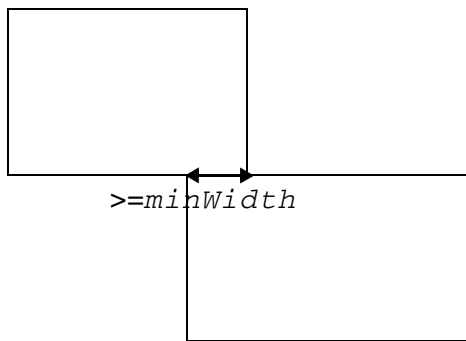
Specifies the minimum width for this layer. This value affects the legal cell placement.

Type: Float, specified in microns

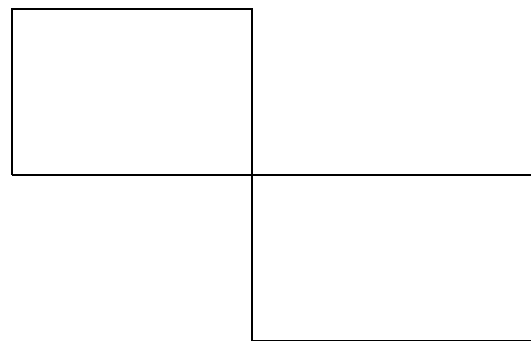
ZEROPRL

Specifies that the width requirement applies only for two implant shapes on adjacent rows with zero PRL in the direction perpendicular to the row direction.

**Figure 1-121 Example of the Width Rule**



Horizontal width must be  $\geq \text{minWidth}$



Corner touching is allowed. Violation if EXCEPTCORNERTOUCH is omitted.

Illustration of WIDTH *minWidth* ZEROPRL EXCEPTCORNERTOUCH

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## Layer (Masterslice or Overlap)

```
LAYER layerName
  TYPE {MASTERSLICE | OVERLAP} ;
  [MASK maskNum ;]
  [PROPERTY propName propVal ;] ...
  [PROPERTY LEF58 TYPE
    "TYPE [NWEELL | PWEELL | ABOVEEDGE | BELOWDIEEDGE | DIFFUSION | TRIMPOLY
      | TRIMMETAL | REGION | MEOL]
    "; " ;
  [PROPERTY LEF58 AREA
    "AREA minArea
    "; " ;
  [PROPERTY LEF58 COREEDGELENGTH
    "COREEDGELENGTH minLength [MAXLENGTH maxLength]
    [CONCAVECORNER | CONVEXCORNER | MIXEDCORNER]
    [EXTENDTOALIGN width spacing]
    "; " ;
  [PROPERTY LEF58 ENCLOSURE
    "ENCLOSURE overhang LAYER secondLayerName;..." ;]
  [PROPERTY LEF58 MAXLENGTH
    "MAXLENGTH maxLength
    "; " ;]
  [PROPERTY LEF58 REGION
    "REGION regionLayerName BASEDLAYER trimLayerName
    "; " ;]
  [PROPERTY LEF58 SPACING
    "SPACING minSpacing
    "; " ;
  [PROPERTY LEF58 SPACING
    "SPACING minSpacing [SAMENET | LAYER secondLayerName];..." ;]
  [PROPERTY LEF58 SPACING
    "SPACING spacing [PRLSPACING prlSpacing1 prlSpacing2]
    [ENDTOEND endToEndSpacing [PRL prl]
    [EXACTALIGNED exactAlignedSpacing
    [EXCEPTDIFFMASKALIGNED]]
    [EDGEALIGNED edgeAlignedSpacing]
    [EXCEPTMIDMETALWIDTH width]]
    [SAMEMASK]
    "; " ;...
  [PROPERTY LEF58 TRIMMEDMETAL
    "TRIMMEDMETAL metalLayer [MASK maskNum]
    "; " ;
  [PROPERTY LEF58 TRIMSHAPE
    "TRIMSHAPE EXTENSIONMODEL
    {ADJACENTTRACK | MIDTRACK | EXTENSION extension [METALEDGE]}
    EXACTWIDTH exactWidth
    [MAXLENGTH maxLength [MASK maskNum]]
    [EXCEPTSPACING spacing] [EXCEPTWIDTH width]
    [USEMETALMASK]
    "; " ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[PROPERTY LEF58 WIDTH  
  "WIDTH minWidth [ZEROPRL]  
  ;" ;]
```

```
END layerName
```

Defines masterslice (nonrouting) or overlap layers in the design. Masterslice layers are typically polysilicon layers and are only needed if the cell `MACROs` have pins on the polysilicon layer.

The overlap layer should normally be named `OVERLAP`. It can be used in `MACRO` definitions to form rectilinear-shaped cells and blocks (that is, an “L”-shaped block).

Each layer is defined by assigning it a name and design rules. You must define masterslice or overlap layers separately, with their own layer statements.

You must define layers in process order from bottom to top. For example:

```
poly      masterslice  
cut01     cut  
metal1    routing  
cut12     cut  
metal2    routing  
cut23     cut  
metal3    routing
```

```
LAYER layerName
```

Specifies the name for the layer. This name is used in later references to the layer.

```
TYPE
```

Specifies the purpose of the layer.

**MASTERSLICE**      Layer is fixed in the base array. If pins appear in the masterslice layers, you must define vias to permit the routers to connect those pins and the first routing layer. Wires are not allowed on masterslice layers.

Routing tools can use only one masterslice layer. If a masterslice layer is defined, exactly one cut layer must be defined between the masterslice layer and the adjacent routing layers.

**OVERLAP**          Layer used for overlap checking for rectilinear blocks. Obstruction descriptions in the macro obstruction statements refer to the overlap layer.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`MASK maskNum`

Specifies how many masks for double- or triple-patterning will be used for this layer. The *maskNum* variable must be an integer greater than or equal to 2. Most applications only support values of 2 or 3.

`PROPERTY propName propVal`

Specifies a numerical or string value for a layer property defined in the `PROPERTYDEFINITIONS` statement. The *propName* you specify must match the *propName* listed in the `PROPERTYDEFINITIONS` statement.

#### Defining Masterslice Layer Properties to Create 32/28 nm and Smaller Nodes Rules

You can include masterslice layer properties in your LEF file to create 32/28nm and smaller nodes rules that currently are not supported by existing LEF syntax. The properties are specified inside the `LAYER MASTERSLICE` statements, where they can be seen with other rules.

Before you can reference them, properties must be defined at the beginning of the LEF file in the `PROPERTYDEFINITIONS` statement, immediately before the first `LAYER` statement.

- Properties belong to the `LAYER` object and have a type of `STRING`.
- The property names used for these rules all start with `LEF58_`.

All properties use the following syntax within the LEF `PROPERTYDEFINITIONS` statement:

```
PROPERTYDEFINITIONS
    LAYER propName STRING ["stringValue"] ;
END PROPERTYDEFINITIONS
```

The property definitions for the masterslice layer properties are as follows:

```
PROPERTYDEFINITIONS
    LAYER LEF58_AREA STRING ;
    LAYER LEF58_COREEDGELENGTH STRING ;
    LAYER LEF58_ENCLOSURE STRING ;
    LAYER LEF58_MAXLENGTH STRING ;
    LAYER LEF58_REGION STRING ;
    LAYER LEF58_SPACING STRING ;
    LAYER LEF58_TRIMMEDMETAL STRING ;
    LAYER LEF58_TRIMSHAPE STRING ;
    LAYER LEF58_TYPE STRING ;
    LAYER LEF58_WIDTH STRING ;
END PROPERTYDEFINITIONS
```

#### Type Rule

A type rule can be used to further classify a masterslice layer.

You can create a type rule by using the following property definition:

```
TYPE MASTERSLICE;
    PROPERTY LEF58_TYPE
        "TYPE [NWEELL | PWEELL | ABOVEEDGE | BELOWDIEEDGE | DIFFUSION | TRIMPOLY
            | TRIMMETAL | REGION | MEOL]
        ;" ;
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

ABOVEDIEEDGE | BELOWDIEEDGE

Specifies that the masterslice layer is a special one for a hard macro to define an OBS on the layer such that the die boundary of the above or below die do not overlap.

The following property statement can be defined on layers with type ABOVEDIEEDGE or BELOWDIEEDGE:

```
PROPERTY LEF58_SPACING
    "SPACING minSpacing
    ; " ;
```

DIFFUSION

Defines a diffusion layer.

The following property statements can be defined on this layer:

```
PROPERTY LEF58_COREEDGELENGTH
    "COREEDGELENGTH minLength [MAXLENGTH maxLength]
    [CONCAVECORNER | CONVEXCORNER | MIXEDCORNER]
    [EXTENDTOALIGN width spacing]
    ; " ;
```

```
PROPERTY LEF58_SPACING
    "SPACING minSpacing
    ; " ;
```

NWELL

Indicates that the layer is a nwell layer.

The following property statements can be defined on this layer:

```
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE overhang LAYER secondLayerName;..." ;
```

```
PROPERTY LEF58_SPACING
    "SPACING minSpacing [SAMENET | LAYER
    secondLayerName] ;..." ;
```

```
PROPERTY LEF58_WIDTH
    "WIDTH minWidth;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

PWELL

Indicates that the layer is a pwell layer.

The following property statements can be defined on this layer:

```
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE overhang LAYER secondLayerName;..." ;

PROPERTY LEF58_SPACING
    "SPACING minSpacing [SAMENET | LAYER
        secondLayerName] ;..." ;

PROPERTY LEF58_WIDTH
    "WIDTH minWidth;" ;
```

REGION

Defines a special masterslice layer that is used to define the areas of a region on which a set of rules defined in the metal, cut, and/or trim metal layers with the REGION property would be applied.

MEOL

Specifies that the masterslice layer is a middle-end-of-line (MEOL) layer, which is the front-end metal layer on a 3D IC design.

The following property statement can be defined on this layer:

```
PROPERTY LEF58_MAXLENGTH
    "MAXLENGTH maxLength
        ; " ;
```

where

MAXLENGTH *maxLength*

Specifies that the maximum length of any shape on an MEOL layer must be less than or equal to *maxLength*. Typically, some standard cells would have shapes on an MEOL layer. If placement abuts too many such cells, it could violate this rule. Placement needs to avoid such violations by honoring the rule.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### TRIMMETAL

Defines a trim metal layer. This layer type is only used along with metal layers manufactured with self-aligned double patterning (SADP) technology. The `TRIMMETAL` layer has the shapes for the SADP mask used to “trim” or “cut” or “block” the self-aligned metal lines created during the first mask step of SADP processing. These shapes could be pre-defined in macros/cells or added at the line-end of a wires during routing. There are additional rules in cut and metal layers to define constraints to those shapes on a `TRIMMETAL` layer.

The `TRIMMETAL` layer can have the following property to indicate which metal layer or which mask of the metal layer can be trimmed by the `TRIMMETAL` shapes. The metal layer should have the `MASK` construct with value of 2 or larger to indicate that it is SADP (or DPT) layer. The `TRIMMETAL` layer could also have the `MASK` construct to indicate number of masks on that layer.

```
PROPERTY LEF58_TRIMMEDMETAL
    "TRIMMEDMETAL metalLayer [MASK maskNum]
    ; " ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

In addition, the following property statements can be defined on this layer:

```
PROPERTY LEF58_TRIMSHAPE
    "TRIMSHAPE EXTENSIONMODEL
    {ADJACENTTRACK | MIDTRACK | EXTENSION extension
    [METAEDGE]}
    EXACTWIDTH exactWidth
    [MAXLENGTH maxLength [MASK maskNum]]
    [EXCEPTSPACING spacing] [EXCEPTWIDTH width]
    [USEMETALMASK]
    ; " ;

PROPERTY LEF58_SPACING
    "SPACING spacing [PRLSPACING prlSpacing1
    prlSpacing2]
    [ENDTOEND endToEndSpacing [PRL prl]
    [EXACTALIGNED exactAlignedSpacing
    [EXCEPTDIFFMASKALIGNED]]
    [EDGEALIGNED edgeAlignedSpacing]
    [EXCEPTMIDMETALWIDTH width]]
    [SAMEMASK]
    ; " ;...

PROPERTY LEF58_AREA
    "AREA minArea
    ; " ;

PROPERTY LEF58_SPACING
    "SPACING minSpacing
    ; " ;
```

Typically, shapes on a trim metal layer could be defined in macros/cells only. There are additional rules in cut and metal layers to define constraints for the shapes on a TRIMMETAL layer.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

TRIMPOLY Defines a TPO layer for self-aligned double patterning (SADP) technology. Only cells would have shapes on that layer. The following property statements define the constraints that placement needs to honor:

```
PROPERTY LEF58_AREA
    "AREA minArea
    ; " ;

PROPERTY LEF58_SPACING
    "SPACING minSpacing
    ; " ;

PROPERTY LEF58_COREEDGELENGTH
    "COREEDGELENGTH minLength
    ; " ;

PROPERTY LEF58_WIDTH
    "WIDTH minWidth [ZEROPRL]
    ; " ;
```

### Type Rule Examples

- The following example indicates that macro A defines a region of (0,0) to (100, 100) with respect to the placement of that macro, such that the boundary of the above die does not overlap:

```
LAYER TOPDIE
    TYPE MASTERSLICE
    PROPERTY LEF58_TYPE "TYPE ABOVE DIE EDGE ;" ;
END TOPDIE

MACRO A
...
    OBS
        LAYER TOPDIE
        RECT 0.000 0.000 100.000 100.000 ;
    END
...
END A
```

### Area Rule for a Trim Metal Layer

You can create an area rule for the masterslice TRIMMETAL layer by using the following property definition:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_AREA
    "AREA minArea
]; "
```

Where:

*AREA minArea* Specifies the minimum area on the masterslice TRIMMETAL layer.  
*Type:* Float, specified in microns

### **Core Edge Length Rule with Diffusion**

A core edge length rule can be used to indicate that the edge length of a diffusion shape of a standard cell must be greater than or equal to the specified minimum length.

You can create a core edge length rule by using the following property definition:

```
PROPERTY LEF58_COREEDGELENGTH
    "COREEDGELENGTH minLength [MAXLENGTH maxLength]
    [CONCAVECORNER | CONVEXCORNER | MIXEDCORNER]
    [EXTENDTOALIGN width spacing]
; "
```

Where:

[CONCAVECORNER | CONVEXCORNER | MIXEDCORNER]

Specifies that the edge length is checked if the edge is between 2 concave corners in CONCAVECORNER, or if the edge is between two convex corners in CONVEXCORNER, or if the edge is between a concave and a convex corners in MIXEDCORNER. See [Figure 1-100](#) on page 188.

COREEDGELENGTH *minLength*

Specifies that the edge length of the diffusion shape of a CORE (standard) cell, in the direction of the cell row, must be greater than or equal to the *minLength*.  
*Type:* Float, specified in microns

EXTENDTOALIGN *width spacing*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that if the diffusion shapes on the top and bottom edges within a cell row have width exactly equal to *width*, the shapes need to be extended and aligned. In addition, any gap on the same edge that is less than or equal to *spacing* should be filled and merged for length consideration.

*Type:* Float, specified in microns

MAXLENGTH *maxLength*

Specifies that the edge length of the diffusion shape of a CORE (standard) cell, in the direction of the cell row, must be less than or equal to *maxLength*.

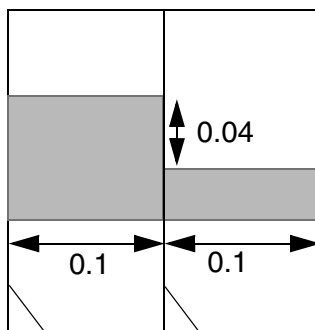
*Type:* Float, specified in microns

### Core Edge Length Rule Examples

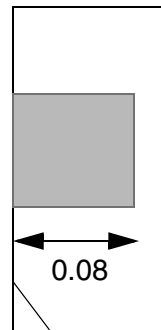
- The following example illustrates the core edge length rule:

```
PROPERTY LEF58_COREEDGELENGTH
"COREEDGELENGTH 0.1 ;
```

**Figure 1-122 Example of Core Edge Length Rule**



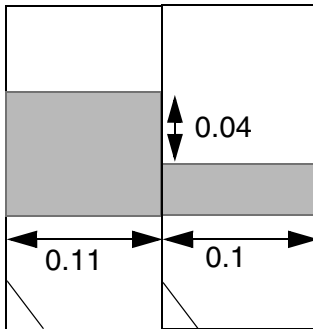
a) OK, the edge length is only checked in the direction of the cell row, which is horizontal.



b) Violation, any horizontal (direction of the cell row) length must be checked.

**Figure 1-123 Example of Core Edge Length Rule**

```
PROPERTY LEF58_COREEDGELENGTH
  "COREEDGELENGTH 0.12 CONVEXCORNER ;
  COREEDGELENGTH 0.1 MIXEDCORNER ; "
```



a) Violation, the 0.1 edge fulfills the MIXEDCORNER requirement ( $\geq 0.1$ ), but the 0.11 edge fails the CONVEXCORNER requirement

**Figure 1-124 Example of Core Edge Length Rule with EXTENDTOALIGN**

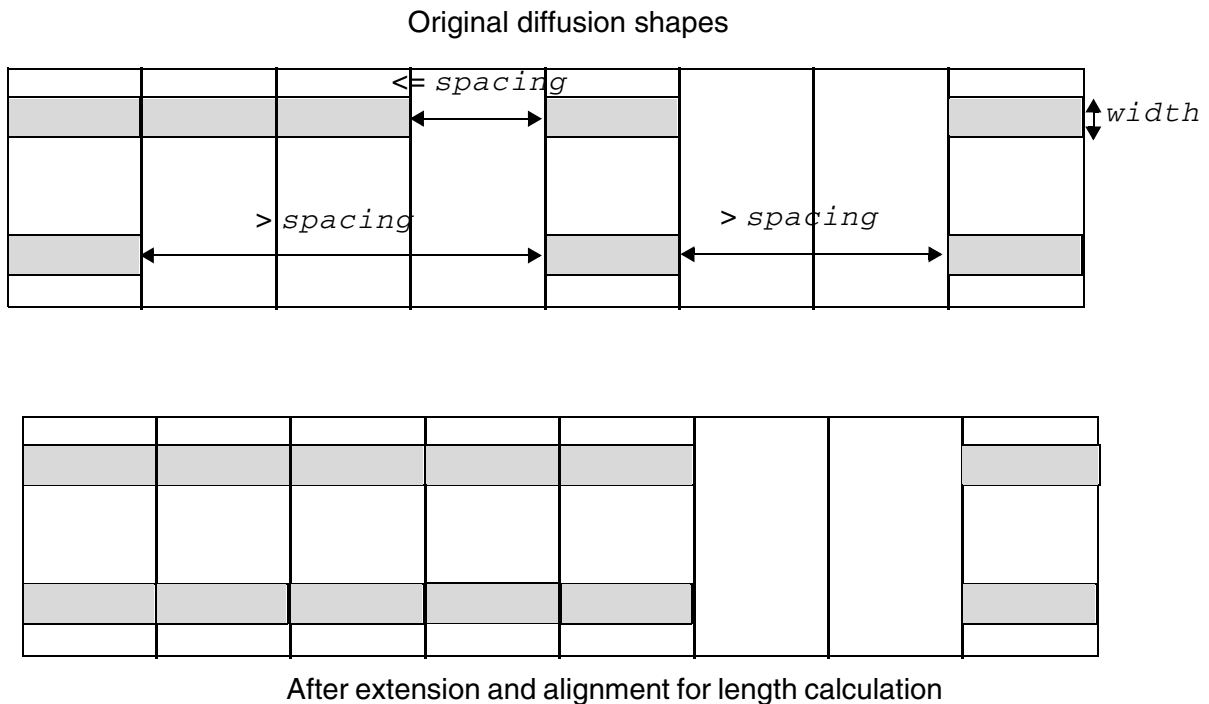


Illustration of COREEDGELENGTH ... EXTENDTOALIGN *width spacing*



#### ***Enclosure Rule for Well Layers***

Enclosure rules can be used to specify that objects on the current well layer must be enclosed by at least overhang amount of objects in the specified second well layer on all the four sides.

You can create an enclosure rule by using the following property definition:

```
PROPERTY LEF58_ENCLOSURE
    "ENCLOSURE overhang LAYER secondLayerName;..." ;
```

Where:

```
ENCLOSURE overhang LAYER secondLayerName
```

Specifies that objects on the current well layer must be enclosed by at least *overhang* amount of objects in the *secondLayerName* well layer on all the four sides. The *secondLayerName* can be any well layer, including forward reference of more than one layer.

This syntax is only allowed on a well masterslice layer with property statement of LEF58\_TYPE "TYPE NWELL ; " or LEF58\_TYPE "TYPE PWELL ; ". This check is optional and is not performed if the enclosing layer does not exist around the well layer.

*Type*: Float, specified in microns

#### **Examples**

- The following example indicates that an enclosure of 0.1  $\mu\text{m}$  is required for objects in PWELLA layer by objects in NWELLA layer:

```
LAYER NWELLA
    TYPE MASTERSLICE
    PROPERTY LEF58_TYPE "TYPE NWELL ;" ;
END NWELLA
LAYER PWELLA
    TYPE MASTERSLICE
    PROPERTY LEF58_TYPE "TYPE PWELL ;" ;
    PROPERTY LEF58_ENCLOSURE "ENCLOSURE 0.1 LAYER NWELLA;" ;
END PWELLA
```

## **Region Rule**

You can create a region rule to define area-based rules on a trim layer.

You can use the following property definition:

```
PROPERTY LEF58_REGION
    "REGION regionLayerName BASEDLAYER trimLayerName
    ; " ;
```

Where:

```
REGION regionLayerName BASEDLAYER trimLayerName
```

Specifies a set of region- or area-based rules on a trim layer *trimLayerName*. *regionLayerName* is a masterslice layer with type REGION. Users can create a blockage on that masterslice layer on a design to indicate the areas of the region. In addition, MACRO may also have OBS on that masterslice layer to indicate that the areas of its instance should be subjected to these region-based rules. All the rules specified on this trim layer with REGION are applied to the corresponding areas of *regionLayerName* on *trimLayerName*. The areas outside *trimLayerName* honor the set of rules on the given trim layer without type REGION. If a rule check crosses the boundaries of a region, it is subjected to the rules for both inside and outside the region, with the tighter rule values of the two sets dominating.

The set of rules defined in the layer with type REGION should be very small as compared to the layer without type REGION. The rules defined in this layer with type REGION should be a replacement of the corresponding rule (having the same set of the constructs of a LEF statement or a complete replacement on a table, like SPACINGTABLE) on the layer without type REGION. All the other rules defined on the layer without type REGION are also applicable on the layer with type REGION.

*Type:* String

## **Region Rule Example**

The following example means that a trim metal completely inside region R1 should have spacing of 0.4. Otherwise, spacing of 0.6 should be applied. Any other rules on layer TM1 (without REGION) would also be applied on region R1:

```
LAYER R1
```

## LEF/DEF 5.8 Language Reference

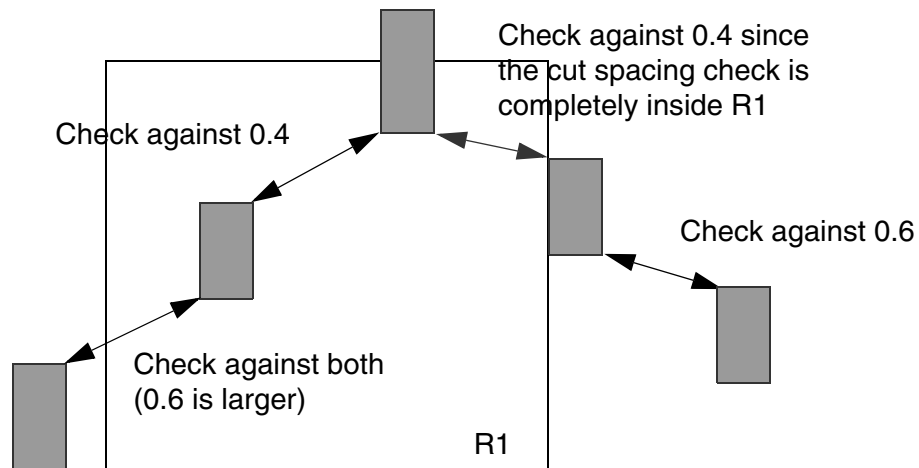
### LEF Syntax

---

```
TYPE MASTERSLICE ;
PROPERTY LEF58_TYPE "TYPE REGION ; " ;
END R1
...
LAYER TM1
TYPE MASTERSLICE ;
PROPERTY LEF58_TYPE "TYPE TRIMMETAL ; " ;
...
PROPERTY LEF58_SPACING "SPACING 0.6 ;" ;
...
END TM1

LAYER TM1R1
TYPE MASTERSLICE ;
PROPERTY LEF58_REGION "REGION R1 BASEDLAYER TM1 ; " ;
...
PROPERTY LEF58_SPACING "SPACING 0.4 ;" ;
...
END TM1R1
```

**Figure 1-125 Illustration of Region Rule**



### ***Spacing Rule for Well Layers***

A spacing rule can be used to specify the minimum spacing allowed between objects on the same well layer or objects on different well layers.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_SPACING
    "SPACING minSpacing [SAMENET | LAYER secondLayerName] ;..." ;
```

Where:

<i>minSpacing</i>	Specifies the default minimum spacing in a well layer. <i>Type:</i> Float, specified in microns
SAMENET	Indicates that the minimum spacing only applies to same-net cuts. The SAMENET <i>minSpacing</i> value should be smaller than the normal cut spacing value that applies to different-net cuts.
LAYER <i>secondLayerName</i>	Applies the spacing rule between objects on this well layer and objects on another previously defined well layer ( <i>secondLayerName</i> ) for specifying an inter-well layer spacing.

#### ***Spacing Rule for ABOVE DIE EDGE, BELOW DIE EDGE, and DIFFUSION Layers***

The following spacing rule can be defined on a masterslice layer for which ABOVE DIE EDGE, BELOW DIE EDGE, or DIFFUSION type rules have been defined:

```
PROPERTY LEF58_SPACING
    "SPACING minSpacing
    ; " ;
```

Where:

SPACING <i>minSpacing</i>	Specifies a spacing between an OBS on the masterslice layer with type ABOVE DIE EDGE, BELOW DIE EDGE, or DIFFUSION to the die boundary of the above or below die to be <i>minSpacing</i> . <i>Type:</i> Float, specified in microns
---------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### ***Spacing Rule for a Trim Metal Layer***

A TRIMMETAL spacing rule can be used to specify the spacing among trim metal shapes.

```
PROPERTY LEF58_SPACING
    "SPACING spacing [PRLSPACING prlSpacing1 prlSpacing2]
    [ENDTOEND endToEndSpacing [PRL prl]
    [EXACTALIGNED exactAlignedSpacing
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[EXCEPTDIFFMASKALIGNED]]  
[EDGEALIGNED edgeAlignedSpacing]  
[EXCEPTMIDMETALWIDTH width]]  
[SAMEMASK]; " ;...
```

Where:

ENDTOEND *endToEndSpacing* [PRL *prl*]

Specifies the end-to-end spacing with parallel run length greater than *prl*, if specified, or 0 in the preferred routing direction on the metal layer that is trimmed.

*Type*: Float, specified in microns

EDGEALIGNED *edgeAlignedSpacing*

Specifies that the end-to-end spacing is *edgeAlignedSpacing* if the opposite long/side edges of the trim metal shapes are exactly aligned. That is the left/top edge of a trim metal shape is exactly aligned to the right/bottom edge of another trim metal. Otherwise, *endToEndSpacing* is applied.

*Type*: Float, specified in microns

EXACTALIGNED *exactAlignedSpacing*

Specifies that the end-to-end spacing is *exactAlignedSpacing* if the short/end side of the trim metal shapes are exactly aligned. Otherwise, *endToEndSpacing* is applied in Euclidean measurement independent of parallel run length if PRL is not defined. If PRL is also defined, *endToEndSpacing* is applied as MAXXY style in the orthogonal direction of the parallel run length.

*Type*: Float, specified in microns

EXCEPTDIFFMASKALIGNED

Specifies that there is no spacing requirement on two same-mask, exactly aligned trims if there is another exactly aligned, different-mask trim between them. In other words, it is legal to have perfectly aligned, consecutive same-mask and different-mask trims. This construct must be specified with SAMEMASK.

EXCEPTMIDMETALWIDTH *width*

Specifies that all the end-to-end spacing rules would be exempted if there is a metal of any mask with width greater than or equal to width on a layer that the trim metal trims.

*Type*: Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**SAMEMASK** Specifies that the spacing values are applied to same-mask objects. It is allowed to have just one spacing statement with **SAMEMASK**, which means that there is no constraint on different-mask shapes.  
*Type:* Float, specified in microns

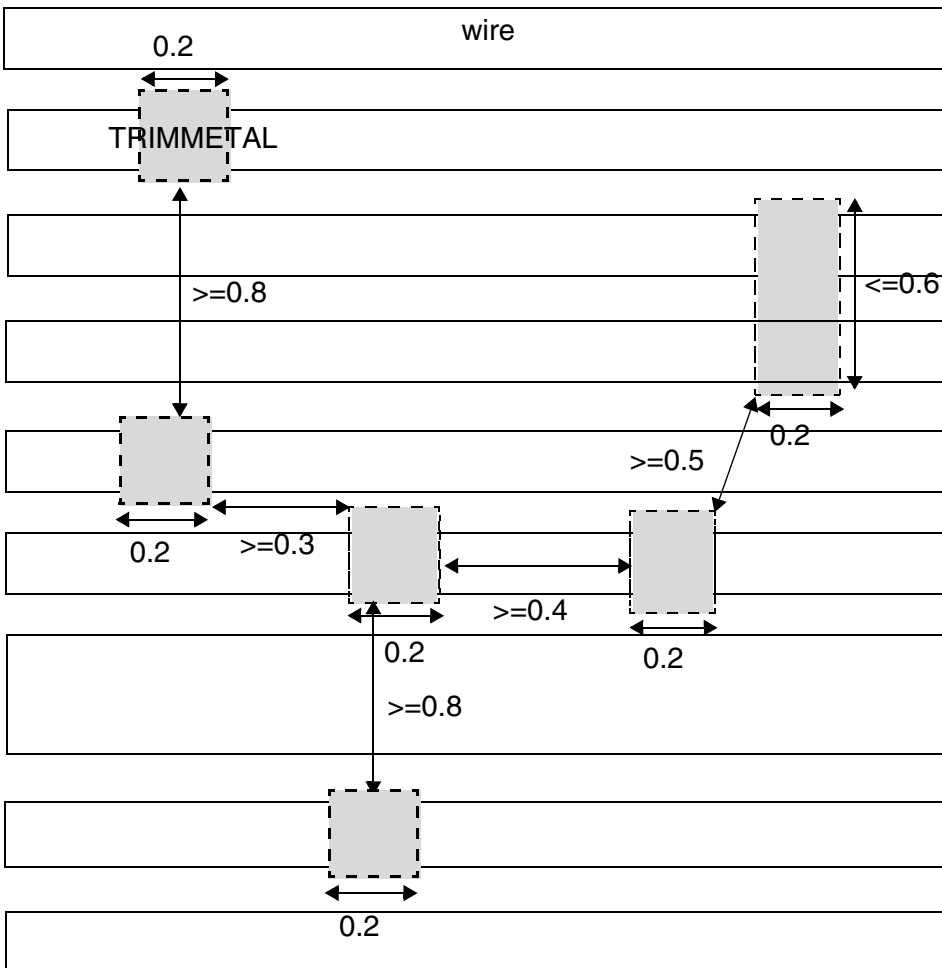
**SPACING** *spacing* [**PRLSPACING** *prlSpacing1 prlSpacing2*]  
Specifies the minimum spacing to be *spacing*. If **PRLSPACING** is specified, *prlSpacing1* is applied with parallel run length equal to 0, *prlSpacing2* with parallel run length greater than 0 and *spacing* with parallel run length less than 0.  
*Type:* Float, specified in microns

### Example of Spacing Rule for a Trim Metal Layer

- The following example illustrates the spacing rule for **TRIMMETAL** layer:

```
# No MASK
PROPERTY LEF58_TRIMSHAPE `
  TRIMSHAPE EXTENSIONMODEL MIDTRACK
  EXACTWIDTH 0.2 MAXLENGTH 0.6 ; ` ;
PROPERTY LEF58_SPACING `
  SPACING 0.5 PRLSPACING 0.3 0.4 ENDTOEND 0.8 ; ` ;
```

**Figure 1-126 Illustration of Spacing Rule for a Trim Metal Layer**



TRIMMETAL shapes (in gray color) must have width of 0.2 & max length of 0.6. Spacing with `PRL > 0` must be  $\geq 0.4$ , `PRL = 0` must be  $\geq 0.3$ , `PRL < 0` must be  $\geq 0.5$  and end to end with `PRL > 0` must be  $\geq 0.8$ .

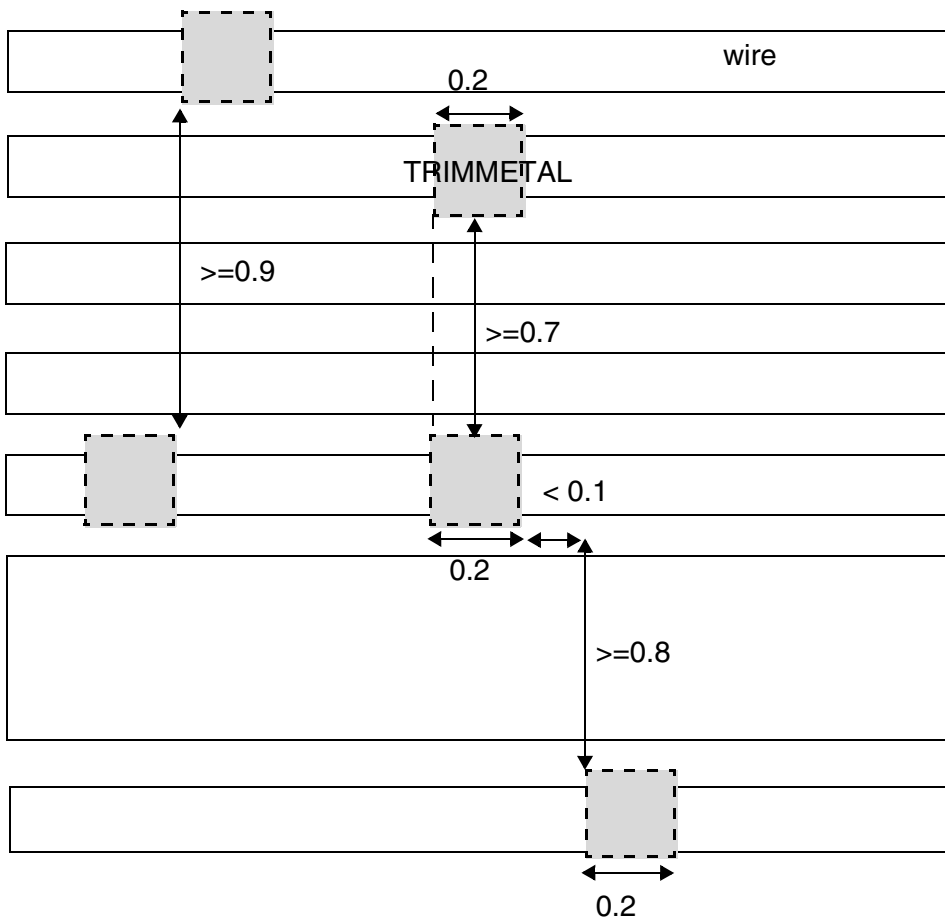
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- The following example illustrates the spacing rule for TRIMMETAL layer with EXACTALIGNED and EDGEALIGNED:

```
# No MASK
PROPERTY LEF58_TRIMSHAPE `
    TRIMSHAPE EXTENSIONMODEL MIDTRACK
        EXACTWIDTH 0.2 ; ` ;
PROPERTY LEF58_SPACING `
    SPACING 0.5 ENDTOEND 0.8 PRL -0.1 EXACTALIGNED 0.7
        EDGEALIGNED 0.9 ; ` ;
```

**Figure 1-127 Illustration of Spacing Rule for a Trim Metal Layer**



- The following example illustrates the spacing rule for TRIMMETAL layer with EXCEPTDIFFMASKALIGNED:

```
PROPERTY LEF58_SPACING `
    SPACING 0.05 ENDTOEND 0.1 EXACTALIGNED 0.15
```



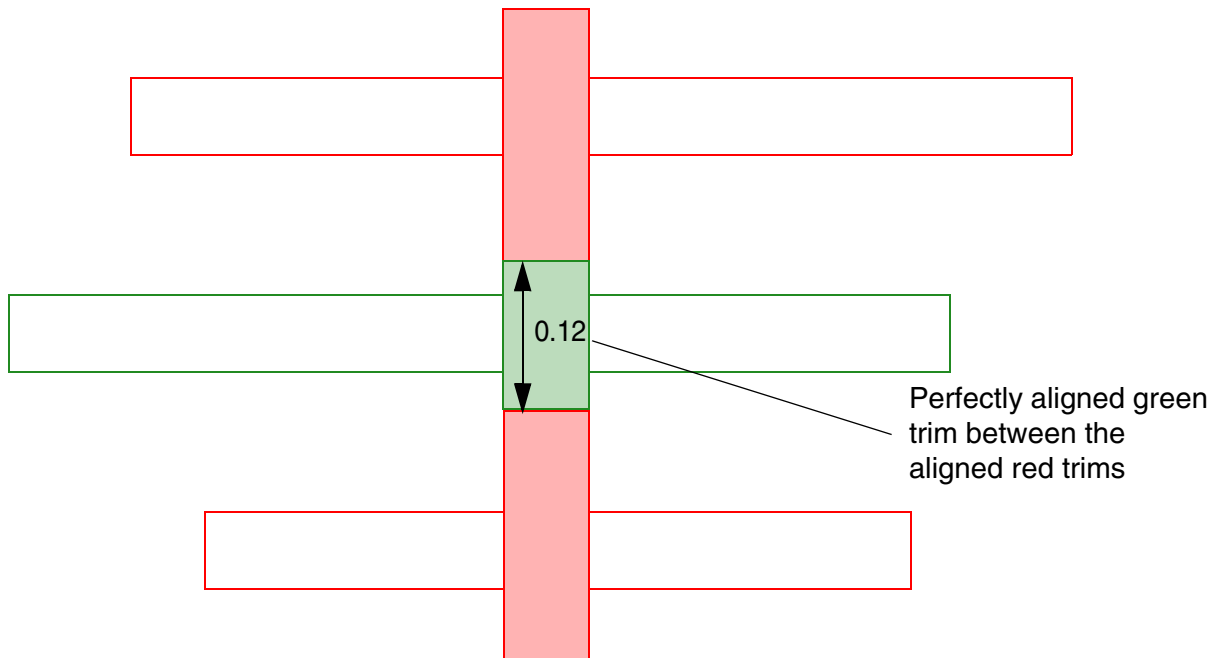
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`EXCEPTDIFFMASKALIGNED SAMEMASK ; " ;`

**Figure 1-128 Illustration of Spacing Rule for a Trim Metal Layer with EXCEPTDIFFMASKALIGNED**



OK, the perfectly aligned red trims need to honor 0.15 spacing, but with the presence of another perfectly aligned green trim between them, this spacing rule is exempted. Violation if `EXCEPTDIFFMASKALIGNED` is omitted.

## LEF/DEF 5.8 Language Reference

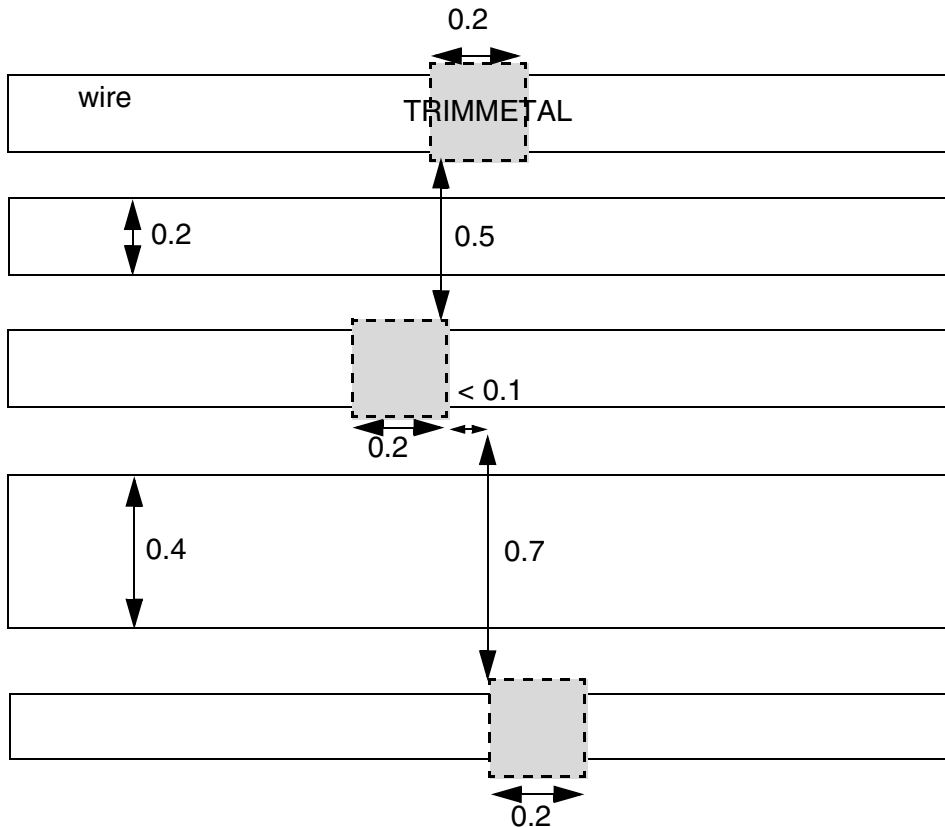
### LEF Syntax

---

- The following example illustrates the spacing rule for `TRIMMETAL` layer with `EXCEPTMIDMETALWIDTH`:

```
# No MASK
PROPERTY LEF58_TRIMSHAPE `
    TRIMSHAPE EXTENSIONMODEL MIDTRACK
    EXACTWIDTH 0.2 ; ` ;
PROPERTY LEF58_SPACING `
    SPACING 0.5 ENDTOEND 0.8 PRL -0.1
    EXCEPTMIDMETALWIDTH 0.4 ; ` ;
```

**Figure 1-129 Illustration of Spacing Rule for a Trim Metal Layer with EXCEPTMIDMETALWIDTH**



The top 0.5 spacing is a violation, but the bottom 0.7 spacing is OK by `EXCEPTMIDMETALWIDTH` exemption.

### ***Trimmed Metal Rule for a Trim Metal Layer***

Trimmed metal rules can be used to specify the metal layer that the shapes on the TRIMMETAL layer tries to trim.

You can create a trimmed metal rule by using the following property definition:

```
PROPERTY LEF58_TRIMMEDMETAL
    "TRIMMEDMETAL metalLayer [MASK maskNum]; "
```

Where:

```
TRIMMEDMETAL metalLayer [MASK maskNum]
```

Specifies the metal layer *metalLayer* that the shapes on the TRIMMETAL layer tries to trim. If *maskNum* is given, only *maskNum* on *metalLayer* is trimmed.  
*Type*: Integer

### **Example of Trimmed Metal Rule**

- The following is an example of a double patterned layer TM1 used to trim both masks of M1. As both TM1 and M1 are double-patterned, and the TRIMMEDMETAL property does not specify a mask, it implies that MASK 1 of TM1 trims MASK 1 of M1, and MASK 2 of TM1 trims MASK 2 of M1.

```
LAYER TM1
    TYPE MASTERSLICE ;
    MASK 2 ;
    PROPERTY LEF58_TYPE "TYPE TRIMMETAL ; " ;
    PROPERTY LEF58_TRIMMEDMETAL "TRIMMEDMETAL M1 ; " ;
    ...
END TM1

...
LAYER M1
    TYPE ROUTING ;
    MASK 2 ;
    ...
END M1
```

- The following example is of a single patterned TRIMMETAL TM2 layer, which trims only one mask of the double-patterned M2 layer. This is indicated by the MASK 1 portion of TM2's TRIMMEDMETAL property:

```
LAYER TM2
    TYPE MASTERSLICE ;
    PROPERTY LEF58_TYPE "TYPE TRIMMETAL ; " ;
    PROPERTY LEF58_TRIMMEDMETAL "TRIMMEDMETAL M2 MASK 1 ; " ;
    ...
END TM2
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
LAYER M2
  TYPE ROUTING ;
  MASK 2 ;
...
END M2
```

#### ***Trim Shape Rule for a Trim Metal Layer***

Trim shape rules can be used to specify that shapes on this layer should be formed based on the line-end wires on the metal layer that is trimmed.

You can create a trim shape rule by using the following property definition:

```
PROPERTY LEF58_TRIMSHAPE
  "TRIMSHAPE EXTENSIONMODEL
  {ADJACENTTRACK | MIDTRACK | EXTENSION extension [METALEDGE]}
  EXACTWIDTH exactWidth
  [MAXLENGTH maxLength [MASK maskNum]]
  [EXCEPTSPACING spacing] [EXCEPTWIDTH width]
  [USEMETALMASK]; "
```

Where:

**MASK *maskNum*** Specifies that the maximum length rule applies on the given mask. *maskNum* must be a positive integer, and most applications support only the values 1, 2, or 3.

*Type:* Integer

**METALEDGE** Specifies that extension is extended from the metal edges that the trim metal trims.

*Type:* Float, specified in microns

```
TRIMSHAPE EXTENSIONMODEL
  {ADJACENTTRACK | MIDTRACK | EXTENSION extension}
  EXACTWIDTH exactWidth [MAXLENGTH maxLength]
  [EXCEPTSPACING spacing] [EXCEPTWIDTH width]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that shapes on this layer should be formed based on the line-end wires on the metal layer (`TRIMMEDMETAL`) that is trimmed. Typically, this trim metal technology would impose a strict 1D on grid routing methodology on `TRIMMEDMETAL`. Tracks in preferred routing direction of `TRIMMEDMETAL` could be non-uniform, but must be homogeneous across the entire design. In addition, wires with a same width are always used on any track.

The shapes must be a rectangle with width measured in the preferred routing layer on `TRIMMEDMETAL` exactly equal to *exactWidth* and length measured in the orthogonal direction less than or equal to *maxLength*, if specified. Shapes are always formed at the line-end of wires on `TRIMMEDMETAL`. The 'width' dimension is always sandwiched between two line-end wires along the preferred direction on `TRIMMEDMETAL`.

With `ADJACENTTRACK` specified in `EXTENSIONMODEL`, the 'height' would extend to the center line of the adjacent tracks on the orthogonal direction.

With `MIDTRACK` specified in `EXTENSIONMODEL`, the 'height' would extend by half of the required spacing of the wires. The shapes could be merged if they are perfectly aligned.

With `EXTENSION` *extension* specified in `EXTENSIONMODEL`, the 'height' of a trim metal would extend on both sides from the center by *extension*.

When the line-end to line-end spacing is greater than or equal to *spacing*, if specified, the shape is not formed.

When the width of line-end is greater than *width*, if specified, the shape is not formed.

`USEMETALMASK`

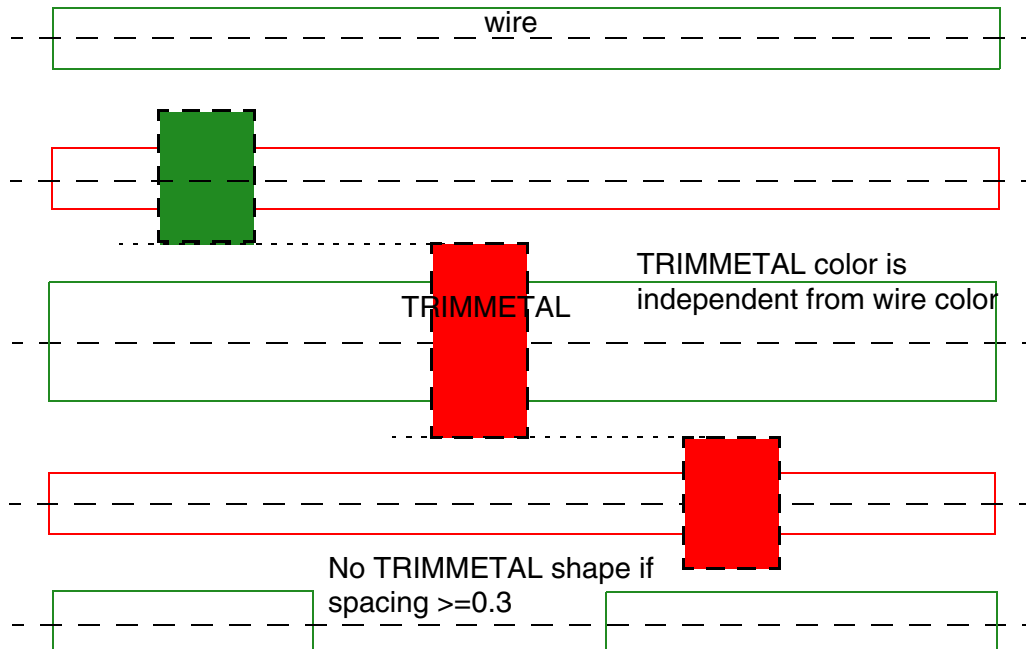
Specifies that the mask of a shape on this layer must follow the mask of a wire on `TRIMMEDMETAL`. This means that *maskNum* on this layer must match the mask number on the `TRIMMEDMETAL`.

## Trim Shape Rule Examples

- The following example is an illustration of TRIMSHAPE rule with MIDTRACK extension model:

```
MASK 2 ;  
PROPERTY LEF58_TRIMSHAPE "  
    TRIMSHAPE EXTENSIONMODEL MIDTRACK  
    EXACTWIDTH 0.2 EXCEPTSPACING 0.3 ; "
```

**Figure 1-130 Illustration of Trim Shape Rule with MIDTRACK Extension Model**



## LEF/DEF 5.8 Language Reference

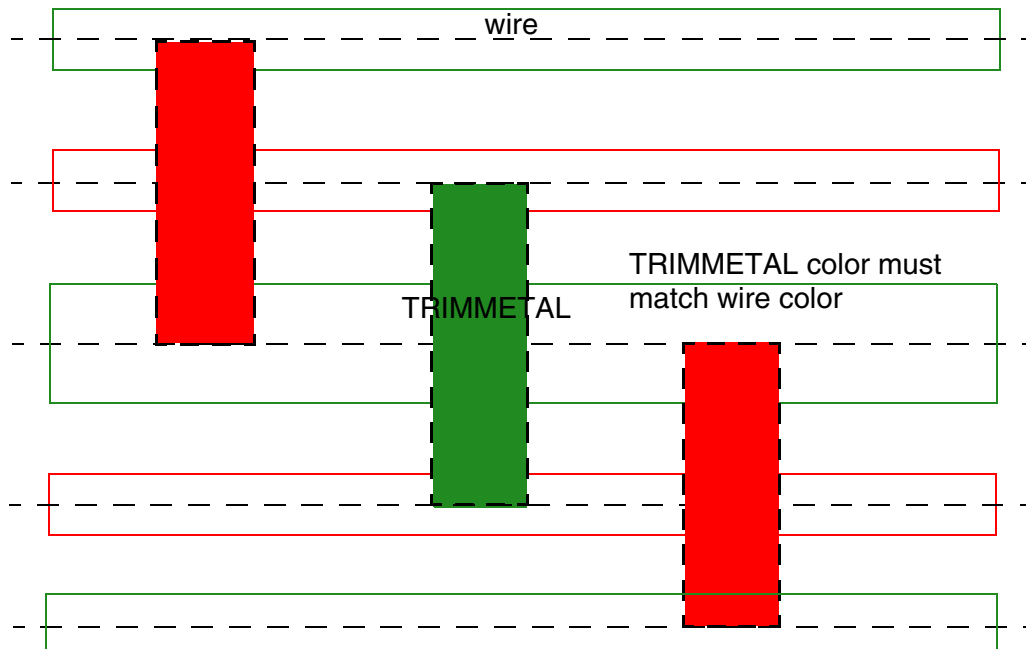
### LEF Syntax

---

- The following example is an illustration of TRIMSHAPE rule with ADJACENTTRACK extension model:

```
MASK 2 ;  
PROPERTY LEF58_TRIMSHAPE "  
    TRIMSHAPE EXTENSIONMODEL ADJACENTTRACK  
    EXACTWIDTH 0.2 USEMETALMASK ; "
```

**Figure 1-131 Illustration of Trim Shape Rule with ADJACENTTRACK Extension Model**



## LEF/DEF 5.8 Language Reference

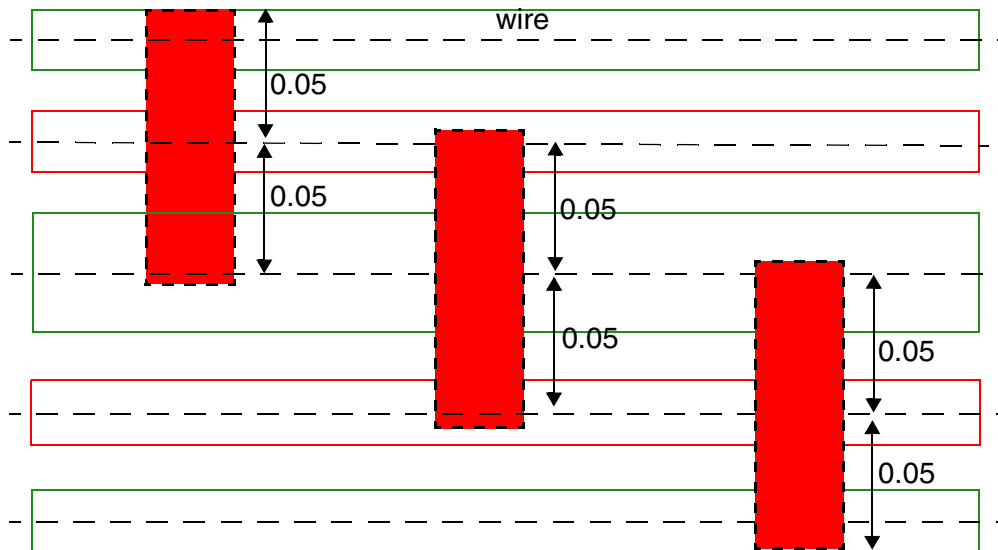
### LEF Syntax

---

- The following example is an illustration of TRIMSHAPE rule with EXTENSION extension model:

```
PROPERTY LEF58_TRIMSHAPE "  
    TRIMSHAPE EXTENSIONMODEL EXTENSION 0.05  
    EXACTWIDTH 0.2 ; "
```

**Figure 1-132 Illustration of Trim Shape Rule with EXTENSION Extension Model**



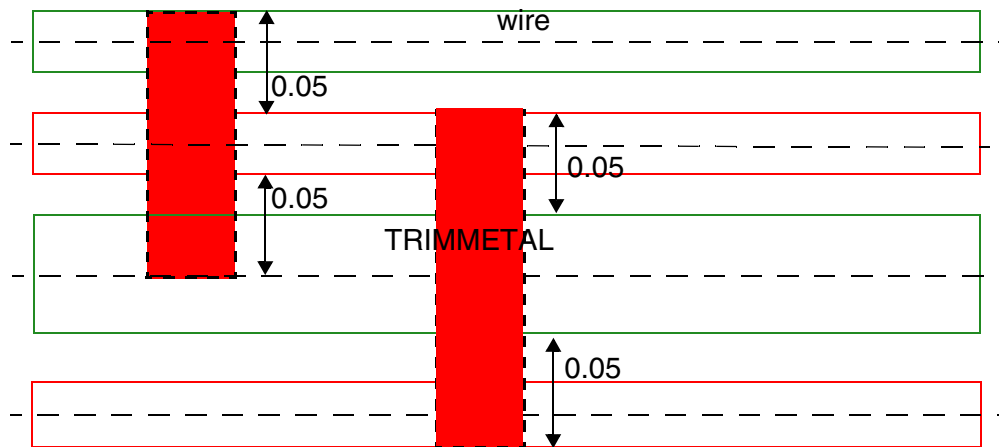
Trim metal is always extended by 0.05 on both ends.



- The following example is an illustration of TRIMSHAPE rule with extension from metal edges:

```
PROPERTY LEF58_TRIMSHAPE `
    TRIMSHAPE EXTENSIONMODEL EXTENSION 0.05
    METAEDGE EXACTWIDTH 0.2 ; ` ;
```

**Figure 1-133 Example of Trim Shape Rule with Extension from Metal Edges**



Trim metal is always extended by 0.05 on metal edges.

### **Width Rule for a TRIMPOLY Layer**

A width rule can be used to specify a minimum width for a TRIMPOLY layer.

You can create a width rule by using the following property definition:

```
PROPERTY LEF58_WIDTH
    "WIDTH minWidth [ZEROPRL]
    ;" ;
```

Where:

*WIDTH minWidth*

Specifies the minimum width for the TRIMPOLY layer.

*ZEROPRL*

Specifies that the width requirement applies only for two TRIMPOLY shapes on adjacent rows with zero parallel run length in the direction perpendicular to the row direction.

### ***Width Rule for a Well Layer***

A width rule can be used to specify a default width for a well layer.

You can create a width rule by using the following property definition:

```
PROPERTY LEF58_WIDTH  
    "WIDTH minWidth;" ;
```

Where:

*WIDTH minWidth*

Specifies the default width for the well layer.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## Layer (Routing)

```
LAYER layerName
    TYPE ROUTING ;
    [MASK maskNum ;]
    DIRECTION {HORIZONTAL | VERTICAL | DIAG45 | DIAG135} ;
    PITCH {distance | xDistance yDistance} ;
    [DIAGPITCH {distance | diag45Distance diag135Distance} ;]
    WIDTH defaultWidth ;
    [OFFSET {distance | xDistance yDistance} ;]
    [DIAGWIDTH diagWidth ;]
    [DIAGSPACING diagSpacing ;]
    [DIAGMINEDGELENGTH diagLength ;]
    [AREA minArea ;]
    [MINSIZE minWidth minLength [minWidth2 minLength2] ... ;]
    [[SPACING minSpacing
        [ RANGE minWidth maxWidth
            [ USELENGTHTHRESHOLD
                | INFLUENCE value [RANGE stubMinWidth stubMaxWidth]
                | RANGE minWidth maxWidth]
            | LENGTHTHRESHOLD maxLength [RANGE minWidth maxWidth]
            | ENDOFLINE eolWidth WITHIN eolWithin
                [PARALLELEDGE parSpace WITHIN parWithin [TWOEDGES]]
            | SAMENET [PGONLY]
            | NOTCHLENGTH minNotchLength
            | ENDOFNOTCHWIDTH endOfNotchWidth NOTCHSPACING minNotchSpacing
                NOTCHLENGTH minNotchLength
        ]
    ;] ...
    [SPACINGTABLE
        [PARALLELRUNLENGTH {length} ...
            {WIDTH width {spacing} ...} ... ;
        [SPACINGTABLE
            INFLUENCE {WIDTH width WITHIN distance SPACING spacing} ... ;]
        |TWOWIDTHS {WIDTH width [PRL runLength] {spacing} ...} ... ;
    ]
    ;" ;]
    [WIREEXTENSION value ;]
    [MINIMUMCUT numCuts WIDTH width [WITHIN cutDistance]
        [FROMABOVE | FROMBELOW]
        [LENGTH length WITHIN distance] ;] ...
    [MAXWIDTH width ;]
    [MINWIDTH width ;]
    [MINSTEP minStepLength
        [ [INSIDECORNER | OUTSIDECORNER | STEP] [LENGTHSUM maxLength]
        | [MAXEDGES maxEdges] ;]
    [MINENCLOSEDAREA area [WIDTH width] ;] ...
    [PROTRUSIONWIDTH width1 LENGTH length WIDTH width2 ;]
    [RESISTANCE RPERSQ value ;]
    [CAPACITANCE CPERSQDIST value ;]
    [HEIGHT distance ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[THICKNESS distance ;]
[SHRINKAGE distance ;]
[CAPMULTIPLIER value ;]
[EDGECAPACITANCE value ;]
[MINIMUMDENSITY minDensity ;]
[MAXIMUMDENSITY maxDensity ;]
[DENSITYCHECKWINDOW windowLength windowWidth ;]
[DENSITYCHECKSTEP stepValue ;]
[FILLACTIVESPACING spacing ;]
[ANTENNAMODEL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4} ;] ...
[ANTENNAAREARATIO value ;] ...
[ANTENNADIFFAREARATIO {value | PWL ( ( d1 r1 ) ( d2 r2 ) ... ) } ;] ...
[ANTENNACUMAREARATIO value ;] ...
[ANTENNACUMDIFFAREARATIO {value | PWL ( ( d1 r1 ) ( d2 r2 ) ... ) } ;] ...
[ANTENNAAREAFACOR value [DIFFUSEONLY] ;] ...
[ANTENNASIDEAREARATIO value ;] ...
[ANTENNADIFFSIDEAREARATIO {value | PWL ( ( d1 r1 ) ( d2 r2 ) ... ) } ;] ...
[ANTENNACUMSIDEAREARATIO value ;] ...
[ANTENNACUMDIFFSIDEAREARATIO {value | PWL ( ( d1 r1 ) ( d2 r2 ) ... ) } ;] ...
[ANTENNASIDEAREAFACOR value [DIFFUSEONLY] ;] ...
[ANTENNACUMROUTINGPLUSCUT ;]
[ANTENNAGATEPLUSDIFF plusDiffFactor ;]
[ANTENNAAREAMINUSDIFF minusDiffFactor ;]
[ANTENNAAREADIFFREDUCEPWL ( ( diffArea1 diffMetalFactor1 )
    ( diffArea2 diffMetalFactor2 ) ... ) ;]
[PROPERTY propName propVal ;] ...
[ACCURRENTDENSITY {PEAK | AVERAGE | RMS}
    { value
    | FREQUENCY freq_1 freq_2 ... ;
    [WIDTH width_1 width_2 ... ;]
    TABLEENTRIES
        v_freq_1_width_1 v_freq_1_width_2 ...
        v_freq_2_width_1 v_freq_2_width_2 ...
        ...
    } ;]
[DCCURRENTDENSITY AVERAGE
    { value
    | WIDTH width_1 width_2 ... ;
    TABLEENTRIES value_1 value_2 ...
    } ;]
[PROPERTY LEF58 ACCURRENTDENSITY
    "ACCURRENTDENSITY {PEAK | AVERAGE | RMS}
        [TEMPPWL temp_1 multi_1 temp_2 multi_2 ...]
        [HOURSPWL hours_1 multi_1 hours_2 multi_2 ...]
        ; " ;]
[PROPERTY LEF58 ANTENNAGATEPWL
    "ANTENNAGATEPWL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4
        ((gateArea1 effectiveGateArea1) (gateArea2 effectiveGateArea2)... )
        ;" ;]
[PROPERTY LEF58 ANTENNAGATEPLUSDIFF
    "ANTENNAGATEPLUSDIFF {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
{plusDiffFactor |
PWL ((diffArea1 plusDiffProtect1) (diffArea1 plusDiffProtect2)...)}
;" ;]

[PROPERTY LEF58 AREA
"AREA minArea
[MASK maskNum]
[EXCEPTMINWIDTH minWidth
| [EXCEPTEDGELENGTH {minEdgeLength maxEdgeLength | edgeLength}]
[EXCEPTMINSIZE {minWidth minLength}...]
[EXCEPTSTEP length1 length2]
| RECTWIDTH rectWidth
| EXCEPTRECTANGLE
| LAYER trimLayer OVERLAP {1 | 2}
]
;..." ;]

[PROPERTY LEF58 BACKSIDE
"BACKSIDE ;" ;]

[PROPERTY LEF58 BOUNDARYEOLBLOCKAGE
"BOUNDARYEOLBLOCKAGE size OFFSET offset
;" ;]

[PROPERTY LEF58 CORNEREOLKEEPOUT
"CORNEREOLKEEPOUT WIDTH eolWidth EOLSPACING eolSpacing
{ SPACING spacing1 spacing2 WITHIN within1 within2
| EXTENSION backwardExt sideExt forwardExt }
;" ;]

[PROPERTY LEF58 CORNERFILLSPACING
"CORNERFILLSPACING spacing EDGELENGTH length1 length2
ADJACENTEOL eolWidth
;" ;]

[PROPERTY LEF58 CORNERSPACING
"CORNERSPACING
{CONVEXCORNER [SAMEMASK]
[CORNERONLY within]
[EXCEPTEOL eolWidth
[EXCEPTJOGLLENGTH length [EDGELENGTH]
[INCLUDELSHAPE]]]
|CONCAVECORNER
[MINLENGTH minLength] [EXCEPTNOTCH [notchLength]] }
[EXCEPTSAMENET | EXCEPTSAMEMETAL]
{WIDTH width SPACING spacing
| WIDTH width SPACING horizontalSpacing verticalSpacing}...
;" ;]

[PROPERTY LEF58 ENCLOSURESPACING
"ENCLOSURESPACING
[CUTCLASS cutClass [LONGEDGEONLY] ]
[FROMABOVE | FROMBELOW]
{ENCLOSURE enclosure SPACING spacing}...
;" ;]

[PROPERTY LEF58 EOEXTENSIONSPACING
"EOEXTENSIONSPACING spacing [SAMEMASK]
[PARALLELOLONLY] [NONEOL]]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[OTHERWIDTH otherWidth]
  {ENDOFFLINE eolWidth EXTENSION extension
    [ENDTOEND endToEndExtension]} ...
[MINLENGTH minLength [TWO SIDES]]
  ; " ;]

[PROPERTY LEF58 EOLKEEPOUT
  "EOLKEEPOUT {eolWidth | minEolWidth maxEolWidth}
EXTENSION backwardExt sideExt forwardExt
  [EXCEPTWITHIN lowSideExt highSideExt]
  [CLASS className [OTHER ENDEOL]]
  [CORNERONLY]
    [EXCEPTFROM { BACKEDGE [FORWARDGAP forwardGap]
      | FRONTEDGE }
    [MASK maskNum [TWO SIDES]]]]
  [EXCEPTS AMEMETAL]
  ; ... " ;]

[PROPERTY LEF58 FILLTOFILLSPACING
  "FILLTOFILLSPACING spacing ;" ;]

[PROPERTY LEF58 FIVEWIRESEOLSPACING
  "FIVEWIRESEOLSPACING eolSpacing WITHIN eolWithin
    PRL prl
    ENCLOSECUT {BELOW|ABOVE} encloseDist CUTWITHIN cutWithin
    NOMETALEOLEXTENSION eolSideExtension eolForwradExtension
  ; ... " ;]

[PROPERTY LEF58 FORBIDDEN SPACING
  "FORBIDDEN SPACING minSpacing maxSpacing [minSpacing2 maxSpacing2]
    [WIDTH minWidth WITHIN within PRL prl
    | [SAMEMASK] WIDTH maxWidth PRL prl
      [TWO EDGES within | EXACTSPACINGEDGE exactSpacing
        [SPANLENGTH spanLength [WITHIN within]]]
    | EXACTWIDTH exactWidth PRL prl
      OTHERWIDTH otherWidth EXACTSPACINGEDGE exactSpacing
    | [SAMEMASK | MASK maskNum]
      WIDTHRANGE minWidth maxWidth PRL prl
      OTHERWIDTH otherWidth WITHIN within [OTHERS AMEMASK]]
  ; " ;]

[PROPERTY LEF58 GAP
  "GAP EXACTWIDTH exactWidth MAXLENGTH maxLength
    SPACING {{0 | 1 | 2} spacing}...[ENDTOEND endToEndSpacing]
  ; " ;]

[PROPERTY LEF58 JOINTCORNERSPACING
  "JOINTCORNERSPACING spacing [SAMEMASK]
    JOINTWIDTH jointWidth [MINLENGTH minLength]
    JOINTLENGTH spanLength [EDGELENGTH edgeLength]
  ; " ;]

[PROPERTY LEF58 LITHOMACROHALO
  "LITHOMACROHALO horizontalHalo verticalHalo WIRES PACING wireSpacing
  ; " ;]

[PROPERTY LEF58 MANUFACTURINGGRID
  "MANUFACTURINGGRID value
  ; " ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### [PROPERTY LEF58 MINIMUMCUT

```
"MINIMUMCUT {numCuts | {CUTCLASS className numCuts} ... }  
  WIDTH width [WITHIN cutDistance]  
  [FROMABOVE | FROMBELOW]  
  [LENGTH length WITHIN distance  
  | AREA area [WITHIN distance]  
  | SAMEMETALOVERLAP  
  | FULLYENCLOSED]  
  ;..." ;]
```

#### [PROPERTY LEF58 MINSIZE

```
"MINSIZE [RECTONLY] minWidth minLength [minWidth minLength]...  
  ;..." ;]
```

#### [PROPERTY LEF58 MINSTEP

```
"MINSTEP minStepLength  
  [[INSIDECORNER | OUTSIDECORNER | STEP] [LENGTHSUM maxLength]]  
  [MAXEDGES maxEdges]  
  [ MINADJACENTLENGTH minAdjLength  
    [CONVEXCORNER [EXCEPTWITHIN exceptWithin]  
    | CONCAVECORNER  
    | THREECONCAVECORNERS [CENTERWIDTH width]  
    | minAdjLength2]  
  | MINBETWEENLENGTH minBetweenLength  
    [EXCEPTSAMECORNERS]  
  | NOADJACENTEOL eolWidth]  
    [EXCEPTADJACENTLENGTH minAdjLength]  
    [MINADJACENTLENGTH minAdjLength]  
    [CONCAVECORNERS]  
  | NOBETWEENEOL eolWidth]  
  ] ;..." ;]
```

#### [PROPERTY LEF58 OPPOSITEEOLSPACING

```
"OPPOSITEEOLSPACING [SAMEMASK] WIDTH width  
  ENDWIDTH eolWidth [MINLENGTH minLength]  
  [JOINTWIDTH jointWidth] JOINTLENGTH spanLength  
    {[JOINTTOEDGEEND jointToEdgeEndLength]  
    [JOINTEXTENSION jointExtension]  
    [JOINTCORNERONLY]  
  [SIDELENGTH length [TOSIDE toSideLength]]  
  [SIDEEXTENSION sideExtension [TOSIDE toSideExtension] ]  
  [SIDEEDGELENGTH sideEdgeLength]  
  [PRL prl]  
  {[EXCEPTEDGELENGTH edgeLength [PRL maxPRL]]}...  
  ENDTOEND endSpacing endSpacing [PRL individualPrl]  
  ENDTOJOINT endSpacing jointSpacing [PRL individualPrl]  
  JOINTTOEND jointSpacing endSpacing [PRL individualPrl]  
  JOINTTOJOINT jointSpacing jointSpacing [PRL individualPrl]  
  [SIDETOEND sideSpacing endSpacing [PRL individualPrl]  
  SIDETOJOINT sideSpacing jointSpacing] [PRL individualPrl]  
  [JOINTTOSIDE jointSpacing sideSpacing] [PRL individualPrl]  
  [SIDETOSIDE sideSpacing sideSpacing] [PRL individualPrl]  
  ;" ;]
```

#### [PROPERTY LEF58 PITCH

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
"PITCH {distance | xDistance yDistance} [FIRSTLASTPITCH firstLastPitch]
; " ;]

[PROPERTY LEF58 PROTRUSIONWIDTH
  "PROTRUSIONWIDTH width1
    {LENGTH length WIDTH width2
    | {WIDTH width2
      {MINSIZE {minWidth minLength | minLength
        CUTCLASS className {FROMABOVE | FROMBELOW}}
      | MINLENGTH minLength
        [EXCEPTCUT cutDistance
          [FROMABOVE | FROMBELOW]]}...}
  ; " ;]

[PROPERTY LEF58 RECTONLY
  "RECTONLY [EXCEPTNONCOREPINS]
; " ;]

[PROPERTY LEF58 REGION
  "REGION regionLayerName BASEDLAYER routingLayerName
; " ;]

[PROPERTY LEF58 RIGHTWAYONGRIDONLY
  "RIGHTWAYONGRIDONLY [CHECKMASK]
; " ;]

[PROPERTY LEF58 SPACING
  "SPACING eolSpace EOLPERPENDICULAR eolWidth perWidth ;" ;]

[PROPERTY LEF58 SPACING
  "SPACING minSpacing AREA maxArea ;" ;]

[PROPERTY LEF58 SPACING
  "SPACING minSpacing LAYER trimLayer [TRIMMASK trimMaskNum]
    [MASK maskNum] EXCEPTOVERLAP overlapLength
    [TRIMENDSPACING trimEndSpacing]
; " ;]

[PROPERTY LEF58 SPACING
  "SPACING minSpacing SAMEMASK ;" ;]

[PROPERTY LEF58 SPACING
  "SPACING minSpacing [WRONGDIRECTION [NONEOL eolWidth]
    [PRL prl] [LENGTH length]]
; " ;]

[PROPERTY LEF58 SPACING
  "SPACING minSpacing NOTCHLENGTH minNotchLength
    [EXCEPTWITHIN lowExcludeSpacing highExcludeSpacing]
    [WITHIN within SPANLENGTH sideOfNotchSpanLength
    | {WIDTH | CONCAVEENDS} sideOfNotchWidth
    | NOTCHWIDTH notchwidth]
; " ;]

[PROPERTY LEF58 SPACING
  "SPACING minSpacing NOTCHSPAN span NOTCHSPACING notchSpacing
    EXCEPTNOTCHLENGTH notchLength
; " ;]

[PROPERTY LEF58 SPACING
  "SPACING eolSpace
    ENDOFLINE eolWidth
    {[EXACTWIDTH]
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[WRONGDIRSPACING wrongDirSpace]
{[OPPOSITEWIDTH oppositeWidth]
  WITHIN eolWithin [wrongDirWithin]
  [SAMEMASK]
  [EXCEPTEXACTWIDTH exactWidth otherWidth]
  [FILLCONCAVECORNER fillTriangle]
  [WITHCUT [CUTCLASS cutClass] [ABOVE] withCutSpace
    [ENCLOSUREEND enclosureEndWidth
      [WITHIN enclosureEndWithin]]]
  [ENDPRLSPACING endPrlSpace PRL endPrl]
  [ENDTOEND endToEndSpace [oneCutSpace twoCutSpace]
    [EXTENSION extension [wrongDirExtension]]
    [OTHERENDWIDTH otherEndWidth]]
  [MAXLENGTH maxLength | MINLENGTH minLength [TWO SIDES]]
  [EQUALRECTWIDTH]
  [PARALLELEDGE [SUBTRACTEOLWIDTH] parSpace
    WITHIN parWithin [PRL prl]
    [MINLENGTH minLength] [TWO EDGES]
    [SAMEMETAL] [NONEOLCORNERONLY]
    [PARALLELSAMEMASK]]
  [ENCLOSECUT [BELOW | ABOVE] encloseDist
    CUTSPACING cutToMetalSpace [ALLCUTS]]
  | TOCONCAVECORNER [MINLENGTH minLength]
    [MINADJACENTLENGTH
      {minAdjLength | minAdjLength1 minAdjLength2}]
  | TONOTCHLENGTH notchLength}
;..." ;]

[PROPERTY LEF58 SPACING
  "SPACING spacing CONVEXCORNERS EXTENSION sideExt orthogonalExt
    [ SAMESIDE extension
      | SINGLE edgeForwardExt cornerForwardExt cornerBackwardExt
        SPANLENGTH spanLength OPPOSITEWIDTH oppWidth
        OPPOSITEEXTENSION oppSideExt
        oppForwardExt1 oppForwardExt2 ]
    ; " ;]

[PROPERTY LEF58 SPACINGTABLE
  "SPACINGTABLE
    [PARALLELRUNLENGTH [WRONGDIRECTION] [SAMEMASK]
      [EXCEPTEOL eolWidth]
      {length} ...
      {WIDTH width
        [EXCEPTWITHIN lowExcludeSpacing highExcludeSpacing]
        {spacing}...} ... ;
      [SPACINGTABLE
        INFLUENCE {WIDTH width WITHIN distance SPACING spacing} ... ;]
      |TWO WIDTHS {WIDTH width [PRL runLength] {spacing} ...} ... ;
      |PARALLELSPANLENGTH PRL runLength {SPANLENGTH spanLength {spacing} ...
      ]
    ;" ;]

[PROPERTY LEF58 SPACINGTABLE
  "SPACINGTABLE JOGTOJOGSPACING jogToJogSpacing
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
JOGWIDTH jogWidth SHORTJOGSPACING shortJogSpacing
  {WIDTH width PARALLEL parLength WITHIN parWithin
   [EXCEPTWITHIN lowExcludeSpacing highExcludeSpacing]
   LONGJOGSPACING longJogSpacing
   [SHORTJOGSPACING widthShortJogSpacing]}
```

]; " ;

[PROPERTY LEF58 SPACINGTABLE

"SPACINGTABLE

DIRECTIONALSPANLENGTH [WRONGDIRECTION]

[SAMEMASK]

[EXCEPTJOGLENGTH *length* [EDGELENGTH] [INCLUDELSHAPE]]

[EXCEPTEOL *eolWidth*]

[EXACTSPANLENGTHSPACING *spanLength1* TO *spanLength2*

[PRL *prl*] {*exactSpacing*}...]...

PRL {*prl*} ...

{SPANLENGTH *spanLength*

[EXACTSPACING *exactSpacing*

| SPACINGTOMINSPAN *spacingToMinSpan*]

[EXACTSELFSPACING *exactSelfSpacing*]

[NOEXCEPTEOL|EOLSPACING *eolspacing*] {*spacing*}...]...

]; " ;

[PROPERTY LEF58 SPACINGTABLE

"SPACINGTABLE

TWOWIDTHS [WRONGDIRECTION]

[SAMEMASK | MASK *maskNum*]

[EXCEPTEOL *eolWidth*]

[FILLCONCAVECORNER *fillTriangle*]

{WIDTH *width* [PRL *runLength*]{*spacing*} ...} ...

]; " ;

[PROPERTY LEF58 SPANLENGTHENCLOSURESPACING

"SPANLENGTHENCLOSURESPACING *spacing*

SPANLENGTH *spanLength* MINLENGTH *minLength*

{FROMABOVE|FROMBELOW} ENCLOSURE *enclosure*

PRL *prl*

; " ;]

[PROPERTY LEF58 SPANLENGTHTABLE

"SPANLENGTHTABLE {*spanLength*}... [MASK *maskNum*] [WRONGDIRECTION]

[ORTHOGONAL *length*]

[EXCEPTOTHERSPAN *otherSpanlength*

| {OTHERSPAN *otherSpanLength*

[MINSPANLENGTH *minSpanLength*}...]...

; " ;]

[PROPERTY LEF58 TWOWIRESFORBIDDENSPPACING

"TWOWIRESFORBIDDENSPPACING [SAMEMASK | MASK *maskNum*]

{*minSpacing maxSpacing*}...

[HORIZONTAL|VERTICAL]

{ MINSPANLENGTH *minSpanLength* [EXACTSPANLENGTH]

MAXSPANLENGTH *maxSpanLength* [EXACTSPANLENGTH]

| FIRSTSPANLENGTH *spanLength1 spanLength2*

[OTHERSPANLENGTH *otherSpanLength1*]

SECONDSPANLENGTH *spanLength3 spanLength4*}

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
        [OTHERSPANLENGTH otherSpanLength2]  
    PRL prl  
    ; " ;]  
[PROPERTY LEF58 TYPE  
    "TYPE {POLYROUTING | MIMCAP | HIGHR } ] ;"  
[PROPERTY LEF58 VOLTAGESPACING  
    "VOLTAGESPACING [TOCUT [ABOVE | BELOW]]  
        {voltage spacing} ...  
    ;..." ;]  
[PROPERTY LEF58 WIDTHTABLE  
    "WIDTHTABLE {width}...[WRONGDIRECTION] [ORTHOGONAL]  
    ] ;"  
[PROPERTY LEF58 WIDTH  
    "WIDTH minWidth [WRONGDIRECTION]  
    ;" ;]  
  
END layerName
```

Defines routing layers in the design. Each layer is defined by assigning it a name and design rules. You must define routing layers separately, with their own layer statements.

You must define layers in process order from bottom to top. For example:

```
poly        masterslice  
cut01       cut  
metal1      routing  
cut12       cut  
metal2      routing  
cut23       cut  
metal3      routing
```

#### ACCURRENTDENSITY

Specifies how much AC current a wire on this layer of a certain width can handle at a certain frequency in units of milliamps per micron (mA/ $\mu\text{m}$ ).

**Note:** The true meaning of current density would have units of milliamps per square micron (mA/ $\mu\text{m}^2$ ); however, the thickness of the metal layer is implicitly included, so the units in this table are milliamps per micron, where only the wire width varies.

The ACCURRENTDENSITY syntax is defined as follows:

```
{PEAK | AVERAGE | RMS}  
{ value  
| FREQUENCY freq_1 freq_2 ... ;  
  [WIDTH width_1 width_2 ... ;]  
  TABLEENTRIES  
    v_freq_1_width_1 v_freq_1_width_2 ...  
    v_freq_2_width_1 v_freq_2_width_2 ...
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

} ; . . .

PEAK	Specifies the peak current limit of the layer.
AVERAGE	Specifies the average current limit of the layer.
RMS	Specifies the root mean square current limit of the layer.
<i>value</i>	Specifies a maximum current for the layer in mA/μm. <i>Type:</i> Float
FREQUENCY	<p>Specifies frequency values, in megahertz. You can specify more than one frequency. If you specify multiple frequency values, the values must be specified in ascending order.</p> <p>If you specify only one frequency value, there is no frequency dependency, and the table entries are assumed to apply to all frequencies.</p> <p><i>Type:</i> Float</p>
WIDTH	<p>Specifies wire width values, in microns. You can specify more than one wire width. If you specify multiple width values, the values must be specified in ascending order.</p> <p>If you specify only one width value, there is no width dependency, and the table entries are assumed to apply to all widths.</p> <p><i>Type:</i> Float</p>
TABLEENTRIES	<p>Defines the maximum current for each of the frequency and width pairs specified in the FREQUENCY and WIDTH statements, in mA/μm.</p> <p>The pairings define each width for the first frequency in the FREQUENCY statement, then the widths for the second frequency, and so on.</p> <p>The final value for a given wire width and frequency is computed from a linear interpolation of the table values. the widths are not adjusted for any process shrinkage, so the should be correct for the “drawn width”.</p> <p><i>Type:</i> Float</p>

#### Example 1-9 AC Current Density Statements

Most LEF files do not include PEAK or AVERAGE limits. The PEAK limits are not a practical problem for digital signal routing. The AVERAGE limits are only needed for DC limits and not AC currents.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Most technologies do not have frequency dependency for RMS limits, but the LEF syntax requires a frequency value, so in practice the frequency value is a single value of 1, as shown in the example below. In this case the RMS limit does not vary with the frequency.

The following examples define AC current density tables:

The RMS current density at 0.7  $\mu\text{m}$  is  $9.0 + (7.5 - 9.0) \times (0.8 - 0.7) / (0.8 - 0.4) = 8.625 \text{ mA}/\mu\text{m}$  at frequency 300Mhz. Therefore, a 0.7  $\mu\text{m}$  wide wire can carry  $8.625 \times 0.7 = 6.035 \text{ mA}$  of RMS current.

The RMS current density at 0.7  $\mu\text{m}$  is  $7.5 + (6.8 - 7.5) \times (0.8 - 0.7) / (0.8 - 0.4) = 7.325 \text{ mA}/\mu\text{m}$  at frequency 600Mhz. Therefore, a 0.7  $\mu\text{m}$  wide wire can carry  $7.325 \times 0.7 = 5.1275 \text{ mA}$  of RMS current.

```
LAYER met1
...
ACCURRENTDENSITY PEAK      #peak AC current limit for met1
FREQUENCY 100 400 ;        #2 freq values in MHz
WIDTH
0.4 0.8 1.6 5.0 10.0 ;     #5 width values in microns
TABLEENTRIES
9.0 7.5 6.5 5.4 4.7        #mA/um for 5 widths and freq_1 (when the frequency
                             #is 100 Mhz)
7.5 6.8 6.0 4.8 4.0 ;      #mA/um for 5 widths and freq_2 (when the frequency
                             #is 400 Mhz)
END met1 ;
```

The PEAK current density at 0.7  $\mu\text{m}$  for 100 Mhz is  $9.0 + (7.5 - 9.0) \times (0.8 - 0.7) / (0.8 - 0.4) = 8.625 \text{ mA}/\mu\text{m}$ , and at 0.7  $\mu\text{m}$  for 400 Mhz is  $7.5 + (6.8 - 7.5) \times (0.8 - 0.7) / (0.8 - 0.4) = 7.325 \text{ mA}/\mu\text{m}$ . Then interpolating between the frequencies at 300Mhz gives  $8.625 + (7.325 - 8.625) \times (400 - 300) / (400 - 100) = 8.192 \text{ mA}/\mu\text{m}$ .

The RMS current density at 0.4  $\mu\text{m}$  is 7.5  $\text{mA}/\mu\text{m}$ . Therefore, a 0.4  $\mu\text{m}$  wide wire can carry  $7.5 \times .4 = 3.0 \mu\text{m}$  of RMS current.

```
LAYER cut12
...
ACCURRENTDENSITY PEAK      #peak AC current limit for one cut
FREQUENCY 10 200 ;         #2 freq values in MHz
CUTAREA 0.16 0.32 ;        #2 cut areas in um squared
TABLEENTRIES
0.5 0.4                    #mA/um squared for 2 cut areas at freq_1 (10 Mhz)
0.4 0.35 ;                 #mA/um squared for 2 cut areas at freq_2 (200 Mhz)
ACCURRENTDENSITY AVERAGE  #average AC current limit for via cut12
10.0 ;                     #mA/um squared for any cut area at any frequency
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
ACCURRENTDENSITY RMS      #RMS AC current limit for via cut12
FREQUENCY 1 ;              #1 freq (required by syntax; not really used)
    CUTAREA 0.16 1.6 ;     #2 cut areas in um squared
TABLEENTRIES
10.0 9.0 ;                 #mA/um squared for 2 cut areas at any frequency
....
END cut12 ;
```

```
ANTENNAAREADIFFREDUCEPWL ( ( diffArea1 diffMetalFactor1 )
( diffArea2 diffMetalFactor2 ) ...)
```

Indicates that the metal area is multiplied by a *diffMetalReduceFactor* that is computed from a piece-wise linear interpolation based on the *diff\_area* attached to the metal. (See Example 4 in [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)) This means that the ratio is calculated as:

$$\text{ratio} = (\text{metalFactor} \times \text{metal\_area} \times \text{diffMetalReduceFactor}) / \text{gate\_area}$$

The *diffArea* values are floats, specified in microns squared. The *diffArea* values should start with 0 and monotonically increase in value to the maximum size *diffArea* allowed. The *diffMetalFactor* values are floats with no units. The *diffMetalFactor* values are normally between 0.0 and 1.0. If no rule is defined, the *diffMetalReduceFactor* value in the PAR(*m<sub>i</sub>*) equation defaults to 1.0.

For more information on the PAR(*m<sub>i</sub>*) equation and process antenna models, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

```
ANTENNAAREAFACOR value [DIFFUSEONLY]
```

Specifies the multiply factor for the antenna metal area calculation. DIFFUSEONLY specifies that the current antenna factor should only be used when the corresponding layer is connected to the diffusion.

*Default:* 1.0

*Type:* Float

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

**Note:** If you specify a value that is greater than 1.0, the computed areas will be larger, and violations will occur more frequently.

```
ANTENNAAREAMINUSDIFF minusDiffFactor
```

Indicates that the antenna ratio metal area should subtract the diffusion area connected to it. This means that the ratio is calculated as:

$$\text{ratio} = (\text{metalFactor} \times \text{metal\_area} - \text{minusDiffFactor} \times \text{diff\_area}) / \text{gate\_area}$$

If the resulting value is less than 0, it should be truncated to 0. For example, if a *metal2* shape has a final ratio that is less than 0 because it connects to a diffusion shape, then the cumulative check for *metal3* (or *via2*) connected to the *metal2* shape adds in a cumulative value of 0 from the *metal2* layer. (See Example 1 in [Appendix C, “Calculating](#)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

and Fixing Process Antenna Violations.”)

*Type:* Float

*Default:* 0.0

For more information on process antenna models, see [Calculating a PAR](#), in [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAAREARATIO *value*

Specifies the maximum legal antenna ratio, using the area of the metal wire that is not connected to the diffusion diode. For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

*Type:* Integer

ANTENNACUMAREARATIO *value*

Specifies the cumulative antenna ratio, using the area of the wire that is not connected to the diffusion diode. For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

*Type:* Integer

ANTENNACUMDIFFAREARATIO {*value* | PWL ( ( *d1 r1* ) ( *d2 r2* )... ) }

Specifies the cumulative antenna ratio, using the area of the metal wire that is connected to the diffusion diode. You can supply an explicit ratio *value* or specify piece-wise linear format (PWL), in which case the cumulative ratio value is calculated using linear interpolation of the diffusion area and ratio input values. The diffusion input values must be specified in ascending order.

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNACUMDIFFSIDEAREARATIO {*value* | PWL ( ( *d1 r1* ) ( *d2 r2* )... ) }

Specifies the cumulative antenna ratio, using the side wall area of the metal wire that is connected to the diffusion diode. You can supply an explicit ratio *value* or specify piece-wise linear format (PWL), in which case the cumulative ratio value is calculated using linear interpolation of the diffusion area and ratio input values. The diffusion input values must be specified in ascending order.

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNACUMROUTINGPLUSCUT

Indicates that the cumulative ratio rules (ANTENNACUMAREARATIO and ANTENNACUMDIFFAREARATIO) accumulate with the previous cut layer instead of the

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

previous metal layer. Use this to combine metal and cut area ratios into one cumulative ratio rule.

**Note:** This rule does not affect `ANTENNACUMSIDEAREARATIO` and `ANTENNACUMDIFFSIDEAREA` models.

For more information on process antenna models, see [Calculating a CAR](#), in [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

`ANTENNACUMSIDEAREARATIO` *value*

Specifies the cumulative antenna ratio, using the side wall area of the metal wire that is not connected to the diffusion diode. For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

`ANTENNADIFFFAREARATIO` {*value* | `PWL ( ( d1 r1 ) ( d2 r2 ) ... )`}

Specifies the antenna ratio, using the area of the metal wire that is connected to the diffusion diode. You can supply an explicit ratio *value* or specify piece-wise linear format (`PWL`), in which case the ratio value is calculated using linear interpolation of the diffusion area and ratio input values. The diffusion input values must be specified in ascending order.

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

`ANTENNADIFFSIDEAREARATIO` {*value* | `PWL ( ( d1 r1 ) ( d2 r2 ) ... )`}

Specifies the antenna ratio, using the side wall area of the metal wire that is connected to the diffusion diode. You can supply an explicit ratio *value* or specify piece-wise linear format (`PWL`), in which case the ratio value is calculated using linear interpolation of the diffusion area and ratio input values. The diffusion input values must be specified in ascending order.

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

`ANTENNAGATEPLUSDIFF` *plusDiffFactor*

Indicates that the antenna ratio gate area includes the diffusion area multiplied by *plusDiffFactor*. This means that the ratio is calculated as:

$$\text{ratio} = (\text{metalFactor} \times \text{metal\_area}) / (\text{gate\_area} + \text{plusDiffFactor} \times \text{diff\_area})$$

The ratio rules without “DIFF” (the `ANTENNAAREARATIO`, `ANTENNACUMAREARATIO`, `ANTENNASIDEAREARATIO`, and `ANTENNACUMSIDEAREARATIO` statements), are unnecessary for this layer if the `ANTENNAGATEPLUSDIFF` rule is specified because a zero diffusion area already is accounted for by the `ANTENNADIFF*RATIO` statements. (See Example 3 in [Routing Layer Process Antenna Model Examples](#) in [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#))



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

*Type:* Float

*Default:* 0.0

For more information on process antenna models, see [Calculating a PAR](#), in [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAMODEL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}

Specifies the oxide model for the layer. If you specify an ANTENNAMODEL statement, that value affects all ANTENNA\* statements for the layer that follow it until you specify another ANTENNAMODEL statement.

*Default:* OXIDE1, for a new LAYER statement

Because LEF is sometimes used incrementally, if an ANTENNA statement occurs twice for the same oxide model, the last value specified is used. For any given ANTENNA keyword, only one value or PWL table is stored for each oxide metal on a given layer.

#### Example 1-10 Antenna Model Statement

The following example defines antenna information for oxide models on layer *metal1*.

```
LAYER metal1
    ANTENNAMODEL OXIDE1 ;           #OXIDE1 not required, but good practice
    ANTENNACUMAREARATIO 5000 ;      #OXIDE1 values
    ANTENNACUMDIFFAREARATIO 8000 ;
    ANTENNAMODEL OXIDE2 ;           #OXIDE2 model starts here
    ANTENNACUMAREARATIO 500 ;        #OXIDE2 values
    ANTENNACUMDIFFAREARATIO 800 ;
    ANTENNAMODEL OXIDE3 ;
    ANTENNACUMAREARATIO 300 ;
    ANTENNACUMDIFFAREARATIO 600 ;
    ...
END metal1
```

ANTENNASIDEAREAFACOR *value* [DIFFUSEONLY]

Specifies the multiply factor for the antenna metal side wall area calculation.

DIFFUSEONLY specifies that the current antenna factor should only be used when the corresponding layer is connected to the diffusion.

*Default:* 1.0

*Type:* Float

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNASIDEAREARATIO *value*

Specifies the antenna ratio, using the side wall area of the metal wire that is not connected to the diffusion diode. For more information on process antenna calculation,

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

*Type:* Integer

AREA *minArea*

Specifies the minimum metal area required for polygons on the layer. All polygons must have an area that is greater than or equal to *minArea*, if no MINSIZE rule exists. If a MINSIZE rule exists, all polygons must meet either the MINSIZE or the AREA rule. For an example using these rules, see [Example 1-15](#) on page 275.

*Type:* Float, specified in microns squared

CAPACITANCE CPERSQDIST *value*

Specifies the capacitance for each square unit, in picofarads per square micron. This is used to model wire-to-ground capacitance.

CAPMULTIPLIER *value*

Specifies the multiplier for interconnect capacitance to account for increases in capacitance caused by nearby wires.

*Default:* 1

*Type:* Integer

DCCURRENTDENSITY

Specifies how much DC current a wire on this layer of a given width can handle in units of milliamps per micron (mA/μm).

The true meaning of current density would have units of milliamps per square micron (mA/μm<sup>2</sup>); however, the thickness of the metal layer is implicitly included, so the units in this table are milliamps per micron, where only the wire width varies.

The DCCURRENTDENSITY syntax is defined as follows:

```
AVERAGE
{ value
| WIDTH width_1 width_2 ... ;
  TABLEENTRIES value_1 value_2 ...
} ;
```

AVERAGE                      Specifies the average current limit of the layer.

*value*                         Specifies the current limit for the layer, in mA/μm.

WIDTH                         Specifies wire width values, in microns. You can specify more than one wire width. If you specify multiple width values, the values must be specified in ascending order.

*Type:* Float

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**TABLEENTRIES** Specifies the value of current density for each specified width, in mA/μm.

The final value for a given wire width is computed from a linear interpolation of the table values. The widths are not adjusted for any process shrinkage, so they should be correct for the “drawn width”.

*Type:* Float

### Example 1-11 DC Current Density Statements

The following examples define DC current density tables:

```
LAYER met1
...
DCCURRENTDENSITY AVERAGE      #avg. DC current limit for met1
50.0 ;                          #mA/um for any width
```

(or)

```
DCCURRENTDENSITY AVERAGE      #avg. DC current limit for met1
WIDTH
  0.4 0.8 1.6 5.0 20.0 ;        #5 width values in microns
TABLEENTRIES
  7.5 6.8 6.0 4.8 4.0 ;        #mA/um for 5 widths
...
END met1 ;
```

The AVERAGE current density at 0.4 μm is 7.5 mA/μm. Therefore, a 0.4 μm wide wire can carry 7.5 x .4 = 3.0 mA of AVERAGE DC current.

```
LAYER cut12
...
DCCURRENTDENSITY AVERAGE      #avg. DC current limit for via cut12
10.0 ;                          #mA/um squared for any cut area
```

(or)

```
DCCURRENTDENSITY AVERAGE      #avg. DC current limit for via cut12
  CUTAREA 0.16 0.32 ;          #2 cut areas in μm2
TABLEENTRIES
  10.0 9.0 ;                    #mA/um squared for 2 cut areas
...
END cut12 ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

DENSITYCHECKSTEP *stepValue*

Specifies the stepping distance for metal density checks, in distance units.

*Type:* Float

DENSITYCHECKWINDOW *windowLength windowWidth*

Specifies the dimensions of the check window, in distance units.

*Type:* Float

DIAGMINEDGELENGTH *diagLength*

Specifies the minimum length for a diagonal edge. Any 45-degree diagonal edge must have a length that is greater than or equal to *diagLength*.

*Type:* Float, specified in microns

DIAGPITCH {*distance* | *diag45Distance diag135Distance*}

Specifies the 45-degree routing pitch for the layer. Pitch is used by the router to get the best routing density.

*Default:* None

*Type:* Float, specified in microns

*distance* Specifies one pitch value that is used for both the 45-degree angle and 135-degree angle directions.

*diag45Distance diag135Distance*

Specifies the 45-degree angle pitch (the center-to-center space between 45-degree angle routes) and the 135-degree angle pitch.

DIAGSPACING *diagSpacing*

Specifies the minimum spacing allowed for a 45-degree angle shape.

*Default:* None

*Type:* Float, specified in microns

DIAGWIDTH *diagWidth*

Specifies the minimum width allowed for a 45-degree angle shape.

*Default:* None

*Type:* Float, specified in microns

DIRECTION {HORIZONTAL | VERTICAL | DIAG45 | DIAG135}

Specifies the preferred routing direction. Automatic routing tools attempt to route in the preferred direction on a layer. A typical case is to route horizontally on layers *metal1* and *metal3*, and vertically on layer *metal2*.

HORIZONTAL Routing parallel to the x axis is preferred.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

VERTICAL	Routing parallel to the y axis is preferred.
DIAG45	Routing along a 45-degree angle is preferred.
DIAG135	Routing along a 135-degree angle is preferred.

**Note:** Angles are measured counterclockwise from the positive x axis.

#### EDGECAPACITANCE *value*

Specifies a floating-point value of peripheral capacitance, in picofarads per micron. The place-and-route tool uses this value in two situations:

- Estimating capacitance before routing
- Calculating segment capacitance after routing

For the second calculation, the tool uses *value* only if you set layer thickness, or layer height, to 0. In this situation, the peripheral capacitance is used in the following formula:

$$\text{segment capacitance} = (\text{layer capacitance per square} \times \text{segment width} \times \text{segment length}) + (\text{peripheral capacitance} \times 2 (\text{segment width} + \text{segment length}))$$

#### FILLACTIVESPACING *spacing*

Specifies the spacing between metal fills and active geometries.

*Type:* Float

#### HEIGHT *distance*

Specifies the distance from the top of the ground plane to the bottom of the interconnect.

*Type:* Float

#### LAYER *layerName*

Specifies the name for the layer. This name is used in later references to the layer.

#### MASK *maskNum*

Specifies how many masks for double- or triple-patterning will be used for this layer. The *maskNum* variable must be an integer greater than or equal to 2. Most applications only support values of 2 or 3.

#### MAXIMUMDENSITY *maxDensity*

Specifies the maximum metal density allowed for the layer, as a percentage. The *minDensity* and *maxDensity* values represent the metal density range within which all areas of the design must fall. The metal density must be greater than or equal to *minDensity* and less than or equal to *maxDensity*.

*Type:* Float

*Value:* Between 0.0 and 100.0

### Example 1-12 Minimum and Maximum Density

The following example specifies a metal density range in which the minimum metal density must be greater than or equal to 20 percent and the maximum metal density must be less than or equal to 70 percent.

```
MINIMUMDENSITY 20.0 ;
```

```
MAXIMUMDENSITY 70.0 ;
```

```
MAXWIDTH width
```

Specifies the maximum wire width, in microns, allowed on the layer. Maximum wire width is defined as the smaller value of the width and height of the maximum enclosed rectangle. For example, MAXWIDTH 10.0 specifies that the width of every wire on the layer must be less than or equal to 10.0  $\mu\text{m}$ .

*Type:* Float

```
MINENCLOSEDAREA area [WIDTH width]
```

Specifies the minimum area size limit for an empty area that is enclosed by metal (that is, a donut hole formed by the metal).

<i>area</i>	Specifies the minimum area size of the hole, in microns squared.
-------------	------------------------------------------------------------------

*Type:* Float

<i>width</i>	Applies the minimum area size limit only when a hole is created from a wire that has a width that is greater than <i>width</i> , in microns. If any of the wires that surround the donut hole are larger than this value, the rule applies.
--------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

*Type:* Float

### Example 1-13 Min Enclosed Area Statement

The following MINENCLOSEDAREA example specifies that a hole area must be greater than or equal to 0.40  $\mu\text{m}^2$ .

```
LAYER m1
```

```
...
```

```
MINENCLOSEDAREA 0.40 ;
```

The following MINENCLOSEDAREA example specifies that a hole area must be greater than or equal to 0.30  $\mu\text{m}^2$ . However, if any of the wires enclosing the hole have a width that is greater than 0.15  $\mu\text{m}$ , then the hole area must be greater than or equal to 0.40  $\mu\text{m}^2$ . If any of

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

the wires enclosing the hole are larger than 0.50  $\mu\text{m}$ , then the hole area must be greater than or equal to 0.80  $\mu\text{m}^2$ .

```
LAYER m1
...
MINENCLOSEDAREA 0.30 ;
MINENCLOSEDAREA 0.40 WIDTH 0.15 ;
MINENCLOSEDAREA 0.80 WIDTH 0.50 ;
```

#### MINIMUMCUT

Specifies the number of cuts a via must have when it is on a wide wire or pin whose width is greater than *width*. The MINIMUMCUT rule applies to all vias touching this particular metal layer. You can specify more than one MINIMUMCUT rule per layer. (See [Example 1-14](#) on page 272.)

The MINIMUMCUT syntax is defined as follows:

```
[MINIMUMCUT numCuts WIDTH width
  [WITHIN cutDistance]
  [FROMABOVE | FROMBELOW]
  [LENGTH length WITHIN distance]
;] ...
```

*numCuts* Specifies the number of cuts a via must have when it is on a wire or pin whose width is greater than *width*.  
*Type:* Integer

WIDTH *width* Specifies the width of the wire or pin, in microns.  
*Type:* Float

WITHIN *cutDistance*

Indicates that *numCuts* via cuts must be less than *cutDistance* from each other in order to be counted together to meet the minimum cut rule. (See [Figure 1-135](#) on page 274.)

FROMABOVE | FROMBELOW

Indicates whether the rule applies only to connections from above this layer or from below.

*Default:* The rule applies to connections from above and below.

LENGTH *length* WITHIN *distance*

Indicates that the rule applies for thin wires directly connected to wide wires, if the wide wire has a width that is greater than *width* and a length that is greater than *length*, and the vias on the thin wire are less than *distance* from the wide wire. (See [Figure 1-134](#) on page 273). The *length* value must be greater than or equal to the *width* value.

If `LENGTH` and `WITHIN` are present, this rule only checks the thin wire connected to a wide wire, and does not check the wide wire itself. A separate `MINIMUMCUT x WIDTH y ;` statement without `LENGTH` and `WITHIN` is required for any wide wire minimum cut rule.

*Type:* Float, specified in microns

### Example 1-14 Minimum Cut Rules

The following `MINIMUMCUT` definitions show different ways to specify a `MINIMUMCUT` rule.

#### ■ Minimum Cut Rule Example 1

The following syntax specifies that two via cuts are required for *metal4* wires that are greater than 0.5  $\mu\text{m}$  when connecting from *metal3* or *metal5*.

```
LAYER metal4
    MINIMUMCUT 2 WIDTH 0.5 ;
```

#### ■ Minimum Cut Rule Example 2

The following syntax specifies that four via cuts are required for *metal4* wires that are greater than 0.7  $\mu\text{m}$ , when connecting from *metal3*.

```
LAYER metal4
    MINIMUMCUT 4 WIDTH 0.7 FROMBELOW ;
```

#### ■ Minimum Cut Rule Example 3

The following syntax specifies that four via cuts are required for *metal4* wires that are greater than 1.0  $\mu\text{m}$ , when connecting from *metal5*.

```
LAYER metal4
    MINIMUMCUT 4 WIDTH 1.0 FROMABOVE ;
```

#### ■ Minimum Cut Rule Example 4

The following syntax specifies that two via cuts are required for *metal4* wires that are greater than 1.1  $\mu\text{m}$  wide and greater than 20.0  $\mu\text{m}$  long, and the via cut is less than 5.0  $\mu\text{m}$  from the wide wire. [Figure 1-134](#) on page 273 illustrates this example.

```
LAYER metal4
```



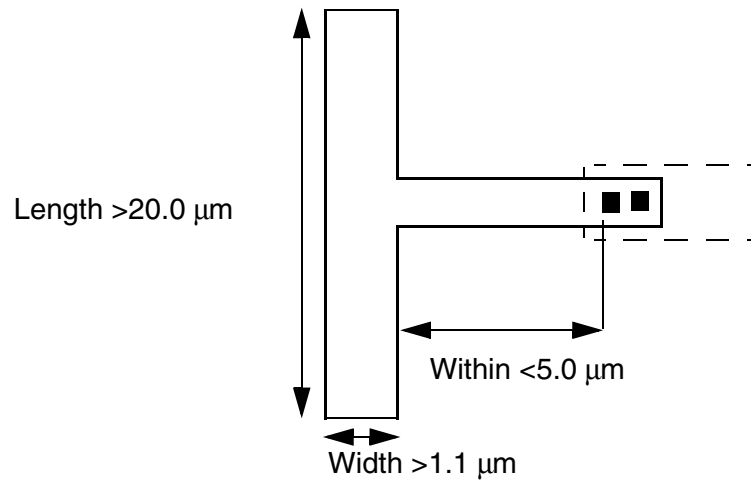
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
MINIMUMCUT 2 WIDTH 1.1 LENGTH 20.0 WITHIN 5.0 ;
```

**Figure 1-134 Minimum Cut Rule**



## ■ Minimum Cut Rule Example 5

The following syntax specifies that two via cuts are required for *metal4* wires that are greater than 1.0  $\mu\text{m}$  wide. The via cuts must be less than 0.3  $\mu\text{m}$  from each other in order to meet the minimum cut rule. [Figure 1-135](#) on page 274 illustrates this example.

```
MINIMUMCUT 2 WIDTH 1.0 WITHIN 0.3 ;
```

**Figure 1-135 Minimum Cut Within Rule**



a) Violation. The wire width is  $> 1.0 \mu\text{m}$ , therefore 2 cuts are needed. However, the 2 cuts are  $\geq 0.3 \mu\text{m}$  apart, therefore they cannot be counted together.

b) Okay. The wire width is  $> 1.0 \mu\text{m}$ , therefore 2 cuts are needed. The 2 cuts are  $< 0.3 \mu\text{m}$  apart, therefore they are counted together and meet the rule.

`MINIMUMDENSITY` *minDensity*

Specifies the minimum metal density allowed for the layer, as a percentage. The *minDensity* and *maxDensity* values represent the metal density range within which all areas of the design must fall. The metal density must be greater than or equal to *minDensity* and less than or equal to *maxDensity*. For an example of this statement, see [Example 1-12](#) on page 270.

*Type:* Float

*Value:* Between 0.0 and 100.0

`MINSIZE` *minWidth minLength [minWidth2 minLength2]*

Specifies the minimum width and length of a rectangle that must be able to fit somewhere within each polygon on this layer (see [Figure 1-136](#) on page 276). All polygons must meet this `MINSIZE` rule, if no `AREA` rule is specified. If an `AREA` rule is specified, all polygons must meet either the `MINSIZE` or the `AREA` rule.

You can specify multiple rectangles by specifying a list of *minWidth2* and *minLength2* values. If more than one rectangle is specified, the `MINSIZE` rule is satisfied if any of the rectangles can fit within the polygon.

*Type:* Float, specified in microns, for all values

### **Example 1-15 Minimum Size and Area Rules**

Assume the following minimum size and area rules:

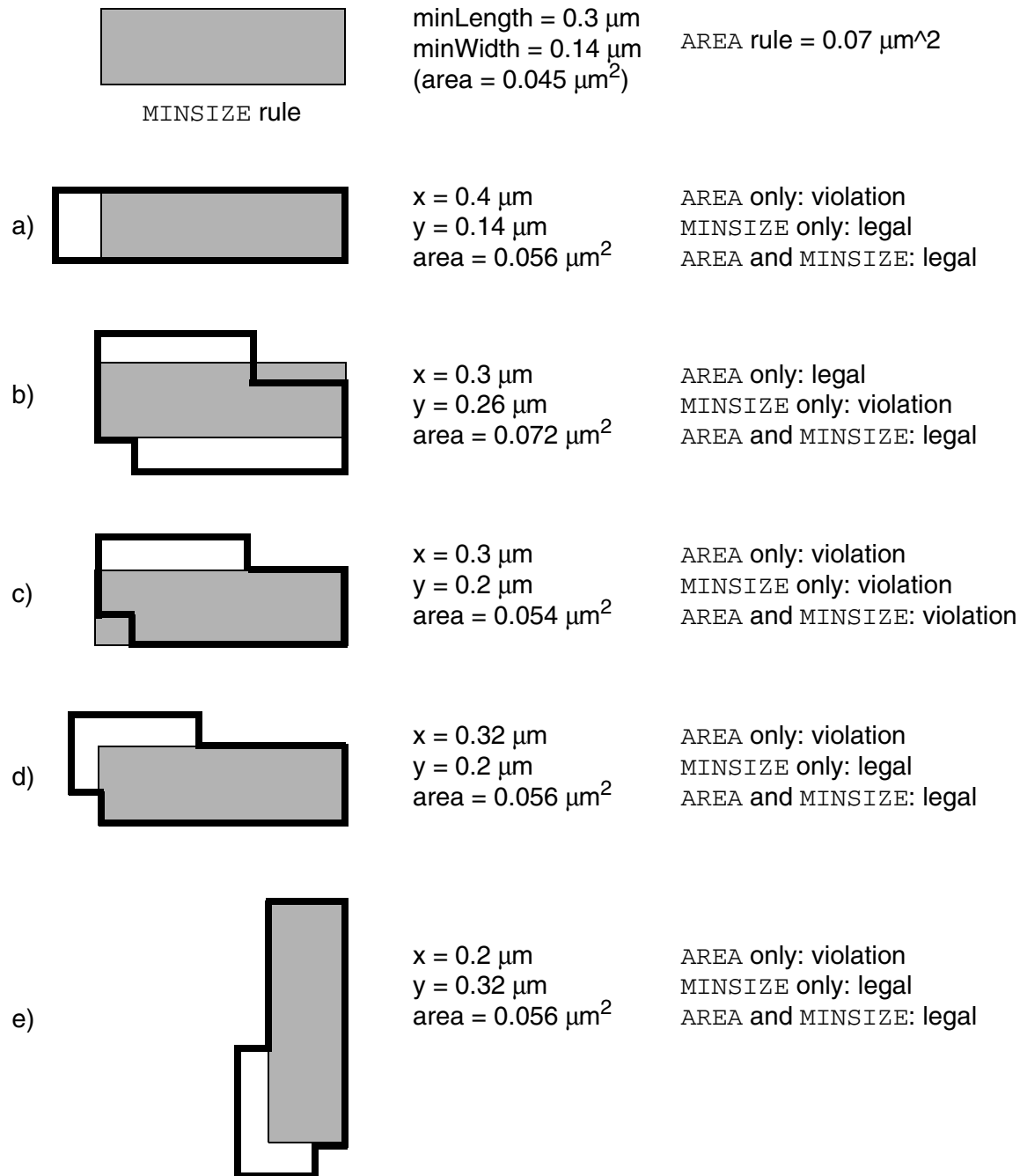
```
LAYER metall
    TYPE ROUTING ;
    AREA 0.07 ;           #0.20 um x 0.35 um = 0.07 um^2
    MINSIZE 0.14 0.30 ;   #0.14 um x 0.30 um = 0.042 um^2
    ....
```

Figure 1-136 on page 276 illustrates how these rules behave when one or both of the rules are present in the `LAYER` statement:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

**Figure 1-136 Minimum Size and Area Rules**



The following statement defines a `MINSIZE` rule that specifies that every polygon must have a minimum area of 0.07  $\mu\text{m}^2$ , or that a rectangle of 0.14 x 0.30  $\mu\text{m}$  must be able to fit within the polygon, or that a rectangle of 0.16 x 0.26  $\mu\text{m}$  must be able to fit within the polygon:

```
LAYER metall
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
TYPE ROUTING ;
AREA 0.07 ;                               #0.20 x 0.35 um = 0.07 um^2
MINSIZE 0.14 0.30 0.16 0.26 ;          #0.14 x 0.30 um = 0.042 um^2
                                         #0.16 x 0.26 um = 0.0416 um^2
...
END metall

MINSTEP
```

Specifies the minimum step size, or shortest edge length, for a shape. The `MINSTEP` rule ensures that Optical Pattern Correction (OPC) can be performed during mask creation for the shape.

**Note:** A single layer should only have one type of `MINSTEP` rule. It should include either `INSIDECORNER`, `OUTSIDECORNER`, or `STEP` statements (with an optional `LENGTHSUM` value), or one `LENGTHSUM` statement, or one `MAXEDGES` statement.

For an illustration of the `MINSTEP` rules, see [Figure 1-137](#) on page 281. For an example, see [Example 1-16](#) on page 279.

The syntax for `MINSTEP` is as follows:

```
[MINSTEP minStepLength
  [ [INSIDECORNER | OUTSIDECORNER | STEP]
    [LENGTHSUM maxLength]
  | [MAXEDGES maxEdges] ;]
```

*minStepLength* Specifies the minimum step size, or shortest edge length, for a shape. The edge of a shape must be greater than or equal to this value, or a violation occurs.

*Type:* Float, specified in microns

`INSIDECORNER` Indicates that a violation occurs if two or more consecutive edges of an inside corner are less than *minStepLength*.

If `LENGTHSUM` is also defined, a violation only occurs if the total length of all consecutive edges (that are less than *minStepLength*) is greater than *maxLength*.

Shape **b** in [Figure 1-137](#) on page 281 shows an inside corner. It is considered an inside corner because the two edges  $\geq$  *minStepLength* (shown with thick lines) that abut the consecutive short edges  $<$  *minStepLength* (shown with dashed lines) form an inside corner (or concave shape).

*Default:* `OUTSIDECORNER`

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

OUTSIDECORNER	<p>Indicates that a violation occurs if two or more consecutive edges of an outside corner are less than <i>minStepLength</i>.</p> <p>If LENGTHSUM is also defined, a violation only occurs if the total length of all consecutive edges (that are less than <i>minStepLength</i>) is greater than <i>maxLength</i>.</p> <p>Shape a in <a href="#">Figure 1-137</a> on page 281 shows an outside corner. It is considered an outside corner because the two edges <math>\geq</math> <i>minStepLength</i> (shown with thick lines) that abut the consecutive short edges <math>&lt;</math> <i>minStepLength</i> (shown with dashed lines) form an outside corner (or convex shape).</p> <p><b>Note:</b> This is the default rule, if INSIDECORNER, OUTSIDECORNER, or STEP is not specified.</p>
STEP	<p>Indicates that a violation occurs if one or more consecutive edges of a step are less than <i>minStepLength</i>.</p> <p>If LENGTHSUM is also defined, a violation only occurs if the total length of all consecutive edges (that are less than <i>minStepLength</i>) is greater than <i>maxLength</i>.</p> <p>Shape f in <a href="#">Figure 1-137</a> on page 281 shows a step. It is considered a step because the two edges <math>\geq</math> <i>minStepLength</i> (shown with thick lines) that abut the consecutive short edges <math>&lt;</math> <i>minStepLength</i> (shown with dashed lines) form a step instead of a corner.</p> <p><i>Default:</i> OUTSIDECORNER</p>
LENGTHSUM <i>maxLength</i>	<p>Specifies the maximum total length of consecutive short edges (edges that are less than <i>minStepLength</i>) that OPC can correct without causing new DRC violations.</p> <p>If the total length of the edges is greater than <i>maxLength</i>, a violation occurs. No violation occurs if the total length is less than or equal to <i>maxLength</i>.</p>
MAXEDGES <i>maxEdges</i>	

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that up to *maxEdges* consecutive edges that are less than *minStepLength* in length are allowed, but more than *maxEdges* in a row is a violation. Typically, most tools only allow a *maxEdges* value of 0, 1, or 2. A *maxEdges* value of 0 means that no edge can be less than *minStepLength*.

*Type:* Integer

**Note:** The *maxEdges* value of 1 will check the cases covered by OUTSIDECORNER and INSIDECORNER. However, there is no relationship between MAXEDGES and STEP.

#### Example 1-16 Minimum Step Rules

- The following table shows the results of the specified MINSTEP rules using the shapes in [Figure 1-137](#) on page 281. For these rules, assume *minStepLength* equals 0.05  $\mu\text{m}$ , and that each dashed edge is 0.04  $\mu\text{m}$  in length.

MINSTEP Rule	Result
MINSTEP 0.05 ;	OUTSIDECORNER is the default behavior. Therefore, shapes a and d are violations because their consecutive edges are less than 0.05 $\mu\text{m}$ . Shapes b, c, e, and f are not outside corner checks.
MINSTEP 0.04 ;	OUTSIDECORNER is the default behavior. Therefore, shapes a and d are checked and are legal because their consecutive edges are greater than or equal to 0.04 $\mu\text{m}$ .
MINSTEP 0.05 LENGTHSUM 0.08 ;	Shape a is legal because its consecutive edges are less than 0.05 $\mu\text{m}$ , and the total length of the edges is less than or equal to 0.08 $\mu\text{m}$ . Shape d is a violation because even though its consecutive edges are less than 0.05 $\mu\text{m}$ , the total length of the edges is greater than 0.08 $\mu\text{m}$ .
MINSTEP 0.05 LENGTHSUM 0.16 ;	Shapes a and d are legal because the total length of their consecutive edges is less than or equal to 0.16 $\mu\text{m}$ .

## LEF/DEF 5.8 Language Reference

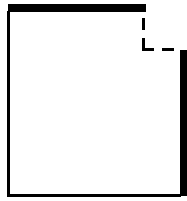
### LEF Syntax

MINSTEP Rule	Result
MINSTEP 0.05 INSIDECORNER ;	Shapes <b>b</b> and <b>e</b> are violations because their consecutive edges are less than 0.05 $\mu\text{m}$ . Shapes <b>a</b> , <b>c</b> , <b>d</b> , and <b>f</b> are not inside corner checks.
MINSTEP 0.05 INSIDECORNER LENGTHSUM 0.15 ;	Shape <b>b</b> is legal because its consecutive edges are less than 0.05 $\mu\text{m}$ , and the total length of the edges is less than or equal to 0.15 $\mu\text{m}$ . Shape <b>e</b> is a violation because even though its consecutive edges are less than 0.05 $\mu\text{m}$ , the total length of the edges is greater than 0.15 $\mu\text{m}$ .
MINSTEP 0.05 STEP ;	Shapes <b>c</b> and <b>f</b> are violations because their consecutive edges are less than 0.05 $\mu\text{m}$ . Shapes <b>a</b> , <b>b</b> , <b>d</b> , and <b>e</b> are not step checks.
MINSTEP 0.05 STEP LENGTHSUM 0.08 ;	Shape <b>c</b> is legal because its consecutive edges are less than 0.05 $\mu\text{m}$ , and the total length of the edges is less than or equal to 0.08 $\mu\text{m}$ . Shape <b>f</b> is a violation because even though its consecutive edges are less than 0.05 $\mu\text{m}$ , the total length of the edges is greater than 0.08 $\mu\text{m}$ .
MINSTEP 0.04 STEP ;	Shapes <b>c</b> and <b>f</b> are legal because their consecutive edges are greater than or equal to 0.04 $\mu\text{m}$ .

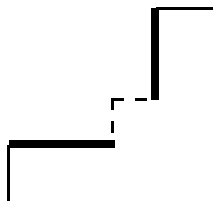


**Figure 1-137**

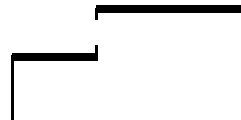
**Note:** All dashed edges are 0.04  $\mu\text{m}$  in length.



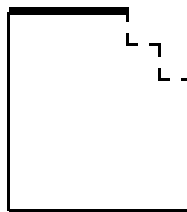
a) OUTSIDECORNER  
0.08  $\mu\text{m}$  LENGTHSUM



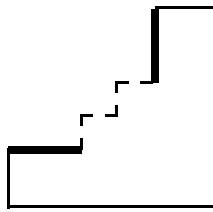
b) INSIDECORNER  
0.08  $\mu\text{m}$  LENGTHSUM



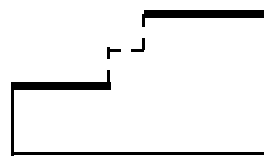
c) STEP  
0.04  $\mu\text{m}$



d) OUTSIDECORNER  
0.16  $\mu\text{m}$  LENGTHSUM



e) INSIDECORNER  
0.16  $\mu\text{m}$  LENGTHSUM

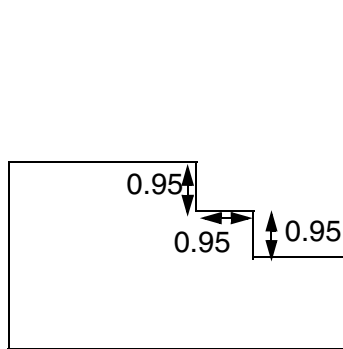


f) STEP  
0.12  $\mu\text{m}$  LENGTHSUM

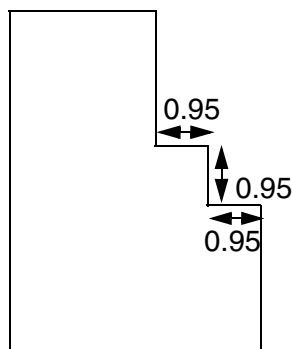
■ Figure 1-138 on page 281 shows the results of the following MINSTEP MAXEDGES rule:

MINSTEP 1.0 MAXEDGES 2 ;

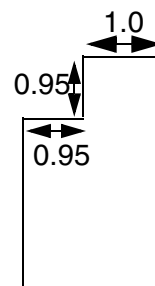
**Figure 1-138**



a) Violation; there is more than two edges in a row that are < 1.0  $\mu\text{m}$  in length.



b) Violation; there is more than two edges in a row that are < 1.0  $\mu\text{m}$  in length.



c) No violation; there are only two edges in a row that are < 1.0  $\mu\text{m}$  in length.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`MINWIDTH` *width*

Specifies the minimum legal object width on the routing layer. For example, `MINWIDTH 0.15` specifies that the width of every object must be greater than or equal to 0.15  $\mu\text{m}$ . This value is used for verification purposes, and does not affect the routing width. The `WIDTH` statement defines the default routing width on the layer.

**Default:** The value of the `WIDTH` statement

**Type:** Float, specified in microns

`OFFSET` {*distance* | *xDistance yDistance*}

Specifies the offset for the routing grid from the design origin for the layer. This value is used to align routing tracks with standard cell boundaries, which helps routers get good on-grid access to the cell pin shapes. For best routing results, most standard cells have a 1/2 pitch offset between the `MACRO SIZE` boundary and the center of cell pins that should be aligned with the routing grid. Normally, it is best to not set the `OFFSET` value, so the software can analyze the library to determine the best offset values to use, but in some cases it is necessary to force a specific offset.

Generally, it is best for all of the horizontal layers to have the same offset and all of the vertical layers to have the same offset, so that routing grids on different layers align with each other. Higher layers can have a larger pitch, but for best results, they should still align with a lower layer routing grid every few tracks to make stacked-vias more efficient.

**Default:** The software is allowed to determine its own offset values for preferred and non-preferred routing tracks.

**Type:** Float, specified in microns

*distance*                      Specifies the offset value that is used for the preferred direction routing tracks.

*xDistance yDistance*  
                                    Specifies the x offset for vertical routing tracks, and the y offset for horizontal routing tracks.

`PITCH` {*distance* | *xDistance yDistance*}

Specifies the required routing pitch for the layer. Pitch is used to generate the routing grid (the `DEF TRACKS`). For more information, see [“Routing Pitch”](#) on page 509.

**Type:** Float, specified in microns

*distance*                      Specifies one pitch value that is used for both the x and y pitch.

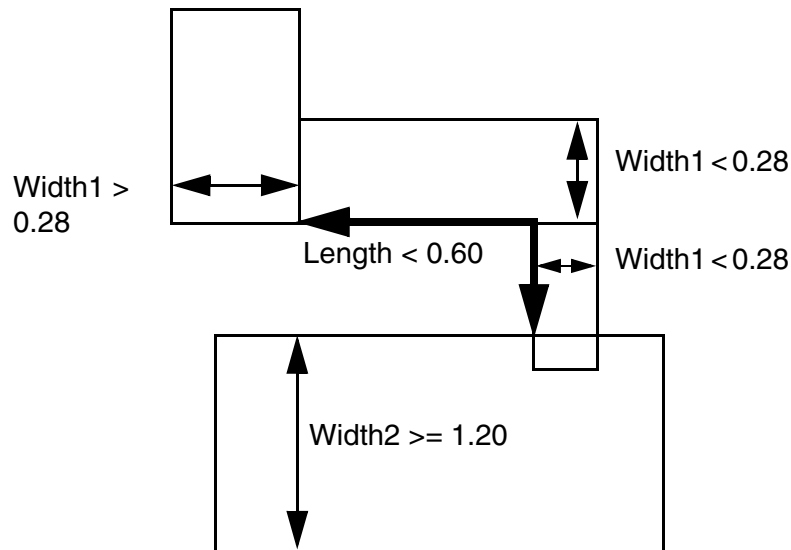
*xDistance yDistance*  
                                    Specifies the x pitch (the space between each vertical routing track), and the y pitch (the space between each horizontal routing track).



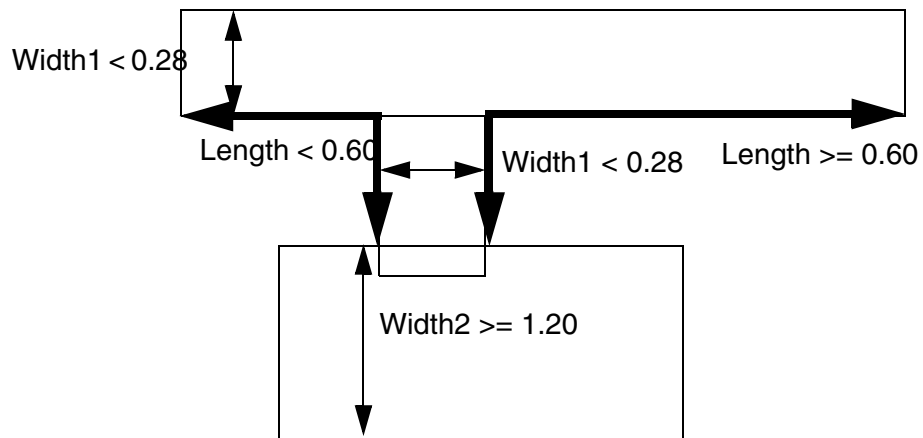
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



b) Violation; length < 0.60 is measured among all protrusion wires with width < 0.28.



c) Violation; length < 0.60 for shortest possible path.

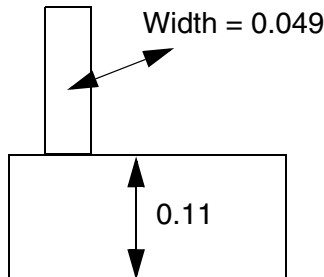
If the given value of `LENGTH` in `PROTRUSIONWIDTH` is zero, then the length of the protrusion wire is irrelevant. In this case, the width of the protrusion wire should always be checked independent of the length of the wire. The following example illustrates this rule:

```
PROTRUSIONWIDTH 0.05 LENGTH 0 WIDTH 0.11 ; " ;
```

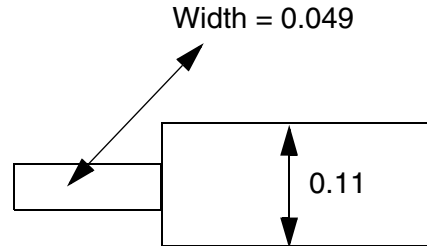
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



a) Violation, the width of the protrusion wire must be  $\geq 0.05$ , and its length is irrelevant.



b) Violation, it does not matter which sides the stub wire protruded from.

RESISTANCE *RPERSQ value*

Specifies the resistance for a square of wire, in ohms per square. The resistance of a wire can be defined as

$$RPERSQU \times \text{wire length} / \text{wire width}$$

SHRINKAGE *distance*

Specifies the value to account for shrinkage of interconnect wiring due to the etching process. Actual wire widths are determined by subtracting this constant value.

*Type:* Float

SPACING

Specifies the spacing rules to use for wiring on the layer. You can specify more than one spacing rule for a layer. See [“Using Spacing Rules”](#) on page 291.

The syntax for describing spacing rules is defined as follows:

```
[SPACING minSpacing
  [ RANGE minWidth maxWidth
    [ USELENGTHTHRESHOLD
      | INFLUENCE influenceLength
        [RANGE stubMinWidth stubMaxWidth]
      | RANGE minWidth maxWidth]
  | LENGTHTHRESHOLD maxLength
    [RANGE minWidth maxWidth]
  | ENDOFLINE eolWidth WITHIN eolWithin
    [PARALLELEDGE parSpace WITHIN parWithin
      [TWOEDGES]]]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
| SAMENET [PGONLY]
| NOTCHLENGTH minNotchLength
| ENDOFNOTCHWIDTH endOfNotchWidth
|   NOTCHSPACING minNotchSpacing
|   NOTCHLENGTH minNotchLength
|
;] ...
```

SPACING *minSpacing*

Specifies the default minimum spacing, in microns, allowed between two geometries on different nets.

*Type:* Float

RANGE *minWidth* *maxWidth*

Indicates that the minimum spacing rule applies to objects on the layer with widths in the indicated RANGE (that is, widths that are greater than or equal to *minWidth* and less than or equal to *maxWidth*). If you do not specify a range, the rule applies to all objects.

*Type:* Float

**Note:** If you specify multiple RANGE rules, the range values should not overlap.

USELENGTHTHRESHOLD

Indicates that the threshold spacing rule should be used if the other object meets the previous LENGTHTHRESHOLD value.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

INFLUENCE *influenceLength*  
[RANGE *stubMinWidth stubMaxWidth*]

Indicates that any length of the stub wire that is less than or equal to *influenceLength* from the wide wire inherits the wide wire spacing.

*Type:* Float

The influence rule applies to stub wires on the layer with widths in the indicated RANGE (that is, widths that are greater than or equal to *stubMinWidth* and less than or equal to *stubMaxWidth*). If you do not specify a range, the rule applies to all stub wires.

*Type:* Float

**Note:** Specifying the INFLUENCE keyword denotes that the statement only checks the influence rule, and does *not* check normal spacing. You must also specify a separate SPACING statement for normal spacing checks.

RANGE *minWidth maxWidth*

Specifies an optional second width range. The spacing rule applies if the widths of both objects fall in the ranges defined (each object in a different range). For an object's width to fall in a range, it must be greater than or equal to *minWidth* and less than or equal to *maxWidth*.

*Type:* Float

**Note:** If you specify multiple RANGE rules, the range values should not overlap.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

LENGTHTHRESHOLD *maxLength*  
[RANGE *minWidth* *maxWidth*]

Specifies the maximum parallel run length or projected length with an adjacent metal object for this spacing value. The *minSpacing* value should be less than or equal to the “default” *minSpacing* value when no LENGTHTHRESHOLD is specified for this range of widths. For an example, see “Using Spacing Rules” on page 291.

The threshold spacing rule applies to objects with widths in the indicated RANGE (that is, widths that are greater than or equal to *minWidth* and less than or equal to *maxWidth*). If you do not specify a range, the rule applies to all objects.

Type: Float

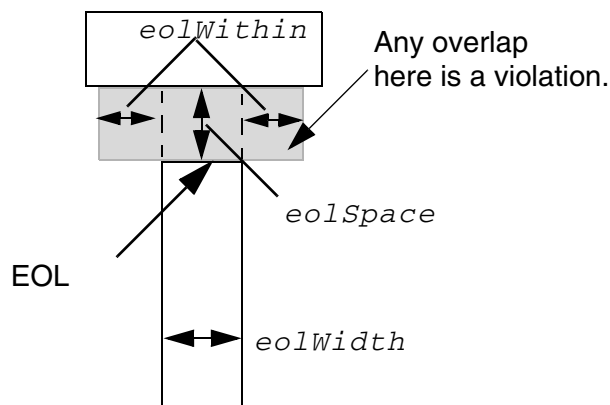
**Note:** If you specify multiple RANGE rules, the range values should not overlap.

ENDOFLINE *eolWidth* WITHIN *eolWithin*

Indicates that an edge that is shorter than *eolWidth*, noted as end-of-line (EOL from now on) edge requires spacing greater than or equal to *eolSpace* beyond the EOL anywhere within (that is, less than) *eolWithin* distance (see Figure 1-140 on page 288).

Typically, *eolSpace* is slightly larger than the minimum allowed spacing on the layer. The *eolWithin* value must be less than the minimum allowed spacing.

**Figure 1-140**



a) EOL width < *eolWidth* requires *eolSpace* beyond EOL to either side by < *eolWithin* distance.



## LEF/DEF 5.8 Language Reference

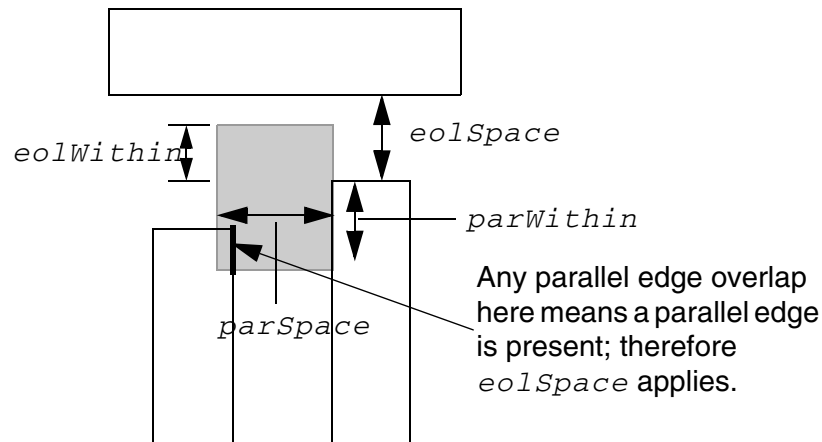
### LEF Syntax

---

PARALLELEDGE *parSpace* WITHIN *parWithin*  
[TWOEDGES]

Indicates the EOL rule applies only if there is a parallel edge that is less than *parSpace* away, and is also less than *parWithin* from the EOL and *eolWithin* beyond the EOL (see [Figure 1-141](#) on page 289).

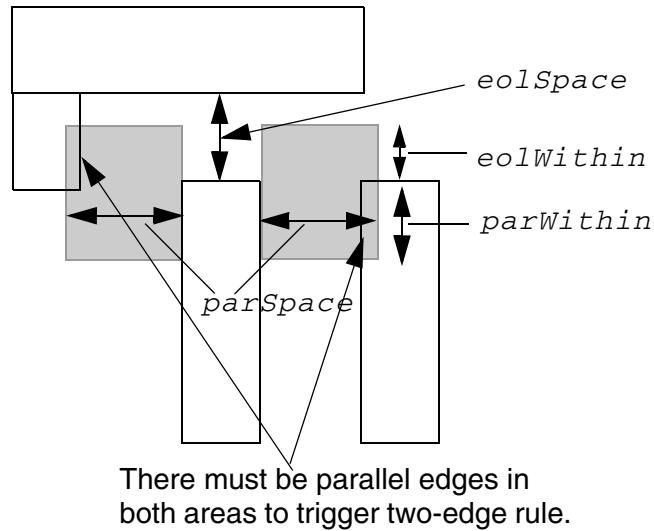
**Figure 1-141**



b) EOL space rule with PARALLELEDGE is triggered only if there is a parallel edge that overlaps inside the illustrated shaded box.

If TWOEDGES is specified, the EOL rule applies only if there are two parallel edges that meet the PARALLELEDGE *parSpace*, *eolWithin*, and *parWithin* parameters (see [Figure 1-142](#) on page 290).

Figure 1-142



c) EOL rule with `TWOEDGES` is triggered only if both sides have parallel edge overlaps inside the illustrated shaded boxes.

`SAMENET` [`PGONLY`]

Indicates that the *minSpacing* value only applies to same-net metal. If `PGONLY` also is specified, the *minSpacing* value only applies to same-net metal that is a power or ground net.

This rule typically is used when a technology has wider spacing for wider width wires; however, it still allows minimum spacing for same-net wires, even if they are wide. (See [Example 1-19](#) on page 298.)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

NOTCHLENGTH *minNotchLength*

Indicates that any notch with a notch length less than *minNotchLength* must have notch spacing greater than or equal to *minSpacing*. (See illustration a in [Figure 1-149](#) on page 299.)

The value you specify for *minSpacing* should be only slightly larger than the normal minimum spacing rule (typically, between 1x and 1.5x minimum spacing).

*Type:* Float, specified in microns

If the value of the specified notch length is zero, then the length of the notch is irrelevant. In other words, the spacing of a notch should always be checked independent of its length.

**Note:** You can specify only one NOTCHLENGTH rule per layer.

ENDOFNOTCHWIDTH *endOfNotchWidth*

NOTCHSPACING *minNotchSpacing*

NOTCHLENGTH *minNotchLength*

Indicates that the notch metal at the bottom end of a U-shaped notch requires spacing that is greater than or equal to *minSpacing*, if the notch has a width that is less than *endOfNotchWidth*, notch spacing that is less than or equal to *minNotchSpacing*, and notch length that is greater than or equal to *minNotchLength*. The spacing is required for the extent of the notch.

The values you specify for *notchSpacing* and *minSpacing* should be only slightly larger than the normal minimum spacing rule (typically between 1x and 1.5x minimum spacing). The value you specify for *endOfNotchWidth* should be only slightly larger than the minimum width rule (typically, between 1x and 1.5x minimum width).

*Type:* Float, specified in microns (for all values)

**Note:** You can specify only one ENDOFNOTCHWIDTH rule per layer.

### Using Spacing Rules

Spacing rules apply to pin-to-wire, obstruction-to-wire, via-to-wire, and wire-to-wire spacing. These requirements specify the default minimum spacing allowed between two geometries on different nets.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

When defined with a `RANGE` argument, a spacing value applies to all objects with widths within a specified range. That is, the rule applies to objects whose widths are greater than or equal to the specified minimum width and less than or equal to the specified maximum width.

**Note:** If you specify multiple `RANGE` arguments, the `RANGE` values should not overlap.

In the following example, the default minimum allowed spacing between two adjacent objects is 0.3  $\mu\text{m}$ . However, for objects between 0.5 and 1.0  $\mu\text{m}$  in width, the spacing is 0.4  $\mu\text{m}$ . For objects between 1.01 and 2.0  $\mu\text{m}$  in width, the spacing is 0.5  $\mu\text{m}$ .

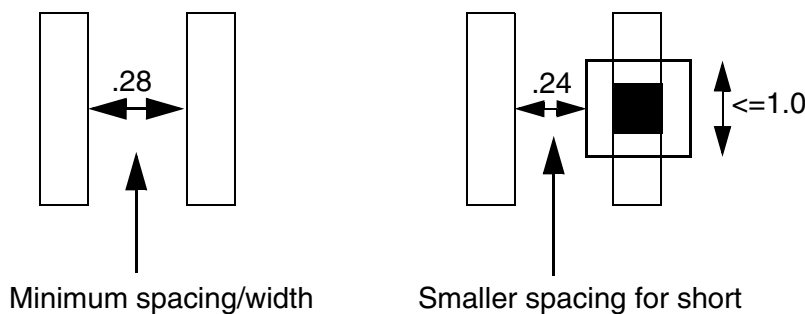
```
SPACING 0.3 ;
SPACING 0.4 RANGE 0.5 1.0 ;
SPACING 0.5 RANGE 1.01 2.0 ; #The RANGE begins at 1.01 and not 1.0 because
                              #RANGE values should not overlap.
```

Threshold spacing is a function of both the wire width and the length of the neighboring object. It is typically used when vias are wider than the wire to allow tighter wire-to-wire spacing, even when the vias are present.

In the following example, a slightly tighter spacing of .24  $\mu\text{m}$  is needed if the other object is less than or equal to 1.0  $\mu\text{m}$  in length (see [Figure 1-143](#) on page 292).

```
SPACING 0.28 ;
SPACING 0.24 LENGTHTHRESHOLD 1.0 ;
```

**Figure 1-143**



The `USELENGTHTHRESHOLD` argument specifies that the threshold spacing rule should be applied if the other object meets the previous `LENGTHTHRESHOLD` value.

In the following example, a larger spacing of 0.32  $\mu\text{m}$  is needed for wire widths between 1.5 and 9.99  $\mu\text{m}$ . However, if the other object is less than or equal to 1.0  $\mu\text{m}$  in length, the smaller .028  $\mu\text{m}$  spacing is applied (see [Figure 1-144](#) on page 293).

```
SPACING 0.28 ; #Default minimum spacing is >=0.28 um.
SPACING 0.28 LENGTHTHRESHOLD 1.0 ; #For short parallel lengths of <= 1.0 um,
```

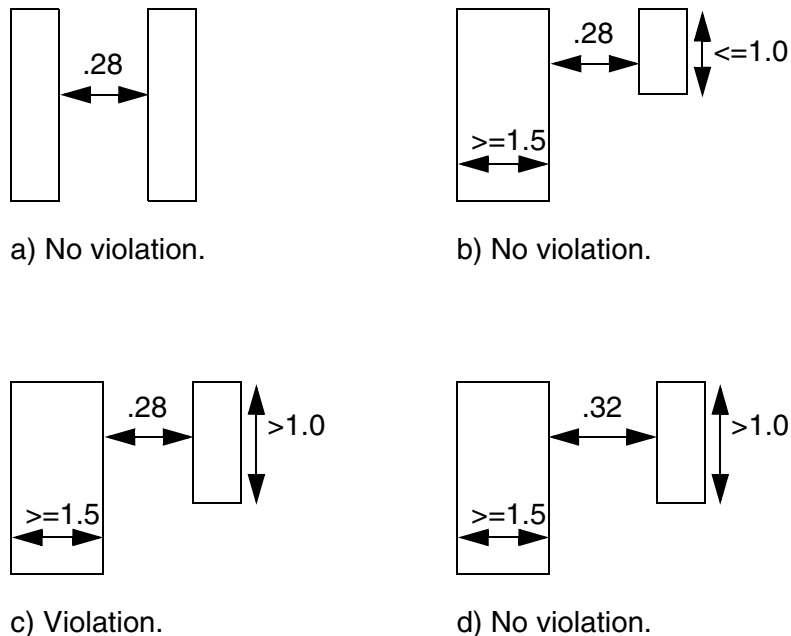
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
#0.28 spacing is allowed.  
SPACING 0.32 RANGE 1.5 9.99 USELENGTHTHRESHOLD ;  
#Wide wires with 1.5 <= width <=9.99 need  
#0.32 spacing unless the parallel run  
#length is <= 1.0 from the previous rule.
```

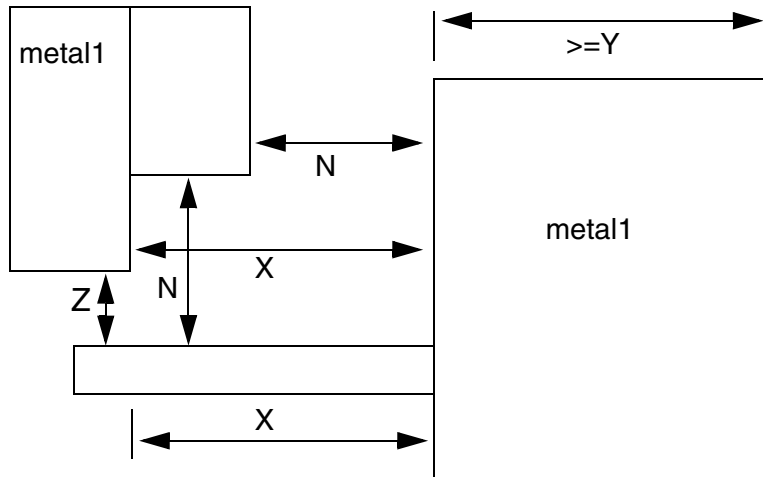
**Figure 1-144**



Influence spacing rules are used to support the inheritance of wide wire spacing by nets connected to the wide wires. For example, a larger spacing is needed for stub wires attached to large objects like pre-routed power wires. A piece of metal connecting to a wider wire will inherit spacing rules for a user-defined distance from the wider wire.

In [Figure 1-145](#) on page 294, a minimum space of N is required between two metal lines when at least one metal line has a width that is  $\geq Y$ . This spacing must be maintained for any small piece of metal ( $< Y$ ) that is connected to the wide metal within X range of the wide metal. Outside of this range, normal spacing rules (Z) apply.

**Figure 1-145**



In the following example, the 0.5  $\mu\text{m}$  spacing applies for the first 1.0  $\mu\text{m}$  of the stub sticking out from the large object. This rule only applies to the stub wire; the previous rule must be included for the wide wire spacing. The `SPACING 0.5 RANGE 2.01 2000.0` statement is required to get extra spacing for the wide-wire itself.

```
SPACING 0.5 RANGE 2.01 2000.0 ;
SPACING 0.28 ;                #Minimum spacing is >= 0.28 um.
SPACING 0.5 RANGE 2.01 2000.0 ;    #wide-wire >= 2.01 um wide requires 0.5um spacing
SPACING 0.5 RANGE 2.01 2000.0 INFLUENCE 1.000 ;
                                   #Stub wires <= 1.0 um from wide wires >= 2.01
                                   #require 0.5 um spacing.
```

Some processes only need the `INFLUENCE` rule for certain widths of the stub wire. In the following example, the 0.5  $\mu\text{m}$  spacing is required only for stub wires between 0.5 and 1.0  $\mu\text{m}$  in width.

```
SPACING 0.28 ;                #Minimum spacing is >= 0.28 um.
SPACING 0.5 RANGE 2.01 2000.0 ;    #wide-wire >= 2.01 um wide requires 0.5um spacing
SPACING 0.5 RANGE 2.01 2000.0 INFLUENCE 1.00 RANGE 0.5 1.0 ;
                                   #Stub wires with 0.5 <= width <= 1.0, and <= 1.0 um from
                                   #wide wide wires >= 2.01 require 0.5 um spacing.
```

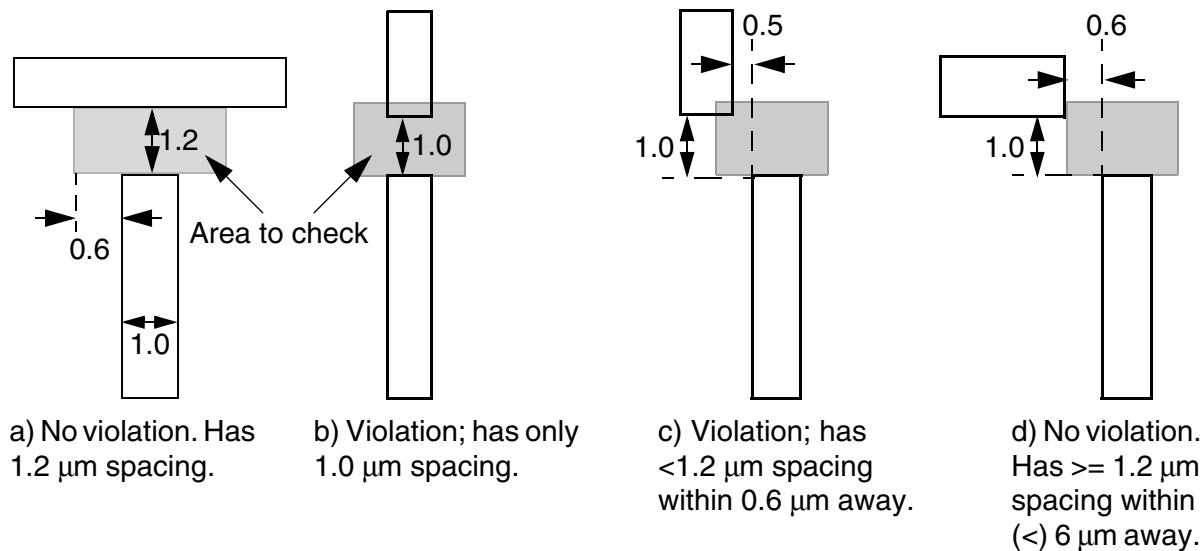
### Example 1-18 EOL Spacing Rules

- If you include the following routing layer rules in your LEF file:

```
SPACING 1.0 ;                #minimum spacing is 1.0 um
SPACING 1.2 ENDOFLINE 1.3 WITHIN 0.6 ;
```

Any EOL that is less than 1.3  $\mu\text{m}$  wide requires spacing that is greater than or equal to 1.2  $\mu\text{m}$  beyond the EOL, within 0.6  $\mu\text{m}$  to either side. [Figure 1-146](#) on page 295 includes examples of legal spacing for, and violations of, this rule.

**Figure 1-146**

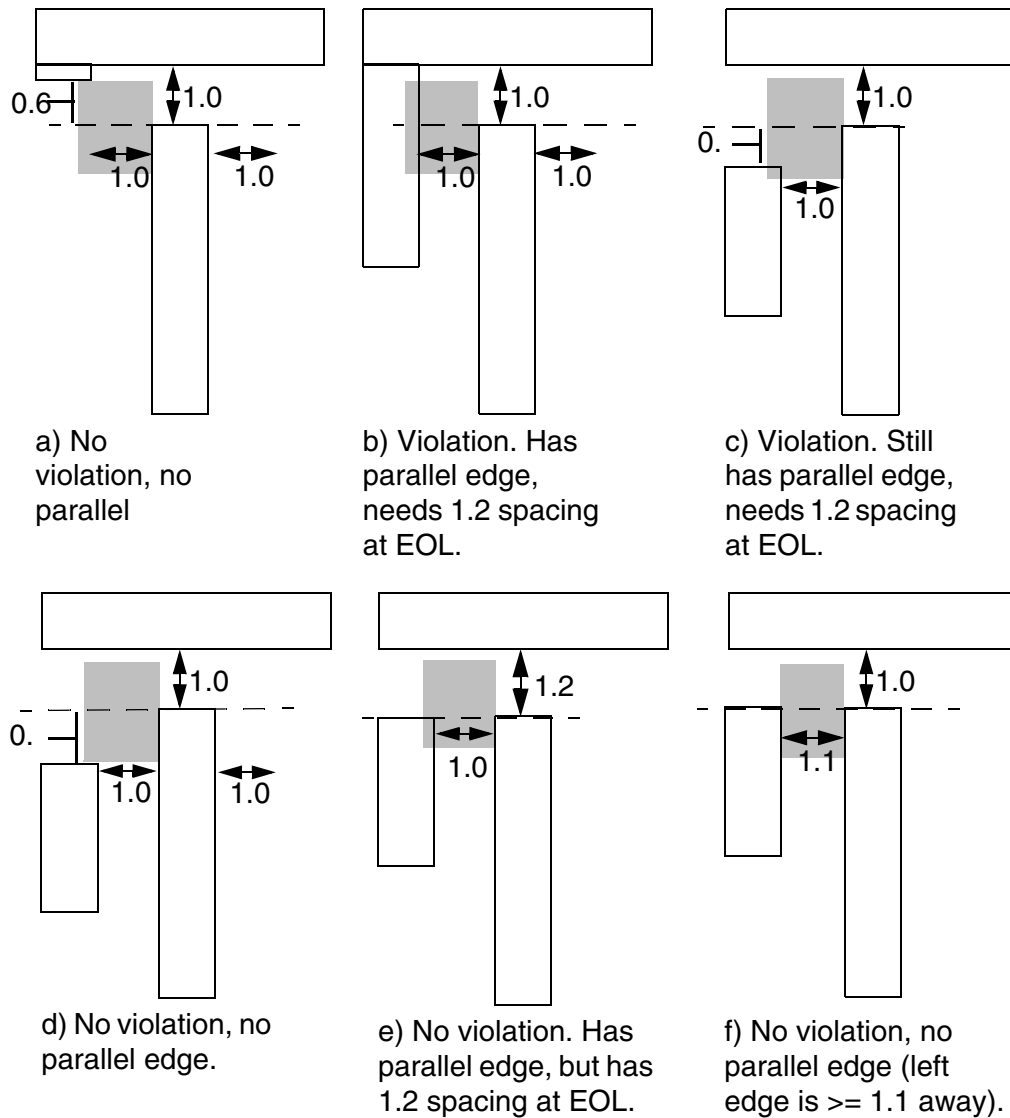


■ If you include the following routing layer rules in your LEF file:

```
SPACING 1.0 ;                               #minimum spacing is 1.0  $\mu\text{m}$ 
SPACING 1.2 ENDOFLINE 1.3 WITHIN 0.6 PARALLELEDGE 1.1 WITHIN 0.5 ;
```

Any line that is less than 1.3  $\mu\text{m}$  wide, with a parallel edge that is less than 1.1  $\mu\text{m}$  away, and is within 0.5  $\mu\text{m}$  of the EOL, requires spacing greater than or equal to 1.2  $\mu\text{m}$  beyond the EOL, within 0.6  $\mu\text{m}$  to either side of the EOL. [Figure 1-147](#) on page 296 includes examples of legal spacing for, and violations of, this rule.

Figure 1-147

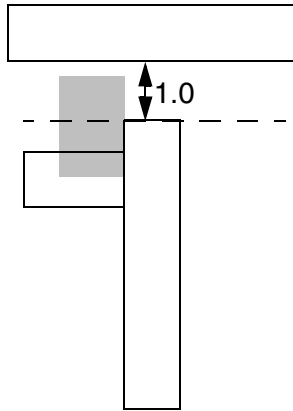




## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



g) No violation. Has overlap on the left side, but no parallel edge.

- The following routing layer rule creates an EOL spacing rule for two parallel edges:

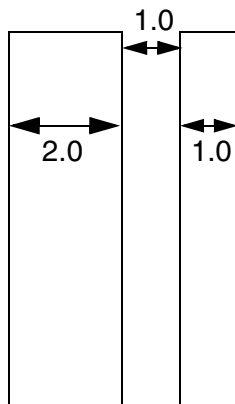
```
SPACING 1.0 ;                #minimum spacing is 1.0 μm
SPACING 1.2 ENDOFLINE 1.3 WITHIN 0.6 PARALLELEDGE 1.1 WITHIN 0.5 TWOEDGES ;
```

### Example 1-19 Same Net Spacing Rule

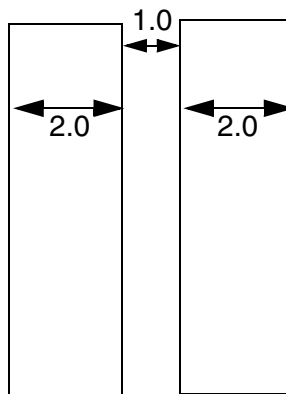
If you include the following routing layer rules in your LEF file, same-net power or ground nets can use 1.0  $\mu\text{m}$  spacing, even if they are 2  $\mu\text{m}$  to 5  $\mu\text{m}$  wide, as shown in [Figure 1-148](#) on page 298:

```
LAYER M1
TYPE ROUTING ;
SPACING 1.0 ;                #min spacing is 1.0
SPACING 1.5 RANGE 2.0 5.0 ;  #need 1.5 spacing for 2 to 5  $\mu\text{m}$  wide wires
SPACING 1.0 SAMENET PGONLY ;
```

**Figure 1-148**



a) Okay if both wires are the same net and are either a power or ground net.



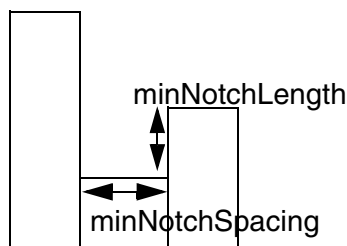
b) Okay if both wires are the same net and are either a power or ground net.

### Example 1-20 Notch Length Spacing Rule

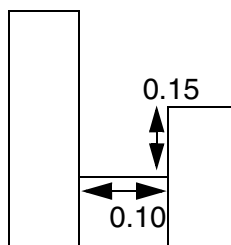
The figure below illustrates the following routing layer rules:

```
SPACING 0.10 ;  
SPACING 0.12 NOTCHLENGTH 0.15 ;
```

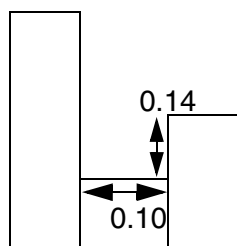
**Figure 1-149 Notch Length Rule Definitions**



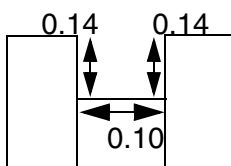
a) Illustration of notch spacing rule.



b) Okay; notchLength is not < 0.15.



c) Violation



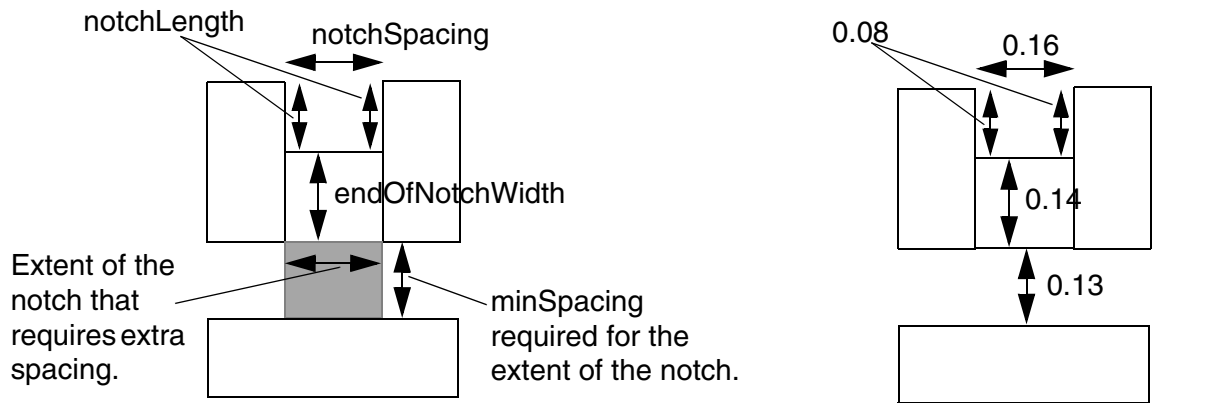
d) Violation

### Example 1-21 End Of Notch Width Spacing Rule

If you include the following routing layer rules in your LEF file, the notch metal at the bottom end of a U-shaped notch must have spacing that is greater than or equal to 0.14  $\mu\text{m}$ , if the notch metal has a width that is less than 0.15  $\mu\text{m}$ , notch spacing that is less than or equal to 0.16  $\mu\text{m}$ , and notch length that is greater than or equal to 0.08  $\mu\text{m}$ . See [Figure 1-150](#) on page 300 for different layout examples for these rules.

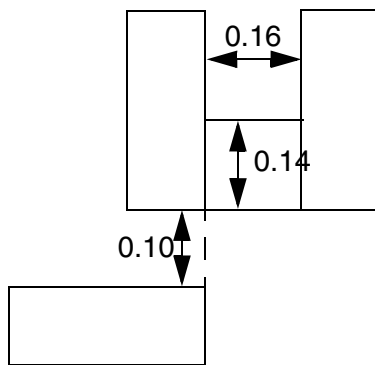
```
SPACING 0.10 ;           #default spacing  
SPACING 0.14 ENDOFNOTCHWIDTH 0.15 NOTCHSPACING 0.16 NOTCHLENGTH 0.08 ;
```

**Figure 1-150 End Of Notch Width Rule Definitions**

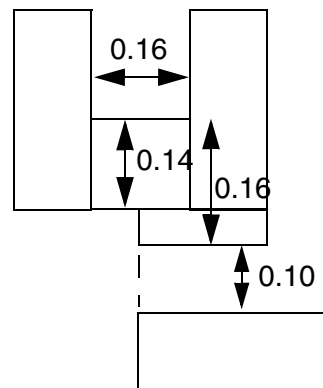


a) ENDOFNOTCHWIDTH rule definitions. The gray box shows where extra space is required.

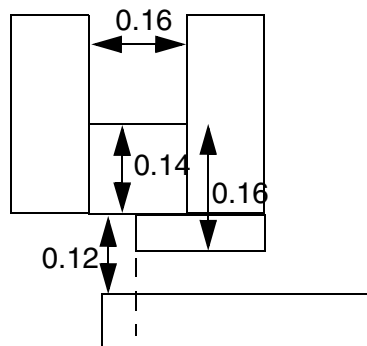
b) Violation; at least 0.14  $\mu\text{m}$  spacing required.



c) Okay. No overlap with notch above; therefore only default spacing of 0.10 required.



d) Okay. Notch metal width is  $\geq 0.15$  for the overlap; therefore extra space is not required.



e) Violation. Notch metal width is  $< 0.15$  for part of the overlap; therefore 0.14 spacing is required.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### SPACINGTABLE

Specifies the spacing tables to use for wiring on the layer. You can specify only one parallel run length and one influence spacing table for a layer. For information on and examples of using spacing tables, see [“Using Spacing Tables”](#) on page 302.

The syntax for describing spacing tables is defined as follows:

```
[SPACINGTABLE
  PARALLELRUNLENGTH {length} ...
  {WIDTH width {spacing} ...}... ;
  [SPACINGTABLE
    INFLUENCE {WIDTH width WITHIN distance
      SPACING spacing} ... ;]
  | TWOWIDTHS {WIDTH width [PRL runLength]
    {spacing} ...} ... ;
;]

PARALLELRUNLENGTH {length} ...
{WIDTH width {spacing} ...}
```

Specifies the maximum parallel run length between two objects, in microns. If the maximum width of the two objects is greater than *width*, and the parallel run length is greater than *length*, then the spacing between the objects must be greater than or equal to *spacing*. The first spacing value is the minimum spacing for a given width, even if the PRL value is not met.

You must specify *length*, *width*, and *spacing* values in increasing order.

*Type:* Float, specified in microns (for all values)

```
TWOWIDTHS {WIDTH width [PRL runLength] {spacing} ...}
```

Creates a table in which the spacing between two objects depends on the widths of both objects (instead of just the widest width). Optionally, it also can depend on the parallel run length between the two objects (PRL). For more information, see ["Two-Width Spacing Tables."](#)

The first width value should be 0 without an accompanied run length definition.

The PRL values in SPACINGTABLE TWOWIDTHS statement can be negative, which should be interpreted in the same way as in SPACINGTABLE PARALLELRUNLENGTH rules.

*Type:* Float, specified in microns (for all values)

```
INFLUENCE {WIDTH width WITHIN distance SPACING spacing ...}
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Creates a table that enforces wide wire spacing rules between nearby perpendicular wires. If an object has a width that is greater than *width*, and is located less than *distance* from two perpendicular wires, then the spacing between the perpendicular wires must be greater than or equal to *spacing*.

You must specify *width* values in increasing order.

*Type:* Float, specified in microns (for all values)

**Note:** You can only specify an INFLUENCE table if you specify a PARALLELRUNLENGTH table first.

### Specifying SPACING Statements with SPACINGTABLE

You can specify some of the SPACING statements with the SPACINGTABLE statements. For example, the following SPACING statements can be specified with SPACINGTABLE:

```
SPACING x SAMENET ____ ;
SPACING x ENDOFLINE ____ ;
SPACING x NOTCHLENGTH ____ ;
SPACING x ENDOFNOTCHWIDTH ____ ;
```

These SPACING checks are orthogonal to the SPACINGTABLE checks, except SAMENET spacing will override SPACINGTABLE for same-net objects.

However, you cannot specify some SPACING statements (as given below) with SPACINGTABLE as these would generate semantic errors.

```
SPACING x ;
SPACING x RANGE ____ ;
SPACING x LENGTHTHRESHOLD ____ ;
```

### Using Spacing Tables

Some processes have complex width and length threshold rules. Instead of creating multiple SPACING rules with different LENGTHTHRESHOLD and RANGE statements, you can define the information in a spacing table.

For example, for [Figure 1-151](#) on page 303, a typical 90nm DRC manual might have the following rules described:

Minimum spacing	0.15 $\mu\text{m}$ spacing
Either width>0.25 $\mu\text{m}$ and parallel length>0.50 $\mu\text{m}$	0.20 $\mu\text{m}$ spacing

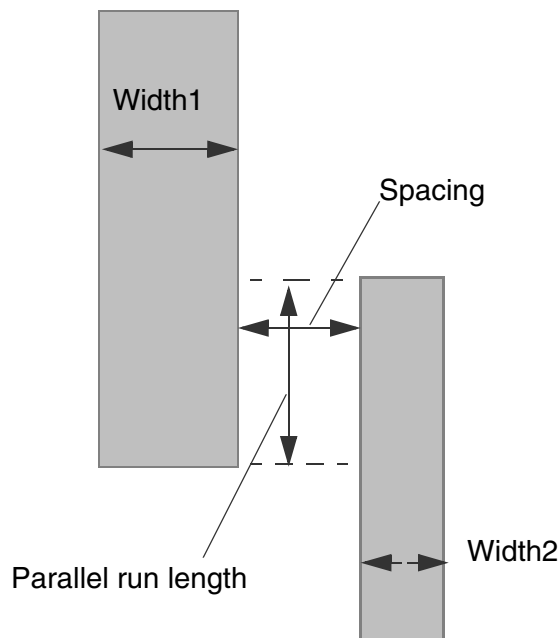
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Either width>1.50 $\mu\text{m}$ and parallel length>0.50 $\mu\text{m}$	0.50 $\mu\text{m}$ spacing
Either width>3.00 $\mu\text{m}$ and parallel length>3.00 $\mu\text{m}$	1.00 $\mu\text{m}$ spacing
Either width>5.00 $\mu\text{m}$ and parallel length>5.00 $\mu\text{m}$	2.00 $\mu\text{m}$ spacing

**Figure 1-151**



These rules translate into the following SPACINGTABLE PARALLELRUNLENGTH statement:

```
LAYER metall
...
SPACINGTABLE
  PARALLELRUNLENGTH 0.00 0.50 3.00 5.00           #lengths must be increasing
  WIDTH 0.00          0.15 0.15 0.15 0.15         #max width>0.00
  WIDTH 0.25          0.15 0.20 0.20 0.20         #max width>0.25
  WIDTH 1.50          0.15 0.50 0.50 0.50         #max width>1.50
  WIDTH 3.00          0.15 0.50 1.00 1.00         #max width>3.00
  WIDTH 5.00          0.15 0.50 1.00 2.00 ;        #max width>5.00
...
END metall
```

Using the SPACINGTABLE PARALLELRUNLENGTH statement, the rules can be described in the following way:

1. Find the maximum width of the two objects.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

2. Find the lowest table row where the maximum width is greater than the table-row width value. The first row is used even if the maximum width is less than and equal to the table-row width.
3. Find the right-most table column where the parallel run length is greater than the table PRL value. The first column spacing value is used even if the object's parallel run length is less than and equal to the table PRL value. The spacing value listed where the row and column intersect is the required spacing for that maximum width and parallel run length.

By definition, the width is the smaller dimension of the object (that is, the width of each object must be less than or equal to its length).

### Influence Spacing Tables

Processes often require a second spacing table to enforce the wide wire spacing rules between nearby perpendicular wires, even if the wires are narrow. [Figure 1-152](#) on page 305 illustrates this situation. Use the following SPACINGTABLE INFLUENCE syntax to describe this table:

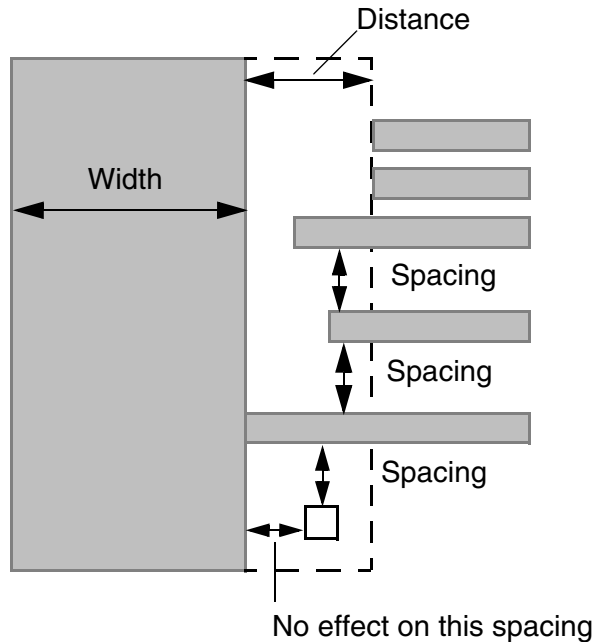
```
SPACINGTABLE INFLUENCE
    {WIDTH width WITHIN distance SPACING spacing} ... ;
```

If a wire has a width that is greater than *width*, and the distance between it and two other wires is less than *distance*, the other wires must be separated by spacing that is greater than or equal to *spacing*. Typically, the *distance* and *spacing* values are the same. Note that the distance halo extends horizontally, but not into the corners.

By definition, the width is the smaller dimension of the object (that is, the width is less than or equal to the length of the large wire).



**Figure 1-152**



The wide wire rules often match the larger width and spacing values in the `SPACINGTABLE PARALLEL RUNLENGTH` values. The previously described rules translate into the following `SPACINGTABLE INFLUENCE` statement:

```
LAYER metall
...
SPACINGTABLE INFLUENCE
  WIDTH 1.50 WITHIN 0.50 SPACING 0.50 #w>1.50, dist<0.50, needs sp>=0.50
  WIDTH 3.00 WITHIN 1.00 SPACING 1.00 #widths must be increasing
  WIDTH 5.00 WITHIN 2.00 SPACING 2.00 ;
...
END metall
```

## Two-Width Spacing Tables

You can create a table that enforces spacing rules that depends on the width of both objects instead of just the widest width, and optionally depends on the parallel run length between the two objects. You can use this table to replace existing `SPACING ... RANGE ... RANGE` rules to make it easier to read, and to include parallel run length effects in one common table. Use the following `SPACINGTABLE TWOWIDTHS` syntax to describe this table:

```
SPACINGTABLE
  TWOWIDTHS {WIDTH width [PRL runLength] {spacing} ... } ... ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

To find the required spacing, a 2-dimensional table is used that implicitly has the same widths (and optional parallel run lengths) for the row and column headings. There must be exactly as many *spacing* values in each *WIDTH* row as there are *WIDTH* rows. The *width* and *runLength* values must be the same or increasing from top to bottom in the table. The *spacing* values must be the same or increasing from left to right, and from top to bottom in the table.

Given two objects with *width1*, *width2*, and a parallel overlap of *runLength*, you find the spacing using the following method:

1. Find the last row where both *width1* is greater than the table row width, and *runLength* is greater than the table row run length. If no table row run length exists, the *runLength* value is not checked for that row (only that *width1* is greater than table row width is checked).
2. Find the right-most column where both *width2* is greater than table column width and *runLength* is greater than table column run length. If no table column run length exists, the *runLength* value is not checked for that column (only that *width2* is greater than table column width is checked).
3. The intersection of the matching row and column gives the required spacing.

For example, assume a DRC manual has the following rules described:

Minimum spacing	0.15 µm spacing
Either width>0.25 µm and parallel length>0.0 µm	0.20 µm spacing
Both width>0.25 µm and parallel length>0.0 µm	0.25 µm spacing
Either width>1.50 µm and parallel length>1.50 µm	0.50 µm spacing
Both width>1.50 µm and parallel length>1.50 µm	0.60 µm spacing
Either width>3.00 µm and parallel length>3.00 µm	1.00 µm spacing
Both width>3.00 µm and parallel length>3.00 µm	1.20 µm spacing

The rules translate into the following SPACINGTABLE:

```
SPACINGTABLE  TWOWIDTHS
#             width=    0.00  0.25  1.50  3.0
#             prl=     none  0.00  1.50  3.0
#             -----
WIDTH 0.00                0.15  0.20  0.50  1.00
WIDTH 0.25 PRL 0.0        0.20  0.25  0.50  1.00
WIDTH 1.50 PRL 1.50       0.50  0.50  0.60  1.00
WIDTH 3.00 PRL 3.00       1.00  1.00  1.00  1.20 ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Note that both width *and* parallel run length (if specified) must be exceeded to index into the row and column. Therefore, in this example:

If width1 = 0.25, width2 = 0.25, and prl = 0.0, then spacing = 0.15.

If width1 = 0.25, width2 = 0.26, and prl = 0.0, then spacing = 0.15.

If width1 = 0.25, width2 = 0.26, and prl = 0.1, then spacing = 0.20.

If width1 = 0.26, width2 = 0.26, and prl = 0.1, then spacing = 0.25.

THICKNESS *distance*

Specifies the thickness of the interconnect.

*Type:* Float

TYPE ROUTING

Identifies the layer as a routable layer.

WIDTH *defaultWidth*

Specifies the default routing width to use for all regular wiring on the layer.

*Type:* Float

WIREEXTENSION *value*

Specifies the distance by which wires are extended at vias. You must specify a value that is more than half of the routing width.

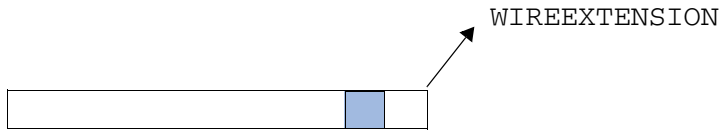
*Default:* Wires are extended half of the routing width

*Type:* Float

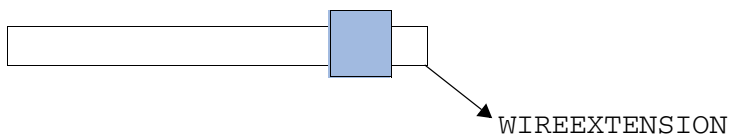
**Note:** The WIREEXTENSION statement only extends wires and not vias. For 65nm and below, WIREEXTENSION is no longer recommended because it may generate some advance rule violations if wires and vias have different widths.

The following figure shows WIREEXTENSION with same and different wire and via widths:

**Figure 1-153 Illustration of WIREEXTENSION**



a) Wire and via with same width



b) Wire and via with different widths

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### Defining Routing Layer Properties to Create 32/28 nm and Smaller Nodes Rules

You can include routing layer properties in your LEF file to create 32/28 nm and smaller nodes rules that currently are not supported by existing LEF syntax. The properties are specified inside the `LAYER ROUTING` statements, where they can be seen with other rules.

Before you can reference them, properties must be defined at the beginning of the LEF file in the `PROPERTYDEFINITIONS` statement, immediately before the first `LAYER` statement.

- Properties belong to the `LAYER` object and have a type of `STRING`.
- The property names used for these rules all start with `LEF58_`.

All properties use the following syntax within the `LEF PROPERTYDEFINITIONS` statement:

```
PROPERTYDEFINITIONS
    LAYER propName STRING ["stringValue"] ;
END PROPERTYDEFINITIONS
```

The property definitions for the routing layer properties are as follows:

```
PROPERTYDEFINITIONS
    LAYER LEF58_ANTENNAGATEPWL STRING ;
    LAYER LEF58_ANTENNAGATEPLUSDIFF STRING ;
    LAYER LEF58_AREA STRING ;
    LAYER LEF58_CORNERSPACING STRING ;
    LAYER LEF58_BACKSIDE STRING ;
    LAYER LEF58_BOUNDARYEOLBLOCKAGE STRING ;
    LAYER LEF58_CORNEREOLKEEPOUT STRING ;
    LAYER LEF58_CORNERFILLSPACING STRING ;
    LAYER LEF58_ENCLOSURESPACING STRING ;
    LAYER LEF58_EOLKEEPOUT STRING ;
    LAYER LEF58_EOLEXTENSIONSPACING STRING ;
    LAYER LEF58_FILLTOFILLSPACING STRING ;
    LAYER LEF58_FIVEWIRESEOLSPACING STRING ;
    LAYER LEF58_FORBIDDENSPACING STRING ;
    LAYER LEF58_GAP STRING ;
    LAYER LEF58_JOINTCORNERSPACING STRING ;
    LAYER LEF58_LITHOMACROHALO STRING ;
    LAYER LEF58_MANUFACTURINGGRID STRING ;
    LAYER LEF58_MINSTEP STRING ;
    LAYER LEF58_MINSIZE STRING ;
    LAYER LEF58_MINIMUMCUT STRING ;
    LAYER LEF58_OPPOSITEEOLSPACING STRING ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
LAYER LEF58_PITCH STRING ;
LAYER LEF58_PROTRUSIONWIDTH STRING;
LAYER LEF58_RECTONLY STRING;
LAYER LEF58_REGION STRING ;
LAYER LEF58_RIGHTWAYONGRIDONLY STRING ;
LAYER LEF58_SPACINGTABLE STRING ;
LAYER LEF58_SPANLENGTHENCLOSURESPACING STRING ;
LAYER LEF58_SPANLENGTHTABLE STRING ;
LAYER LEF58_SPACING STRING ;
LAYER LEF58_TWOWIRESFORBIDDENSPACING STRING ;
LAYER LEF58_TYPE STRING ;
LAYER LEF58_VOLTAGESPACING STRING ;
LAYER LEF58_WIDTHTABLE STRING ;
END PROPERTYDEFINITIONS
```

#### **Boundary EOL Blockage Rule**

A Boundary EOL blockage rule is used to specify line-end blockages. You can create a corner EOL keep-out rule by using the following property definition:

```
PROPERTY LEF58_BOUNDARYEOLBLOCKAGE
    "BOUNDARYEOLBLOCKAGE size OFFSET offset
    ;" ;
```

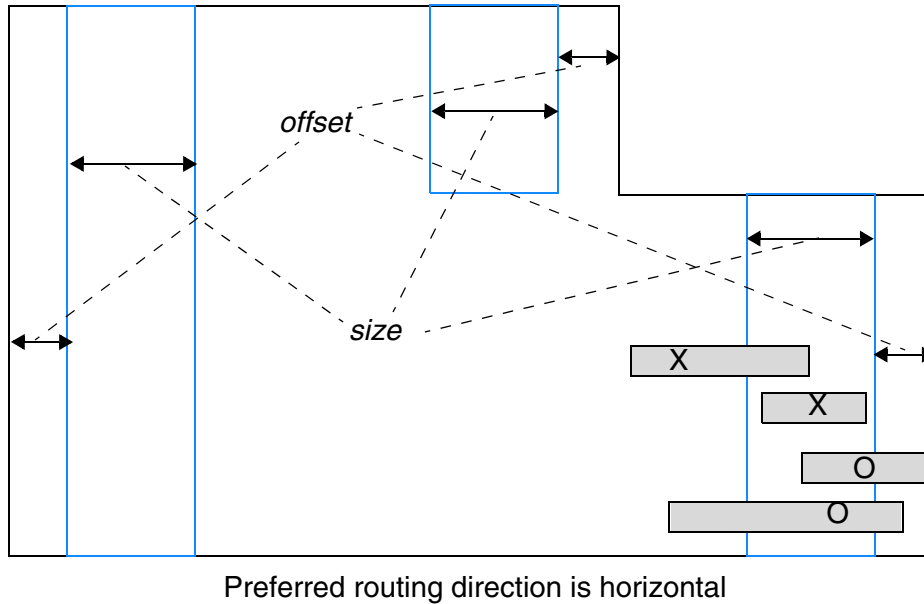
Where:

```
BOUNDARYEOLBLOCKAGE size OFFSET offset
```

Specifies line-end blockages formed near the boundary edges in the orthogonal direction of the preferred routing direction of the layer of a design. The blockage is formed by going inward from the boundary edge to the core by *offset* and having the size of *size* going further inward to the core. These blockages prevent the line-end of a wire from ending inside the blockages if it does not cross the edge closer to the boundary.

*Type:* Float, specified in microns

**Figure 1-154 Illustration of the Boundary EOL Blockage Rule**



The top wire crossing the left edge of the blue line-end blockage on the right boundary would be a violation. The second wire that is completely enclosed by the blockage is also a violation. The third wire crossing the right edge of the blockage and the fourth wire going through the blockage are both legal.

Illustration of `BOUNDARYEOLBLOCKAGE size OFFSET offset`

### **Corner EOL Keep-out Rule**

A corner EOL keep-out rule specifies a keep-out region for an EOL edge. You can create a corner EOL keep-out rule by using the following property definition:

```
PROPERTY LEF58_CORNEREOLKEEPOUT
    "CORNEREOLKEEPOUT WIDTH eolWidth EOLSPACING eolSpacing
      { SPACING spacing1 spacing2 WITHIN within1 within2
        | EXTENSION backwardExt sideExt forwardExt }
    ;"
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

```
CORNEREOLKEEPOUT WIDTH eolWidth EOLSPACING eolSpacing  
    SPACING spacing1 spacing2 WITHIN within1 within2
```

Specifies that if the distance between two corner-to-corner end-of-lines (EOLs) in the direction of the EOL edge is greater than or equal to *within1* and less than or equal to *within2* and the distance between them in the direction perpendicular to the EOL edge is greater than or equal to *spacing1* and less than or equal to *spacing2*, then a neighbor wire is not allowed in the search window formed by their facing corners.

Both of the EOLs must have width less than *eolWidth* and the spacing to a neighbor wire with PRL greater than 0 to the EOL must be greater than *eolSpacing*.

*spacing1* and *spacing2* could be negative, which means that if the two end-of-lines are within  $\text{abs}(\text{spacing1})$ , the rule would be triggered.

*Type:* Float, specified in microns

```
CORNEREOLKEEPOUT WIDTH eolWidth EOLSPACING eolSpacing  
    EXTENSION backwardExt sideExt forwardExt
```

Specifies a keep-out region for an EOL edge with width less than *eolWidth* and spacing greater than *eolSpacing* with PRL greater than 0 to a neighbor wire. The keep-out region is formed by extending *backwardExt* going backward, *sideExt* on the side, and *forwardExt* going forward from the corners of EOL. Any corners falling within the keep-out region will be a violation.

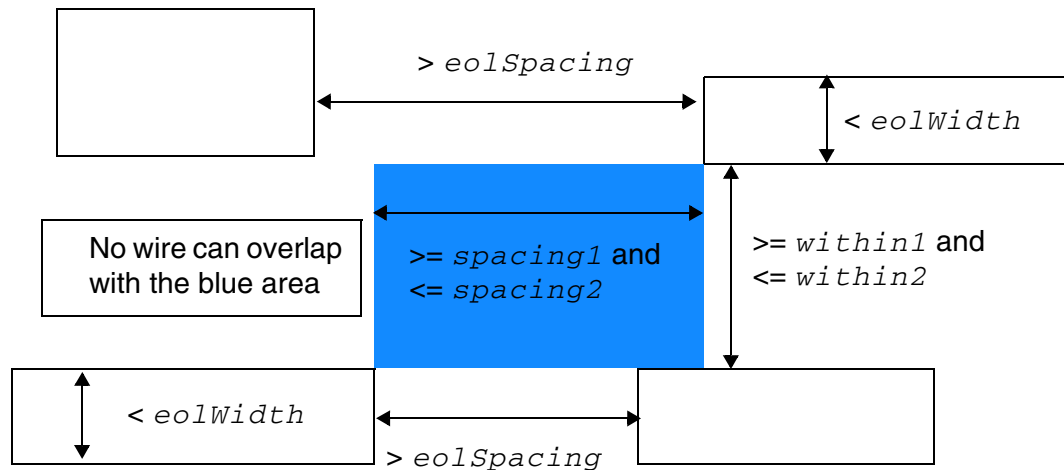
*Type:* Float, specified in microns

### Corner EOL Keep-out Rule Examples

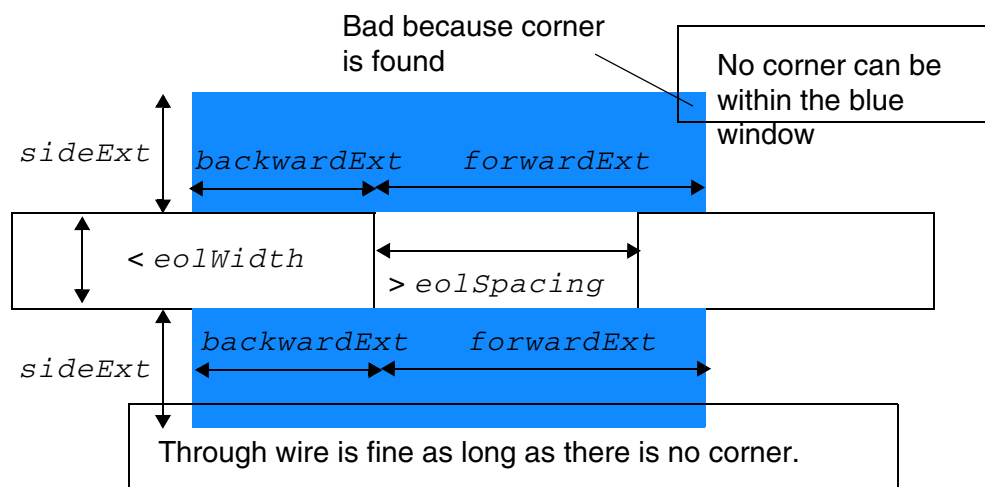
The following diagrams illustrate the corner EOL keep-out rule:



**Figure 1-155 Illustration of the Corner EOL Keep-out Rule with SPACING**



**Figure 1-156 Illustration of the Corner EOL Keep-out Rule with EXTENSION**



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### ***Corner Fill Spacing Rule***

A corner fill spacing rule can be used to define spacing of missing corners.

You can create a corner fill spacing rule by using the following property definition:

```
PROPERTY LEF58_CORNERFILLSPACING
    "CORNERFILLSPACING spacing EDGELENGTH length1 length2
    ADJACENTEOL eolWidth ;" ;
```

Where:

```
CORNERFILLSPACING spacing EDGELENGTH length1 length2
    ADJACENTEOL eolWidth
```

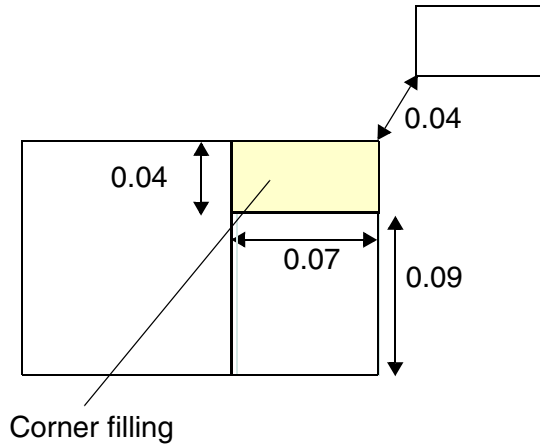
Specifies the spacing of a missing corner after the corner is filled to be *spacing*, if the following conditions are met:

The missing concave corner is composed of two edges with length less than *length1* and *length2* respectively. The edge with length less than *length2* has an adjacent EOL edge with width less than *eolWidth*.

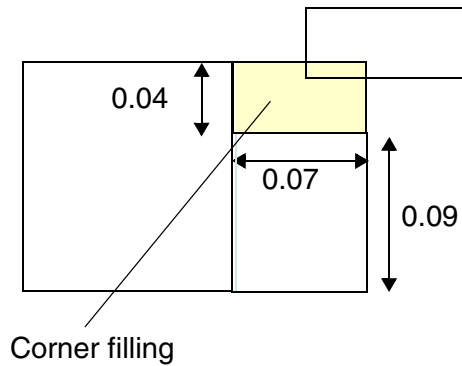
*Type:* Float, specified in microns

**Figure 1-157 Illustration of Corner Fill Spacing Rule**

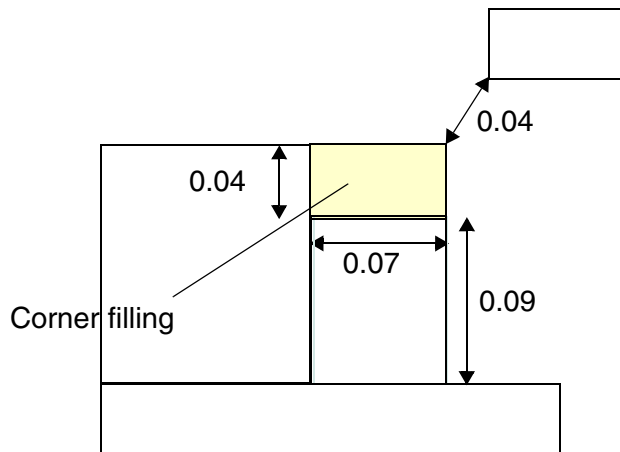
```
PROPERTY LEF58_CORNERFILLSPACING
  "CORNERFILLSPACING 0.05 EDGELENGTH 0.06 0.08
    ADJACENTEOL 0.10 ; " ;
```



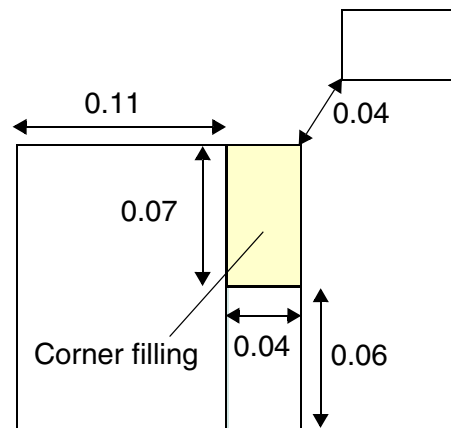
a) Violation, all of the conditions are met, the spacing after the corner filling is 0.04 ( $< 0.05$ )



b) Violation, neighbor wire overlaps with filling area is also bad



c) OK, the 0.09 adjacent edge is not a EOL edge



d) OK, 0.07 cannot be the first edge length, being the second edge length, its adjacent edge length is more than 0.10 ( $0.11 > 0.10$ ).

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### ***EOL Spacing Rule***

An EOL spacing rule ensures that Optical Proximity Correction (OPC) can be performed without interference between the OPC shapes added at the EOLs.

You can create an EOL spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING
    "SPACING eolSpace
        ENDOFLINE eolWidth
        [{EXACTWIDTH}
        [WRONGDIRSPACING wrongDirSpace]
        {[OPPOSITEWIDTH oppositeWidth]
        WITHIN eolWithin [wrongDirWithin]
        [SAMEMASK]
        [EXCEPTEXACTWIDTH exactWidth otherWidth]
        [FILLCONCAVECORNER fillTriangle]
        [WITHCUT [CUTCLASS cutClass] [ABOVE] withCutSpace
            [ENCLOSUREEND enclosureEndWidth
                [WITHIN enclosureEndWithin]]]
        [ENDPRLSPACING endPrlSpace PRL endPrl]
        [ENDTOEND endToEndSpace [oneCutSpace twoCutSpace]
            [EXTENSION extension [wrongDirExtension]]
            [OTHERENDWIDTH otherEndWidth]]
        [MAXLENGTH maxLength | MINLENGTH minLength [TWO SIDES]]
        [EQUALRECTWIDTH]
        [PARALLELEDGE [SUBTRACTEOLWIDTH] parSpace
            WITHIN parWithin [PRL prl]
            [MINLENGTH minLength] [TWO EDGES]
            [SAMEMETAL] [NONEOLCORNERONLY]
            [PARALLELSAMEMASK]]
        [ENCLOSECUT [BELOW | ABOVE] encloseDist
            CUTSPACING cutToMetalSpace [ALLCUTS]]
        | TOCONCAVECORNER [MINLENGTH minLength]
            [MINADJACENTLENGTH
                {minAdjLength | minAdjLength1 minAdjLength2}]
        | TONOTCHLENGTH notchLength
    }
    ;...";
```

Where:

The keywords SPACING (including ENDOFLINE and WITHIN), PARALLELEDGE, and TWOEDGES are the same as the existing LEF syntax.

```
ENCLOSECUT [BELOW | ABOVE] encloseDist CUTSPACING
cutToMetalSpace [ALLCUTS]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates that the rule only applies if there is a cut below or above this metal that is less than *encloseDist* from the end-of-line edge and the cut-edge to metal-edge space beyond the EOL edge less than *cutToMetalSpace*. If there is more than one cut connecting the same metal shapes above and below, only one cut needs to meet this rule, unless *ALLCUTS* is also specified which checks the rule against all of the cuts. (See [Figure 1-166](#) on page 330, and [Figure 1-168](#) on page 332.).

If you specify *BELOW*, the *encloseDist* and *cutToMetalSpace* is checked for the cut layer below this routing layer. If you specify *ABOVE*, they are checked for the cut layer above this routing layer. If you specify neither, the rule applies to both adjacent cut layers.

*Type:* Float, specified in microns (for both values)

ENCLOSUREEND *enclosureEndWidth* [WITHIN *enclosureEndWithin*]

Specifies that the EOL spacing rule applies to a EOL edge touching a via cut with length equal to the wire width that is less than *enclosureEndWidth*.

The *WITHIN* keyword specifies the within search distance for a EOL edge touching a via cut to be *enclosureEndWithin* instead of *eolWithin*.

*Type:* Float, specified in microns

ENDPRLSPACING *endPrlSpace* PRL *endPrl*

Specifies that *endPrlSpace* is applied between the end-of-line (EOL) edge to any neighbor wire with PRL greater than 0 but less than or equal to *endPrl*. Otherwise, *eolSpace* is applied to the end-to-line situation.

*Type:* Float, specified in microns

ENDTOEND *endToEndSpace* [*oneCutSpace* *twoCutSpace*]

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the two EOL spacings. For end-to-end situation when there is a parallel run length greater than 0 between the two EOL edges with *eolWithin* extension on the checking EOL edge, *endToEndSpace* is applied. Otherwise, *eolSpace* is applied to end-to-line situation.

An end-to-end situation with one of the EOL edge touching a via cut (and the other does not touching a via cut) must have spacing greater than or equal to *oneCutSpace*. If both EOL edges touching a via cut, the spacing must be greater than or equal to *twoCutSpace*. These additional spacing values must be specified along with **WITHCUT**.

*Type:* Float, specified in microns

**EQUALRECTWIDTH**

Indicates that if the length of the EOL edge is larger than the wire width, the rule does not apply. If there are multiple EOL statements with the **EQUALRECTWIDTH** keyword for a given layer, they must all have the **EQUALRECTWIDTH** keyword.

**EXACTWIDTH**

Specifies that this end-of-line spacing rule only applies if the edge width/length is exactly equal to *eolWidth*. This construct would typically make sense only if **WIDTHTABLE** or **SPANLENGHTHABLE** is also defined on the layer, and the end-of-line spacing rule will only be triggered for one of the discrete widths.

**EXTENSION** *extension* [*wrongDirExtension*]

Specifies the extension on two EOL edges facing each other as *extension*. The *wrongDirExtension* variable specifies the extension on two EOL edges parallel to the specified direction in **DIRECTION** on a Manhattan routing layer. The end to end spacing is applied as if the extensions are part of the EOL edges. When only the extensions have projected parallel run length, but the EOL edges do not, the end to end spacing still applies.

*Type:* Float, specified in microns

**EXCEPTEXACTWIDTH** *exactWidth otherWidth*

Specifies that the end-of-line spacing rule does not apply if the EOL edge has width/length exactly equal to *exactWidth* and the width/length of the other neighbor edge is less than or equal to the *otherWidth*. See [Figure 1-174](#) on page 337.

*Type:* Float, specified in microns

**FILLCONCAVECORNER** *fillTriangle*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the EOL to any neighbor concave corners should be filled by a triangle with value of *fillTriangle* along the edges to form the corner before spacing is checked. See [Figure 1-173](#) on page 337.

*Type:* Float, specified in microns

MAXLENGTH *maxLength*

Indicates that if the EOL is more than *maxLength* along both sides, the rule does not apply. (See [Figure 1-167](#) on page 331.)

*Type:* Float, specified in microns

MINADJACENTLENGTH {*minAdjLength* | *minAdjLength1* *minAdjLength2*}

Specifies that the EOL to concave corner spacing rule applies only if both of the edge lengths forming the concave corner are greater than *minAdjLength*, or one of the edge length is greater than *minAdjLength1* and the other edge length is greater than *minAdjLength2*. See [Figure 1-171](#) on page 335.

*Type:* Float, specified in microns

MINLENGTH *minLength*

Indicates that if the EOL length is less than *minLength* along both sides, the rule does not apply.

*Type:* Float, specified in microns

MINLENGTH *minLength* [TWOEDGES]

Indicates that if the EOL length is less than *minLength* along the side, then any parallel-edge on that side is ignored, and the rule may not apply. If TWOEDGES is specified, the EOL rule applies only if there are two parallel edges on each side of the EOL edge that meet the PARALLELEDGE.

*Type:* Float, specified in microns

NONEOLCORNERONLY

Specifies that the parallel edge neighbor wire must contain a corner that does not belong to another EOL edge.

OPPOSITEWIDTH *oppositeWidth*

Indicates that the rule applies only if a wire beyond the end of the line edge has a perpendicular span to the EOL edge less than *oppositeWidth*.

*Type:* Float for all parameters, specified in microns

OTHERENDWIDTH *otherEndWidth*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates that the rule only applies if the width of the other wire is less than the *otherEndWidth*.

PARALLELEDGE [SUBTRACTEOLWIDTH] *parSpace* WITHIN *parWithin*

Indicates that the EOL rule applies only if there is a parallel-edge less than *parSpace* away that is also less than *parWithin* from the end of the wire.

PARALLELSAMEMASK

Specifies that the end-of-line spacing rule only applies if there is a side neighbor having the same-mask of the wire with the EOL edge found in the side search window. Having a different-mask side neighbor in the search is irrelevant.

PRL *prl*

Indicates that the end-of-line (EOL) spacing rule applies only if the side neighbor and the neighbor beyond the EOL edge have parallel run length greater than the specified *prl* after extending by the WITHIN values. When there are two nearby EOL edges on opposite or orthogonal sides, their search window for the side neighbor could be merged when each of the individual EOL edges has a neighbor beyond the EOL within *eolSpace*, and the PRL is determined by the merged window.  
*Type:* Float, specified in microns

SAMEMASK

Indicates that the EOL spacing rule only applies between objects on the same mask.

**Note:** The SAMEMASK keyword can only be specified with MINLENGTH, OPPOSITEWIDTH, and ENDTOEND, or PARALLELEDGE keywords.

SAMEMETAL

Specifies that the EOL spacing rule applies only if the neighbors on the side(s) of and beyond the EOL edge are connected as same-metal. In addition, if a side neighbor wire covering the entire length of *parWithin*, only if it is non-zero, then it will be exempted as a valid neighbor.

SUBTRACTEOLWIDTH

Indicates that the *parSpace* value should be subtracted by the width of the EOL edge to define the distance required to search for a parallel neighbor edge.

TOCONCAVECORNER

Specifies that the spacing between a EOL edge with width less than *eolWidth* and a concave corner of neighbor wires must be greater than or equal to *spacing*.  
*Type:* Float, specified in microns

TONOTCHLENGTH *notchLength*



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the end-of-line spacing to a notch with length less than *notchLength*. See [Figure 1-175](#) on page 338.

*Type:* Float, specified in microns

TWOSIDES

Indicates that the rule applies only when the EOL length is greater than and equal to *minLength* along both sides. In other words, if the EOL length is less than *minLength* along any one side, the rule does not apply.

WITHCUT [CUTCLASS *cutClass*] [ABOVE] *withCutSpace*

Specifies the EOL spacing, *withCutSpace*, only applies if a EOL edge touches a via cut in either below or above cut layer.

*Type:* Float, specified in microns

The CUTLCASS keyword defines the EOL spacing rule for certain *cutClass* only. If CUTCLASS is defined in the routing layer, then CUTCLASS must be specified in the SPACING statement.

The ABOVE keyword defines the EOL spacing rule only applies if a EOL edge touches a via cut in above cut layer.

WITHIN *eolWithin* [*wrongDirWithin*]

The *wrongDirWithin* variable specifies the within distance if a EOL edge is parallel to the specified direction in DIRECTION on a Manhattan routing layer, while *eolWithin* is the within distance of a EOL edge perpendicular to that direction.

*Type:* Float, specified in microns

WRONGDIRSPACING *wrongDirSpace*

Specifies an optional end-of-line spacing for a EOL edge parallel to the specified direction in DIRECTION on a Manhattan routing layer to be *wrongDirSpace*.

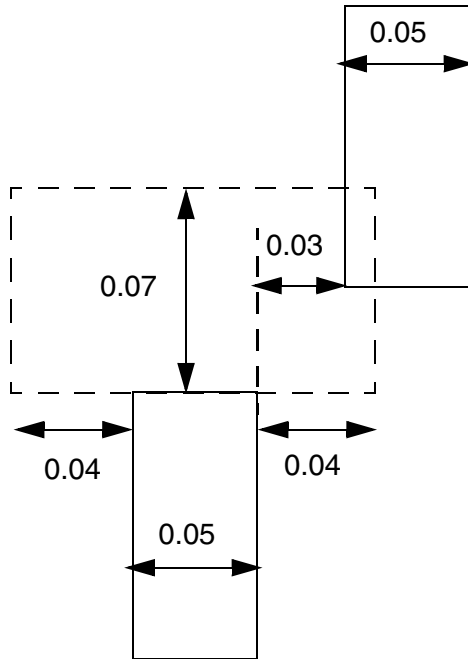
*Type:* Float, specified in microns

## EOL Spacing Rule Examples

The following example illustrates EOL spacing with exact width:

```
PROPERTY LEF58_WIDHTABLE
  "WIDHTABLE 0.05 0.10 ... ; " ;
SPACING 0.09 ENDOFLINE 0.051 WITHIN 0.02 ;
PROPERTY LEF58_SPACING "
  SPACING 0.07 ENDOFLINE 0.10 EXACTWIDTH WITHIN 0.04 ; " ;
```

**Figure 1-158 Example of Spacing Rule with EXACTWIDTH**



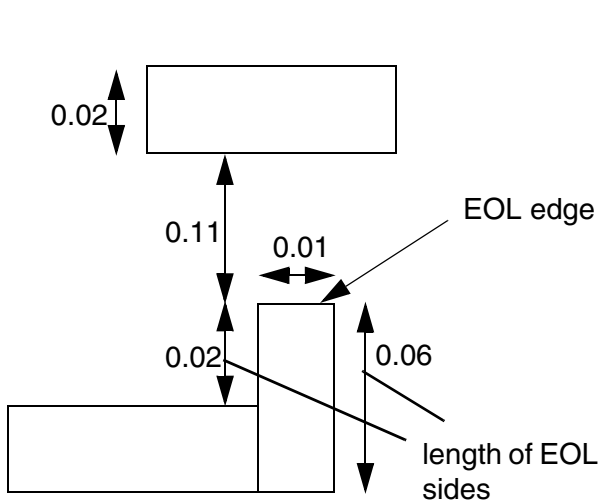
a) OK, the 0.051 EOL width statement does not apply since the neighbor wire is outside within of 0.02, and the 0.1 EOL width statement also does not apply since the edge width/length of 0.05 is not exactly equal to 0.10.

# LEF/DEF 5.8 Language Reference

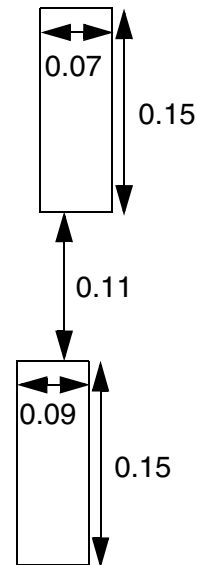
## LEF Syntax

**Figure 1-159 EOL Spacing Rule Illustrations**

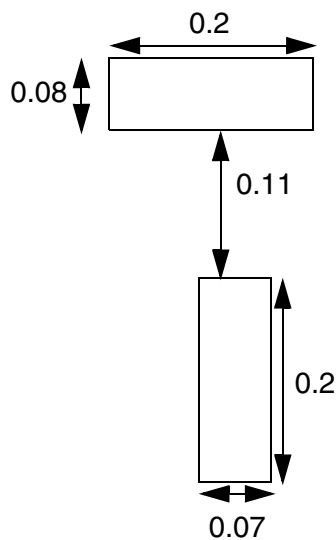
```
PROPERTY LEF58_SPACING "SPACING 0.12 ENDOFLINE 0.1
OPPOSITEWIDTH 0.08 WITHIN 0.05 MINLENGTH 0.06 ;" ;
```



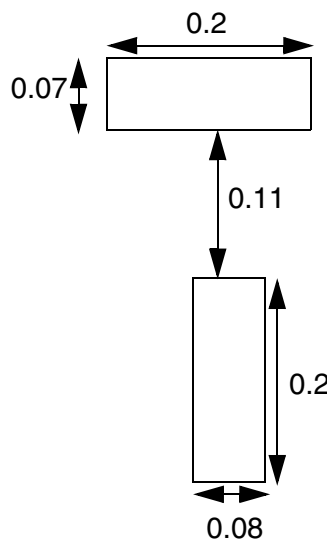
a) Violation, the length on the right side  $\geq 0.06$ , and the opposite wire width  $< 0.08$ . OK if TWOSIDES is used



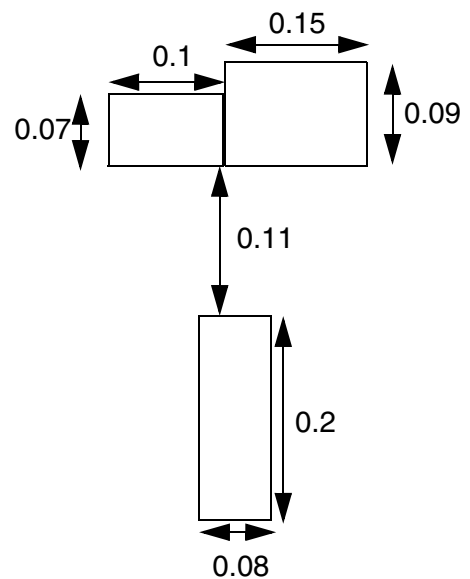
b) OK, the opposite wire has a perpendicular span (to the EOL edge) of  $0.15 \geq 0.08$



c) OK, opposite wire width  $\geq 0.08$ , and the rule does not apply



d) Violation, swapping the wire width of the wires becomes a violation



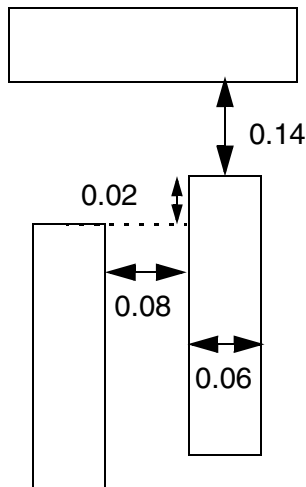
e) Violation, if part of the opposite wire width  $< 0.08$  (the left segment), it is a violation

## LEF/DEF 5.8 Language Reference

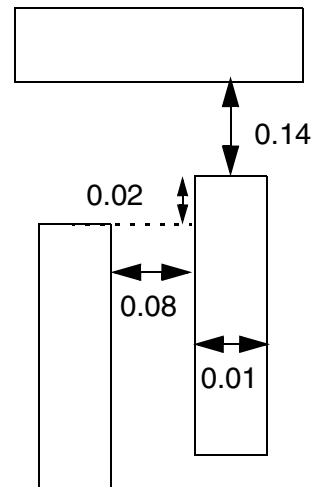
### LEF Syntax

**Figure 1-160 EOL Spacing Rule Illustrations**

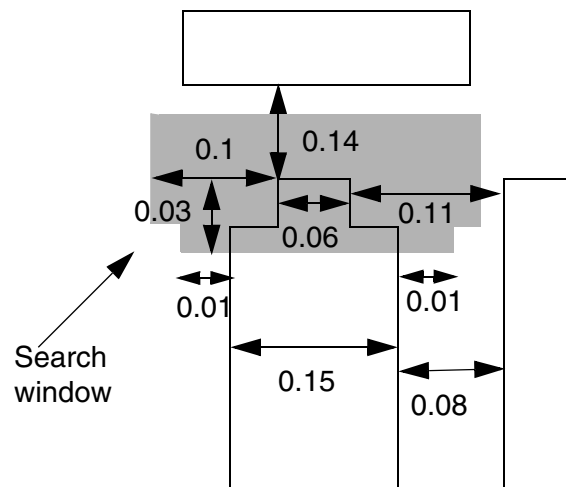
```
PROPERTY LEF58_SPACING "SPACING 0.15 ENDOFLINE 0.1 WITHIN 0.05  
PARALLELEDGE SUBTRACTEOLWIDTH 0.16 WITHIN 0.03 ;" ;
```



a) Violation, the top wire needs to be 0.15 away to avoid the EOL violation

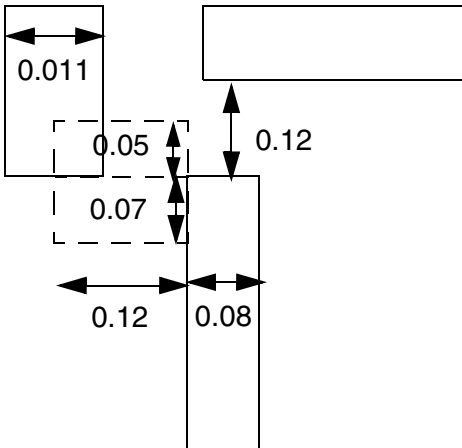


b) OK, the combined EOL wire width & the gap of the parallel edge is 0.17 ( $\geq 0.16$ ), and the EOL rule does not apply

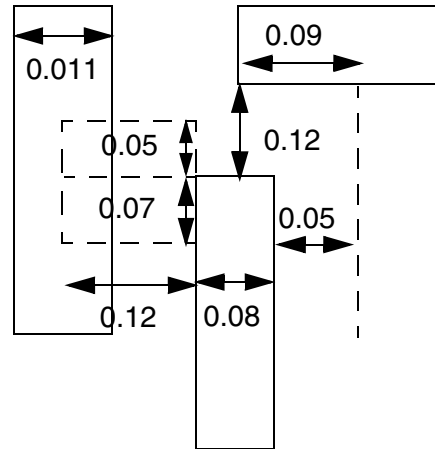


d) Okay, the parallel edge is searched by extending the outline of the wire dynamically based on the wire width. Hence, 0.1 ( $0.16 - 0.06$ ) on top, and 0.01 ( $0.16 - 0.15$ ) on bottom.

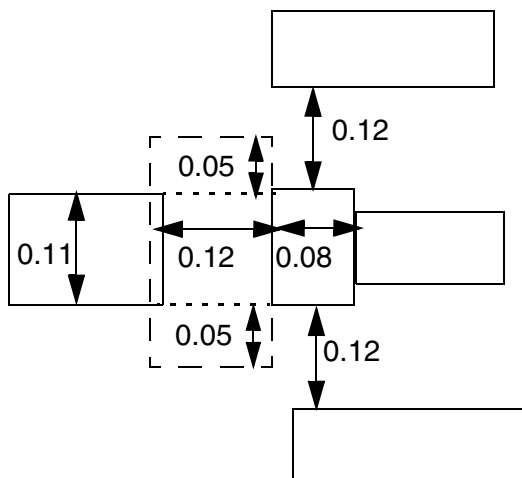
Figure 1-161 EOL Spacing Rule Illustrations



a) OK, the left neighbor only has PRL of 0.05 ( $\leq 0.1$ ). Violation if PRL is omitted.



b) OK, the top neighbor beyond the EOL edge has PRL of 0.09 ( $\leq 0.1$ ). Violation if PRL is omitted.



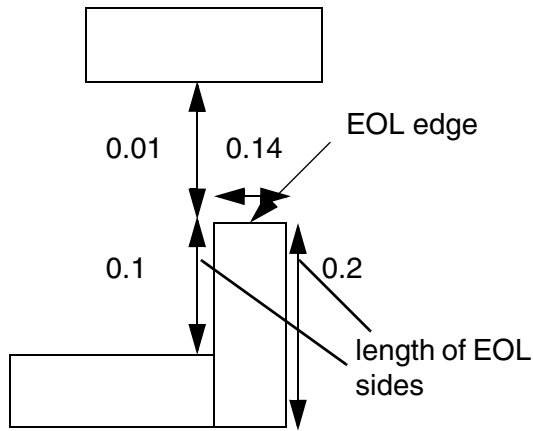
c) Violation, if the side neighbor search window is considered for the top and bottom 0.08 EOL individually, PRL would be 0.07 to not trigger the rule. However, the correct search window should be merged between these two EOL edges as shown in the dotted window, and PRL is 0.11.

```
PROPERTY LEF58_SPACING "  
    SPACING 0.13 ENDOFLINE 0.1 WITHIN 0.05  
    PARALLELEDGE 0.12 WITHIN 0.07  
    PRL 0.1 ;" ;
```

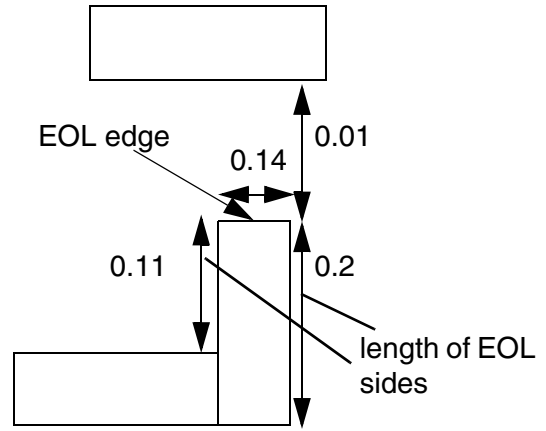
Figure 1-162 on page 326 illustrates the following EOL spacing rule:

```
PROPERTY LEF58_SPACING  
    "SPACING 0.10 ENDOFLINE 0.15 WITHIN 0.05 ENDTOEND 0.12  
    MINLENGTH 0.11 TWOSIDES ;" ;
```

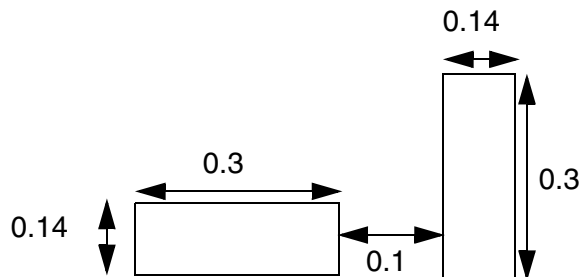
**Figure 1-162 EOL Spacing Rule Illustrations**



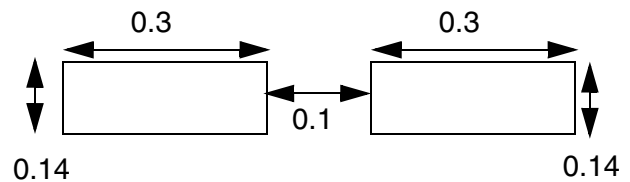
a) OK, the length of the left EOL side  $< 0.11$ , and the rule does not apply. Violation without TWOSIDES



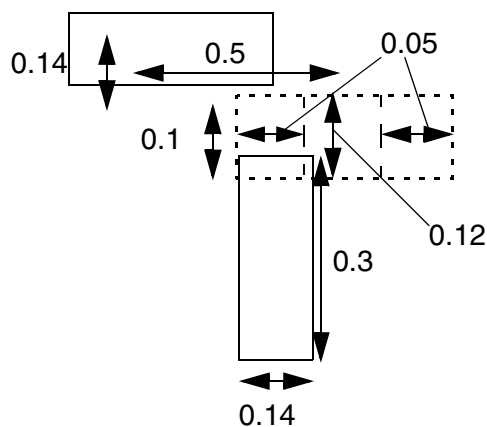
b) Violation, the length of both of the sides  $\geq 0.11$ , and EOL spacing  $< 0.1$



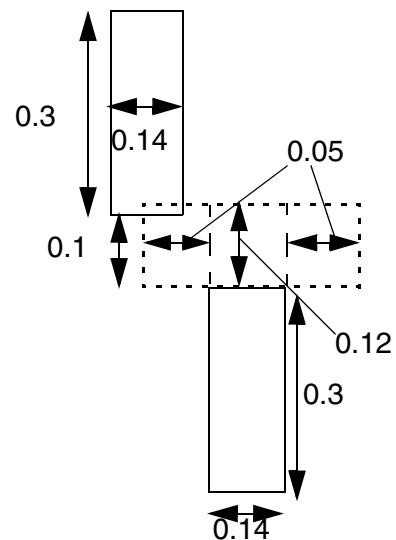
c) OK, the right edge is not a EOL edge, and 0.1 is needed



d) Violation, this is an end-to-end situation, and EOL spacing  $< 0.12$



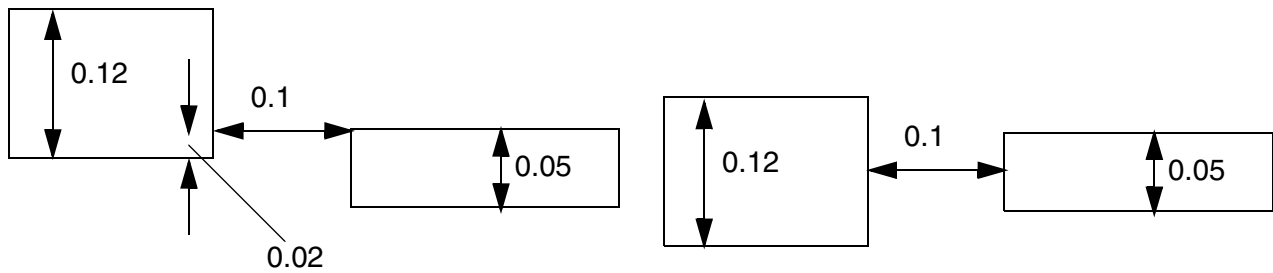
e) OK, it fulfills the end-to-line EOL spacing of 0.1, and the 2 EOL edges do not have common parallel run length such that end-to-end EOL spacing is ignored.



f) Violation, it does not satisfy end-to-end EOL spacing

**Figure 1-163 EOL Spacing Rule Illustrations with ENDPRLSPACING**

```
PROPERTY LEF58_SPACING
  "SPACING 0.1 ENDOFLINE 0.06 WITHIN 0
    ENDPRLSPACING 0.12 PRL 0.03 ;" ;
```



a) Violation, ENDPRLSPACING of 0.12 fails.

b) OK, PRL of the EOL is 0.05 (> 0.03), only EOL spacing of 0.1 is required and is met.

# LEF/DEF 5.8 Language Reference

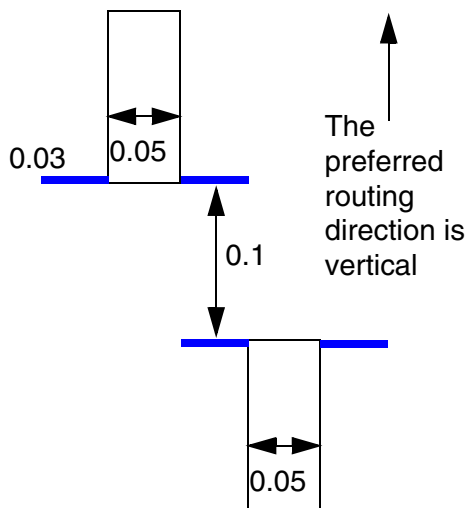
## LEF Syntax

**Figure 1-164 EOL Spacing Rule Illustrations**

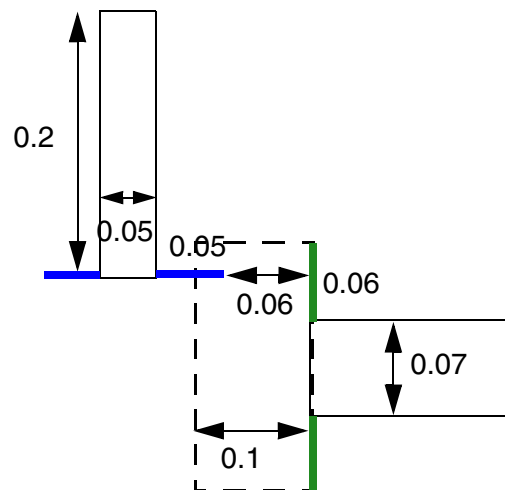
```

DIRECTION VERTICAL ;
PROPERTY LEF58_SPACING
  "SPACING 0.1 ENDOFLINE 0.08 WITHIN 0.05 0.06
    WITHOUT CUTCLASS VA 0.12 ENCLOSUREEND 0.14
    ENDTOEND 0.11 0.13 0.15
    EXTENSION 0.03 0.08 ; " ;

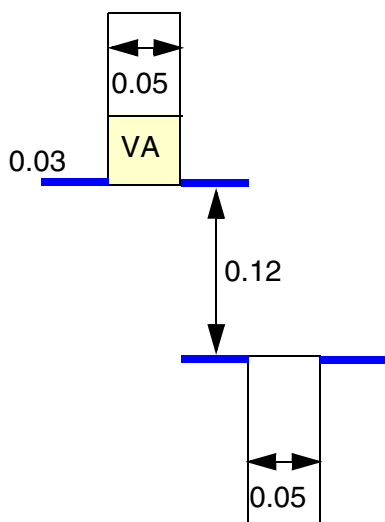
```



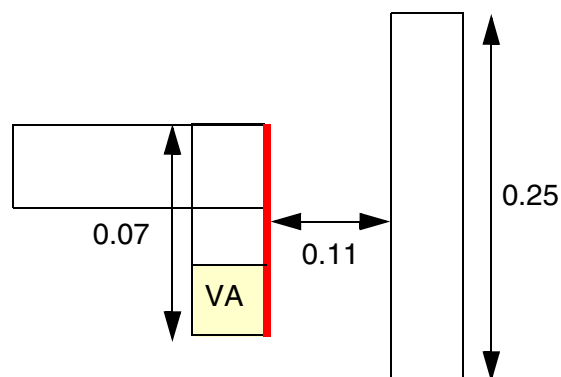
a) Violation, as long as the extensions have common projected run length, 0.11 spacing is needed



b) OK, the blue 0.05 is merely a within distance for searching the neighbor, 0.11 ( $\geq 0.1$ ) is the spacing between the vertical edge of to the green 0.06 within extension.



c) Violation, with touching one cut in one of the EOL edges, 0.13 spacing is needed

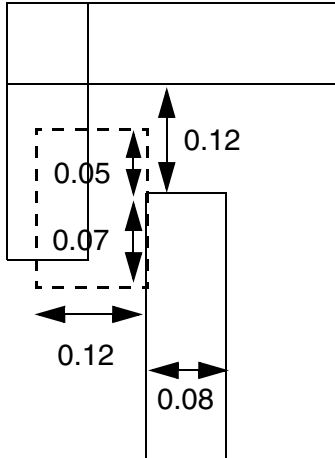


d) OK, the red edge is not an enclosure end edge, and 0.12 spacing does not apply. It is a EOL edge, and 0.1 spacing is met.

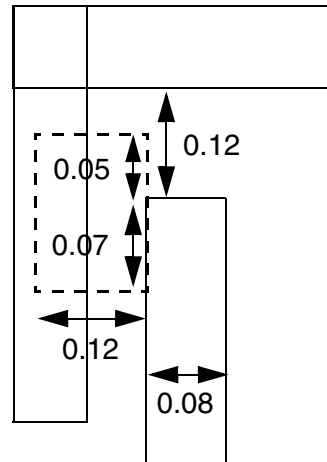


**Figure 1-165 EOL Spacing Rule Illustrations**

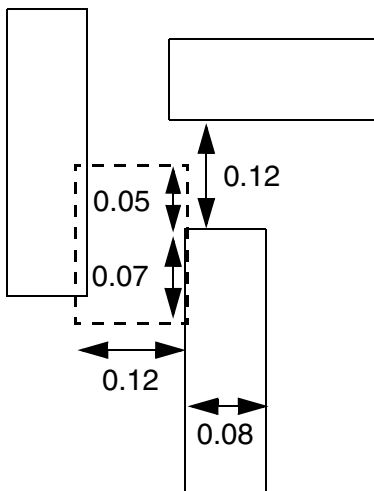
```
PROPERTY LEF58_SPACING
  "SPACING 0.13 ENDOFLINE 0.1 WITHIN 0.05
    PARALLELEDGE 0.12 WITHIN 0.07
    SAMEMETAL ; " ;
```



a) Violation, neighbor on the side is found and EOL spacing is 0.12 ( $\leq 0.13$ )

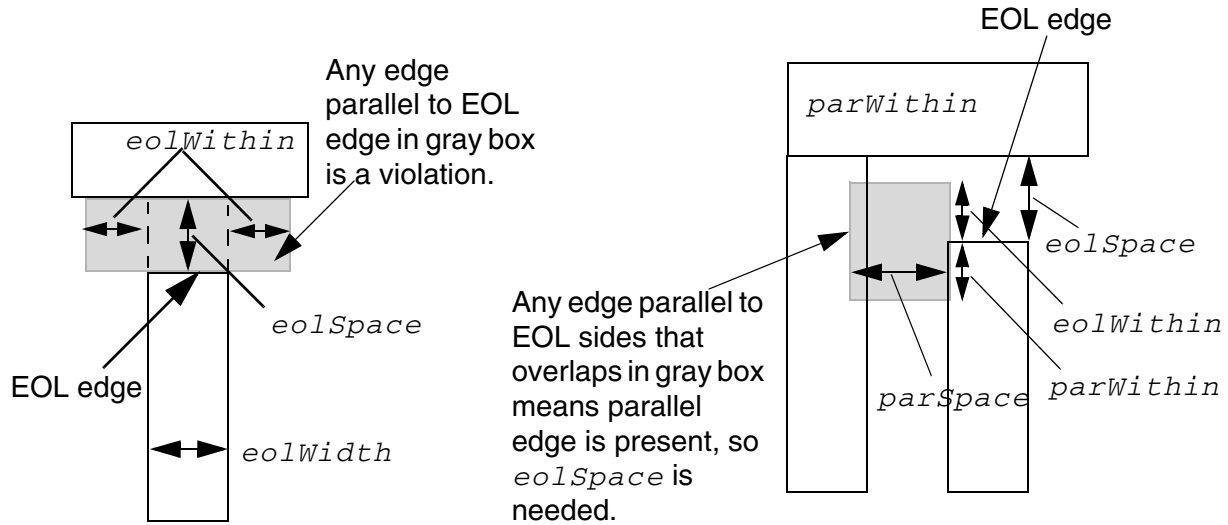


b) OK, the neighbor covers the entire length of the searching window, the rule does not apply



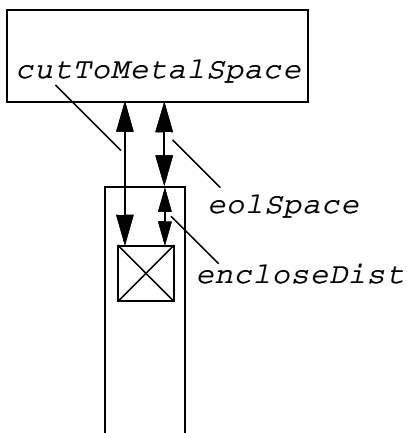
c) OK, the neighbors do not belong to same-metal. Violation, if SAMEMETAL is not specified.

Figure 1-166 EOL Spacing Rule Illustrations



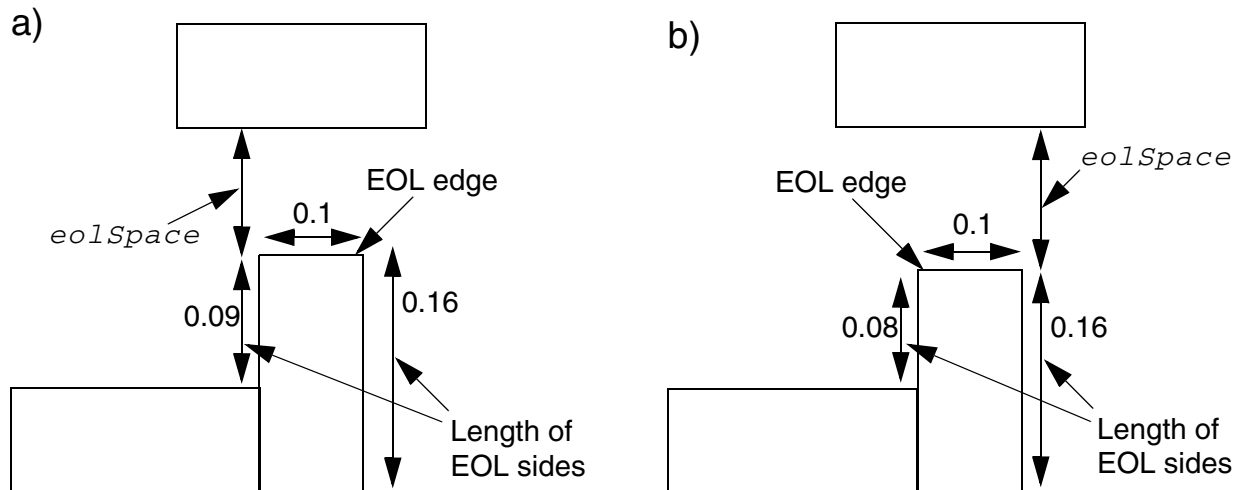
a) Basic EOL rule: EOL width < *eolWidth* requires  $\geq$  *eolSpace* beyond EOL edge to opposite edges to either side by < *eolWithin* distance (gray box).

b) EOL with PARALLELEDGE is triggered only if an EOL side also has a parallel edge < *parSpace* and < *eolWithin* above, or < *parWithin* below, EOL edge (gray box). If *minLength* is given, and the EOL side is < *minLength*, the parallel edge does not matter, and no extra space is needed.



c) EOL with ENCLOSECUT rule is triggered if EOL edge has a cut edge that has < *encloseDist* enclosure, and the cut edge is < *cutToMetalSpace* from the metal beyond the EOL edge.

Figure 1-167 Examples of EOL Spacing Rule with MaxLength and MinLength



PROPERTY LEF58\_SPACING

"SPACING 0.12 ENDOFLINE 0.11 WITHIN 0.05 MAXLENGTH 0.08 ;" ;

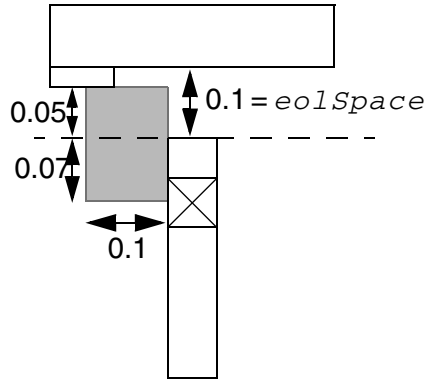
a) Both sides have a maximum length  $> 0.08$ , so do not check *eolSpace*.

b) One side has a maximum length  $\leq 0.08$ , so check *eolSpace*  $\geq 0.12$ .

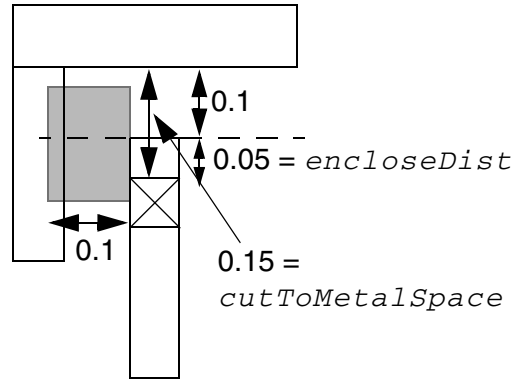
**Figure 1-168 Example of More Complex EOL Spacing Rule**

PROPERTY LEF58\_SPACING

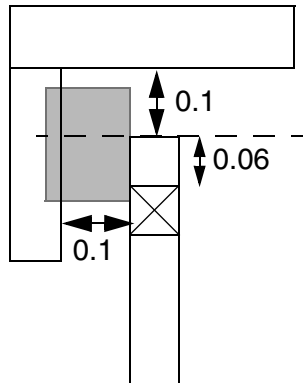
```
"SPACING 0.15 ENDOFLINE 0.10 WITHIN 0.05 PARALLELEDGE 0.10
  WITHIN 0.07 MINLENGTH 0.10 ENCLOSECUT BELOW 0.06 CUTSPACING 0.16 ;"
```



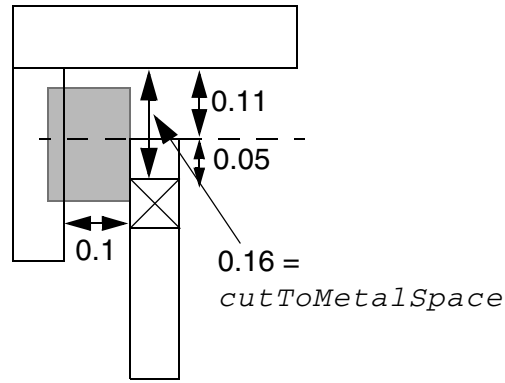
a) No violation. No parallel edge inside gray box ( $parSpace < 0.10$ ,  $eolWithin < 0.05$ , and  $parWithin < 0.07$ ), so do not check *eolSpace*.



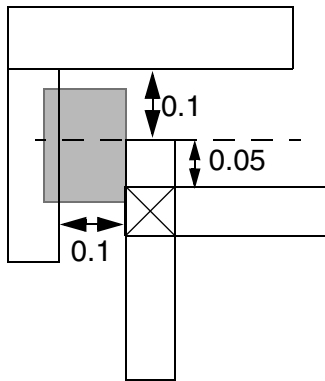
b) Violation. Meets min length, has parallel edge, cut enclosure  $encloseDist < 0.06$ ,  $cutToMetalSpace < 0.16$ , so needs 0.15 *eolSpace*.



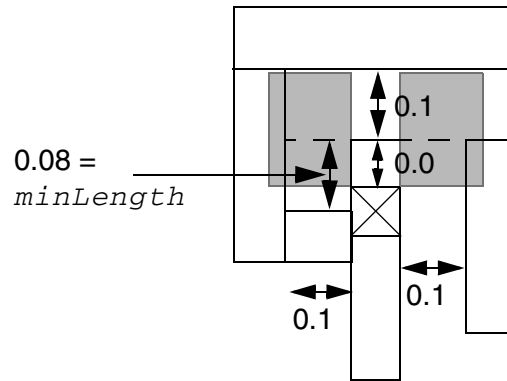
c) No violation. Meets min length, has parallel edge, but cut enclosure  $encloseDist \geq 0.06$ , so do not check *eolSpace*.



d) No violation. Meets min length, has parallel edge, cut enclosure  $encloseDist < 0.06$ , but  $cutToMetalSpace \geq 0.16$ , so do not check *eolSpace*.

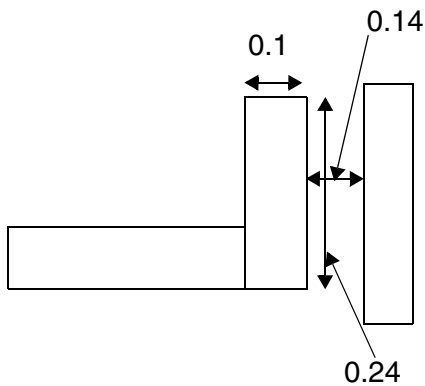


e) No violation. Does not meet  $minLength \geq 0.06$ , so do not check *eolSpace*.

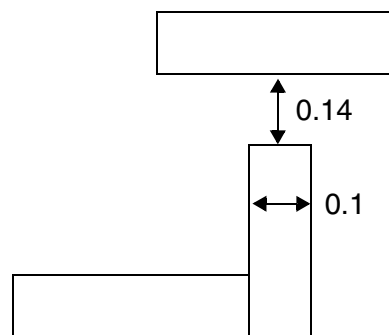


f) Violation. The  $minLength = 0.08$  means no left parallel edge, but the right side has a parallel edge, so needs 0.15 *eolSpace* or *cutToMetalSpace*  $\geq 0.16$ .

**Figure 1-169 Example of EOL Spacing Rule with EQUALRECTWIDTH**



a) No violation. The 0.24 edge is not a valid EOL edge with the `EQUALRECTWIDTH` keyword since its length is not the same as the wire width of 0.1.



b) Violation. The 0.1 edge is a valid EOL edge, even with the `EQUALRECTWIDTH` keyword, and EOL spacing is  $< 0.15$ .

PROPERTY LEF58\_SPACING

"SPACING 0.15 ENDOFLINE 0.25 WITHIN 0.10 EQUALRECTWIDTH;

## LEF/DEF 5.8 Language Reference

### LEF Syntax

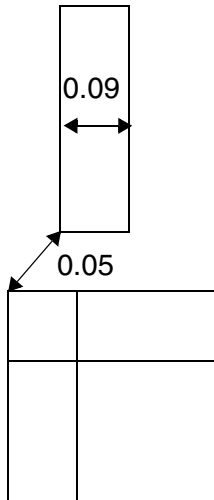
---

- The following example indicates that the 0.15  $\mu\text{m}$  EOL spacing is applied to an EOL edge less than 0.10  $\mu\text{m}$  to objects on the same mask in the routing layer. This statement does not apply to objects on different masks, and a separate EOL rule without `SAMEMASK` can be defined to an EOL spacing for different mask objects.

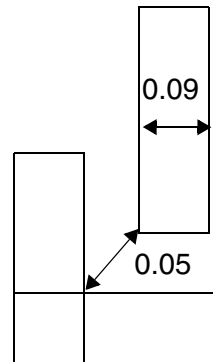
```
PROPERTY LEF58_SPACING
    "SPACING 0.15 ENDOFLINE 0.10 WITHIN 0.0 SAMEMASK ; " ;
```

**Figure 1-170 Illustration of Spacing Rule with TOCONCAVECORNER**

```
PROPERTY LEF58_SPACING
    "SPACING 0.06 ENDOFLINE 0.1 TOCONCAVECORNER ; " ;
```



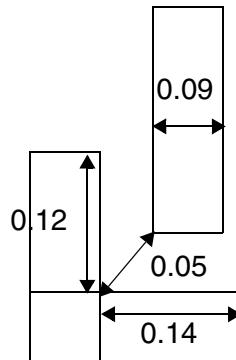
a) OK, the spacing applies to concave corner only



b) Violation, the spacing to the concave corner is 0.05 ( $\leq 0.06$ )

**Figure 1-171 Illustration of Spacing Rule with MINADJACENTLENGTH**

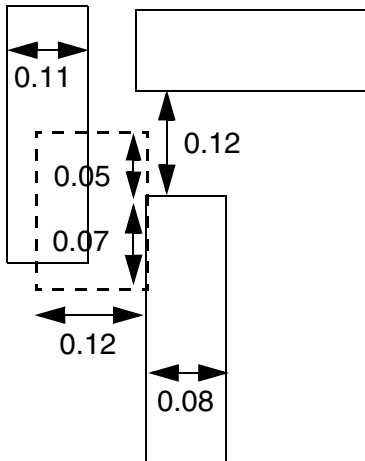
```
PROPERTY LEF58_SPACING
  "SPACING 0.06 ENDOFLINE 0.1
    TOCONCAVECORNER
    MINADJACENTLENGTH 0.12 ; " ;
```



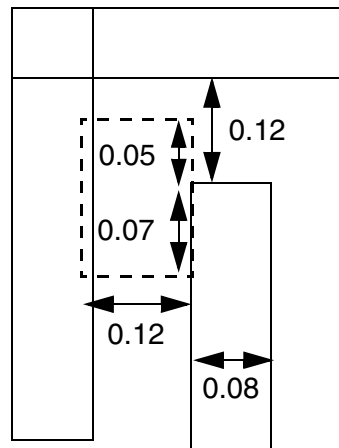
a) OK, the vertical edge forming the concave corner is 0.12 ( $\leq 0.12$ ). It is a violation if MINADJACENTLENGTH is omitted.

**Figure 1-172 EOL Spacing Rule with NONEOLCORNERONLY**

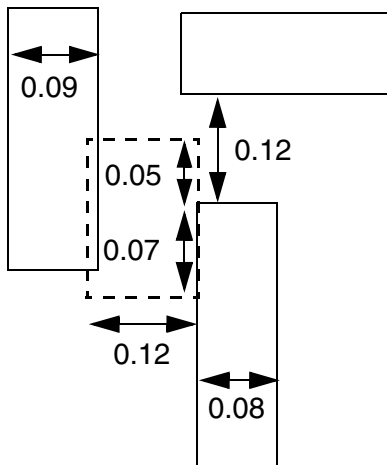
```
PROPERTY LEF58_SPACING  
  "SPACING 0.13 ENDOFLINE 0.1 WITHIN 0.05  
    PARALLELEDGE 0.12 WITHIN 0.07  
    NONEOLCORNERONLY ; " ;
```



a) Violation, a neighbor corner is found and EOL spacing is 0.12 ( $\leq 0.13$ ).



b) OK, the neighbor does not have a corner in the search window

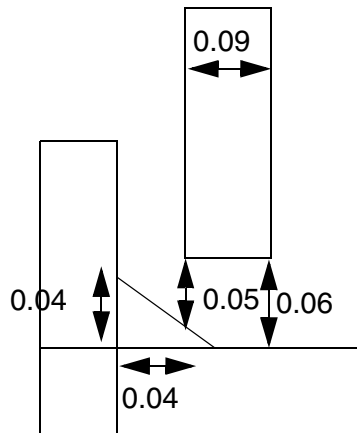


c) OK, the neighbor corner belonging to another EOL edge



**Figure 1-173 EOL Spacing Rule with FILLCONCAVECORNER**

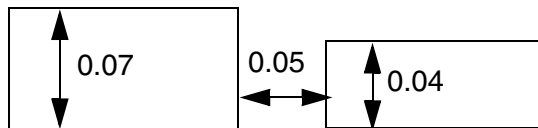
```
PROPERTY LEF58_SPACING "
    SPACING 0.06 ENDOFLINE 0.1 WITHIN 0
    FILLCONCAVECORNER 0.04 ; " ;
```



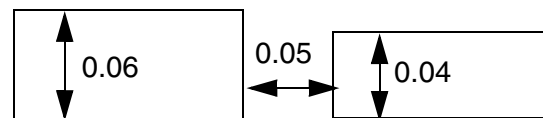
a) Violation, spacing is only 0.05 to the triangular fill with length of 0.04 along the edges of the concave corner

**Figure 1-174 EOL Spacing Rule with EXCEPTEXACTWIDTH**

```
SPACING 0.05 ;
PROPERTY LEF58_SPACING "
    SPACING 0.06 ENDOFLINE 0.1 WITHIN 0
    EXCEPTEXACTWIDTH 0.07 0.05 ; " ;
```



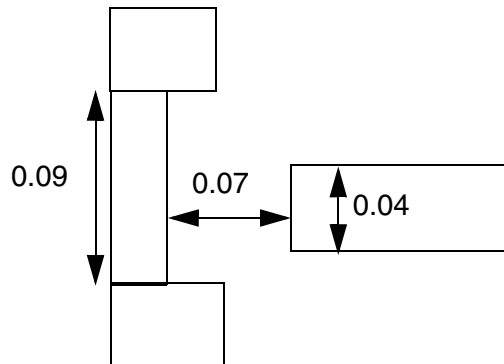
a) OK, the width of the left EOL edge is exactly equal to 0.07 while the right edge has width less than 0.05 to exempt the end-of-line spacing rule



b) Violation, there is no EOL edge with width exactly equal to 0.07

**Figure 1-175 EOL Spacing Rule with TONOTCHLENGTH**

```
PROPERTY LEF58_SPACING "  
    SPACING 0.08 ENDOFLINE 0.05  
    TONOTCHLENGTH 0.1 ; "
```



a) Violation, the EOL edge is 0.07 away from a notch with length of 0.09 (< 0.1)

### **Type Rule**

A type rule can be used to further classify a routing layer.

You can create a type rule by using the following property definition:

```
TYPE ROUTING;  
    PROPERTY LEF58_TYPE  
        "TYPE {POLYROUTING | MIMCAP | HIGHR};"
```

Where:

HIGHR	Specifies that the layer is a high resistance routing layer that is not used as a regular routing layer.
POLYROUTING	Indicates that the polysilicon layer should be considered as a routing layer. Polysilicon layers provide extra routing resources for designs with limited metal routing layers.
MIMCAP	Indicates that the layer is a mimcap layer. A mimcap layer is a metal layer that is not to be used as a routing layer.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

The `POLYROUTING` and `MIMCAP` keywords are mutually exclusive; you cannot specify them together.

#### **Convex Corners Spacing Rule**

A convex corners spacing rule can be used to define spacing between two convex corners of two parallel wires

You can create convex corners spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING
    "SPACING spacing CONVEXCORNERS EXTENSION sideExt orthogonalExt
        [ SAMESIDE extension
            | SINGLE edgeForwardExt cornerForwardExt cornerBackwardExt
                SPANLENGTH spanLength OPPOSITEWIDTH oppWidth
                OPPOSITEEXTENSION oppSideExt
                    oppForwardExt1 oppForwardExt2 ]
        ; " ;
```

Where:

```
SPACING spacing CONVEXCORNERS EXTENSION sideExt orthogonalExt
```

Specifies that the spacing between two convex corners of two parallel wires in the opposite sides facing each other must be greater than or equal to *spacing* in the direction of the parallel wires, if the window formed by the corners has exactly two minimum width neighbor wires fully across the window. Also, all the wires, including the ones containing the corners, are exactly minimum spacing apart.

In addition, both of the search windows by extending side ways away from the two middle neighbor wires by *sideExt* and forward along the parallel wires by *orthogonalExt* from the corners do not contain any neighbor wires.

*Type:* Float, specified in microns

```
SAMESIDE extension
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the parallel wires are on the same side instead of facing each other, and the window formed by the corners should be extended by *extension* along the wire direction. In addition, if there is a neighbor wire touching the edge of the search windows from the convex corners that faces the two middle parallel wires, it is considered a valid neighbor. If a neighbor wire touches any of the other edges of the search windows, the wire is not counted as neighbor.

*Type:* Float, specified in microns

```
SINGLE edgeForwardExt cornerForwardExt cornerBackwardExt  
SPANLENGTH spanLength OPPOSITEWIDTH oppWidth  
OPPOSITEEXTENSION oppSideExt oppForwardExt1 oppForwardExt2
```

Specifies a forbidden zone of a single convex corner to any wires to be a *spacing* distance beyond a virtual edge from the corner and extending minimum spacing in the direction parallel to the edge that contains the corner, if all of the conditions described below are met.

The wire containing the corner must have width exactly equal to minimum width and span length greater than or equal to *spanLength*. There is no wire beyond the edge containing the corner by *edgeForwardExt*. A search window by extending side way away from the corner by *sideExt* and backward by *orthogonalExt* does not contain any neighbor wires. There is a neighbor wire with width less than or equal to the *oppWidth* fully abutted to a search window formed from the opposite corner by extending minimum spacing on the side and *cornerForwardExt* and *cornerBackwardExt* in forward and backward direction. A search window by extending sideways from the opposite side of that neighbor wire by *oppSideExt* and projected forward from the corner from *oppForwardExt1* to *oppForwardExt2* must overlap or touch a wire.

*Type:* Float, specified in microns

Figure 1-176 Illustration of Spacing Rule with CONVEXCORNERS

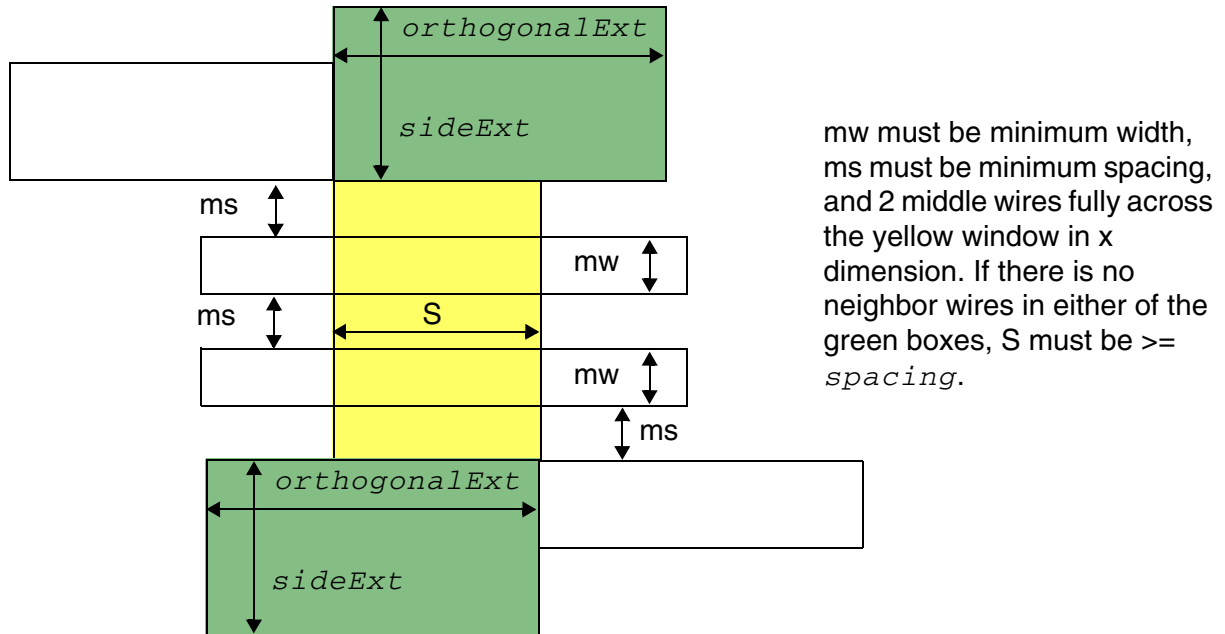
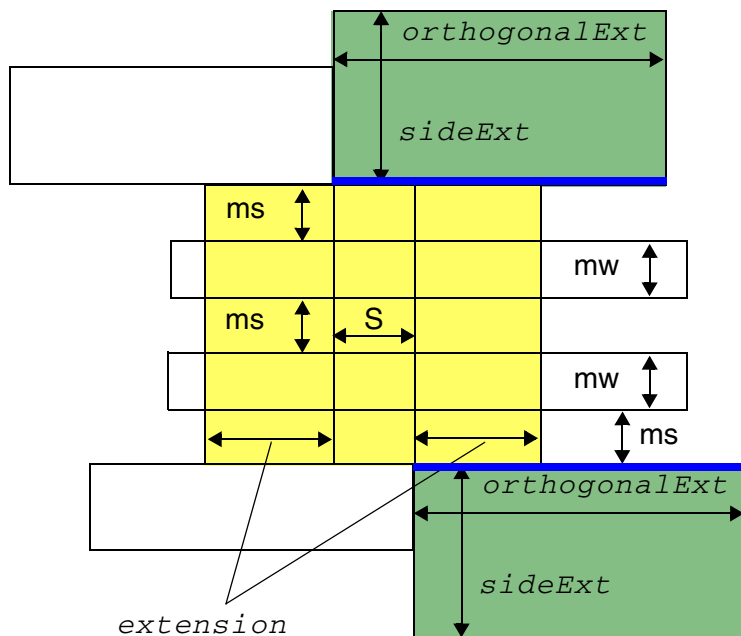


Illustration of SPACING *spacing* CONVEXCORNERS  
EXTENSION *sideExt* *orthogonalExt*

### Figure 1-177 Illustration of Spacing Rule with CONVEXCORNERS



mw must be minimum width, ms must be minimum spacing, and 2 middle wires fully across the yellow windows in x dimension. If there is no neighbor wires in either of the green boxes, S must be  $\geq$  spacing. Objects barely touches the blue edges are considered as neighbor.

Illustration of SPACING *spacing* CONVEXCORNERS  
EXTENSION *sideExt orthogonalExt* SAME SIDE extension

**Figure 1-178 Illustration of Spacing Rule with CONVEXCORNERS**

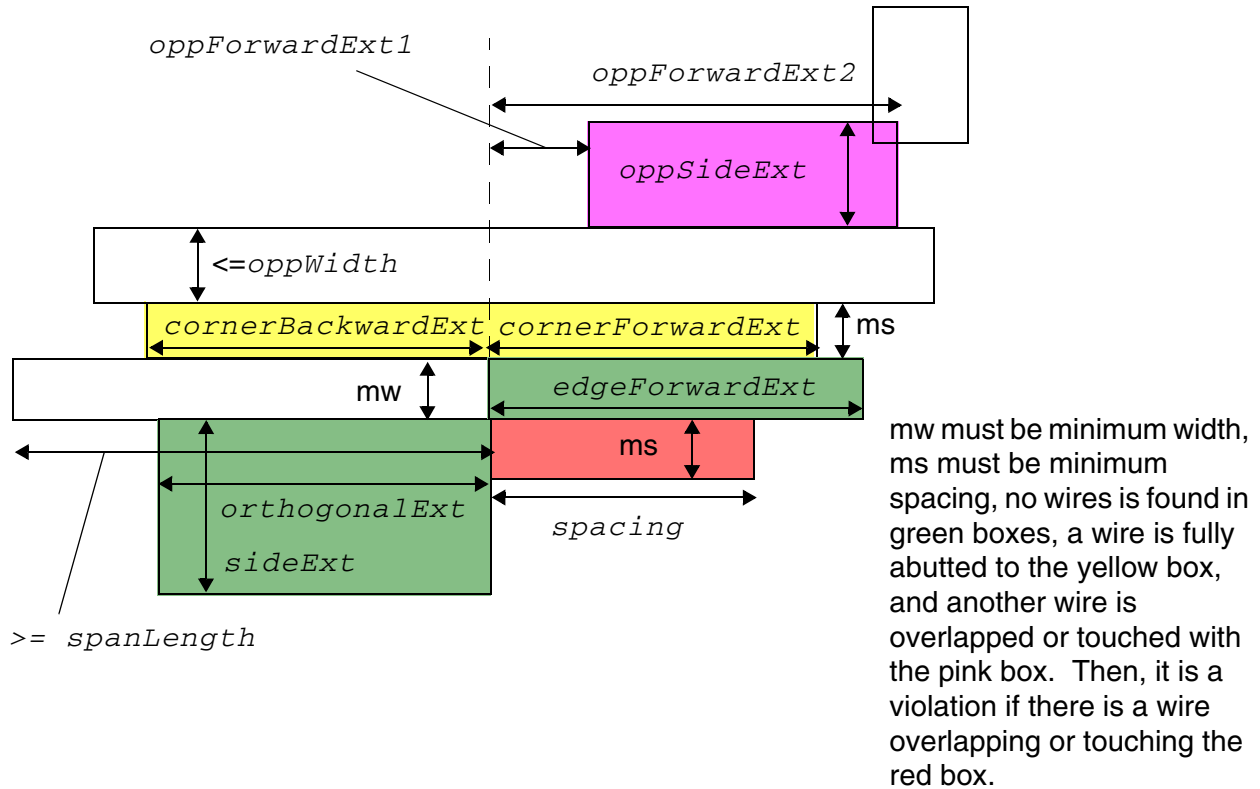


Illustration of SPACING *spacing* CONVEXCORNERS

```
EXTENSION sideExt orthogonalExt
SINGLE edgeForwardExt cornerForwardExt
cornerBackwardExt
SPANLENGTH spanLength
OPPOSITEWIDTH oppWidth
OPPOSITEEXTENSION oppSideExt
oppForwardExt1 oppForwardExt2
```

### **Span Length Enclosure Spacing Rule**

A span length enclosure spacing rule can be used to specify spacing on an edge with a certain span length and edge length having a cut with a certain extension on that edge to any wires with given PRL.

You can create a span length enclosure spacing rule by using the following property definition:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_SPANLENGTHENCLOSURESPACING
  "SPANLENGTHENCLOSURESPACING spacing
    SPANLENGTH spanLength MINLENGTH minLength
    {FROMABOVE|FROMBELOW} ENCLOSURE enclosure
    PRL prl
  ; " ;
```

Where:

```
SPANLENGTHENCLOSURESPACING spacing
  SPANLENGTH spanLength MINLENGTH minLength
  {FROMABOVE|FROMBELOW} ENCLOSURE enclosure
  PRL prl
```

Specifies the spacing of an edge with span length greater than or equal to *spanLength* to any parallel edge of a wire with a common parallel run length greater than *prl*, which must be zero or a positive value if:

- Both edges have length greater than *minLength* and
- The edge with span length greater than or equal to *spanLength* has a cut in the range of the common parallel run length with the neighbor edge from above or below this layer, depending on whether FROMABOVE or FROMBELOW is specified, with enclosure less than enclosure from that edge.

*Type:* Float, specified in microns

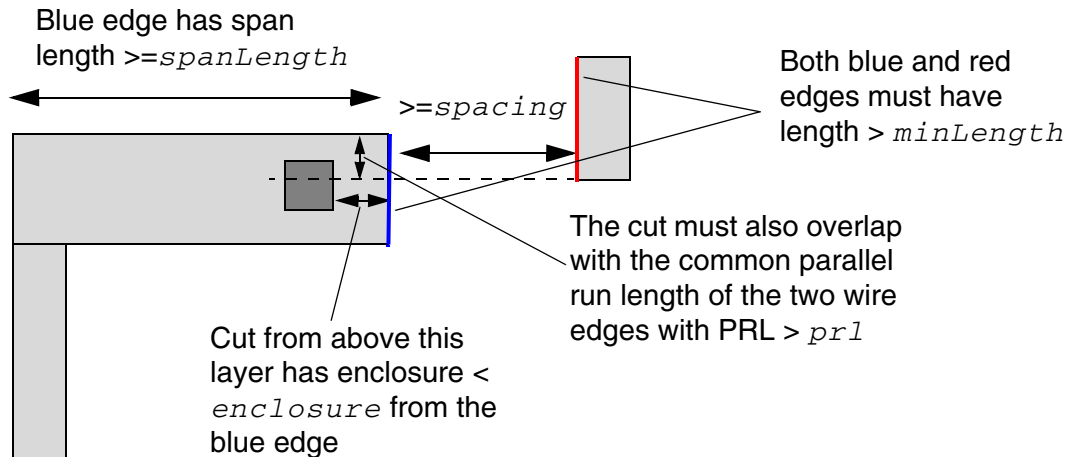
### Span Length Enclosure Spacing Rule Example

- The following example is an illustration of SPANLENGTHENCLOSURESPACING:

```
SPANLENGTHENCLOSURESPACING spacing
  SPANLENGTH spanLength MINLENGTH minLength
  FROMABOVE ENCLOSURE enclosure
  PRL prl
```



**Figure 1-179 Illustration of SPANLENGTHENCLOSURESPACING**



### Span Length Table Rule

A span length table rule can be used to specify all the allowable legal span lengths on the routing layer.

You can create a span length table rule by using the following property definition:

```
PROPERTY LEF58_SPANLENGTHTABLE
    "SPANLENGTHTABLE {spanLength} ... [MASK maskNum] [WRONGDIRECTION]
        [ORTHOGONAL length]
        [EXCEPTOTHERSPAN otherSpanlength]
        | {OTHERSPAN otherSpanLength
            [MINSPANLENGTH minSpanLength]...}
    ; " ;
```

Where:

<code>MASK maskNum</code>	Specifies that the span length rule applies only on the given mask of the current layer. <i>maskNum</i> must be a positive integer, and most applications support only the values 1, 2, or 3.
<code>ORTHOGONAL length</code>	Specifies that the length between two inside facing corners of a rectilinear object must be greater than or equal to the <i>length</i> .  <i>Type:</i> Float, specified in microns
<code>{OTHERSPAN otherSpanLength MINSPANLENGTH minSpanLength}...</code>	

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that if the span length on the orthogonal direction of the span length being checked is less than or equal to *otherSpanlength*, the span length being checked needs to be only greater than or equal to the *minSpanLength*. In addition, if the span length is smaller than *minSpanLength* but is exactly equal to one of the values in *spanLength*, it is also legal. If multiple OTHERSPAN and SPANLENGTH are specified, the first *otherSpanLength* that the current span length is less than or equal to would be applied. This construct must be defined on a layer with RECTONLY. In a polygon shape, the span length in the orthogonal direction could be ambiguous.

*Type:* Float, specified in microns

EXCEPTOTHERSPAN *otherSpanlength*

Indicates that the span length rule only applies if the span length on the perpendicular direction is greater than the specified *otherSpanlength*. This construct must be defined on a layer with RECTONLY. Otherwise, the span length in the orthogonal direction could be ambiguous in a polygon shape.

*Type:* Float, specified in microns

SPANLENGTHTABLE {*spanLength*}...

Specifies all of the allowable legal span lengths on the routing layer. All of the given span lengths are exact values, except for the last one, which is greater than equal to the value. All of the possible span lengths of an object in both directions should be checked against the given span length values. At most, two SPANLENGTHTABLE spacing tables can be defined - one with WRONGDIRECTION and the other without.

*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### WRONGDIRECTION

Specifies all of the allowable legal span lengths in the direction parallel to the specified direction in `DIRECTION` on a Manhattan routing layer.

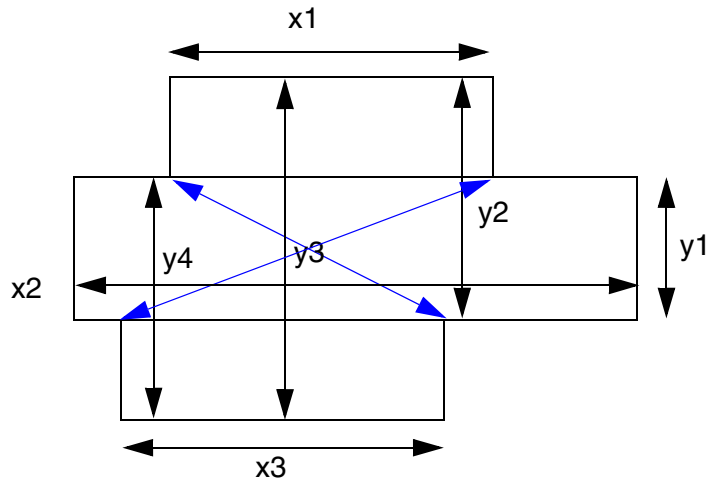
Note that using `WRONGDIRECTION` changes the interpretation of any wrong-way routing widths in the `DEF NETS` section. If `WRONGDIRECTION` is specified, then any wrong-way routing in the `DEF NETS` section will use the `WRONGDIRECTION` width for that layer unless the net or route has a `NONDEFAULTRULE` with a `WIDTH` greater than the `WRONGDIRECTION` width. But, the implicit default route-extension is still half of the preferred direction width.

Some older tools may not understand this behavior. Normally, they will still read/write and round-trip the DEF routing properly, but will not understand that the width is slightly larger for wrong-way routes. If these tools check wrong-way width, then the DRC rules may flag false violations. RC extraction with the wrong width will also flag errors, although wrong-way routes are generally short and the width difference is small, so the RC error is normally negligible.

#### Span Length Table Rule Examples

- The following example is an illustration of `SPANLENGHTABLE {spanLength}... ORTHOGONAL length` and `SPANLENGHTABLE {spanLength}... WRONGDIRECTION` with preferred direction being horizontal:

**Figure 1-180 Illustration of SPANLENGHTHABLE**

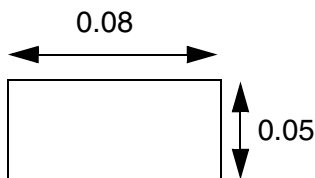


$y1$ ,  $y2$ ,  $y3$ , and  $y4$  should be checked against the given span lengths in SPANLENGHTHABLE without WRONGDIRECTION while  $x1$ ,  $x2$ , and  $x3$  should be checked against the span lengths in SPANLENGHTHABLE with WRONGDIRECTION. In addition, the length of the blue arrows should be checked against length.

- The following example is an illustration of SPANLENGHTHABLE with EXCEPTOTHERSPAN:

```
DIRECTION HORIZONTAL ;
PROPERTY LEF58_RECTONLY "
    RECTONLY ; " ;
PROPERTY LEF58_SPANLENGHTHABLE "
    SPANLENGHTHABLE 0.05 0.06 0.07 0.080 0.090 0.10 ;
    SPANLENGHTHABLE 0.07 0.10 WRONGDIRECTION
    EXCEPTOTHERSPAN 0.05 ; " ;
```

**Figure 1-181 Illustration of SPANLENGHTHABLE with EXCEPTOTHERSPAN**



a) OK, span length of 0.05 fulfills the first span length table, and span length of 0.08 is not checked because EXCEPTOTHERSPAN condition is not fulfilled. Violation if EXCEPTOTHERSPAN is omitted.

- The following example is an illustration of SPANLENGHTHABLE with OTHERSPAN:

```
DIRECTION HORIZONTAL ;
```

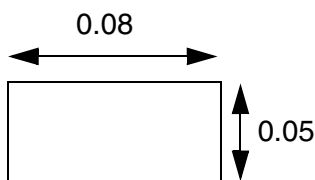
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_RECTONLY `
  RECTONLY ; ` ;
PROPERTY LEF58_SPANLENGHTHABLE `
  SPANLENGHTHABLE 0.04 0.05 0.06 0.07 0.080 0.090 0.10 ;
  SPANLENGHTHABLE 0.07 0.10 WRONGDIRECTION
  OTHERSPAN 0.04 MINSPANLENGTH 0.09
  OTHERSPAN 0.06 MINSPANLENGTH 0.08 ; ` ;
```

**Figure 1-182 Illustration of SPANLENGHTHABLE with OTHERSPAN**



a) OK, span length of 0.05 fulfills the first span length table, and span length of 0.08 is checked against OTHERSPAN of 0.06, which would pass the corresponding requirement of  $\geq 0.08$ .

### **Fill to Fill Spacing Rule**

A fill to fill spacing rule can be used to define spacing between metal fills.

You can create a fill to fill spacing rule by using the following property definition:

```
PROPERTY LEF58_FILLTOFILLSPACING
  "FILLTOFILLSPACING spacing ;" ;
```

Where:

**FILLTOFILLSPACING** Specifies the spacing between metal fills.  
*Type:* Float, specified in microns

### **Five Wires EOL Spacing Rule**

The five wires EOL spacing rule is used to define spacing constraints for five consecutive wires meeting certain conditions.

You can create a five wire EOL spacing rule by using the following property definition:

```
PROPERTY LEF58_FIVEWIRESEOLSPACING
  "FIVEWIRESEOLSPACING eolSpacing WITHIN eolWithin
  PRL prl
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
ENCLOSECUT {BELOW|ABOVE} encloseDist CUTWITHIN cutWithin  
NOMETALEOLEXTENSION eolSideExtension eolForwradExtension  
;...";
```

Where:

```
FIVEWIRESEOLSPACING eolSpacing WITHIN eolWithin  
PRL prl  
ENCLOSECUT {BELOW|ABOVE} encloseDist CUTWITHIN cutWithin  
NOMETALEOLEXTENSION eolSideExtension eolForwradExtension
```

Specifies that if there are five consecutive minimum default width wires that are minimum spacing apart and have a parallel run length greater than *prl*, it is a violation to have a neighbor wire overlapping with the region formed by the line end of the middle wire by extending *eolWithin* along the EOL edge and *eolSpacing* along the orthogonal direction. In addition, the following conditions must also be met to be a violation:

- There is a cut on below or above cut layer in case BELOW or ABOVE is specified on the middle wire within *cutWithin* of parallel wires and having enclosure less than *encloseDist* from the line end.
- There is no wire completely covering or touching the entire range along the orthogonal direction of the EOL of one of the two regions by extending *eolSideExtension* along the EOL corners and *eolForwradExtension* along the orthogonal direction.

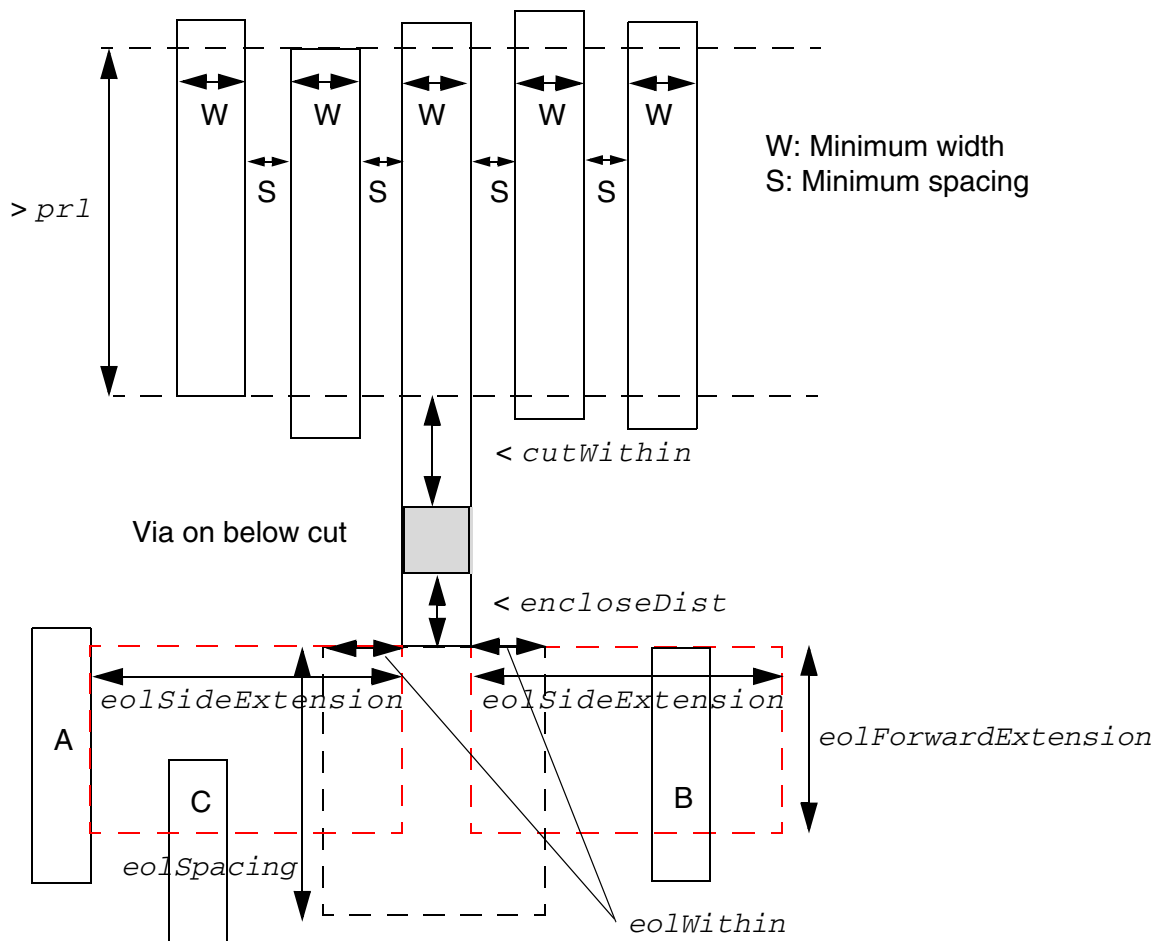
*Type:* Float, specified in microns

## Five Wires EOL Spacing Rule Example

- The following example is an illustration of FIVEWIRESEOLSPACING:

```
FIVEWIRESEOLSPACING eolSpacing WITHIN eolWithin
PRL prl
ENCLOSECUT BELOW encloseDist CUTWITHIN cutWithin
NOMETALEOLEXTENSION eolSideExtension eolForwardExtension
```

Figure 1-183 Illustration of FIVEWIRESEOLSPACING



It is a violation if there is no metal wire covering the entire y range of the red regions, like A or B (C is irrelevant) and a metal wire overlaps with the black region.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### ***Forbidden Spacing Rule***

A forbidden spacing rule can be used to define spacing between two wires, such that under certain conditions a violation may occur if there is a different-metal polygon wire between the wires.

You can create a forbidden spacing rule by using the following property definition:

```
PROPERTY LEF58_FORBIDDENSPACING
    "FORBIDDENSPACING minSpacing maxSpacing [minSpacing2 maxSpacing2]
        [WIDTH minWidth WITHIN within PRL prl
        | [SAMEMASK] WIDTH maxWidth PRL prl
            [TWOEDGES within | EXACTSPACINGEDGE exactSpacing
            [SPANLENGTH spanLength [WITHIN within]]]
        | EXACTWIDTH exactWidth PRL prl
            OTHERWIDTH otherWidth EXACTSPACINGEDGE exactSpacing
        | [SAMEMASK | MASK maskNum]
            WIDTHRANGE minWidth maxWidth PRL prl
            OTHERWIDTH otherWidth WITHIN within [OTHERSAMEMASK]]
    ; " ;
```

Where:

EXACTSPACINGEDGE *exactSpacing*

Specifies forbidden spacing only applies if the wire width, less than the *maxWidth*, has a neighbor with exact spacing of *exactSpacing* on one side, and the forbidden spacing range will be applied on the opposite side. The forbidden spacing measurement will become the traditional edge-to-edge between wires from right/left or top/bottom of a wire to left/right or bottom/top of another wire. In addition, this rule will apply to right way wires only while the forbidden spacing range is measured in the direction perpendicular to the specified direction in DIRECTION on a Manhattan routing layer.

*Type:* Float, specified in microns

EXACTWIDTH *exactWidth* PRL *prl* OTHERWIDTH *otherWidth*  
EXACTSPACINGEDGE *exactSpacing*



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the spacing between a wire with width exactly equal to *exactWidth* and another wire with width greater than or equal to *otherWidth* must not be greater than or equal to *minSpacing* and less than or equal to *maxSpacing* if the first wire has another wire on the other side with the exact spacing of *exactSpacing* and the three wires have parallel run length greater than *prl*.

*Type:* Float, specified in microns

FORBIDDENSPACING *minSpacing maxSpacing WIDTH maxWidth PRL prl*

Specifies that if the spacing between the right/left or top/bottom edge of a wire with width less than the *maxWidth* to the right/left or top/bottom of another wire is greater than or equal to the *minSpacing* and less than or equal to the *maxSpacing* with parallel run length greater than *prl*, then it will be a violation if and only if there is a different-metal polygon wire between these two wires.

*Type:* Float, specified in microns

MASK *maskNum*

Specifies that the forbidden spacing rule applies only to the objects belonging to the given mask. In other words, forbidden spacing is checked against the objects in the given mask, and the mask of other neighboring objects is irrelevant. *maskNum* must be a positive integer, and most applications support only the values 1, 2, or 3.

*Type:* Integer

*minSpacing2 maxSpacing2*

Specifies an additional second set of forbidden spacing range. This second set can be defined only if the TWOEDGES or EXACTSPACINGEDGE keyword is also specified.

*Type:* Float, specified in microns

OTHERSAMEMASK

Specifies that the forbidden spacing rule applies only if the two outer wires have the same mask. The mask of the middle wire is irrelevant. See [Figure 1-191](#) on page 363.

TWOEDGES *within*

Indicates that the forbidden spacing rule only applies if the wire width is less than the *maxWidth* that has neighbors on both the sides within *within*. The forbidden spacing measurement will become the traditional edge-to-edge between wires from right/left or top/bottom of a wire to left/right or bottom/top of another wire.

*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**SAMEMASK** Specifies that the forbidden spacing rule applies only to objects on the same mask. The objects are the ones that the forbidden spacing is checked against, and the colors of other neighbor objects are irrelevant.

[ **SAMEMASK** ]

**WIDTHRANGE** *minWidth* *maxWidth* **PRL** *prl*

**OTHERWIDTH** *otherWidth* **WITHIN** *within*

Specifies that the spacing between a wire with width greater than *minWidth* and less than *maxWidth*, which is sandwiched by two wires within *within* with width greater than *otherWidth*, and any of two wires must not be greater than or equal to *minSpacing* and less than or equal to *maxSpacing* if the three wires have parallel run length greater than *prl*.

*Type*: Float, specified in microns

**SAMEMASK** specifies that the forbidden spacing rule applies only to objects on the same mask. The objects are the ones that the forbidden spacing is checked against, and the mask of the other neighbor object is irrelevant. See [Figure 1-190](#) on page 362.

**SPANLENGTH** *spanLength*

Specifies that the forbidden spacing range rule only applies if the span length of the neighbor wire is less than the *spanLength*.

*Type*: Float, specified in microns

**WIDTH** *minWidth* **WITHIN** *within* **PRL** *prl*

Specifies that it is a violation if two wires are apart by a distance that is greater than or equal to the *minSpacing* and less than or equal to the *maxSpacing* and are within *within* distance from a wire with width greater than or equal to the *minWidth* and has parallel run length greater than *prl*.

*Type*: Float, specified in microns

**WITHIN** *within*

Specifies that the forbidden spacing range rule only applies if the neighbor wire has another neighbor within (less than) a *within* distance on either side. See [Figure 1-189](#) on page 361.

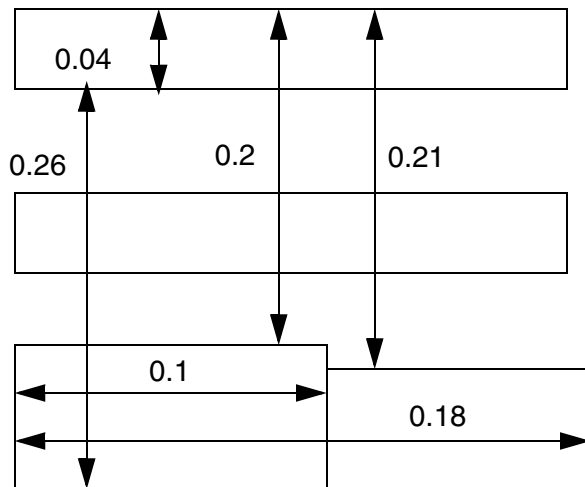
*Type*: Float, specified in microns

## Forbidden Spacing Rule Examples

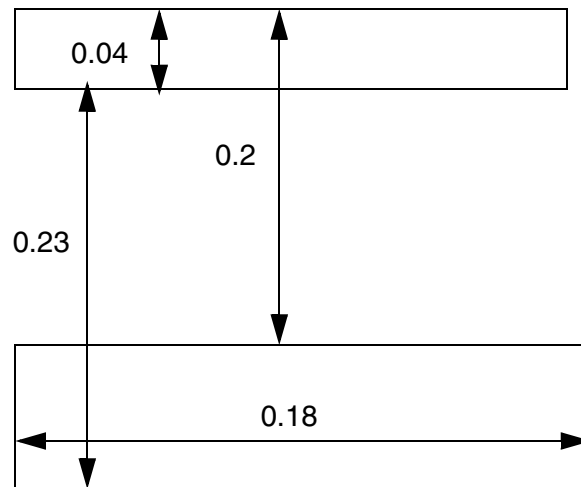
**Figure 1-184 Illustration of FORBIDDENSPACING with WIDTH and PRL**

```
PROPERTY LEF58_FORBIDDENSPACING
```

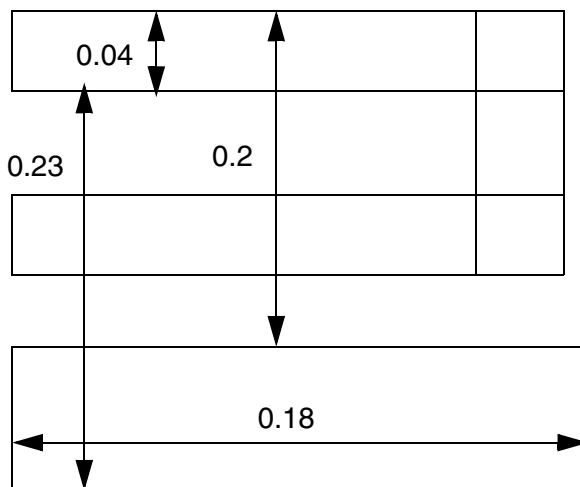
```
"FORBIDDENSPACING 0.2 0.25 WIDTH 0.05 PRL 0.15 ; " ;
```



a) Violation, the bottom to bottom edge is fine, but the top to top edges need to sum up the PRL of 0.18 ( $> 0.15$ ) to trigger the rule



b) OK, there is no wire between the 2 wires.



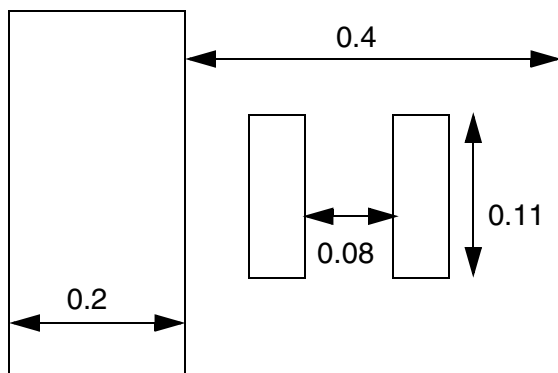
c) OK, the middle is same-metal to one of the 2 wires.

## LEF/DEF 5.8 Language Reference

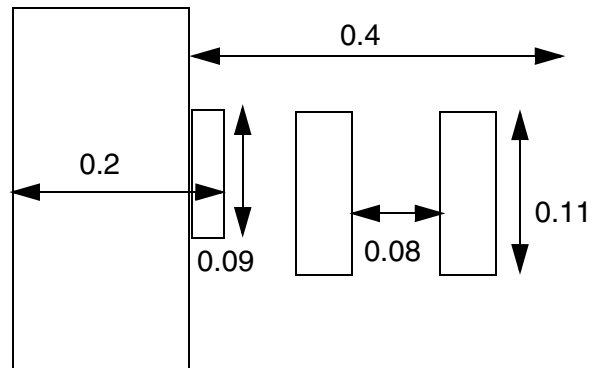
### LEF Syntax

**Figure 1-185 Illustration of FORBIDDENSPACING with WIDTH and WITHIN**

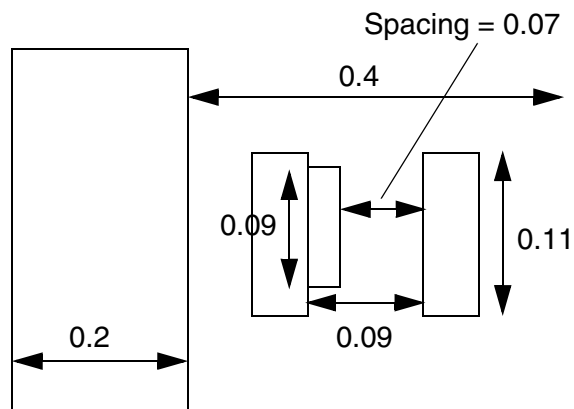
```
PROPERTY LEF58_FORBIDDENSPACING
  "FORBIDDENSPACING 0.07 0.09
    WIDTH 0.2 WITHIN 0.4 PRL 0.1 ; " ;
```



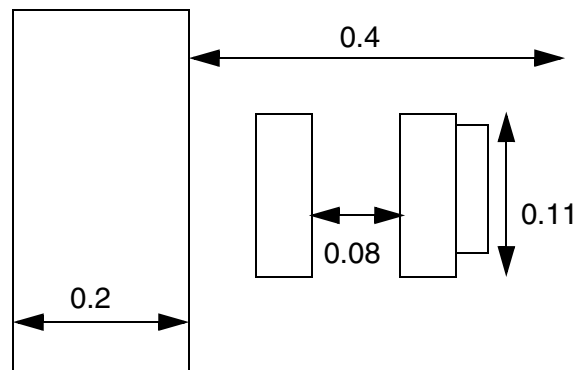
a) Violation, the 2 neighbor wires have a spacing of 0.08, which is inside the forbidden spacing zone



b) OK, after shrink and grow of half width of 0.2 on the wide wire, the final geometry only has width of 0.09, and the rule is not triggered.



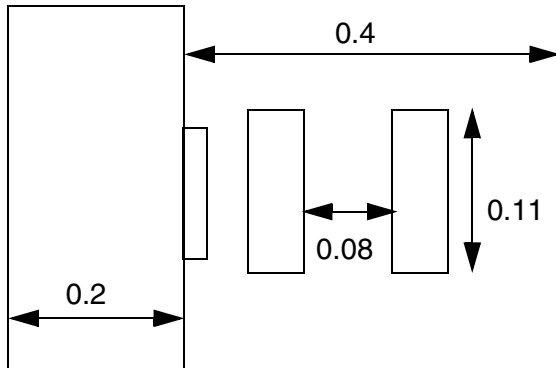
c) Violation, since 0.07 spacing in the jog and 0.09 spacing are in the forbidden range, PRL is the sum of them, 0.11 ( $> 0.1$ ).



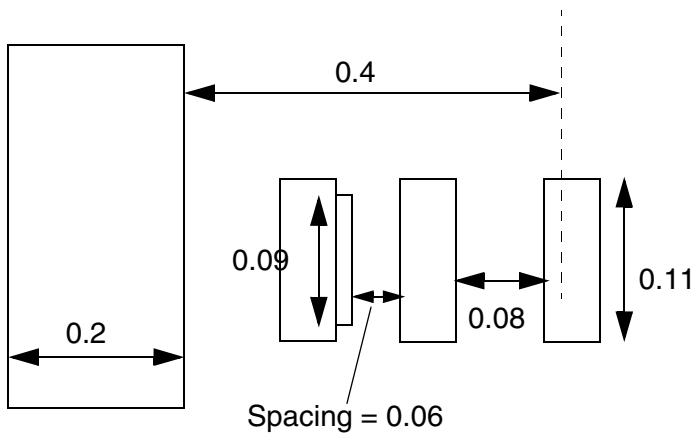
d) Violation, a jog on the other side does not exempt the rule.

**Figure 1-186 Illustration of FORBIDDENSPACING with PRL**

```
PROPERTY LEF58_FORBIDDENSPACING  
  "FORBIDDENSPACING 0.07 0.09  
    WIDTH 0.2 WITHIN 0.4 PRL 0.1 ; " ;
```



a) Violation, parallel run length  $> 0.1$ , and even a small jog is irrelevant.

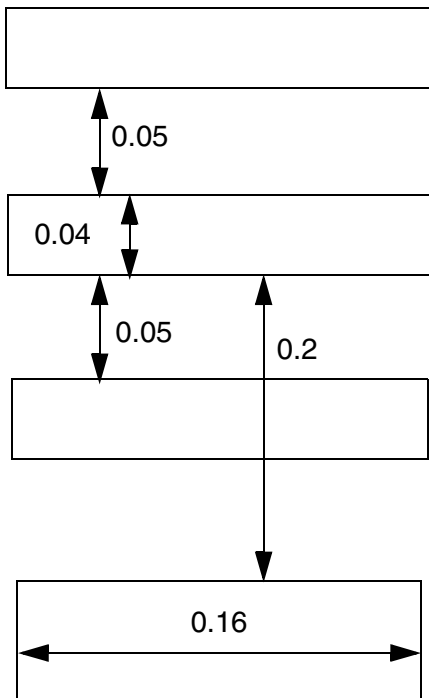


b) Violation, the jog with spacing of 0.06 on the first right wire would exempt the rule to the second wire, but the third wire is within search window, and would be a violation to the second wire.

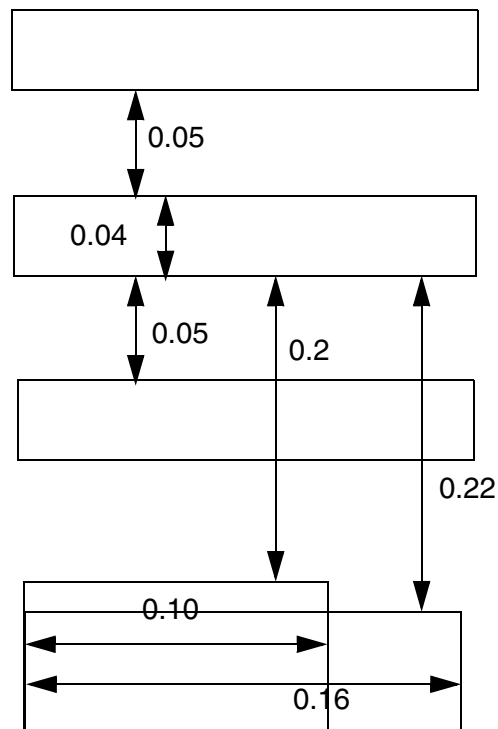
**Figure 1-187 Illustration of FORBIDDENSPACING with TWOEDGES**

PROPERTY LEF58\_FORBIDDENSPACING

```
"FORBIDDENSPACING 0.2 0.25 0.3 0.33 WIDTH 0.05 PRL 0.15  
TWOEDGES 0.06 ; " ;
```



a) Violation, all of the conditions are met, and yet the spacing is 0.2 ( $\geq 0.2$  &  $\leq 0.25$ )

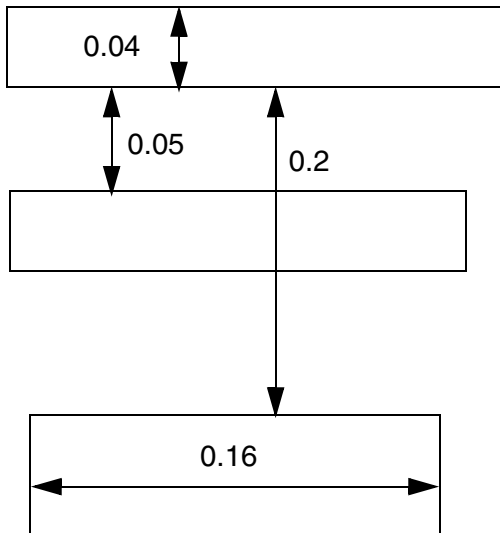


b) Violation, the PRL is considered as 0.16 when both edges are in forbidden spacing range.

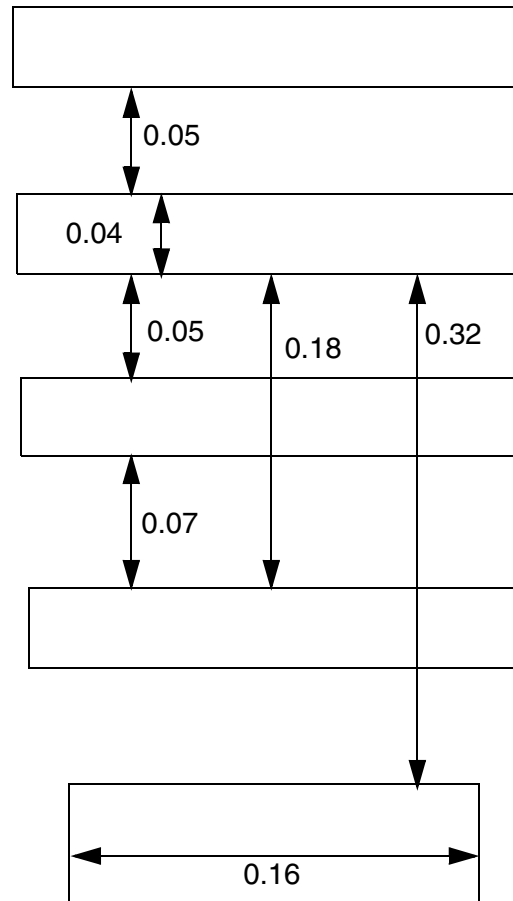
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



c) OK, the top wire only has 1 neighbor.

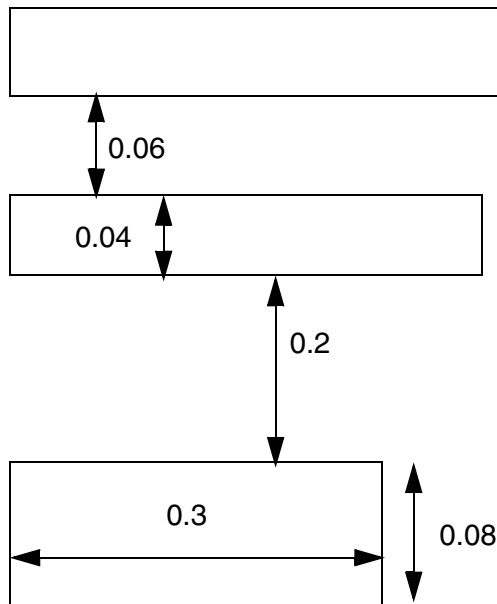


d) Violation, the bottom most wire should be checked independently

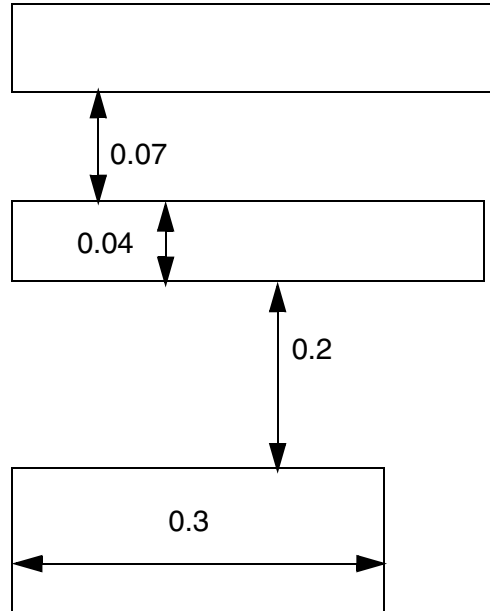
- The following example shows forbidden spacing rule with exact spacing edge and parallel run length:

```
DIRECTION HORIZONTAL ;
PROPERTY LEF58_FORBIDDENSPACING
  "FORBIDDENSPACING 0.2 0.25 WIDTH 0.05 PRL 0.04
    EXACTSPACINGEDGE 0.06 SPANLENGTH 0.09 ; " ;
```

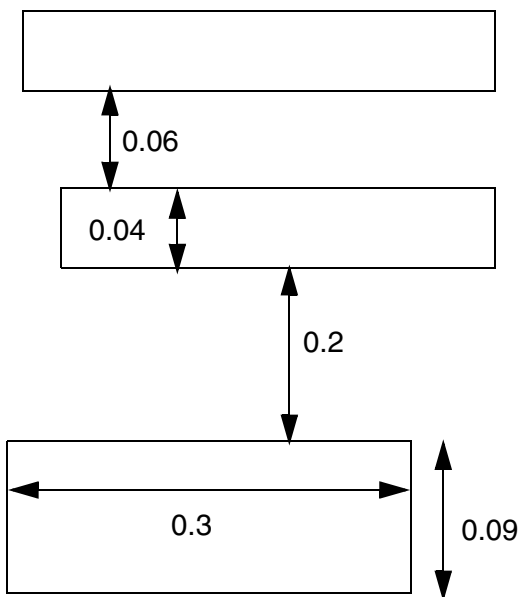
**Figure 1-188 Illustration of FORBIDDENSPACING with EXACTSPACINGEDGE**



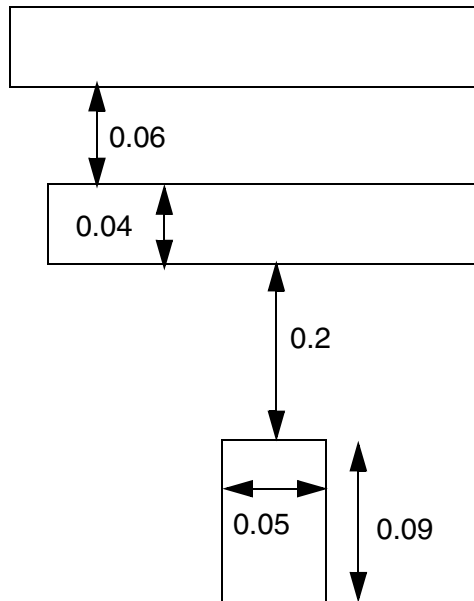
a) Violation, exact spacing of 0.06 neighbor on top & spacing to bottom neighbor is 0.2 ( $\geq 0.2$  &  $\leq 0.25$ )



b) OK, top neighbor is not exactly 0.06 spacing away.



c) OK, the span condition is not met



d) OK, the span, not width, of 0.09 is checked



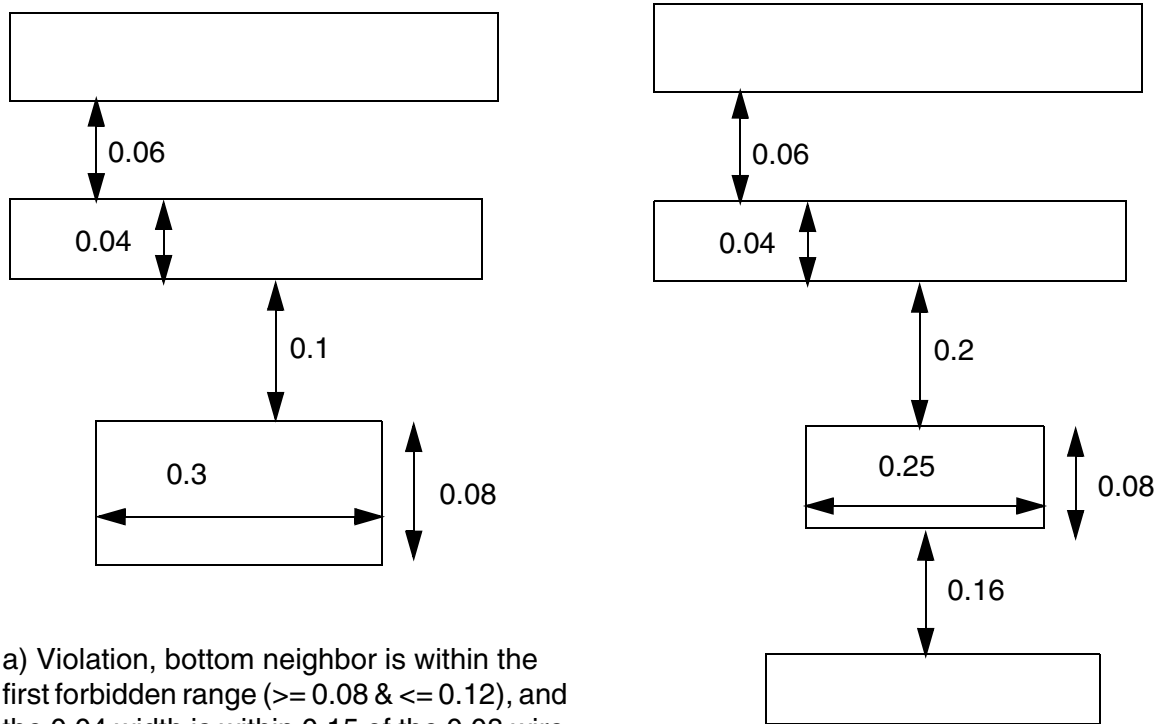
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- The following example shows forbidden spacing rule with within in exact spacing edge span length:

```
DIRECTION HORIZONTAL ;  
PROPERTY LEF58_FORBIDDENSPACING  
  "FORBIDDENSPACING 0.08 0.12 0.2 0.25 WIDTH 0.05 PRL 0.04  
    EXACTSPACINGEDGE 0.06 SPANLENGTH 0.09  
    WITHIN 0.15 ; " ;
```

**Figure 1-189 Illustration of FORBIDDENSPACING with WITHIN**



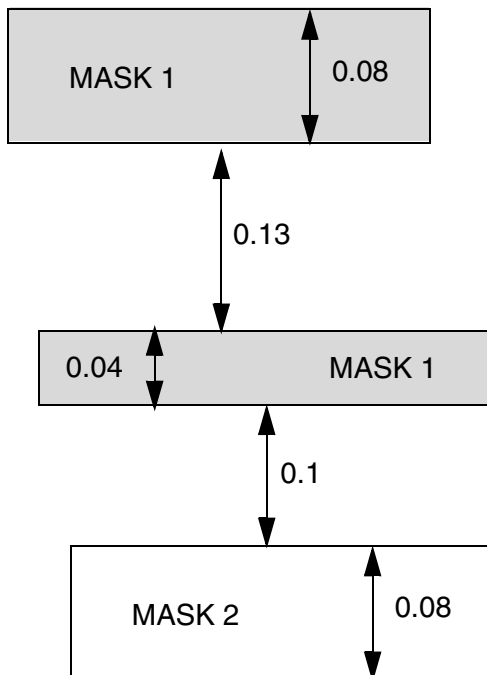
a) Violation, bottom neighbor is within the first forbidden range ( $\geq 0.08$  &  $\leq 0.12$ ), and the 0.04 width is within 0.15 of the 0.08 wire.

b) OK, the 0.08 wire does not have a neighbor within 0.15. It would be a violation if WITHIN is omitted.

- The following example shows forbidden spacing rule with exact width:

```
DIRECTION HORIZONTAL ;  
PROPERTY LEF58_FORBIDDENSPACING "  
    FORBIDDENSPACING 0.08 0.12 EXACTWIDTH 0.04 PRL 0  
    OTHERWIDTH 0.08 EXACTSPACINGEDGE 0.06 ; " ;
```

**Figure 1-190 Illustration of FORBIDDENSPACING with WIDTHRANGE**

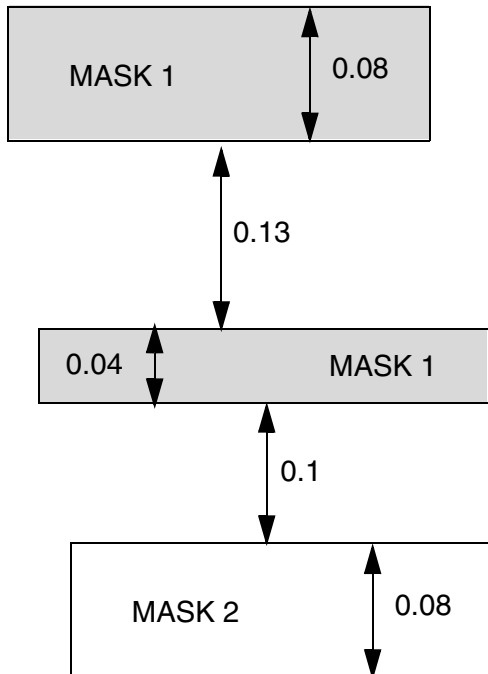


a) OK, the spacing between the bottom two wires are within forbidden spacing range of 0.08 and 0.12, but they are different-mask; the top two same-mask wires have spacing outside the forbidden spacing range. Violation if SAMEMASK is omitted.

Example of

```
DIRECTION HORIZONTAL ;  
PROPERTY LEF58_FORBIDDENSPACING "  
    FORBIDDENSPACING 0.08 0.12 SAMEMASK  
    WIDTHRANGE 0.00 0.05 PRL 0.00  
    OTHERWIDTH 0.07 WITHIN 0.35 ; " ;
```

**Figure 1-191 Illustration of FORBIDDENSPACING with WIDTHRANGE**

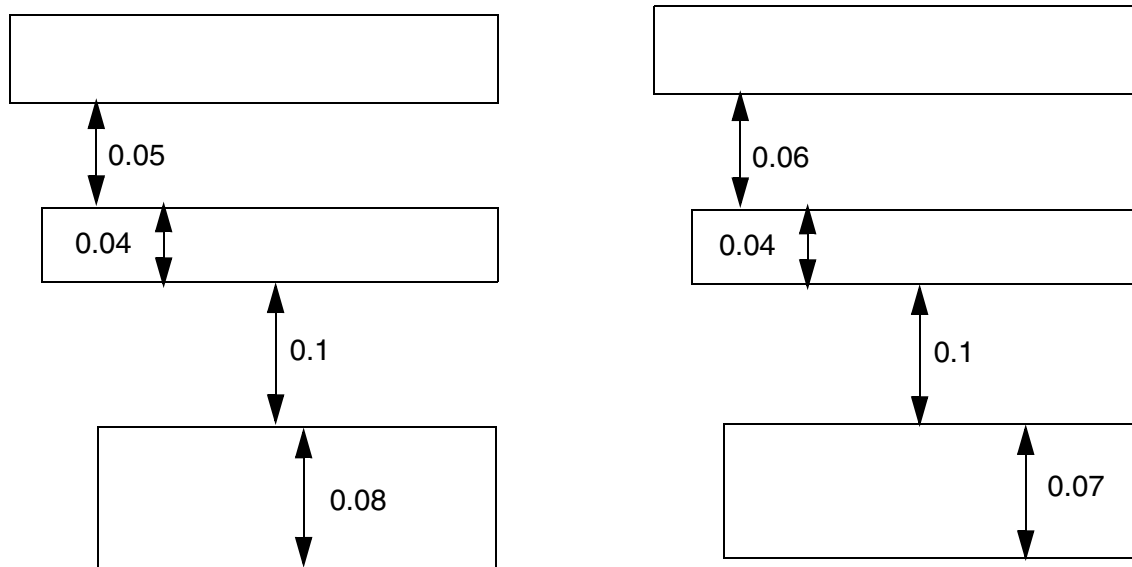


a) OK, the bottom two wires could potentially violate the rule, but the two outer wires must be same-mask to trigger the rule. Violation if OTHERSAMEMASK is omitted.

Example of

```
DIRECTION HORIZONTAL;
PROPERTY LEF58_FORBIDDENSPACING "
  FORBIDDENSPACING 0.08 0.12
  WIDTHRANGE 0.00 0.05 PRL 0.00
  OTHERWIDTH 0.07 WITHIN 0.35 OTHERSAMEMASK; " ;
```

**Figure 1-192 Illustration of FORBIDDENSPACING with EXACTWIDTH**



a) OK, the middle 0.04 wire does not have an exact spacing of 0.05 on the other (top) side

b) OK, the bottom neighbor has width of 0.07 (< 0.08)

### Gap Rule

A gap rule can be used to specify rectangular gaps in a design with certain width and length.

You can create a gap rule by using the following property definition:

```
PROPERTY LEF58_GAP
    "GAP EXACTWIDTH exactWidth MAXLENGTH maxLength
        SPACING {{0 | 1 | 2} spacing}...[ENDTOEND endToEndSpacing]
        ; " ;
```

Where:

ENDTOEND *endToEndSpacing*

Specifies that the end-to-end spacing of two gaps with parallel run length greater than 0 in non-preferred direction of the routing layer must be greater than or equal to *endToEndSpacing*.

*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

GAP EXACTWIDTH *exactWidth* MAXLENGTH *maxLength*

Specifies the empty area without wires must be a rectangle with width exactly equal to *exactWidth* and length less than or equal to *maxLength*. The 'width' is measured along the preferred direction of the routing layer. The gap between two parallel wires with the corresponding required spacing is not formed. The gaps are formed/aligned to line end of wires and are extended half of the required spacing of the wires in the perpendicular direction.

*Type:* Float, specified in microns

SPACING {{0 | 1 | 2} *spacing*}...

Specifies the spacing among those gaps based on whether they are on the same track (0), adjacent track (1), or 2 tracks apart (2), where track is defined along the preferred direction of the routing layer. At the most three spacing values with different track number can be defined.

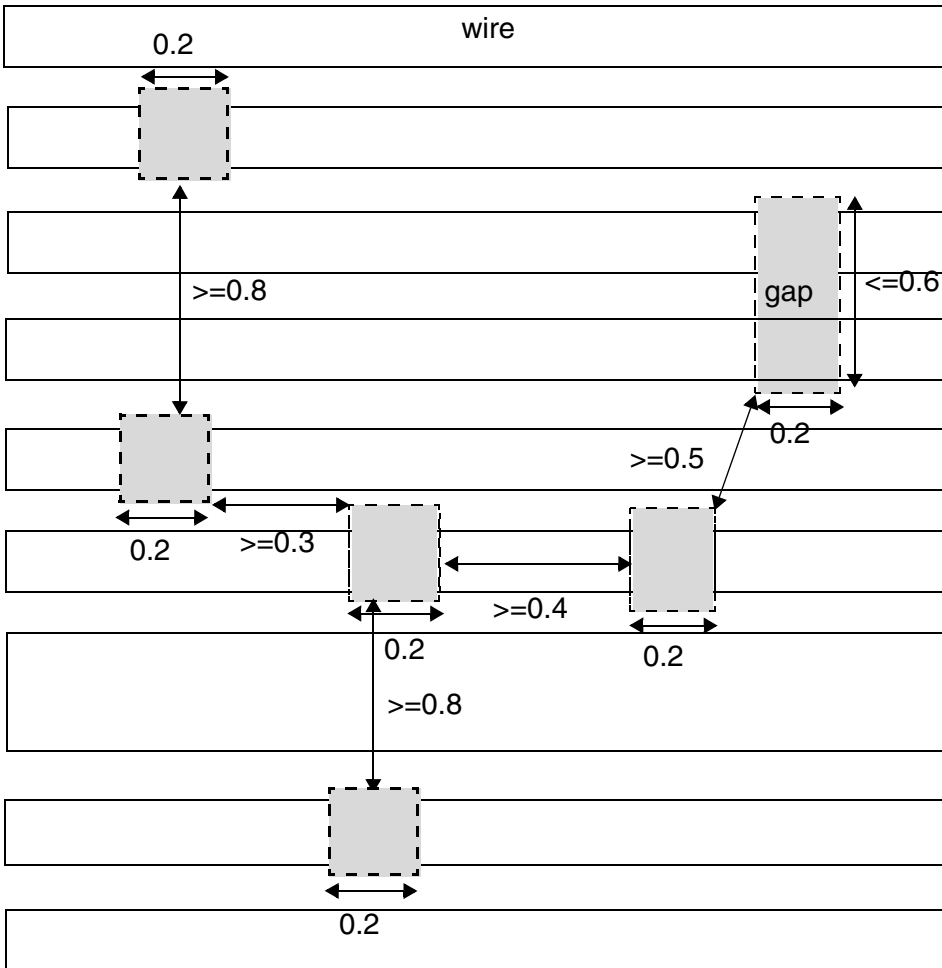
*Type:* Float, specified in microns

### Gap Rule Example

- The following rule illustrates the Gap rule with ENDTOEND:

```
DIRECTION HORIZONTAL ;
PROPERTY LEF58_GAP "
    GAP EXACTWIDTH 0.2 MAXLENGTH 0.6
    SPACING 0 0.4 1 0.3 2 0.5 ENDTOEND 0.8; " ;
```

**Figure 1-193 Illustration of Gap Rule with ENDTOEND**



Gaps (in gray color) must have width of 0.2 & max length of 0.6. Spacing on the same track must be  $\geq 0.4$ , on adjacent track must be  $\geq 0.3$ , two track apart must be  $\geq 0.5$  and end to end with PRL > 0 must be  $\geq 0.8$ .

### ***Joint Corner Spacing Rule***

A joint corner spacing rule can be used to define the spacing between two facing joints of joint corners.

You can create a joint corner spacing rule by using the following property definition:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_JOINTCORNERSPACING
    "JOINTCORNERSPACING spacing [SAMEMASK]
        JOINTWIDTH jointWidth [MINLENGTH minLength]
        JOINTLENGTH spanLength [EDGELENGTH edgeLength]
    ; " ;
```

Where:

All other keywords are the same as the existing LEF routing layer joint corner spacing rule syntax.

EDGELENGTH *edgeLength*

Specifies that the joint corner spacing only applies if the edge length of both the joints is less than or equal to the specified *edgeLength*.

*Type:* Float, specified in microns

```
JOINTCORNERSPACING spacing
JOINTWIDTH jointWidth [MINLENGTH minLength]
JOINTLENGTH spanLength
```

Specifies the spacing between two facing joints of joint corners with parallel run length less than or equal to zero to be *spacing*. A joint corner is a convex corner consisting of two consecutive joints with span greater than *spanLength* and not a EOL edge with length less than the *jointWidth* and having a length greater than or equal to the *minLength* along both the sides, if the MINLENGTH keyword is specified.

*Type:* Float, specified in microns

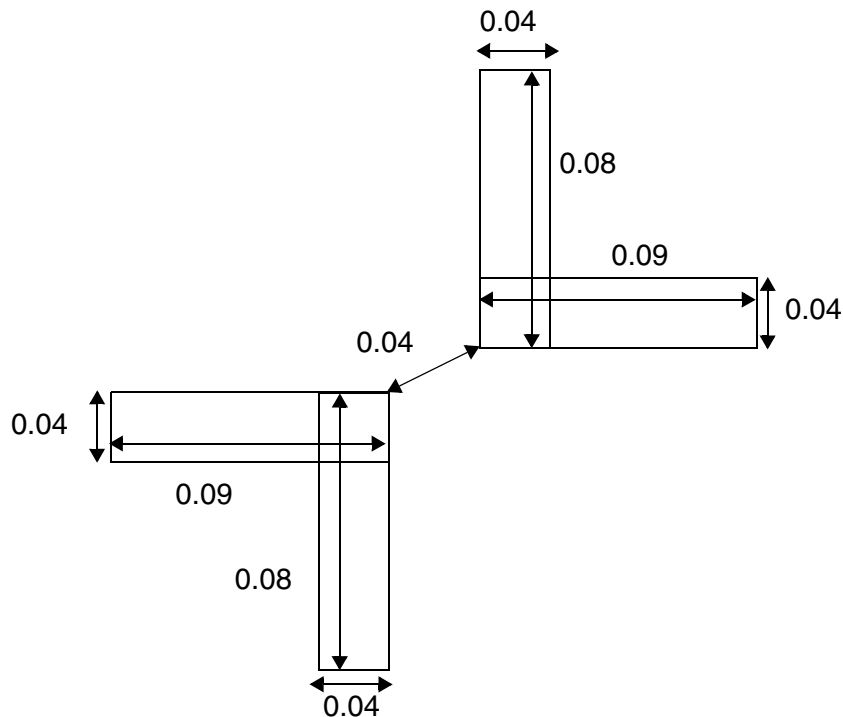
SAMEMASK

Specifies that the corner spacing rule only applies to objects on the same mask.

## Joint Corner Spacing Rule Examples

**Figure 1-194 Illustration of Joint Corner Spacing Rule**

```
PROPERTY LEF58_JOINTCORNERSPACING
  "JOINTCORNERSPACING 0.05
    JOINTWIDTH 0.07 JOINTLENGTH 0.06
    EDGELENGTH 0.1 ; " ;
```



a) Violation, the spacing is applied between two facing joints with no common parallel run length

## ***EOL Keep-out Rule***

EOL keep-out rule can be used to define a keep-out region for an end-of-line edge.

You can create an EOL keep-out rule by using the following property definition:

```
PROPERTY LEF58_EOLKEEPOUT
  "EOLKEEPOUT {eolWidth | minEolWidth maxEolWidth}
    EXTENSION backwardExt sideExt forwardExt
    [EXCEPTWITHIN lowSideExt highSideExt]
    [CLASS className [OTHERENDEOL]]
    [CORNERONLY]
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[EXCEPTFROM { BACKEDGE [FORWARDGAP forwardGap]  
               | FRONTEGE}  
  [MASK maskNum [TWO SIDES]]]  
[EXCEPTSAMEMETAL]  
; ... " ;
```

Where:

**CLASS *className*** Assigns a class name to the rule. It is a violation if all the rules belonging to the same class fail. An entire set of failed class rules will be a violation. If any one of the rules have CLASS, then CLASS must be specified for all of the rules. Typically, CLASS is used along with EXCEPTFROM - one being BACKEDGE and another being FRONTEGE. It is a violation if both of them fail on the same side of the wire.

**CORNERONLY** Specifies that it is a violation if there is a corner falling within the keepout, and a pass-through edge is allowed.

**EXCEPTFROM {BACKEDGE | FRONTEGE}**

Specifies that the rule only applies if the neighbor wire containing the corner is not coming from the front edge of the keepout window in EXCEPTFROM FRONTEGE or not coming from the back edge of the search window in EXCEPTFROM BACKEDGE.

**EXCEPTWITHIN *lowSideExt highSideExt***

Specifies that the rule is not checked on a neighbor object overlapping with a region greater than or equal to *lowSideExt* and less than or equal to *highSideExt* away from the sides of the EOL edge, which covers the entire range of *backwardExt* and *forwardExt*.  
*Type:* Float, specified in microns

**EOLKEEPOUT *eolWidth* EXTENSION *backwardExt sideExt forwardExt***

Defines a keepout region for an end-of-line edge with length less than the *eolWidth* by extending *backwardExt* going backward, *sideExt* on the side, and *forwardExt* going forward. Any objects falling within the region will be a violation.  
*Type:* Float, specified in microns

**EXCEPTSAMEMETAL** Specifies that the keepout region does not apply to same-metal objects.

**FORWARDGAP *forwardGap***

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies if there is a face-to-face object within the search window of *backwardExt* *sideExt* *forwardGap*, the objects must be extended such that the face-to-face gap must become *forwardExt*. See [Figure 1-197](#) on page 373.

*Type:* Float, specified in microns

MASK *maskNum*

Specifies which mask the triggering wire of this rule belongs to, *maskNum* must be a positive integer, and most applications only support values of 1, 2, or 3. See [Figure 1-202](#) on page 378.

*Type:* Integer

*minEolWidth* *maxEolWidth*

Specifies that the rule applies only if the EOL width is greater than or equal to *minEolWidth* and less than *maxEolWidth*.

*Type:* Float, specified in microns

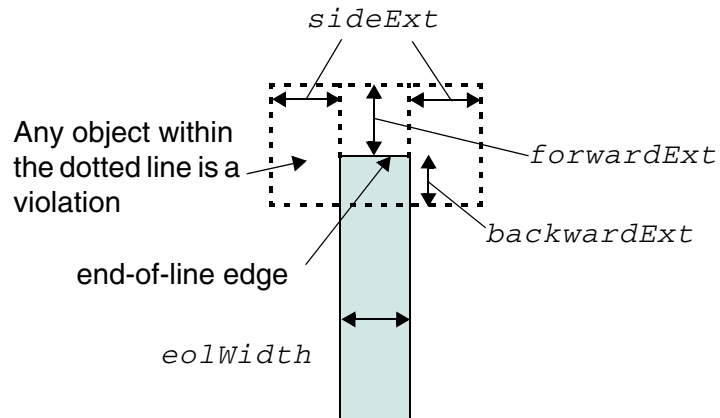
OTHERENDEOL

Specifies that if the object found in the extension window is also an EOL, it is only a violation if that EOL also fails all of the EOLKEEPOUT statements in the same CLASS. OTHERENDEOL must be specified along with EXCEPTFROM BACKEDGE since it would be a symmetrical situation meaning either one of the two EOLs as a trigger edge would find the other EOL, and it makes sense for both EOLs to fail other EOLKEEPOUT statements to be a violation.

TWOSIDES

Specifies that the rule only applies if it has 2 different-mask wires on both sides.

**Figure 1-195 Definition of EOL Keep Out Rule**



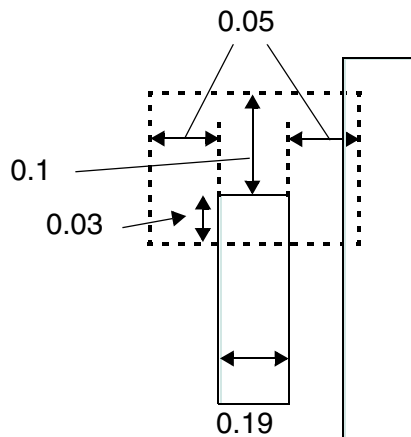
Definition of EOLKEEPOUT *eolWidth* EXTENSION  
*backwardExt sideExt forwardExt*

### EOL Keep Out Rule Examples

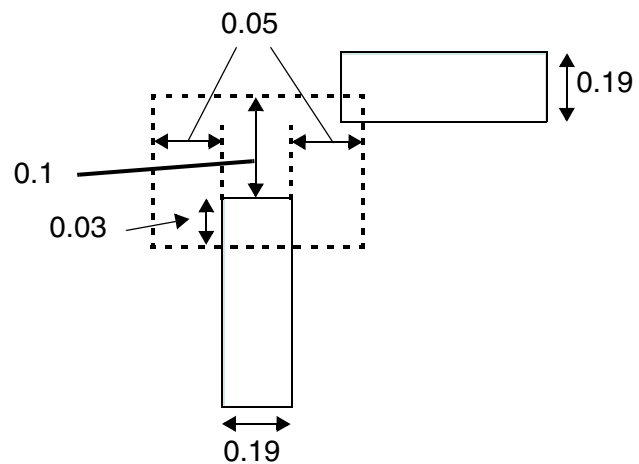
- The following example illustrates EOLKEEPOUT with EXTENSION:

**Figure 1-196 Illustration of EOL Keep Out Rule with EXTENSION**

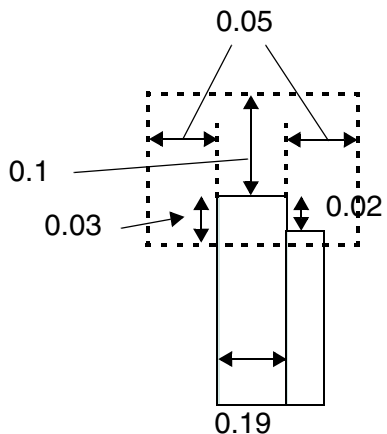
```
PROPERTY LEF58_EOLKEEPOUT  
  "EOLKEEPOUT 0.2 EXTENSION 0.03 0.05 0.1 ; " ;
```



a) Violation, side neighbor is also a violation. With CORNERONLY, it is OK since there is no corner neighbor.



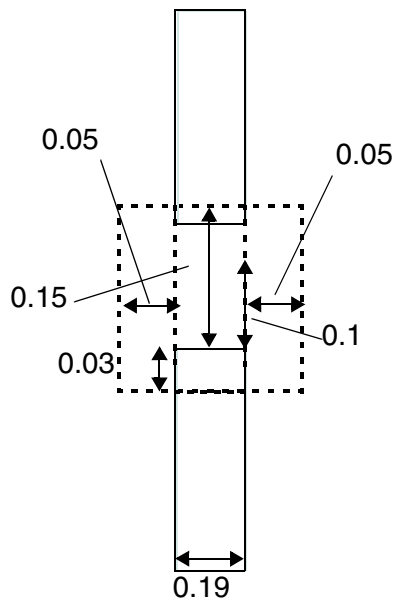
b) Violation, any neighbor in the dotted region is a violation.



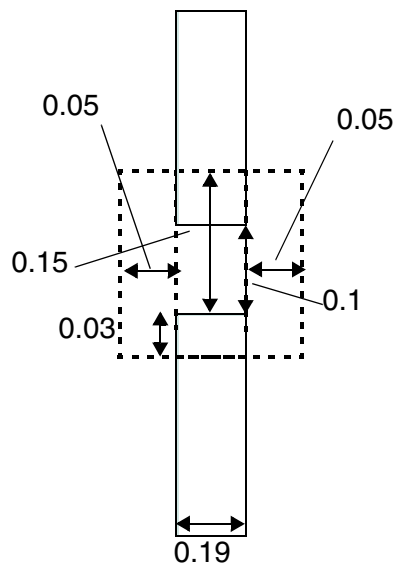
c) Violation, any neighbors, including same-metal, would be a violation. With EXCEPTSAMEMETAL, it would be OK.

**Figure 1-197 Illustration of EOL Keep Out Rule with FORWARDGAP**

```
PROPERTY LEF58_EOLKEEPOUT "  
  EOLKEEPOUT 0.2 EXTENSION 0.03 0.05 0.1  
  CORNERONLY EXCEPTFROM BACKEDGE  
  FORWARDGAP 0.15 ; " ;
```



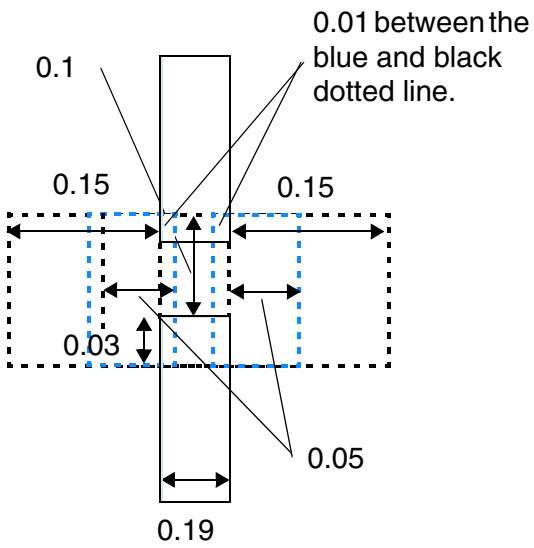
a) Violation, if there is a face-to-face object within the dotted search window, the spacing between the object must be exactly equal to 0.1



b) OK, the spacing is exactly 0.1

**Figure 1-198 Illustration of EOL Keep Out Rule with EXCEPTWITHIN**

```
PROPERTY LEF58_EOLKEEPOUT  
  "EOLKEEPOUT 0.2 EXTENSION 0.03 0.15 0.1 ; " ;  
  EXCEPTWITHIN -0.01 0.05 ;"
```



a) OK, having negative EXCEPTWITHIN value would exclude neighbor on the same track

## LEF/DEF 5.8 Language Reference

### LEF Syntax

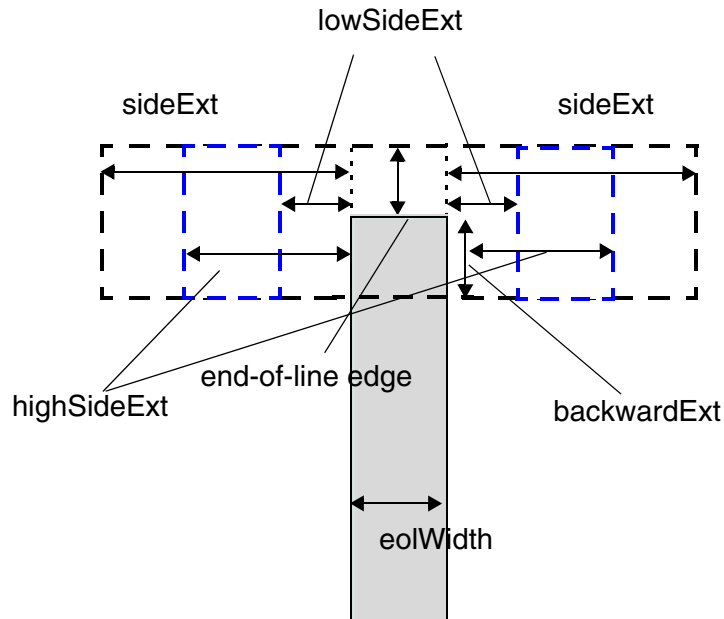
- The following example indicates that it is a violation if all first three rules fail or both the last rules fail :

```
PROPERTY LEF58_EOLKEEPOUT "  
    EOLKEEPOUT 0.06 EXTENSION 0.00 0.08 0.12 CLASS A CORNERONLY  
        EXCEPTFROM FRONTEGE ;  
    EOLKEEPOUT 0.06 EXTENSION 0.04 0.08 0.00 CLASS A CORNERONLY  
        EXCEPTFROM BACKEDGE ;  
    EOLKEEPOUT 0.06 EXTENSION 0.04 0.06 0.12 CLASS A CORNERONLY ;  
    EOLKEEPOUT 0.09 EXTENSION 0.04 0.08 0.12 CLASS B CORNERONLY  
        EXCEPTFROM FRONTEGE ;  
    EOLKEEPOUT 0.09 EXTENSION 0.04 0.00 0.12 CLASS B CORNERONLY  
        EXCEPTFROM BACKEDGE ; "
```

**Figure 1-199 Illustration of EOL Keep Out Rule with eolWidth**

Illustration of EOLKEEPOUT *eolWidth*

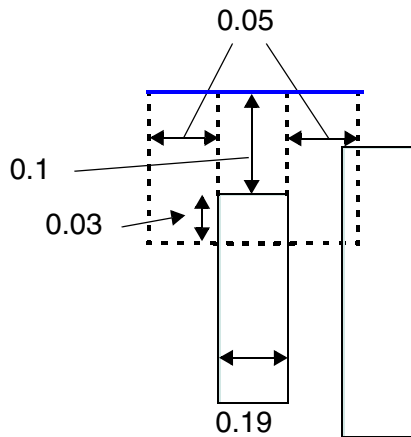
EXTENSION *backwardExt sideExt forwardExt*  
EXCEPTWITHIN *lowSideExt highSideExt*



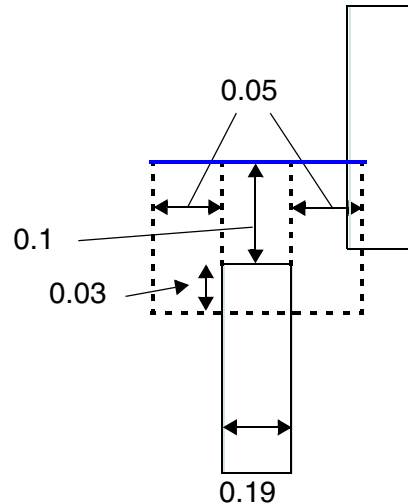
Any objects overlapping with the blue dotted regions is exempted from the rule

**Figure 1-200 Illustration of EOL Keep Out Rule with EXCEPTFROM and FRONTEDGE**

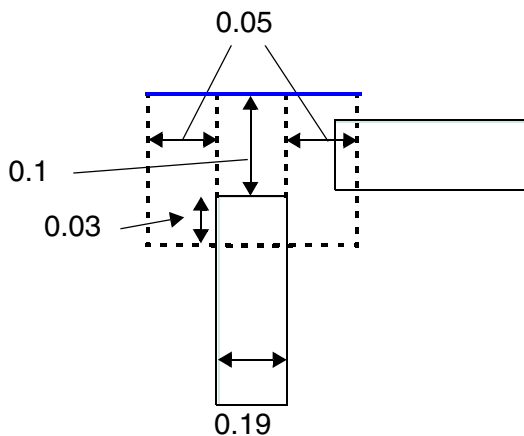
```
PROPERTY LEF58_EOLKEEPOUT  
  "EOLKEEPOUT 0.2 EXTENSION 0.03 0.05 0.1  
    CORNERONLY EXCEPTFROM FRONTEDGE ; " ;
```



a) Violation, neighbor is not coming from the front edge in blue



b) OK, neighbor wire is coming from the front edge. It would be a violation without EXCEPTFROM or with EXCEPTFROM BACKEDGE.



c) Violation, neighbor wire coming from the side edges is also bad



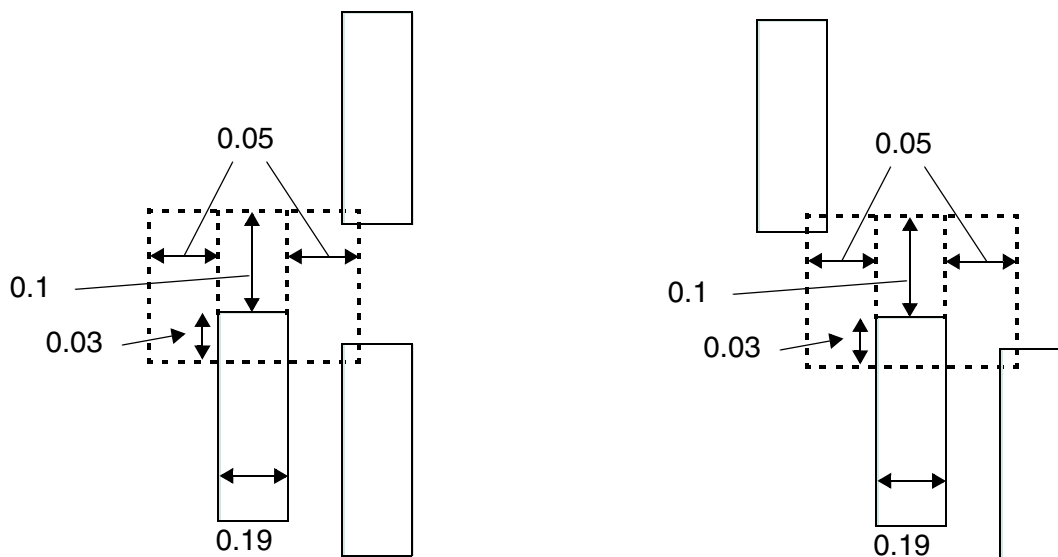
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- The following example indicates that the rule fails if both the statements with neighbors are on the same right side:

```
PROPERTY LEF58_EOLKEEPOUT
  "EOLKEEPOUT 0.2 EXTENSION 0.03 0.05 0.0 CLASS A
    CORNERONLY EXCEPTFROM FRONTEDGE ;
  EOLKEEPOUT 0.2 EXTENSION 0.0 0.05 0.1 CLASS A
    CORNERONLY EXCEPTFROM BACKEDGE ; "
```

**Figure 1-201 Illustration of EOL Keep Out Rule with FRONTEDGE and BACKEDGE**



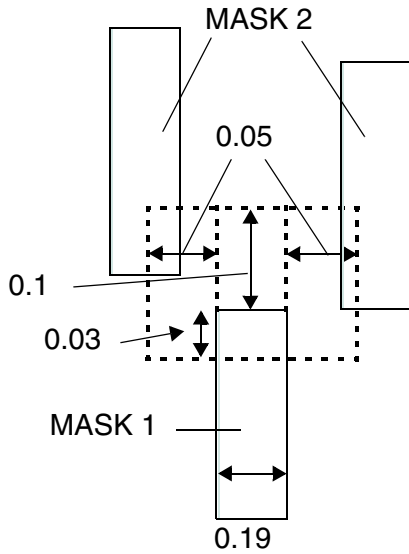
a) Violation, both of the statements fail with neighbors on the same right side

b) OK, failing neighbors are on opposite side of the wire

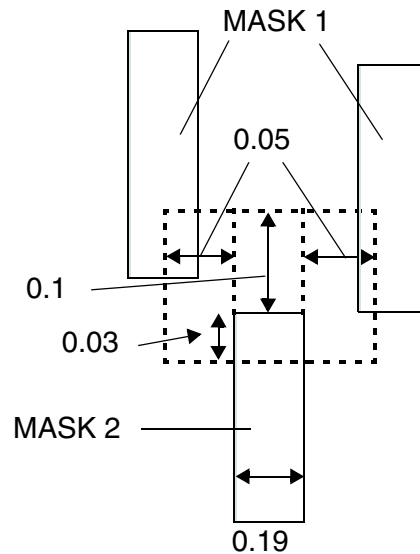
- The following example indicates that the rule fails if mask 1 wire has 2 different-mask (mask 2) wires on both sides:

```
PROPERTY LEF58_EOLKEEPOUT
  "EOLKEEPOUT 0.2 EXTENSION 0.03 0.05 0.1
    CORNERONLY EXCEPTFROM BACKEDGE
    MASK 1 TWOSIDES ; "
```

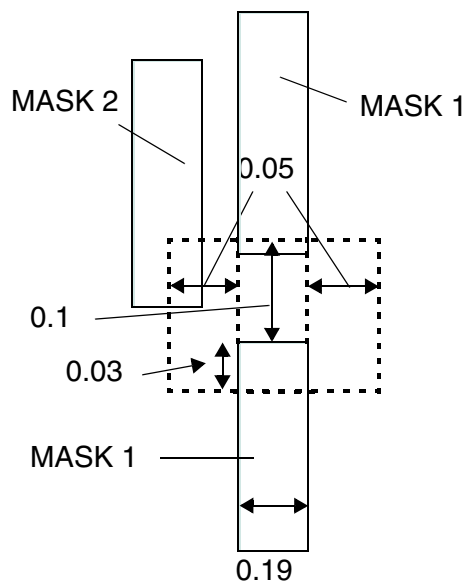
**Figure 1-202 Illustration of EOL Keep Out Rule with Mask and Two Sides**



a) Violation, the mask 1 wire has 2 different-mask (mask 2) wires on both sides.



b) OK, the triggering wire must be a mask 1 wire.



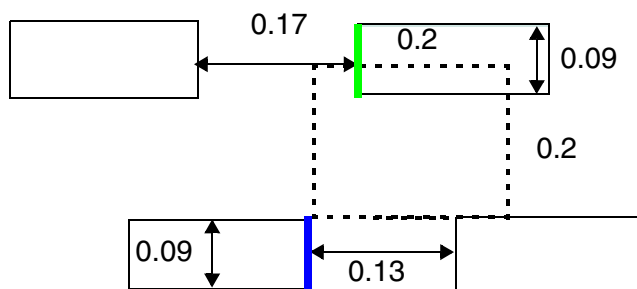
c) OK, only one mask 2 neighbor

**Figure 1-203 Illustration of EOL Keep Out Rule with OTHERENDEOL**

```

DIRECTION HORIZONTAL ;
PROPERTY LEF58_EOLKEEPOUT "
    EOLKEEPOUT 0.1 EXTENSION 0 0.2 0.2
    EXCEPTWITHIN -0.01 0 CLASS A CORNERONLY
    EXCEPTFROM BACKEDGE OTHERENDEOL ;
    EOLKEEPOUT 0.1 EXTENSION 0 0 0.15 CLASS A ; " ;

```



a) OK, when checking the blue EOL, it would fall both CLASS A statements. However, OTHERENDEOL is specified, the neighbor green EOL would pass the second statement, and overall, it would not be a violation.

### ***Litho Macro Halo Rule***

A litho macro halo rule is used to define a routing halo that allows only planar connections to its pins.

You can create a litho macro halo rule by using the following property definition:

```

PROPERTY LEF58_LITHOMACROHALO
    "LITHOMACROHALO horizontalHalo verticalHalo WIRESPPACING wireSpacing
    ; " ;

```

Where:

```

LITHOMACROHALO horizontalHalo verticalHalo WIRESPPACING
wireSpacing

```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies halo values for litho purpose on a block macro. A routing halo with *horizontalHalo* and *verticalHalo* offset horizontally and vertically from the bounding box of the macro could be formed. Inside this routing halo, only planar connections to macro pins are allowed. Any other wires or above/below via insertion is also disallowed. In addition, the wires connected to the macro pins must have spacing greater than or equal to *wireSpacing*. A separate command specifies which block macro should create such a routing halo on them.

*Type:* Float, specified in microns

### ***Opposite EOL Spacing Rule***

An opposite EOL spacing rule can be used to define spacing on a wire with two neighbor wires on the opposite edges.

You can create an opposite EOL spacing rule by using the following property definition:

```
PROPERTY LEF58_OPPOSITEEOLSPACING
    "OPPOSITEEOLSPACING [SAMEMASK] WIDTH width ;" ;
    ENDWIDTH eolWidth [MINLENGTH minLength]
    [JOINTWIDTH jointWidth] JOINTLENGTH spanLength
    [JOINTTOEDGEEND jointToEdgeEndLength]
    [JOINTEXTENSION jointExtension]
    [JOINTCORNERONLY]
    [SIDELENGTH length[TOSIDE toSideLength]]
    [SIDEEXTENSION sideExtension [TOSIDE toSideExtension]]
    [SIDEEDGELENGTH sideEdgeLength]
    [PRL prl]
    {[EXCEPTEDGELENGTH edgeLength [PRL maxPRL]]}...
    ENDTOEND endSpacing endSpacing [PRL individualPrl]
    ENDTOJOINT endSpacing jointSpacing [PRL individualPrl]
    JOINTTOEND jointSpacing endSpacing [PRL individualPrl]
    JOINTTOJOINT jointSpacing jointSpacing [PRL individualPrl]
    [SIDETOEND sideSpacing endSpacing [PRL individualPrl]
    SIDETOJOINT sideSpacing jointSpacing] [PRL individualPrl]
    [JOINTTOSIDE jointSpacing sideSpacing] [PRL individualPrl]
    [SIDETOSIDE sideSpacing sideSpacing] [PRL individualPrl]
    ;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

OPPOSITEEOLSPACING Defines the spacing requirements on a wire with two neighbor wires on opposite edges that have a projected parallel run length greater than 0. The neighbor wires are classified either as a EOL or a T or L joint.

SAMEMASK Specifies that the rule applies only if all three objects to trigger the rule belong to the same mask.

WIDTH *width* Specifies that the rule applies only if the width of the middle wire is less than *width*.  
Type: Float, specified in microns

ENDWIDTH *eolWidth* [MINLENGTH *minLength*]

Specifies that the rule applies only if the neighbor end-of-line has width less than *eolWidth*.

MINLENGTH indicates that the edge is only an end when the end-of-line length is greater than or equal to *minLength* along both the sides. In other words, if end-of-line length is less than *minLength* along any one side, it is not an end, but may be a joint.

Type: Float, specified in microns

JOINTEXTENSION *jointExtension*

Specifies the extension on both sides of a joint to be *jointExtension*. The extension should be treated as if it is a part of the joint.

Type: Float, specified in microns

[JOINTWIDTH *jointWidth*] JOINTLENGTH *spanLength*  
[JOINTTOEDGEEND *jointToEdgeEndLength*]

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the neighbor wire end edge is a joint if its width is less than *jointWidth*, if specified, or less than *eolWidth*, its span is greater than *spanLength*, and it is not a EOL edge.

If `JOINTTOEDGEEND` is specified, then at least one of the distances from the end points of the joint to the ends of the edge, that contain the joint, must be less than or equal to *jointToEdgeEndLength*. A T or L configuration (see [Figure 1-204](#) on page 385) is a typical joint. However, joints are not restricted to such configurations only. A joint can be any edge that fulfills the above definition.

*Type:* Float, specified in microns

`JOINTLENGTH spanLength`

Specifies that the neighbor wire end edge is a joint if it has width less than *eolWidth* and span greater than *spanLength*, and it is not an EOL, that is, it is either a T or L joint pattern.

*Type:* Float, specified in microns

`{EXCEPTEDGELENGTH edgeLength [PRL maxPRL]}...`

Specifies that the rule does not apply if both the end or joint neighbor edges have a length greater than and equal to *edgeLength* and projected parallel run length is less than and equal to *maxPRL*, if PRL is also specified. At the most, two such statements can be specified and supported.

*Type:* Float, specified in microns

`ENDTOEND endSpacing endSpacing`

`ENDTOJOINT endSpacing jointSpacing`

`JOINTTOEND jointSpacing endSpacing`

`JOINTTOJOINT jointSpacing jointSpacing`

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the spacing between the neighbor edges to the middle wire. There are four groups of two spacings. The keywords define the category of the neighbors, either as an end or a joint. For example, in the case of `ENDTOJOINT`, the first spacing, *endSpacing*, specifies the minimum spacing between the end neighbor edge to the middle wire, and the second spacing, *jointSpacing*, specifies the minimum spacing between the joint neighbor edge to the middle wire. To satisfy the rule, for both end/joint neighbors, either both the neighbor spacings must be greater than and equal to the minimum of the specified spacings, or at least one neighbor spacing must be greater than and equal to the maximum of the specified spacings in `ENDTOEND` or `JOINTTOJOINT`. For end and joint neighbors, both `ENDTOJOINT` and `JOINTTOTEND` statements must be fulfilled individually. To fulfill one statement either joint spacing greater than or equal to *jointSpacing*, or end spacing greater than or equal to *endSpacing*, must be true.

*Type:* Float, specified in microns (for all values)

`JOINTCORNERONLY`

Specifies that the joint must form a joint corner, which is a convex corner consisting of two consecutive joints.

`JOINTTOSIDE` *jointSpacing sideSpacing*

Specifies a spacing requirement similar to `SIDETOJOINT`, but having joint spacing to be the first spacing.

*Type:* Float, specified in microns

`PRL` *prl*

Specifies that the rule only applies if the projected parallel run length of a wire and the two neighbor wires is greater than *prl*. If *prl* is negative, it is similar to extending the end, joint, or side edges by `abs(prl)` in a `WITHIN` style.

*Type:* Float, specified in microns

`PRL` *individualPrl*

Specifies that the parallel run length on the given `END`, `JOINT`, and `SIDE` combination must be greater than the specified *individualPrl* value to trigger the rule. If it is not specified on certain combinations, those combinations would be checked against the global *prl*.

*Type:* Float, specified in microns

`SIDEEDGELENGTH` *sideEdgeLength*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that a side fulfilling other conditions must also be an edge with length greater than the *sideEdgeLength*.

Type: Float, specified in microns

SIDEEXTENSION *sideExtension* [TOSIDE *toSideExtension*]

Specifies the extension of a side of EOL edge to be *toSideExtension* only in the direction going beyond the EOL edge in case of *SIDETOSIDE*, and this extension definition is only applied to one of the *SIDE* only while the other *SIDE* would equivalent to having zero extension. The extension should be treated as if it is part of the side. The *sideExtension* is still used for *SIDE* to *END* or *JOINT* cases.

Type: Float, specified in microns

SIDELENGTH *length* [TOSIDE *toSideLength*]

Specifies the length on the sides of a EOL edge to be *toSideLength* in case of *SIDETOSIDE*, and this length definition is only applied to one of the *SIDE* only while the other *SIDE* will be equivalent to having zero length. The *length* is still used for *SIDE* to *END* or *JOINT* cases.

Type: Float, specified in microns

SIDETOEND *sideSpacing* *endSpacing*

SIDETOJOINT *sideSpacing* *jointSpacing*

Specifies the spacing between a side of a EOL edge defined in *SIDELENGTH*, which must be specified along with these constructs, to a wire which has a neighbor end or joint on the opposite side with a certain spacing. The spacings have similar definitions of *jointSpacing* and *endSpacing*.

Type: Float, specified in microns

SIDETOSIDE *sideSpacing* *sideSpacing*

Specifies a spacing requirement between two sides of a EOL edge with a middle wire. In addition, any defined *PRL* value is ignored when considering *SIDETOSIDE* spacing.

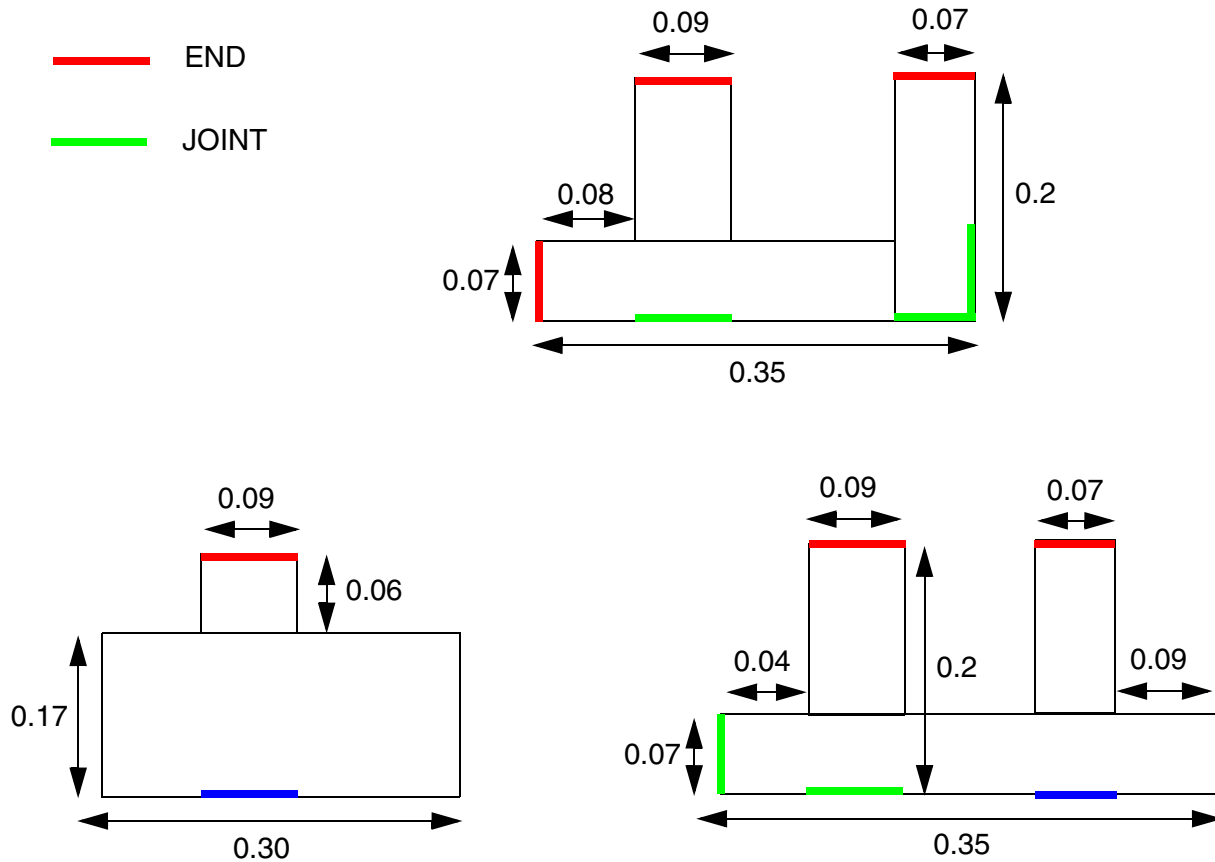
Type: Float, specified in microns

### Opposite EOL Spacing Examples

- **Figure 1-204** on page 385 illustrates *END* and *JOINT* with *ENDWIDTH* 0.1, *MINLENGTH* 0.05, *JOINTLENGTH* 0.15, and *JOINTTOEDGEEND* 0.08:



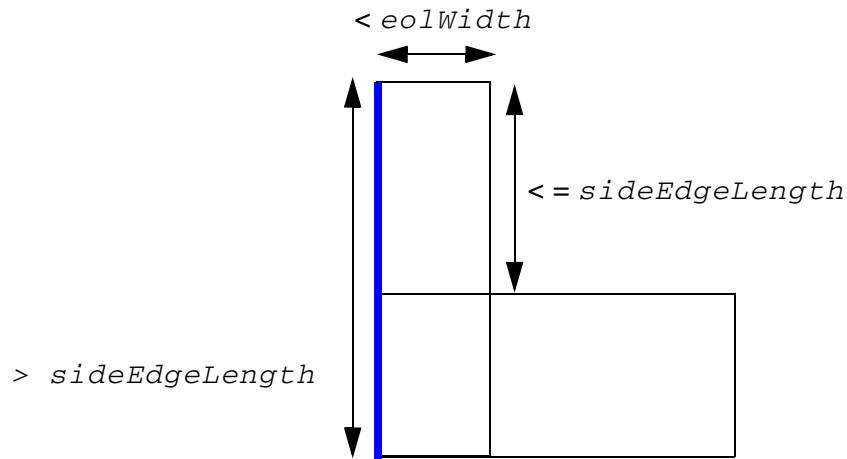
Figure 1-204 Illustration of END and JOINT



Since the whole bottom edge has span length  $> 0.15$ , and its length/width of  $0.3 \geq 0.1$ , it is not a joint. The blue segment is also not a joint since it is merely part of the bottom edge with span length  $> 0.15$ .

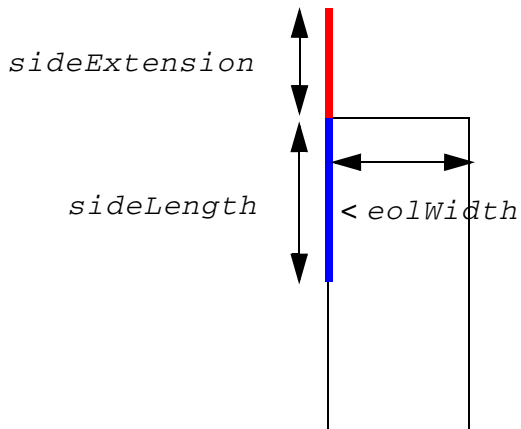
The blue segment is not a joint since both lengths to the end of the edge  $> 0.08$ . The vertical left segment is not an end since one of the end-of-line length is  $0.04 < 0.05$ , but it is a joint.

**Figure 1-205 Illustration of SIDEEDGELENGTH**



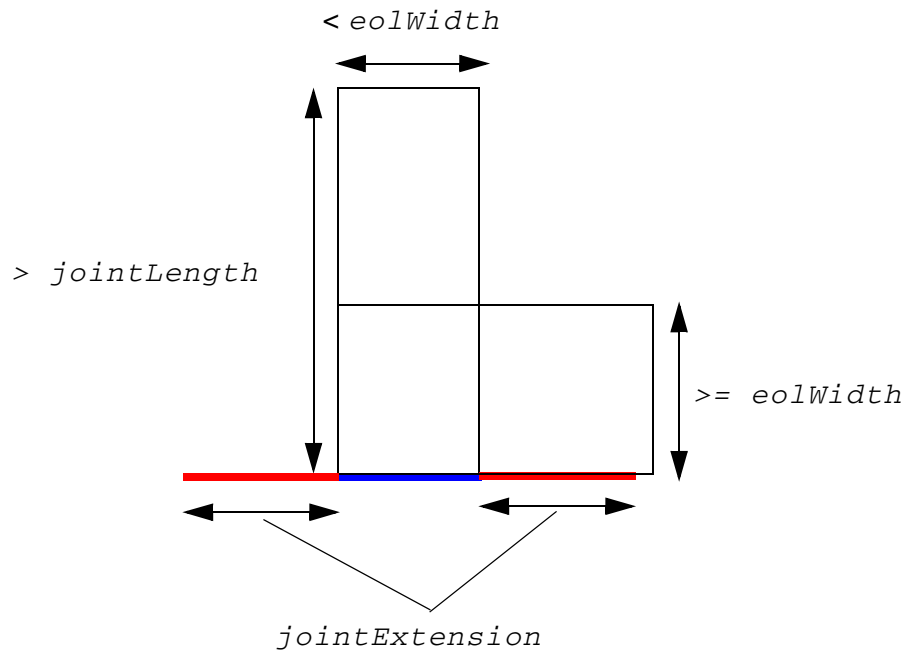
Only the blue edge satisfies the SIDEEDGELENGTH condition to be a SIDE.

**Figure 1-206 Illustration of SIDELENGTH and SIDEEXTENSION**



Blue edge represents the SIDELENGTH while red edge represents the SIDEEXTENSION.

**Figure 1-207 Illustration of JOINTEXTENSION**



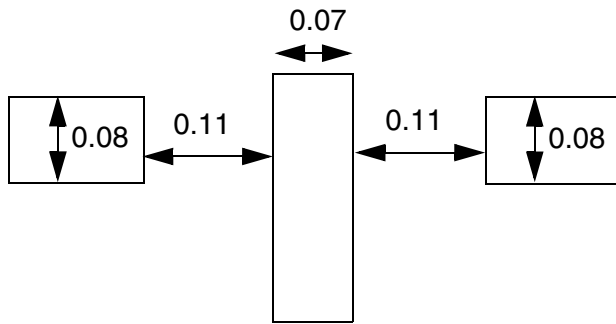
Blue edge is a JOINT while the red edges represent the JOINTEXTENSION.

## LEF/DEF 5.8 Language Reference

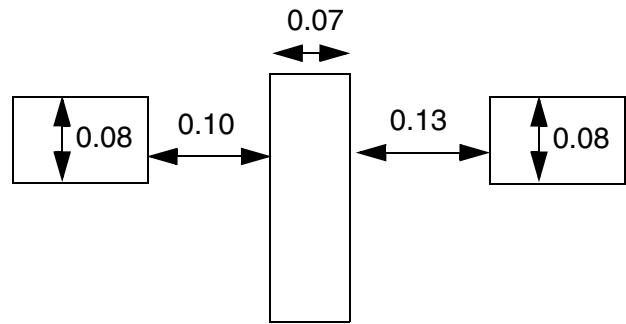
### LEF Syntax

**Figure 1-208 Illustration of OPPOSITEEOLSPACING**

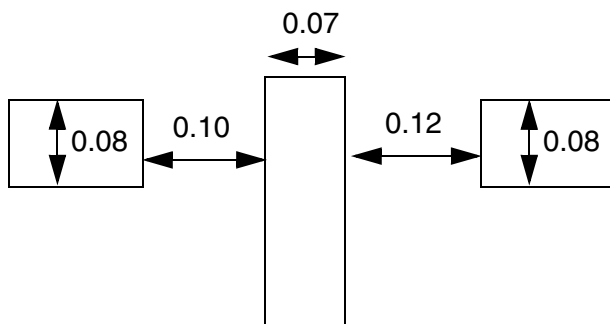
```
PROPERTY LEF58_OPPOSITEEOLSPACING  
  WIDTH 0.08  
  ENDWIDTH 0.1  
  JOINTLENGTH 0.15  
  EXCEPTEDGELENGTH 0.08 PRL 0.03  
  ENDTOEND 0.11 0.13  
  ENDTOJOINT 0.12 0.10  
  JOINTTOEND 0.12 0.10  
  JOINTTOJOINT 0.12 0.16;
```



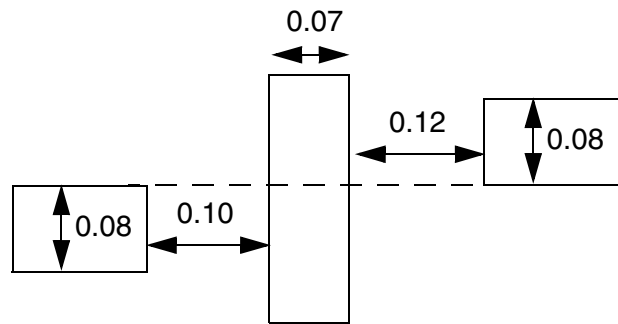
a) Okay, both of the neighbor spacings  
>= 0.11 (minimum of the spacings)



b) Okay, right spacing >= 0.13  
(maximum of the spacings)



c) Violation, left spacing < 0.11 to  
trigger the rule

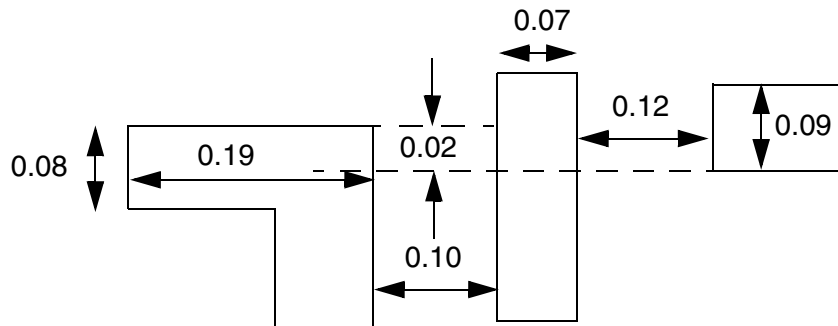


d) Okay, no projected parallel run length

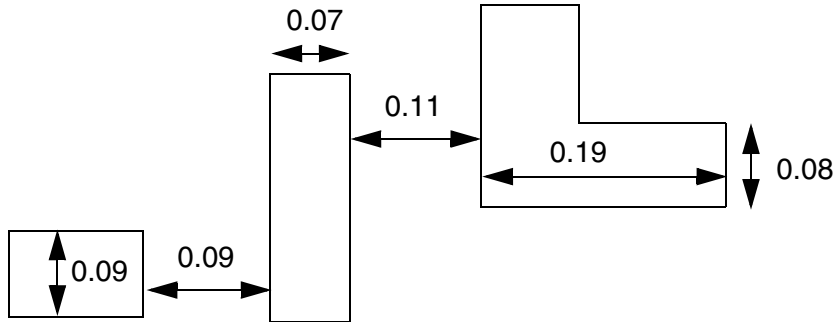
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



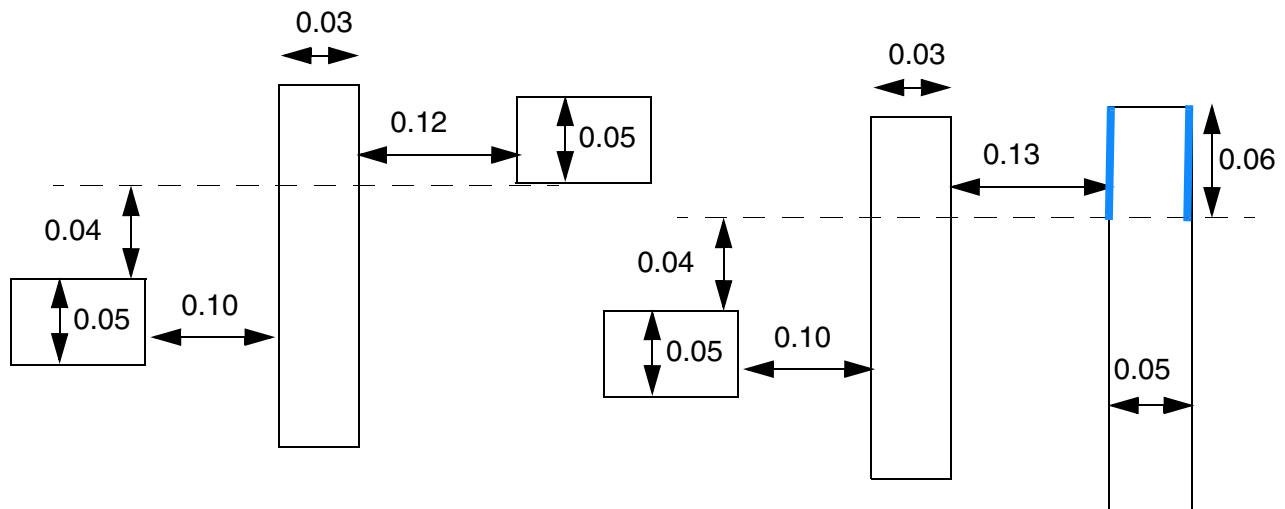
e) Okay, the exception conditions are met



f) Violation, ENDTOJOINT is fine, but JOINTTOEND is not

**Figure 1-209 Illustration of OPPOSITEEOLSPACING**

```
PROPERTY LEF58_OPPOSITEEOLSPACING
"OPPOSITEEOLSPACING
  WIDTH 0.08
  ENDWIDTH 0.1
  JOINTLENGTH 0.15
  SIDELENGTH 0.06
  PRL -0.05
  ENDOTEND 0.11 0.13
  ENDTOJOINT 0.12 0.10
  JOINTTOEND 0.12 0.10
  JOINTTOJOINT 0.12 0.16
  SIDETOEND 0.14 0.15
  SIDETOJOINT 0.14 0.15 ;"
```



a) Violation, the neighbor wires are within 0.05 (PRL -0.05) of each other

b) Violation, the blue edges are SIDE of a EOL edge, with 0.13 (< 0.14) spacing to the middle wire, and the opposite end wire has 0.10 (< 0.15) spacing

**Figure 1-210 Illustration of SIDELENGTH**

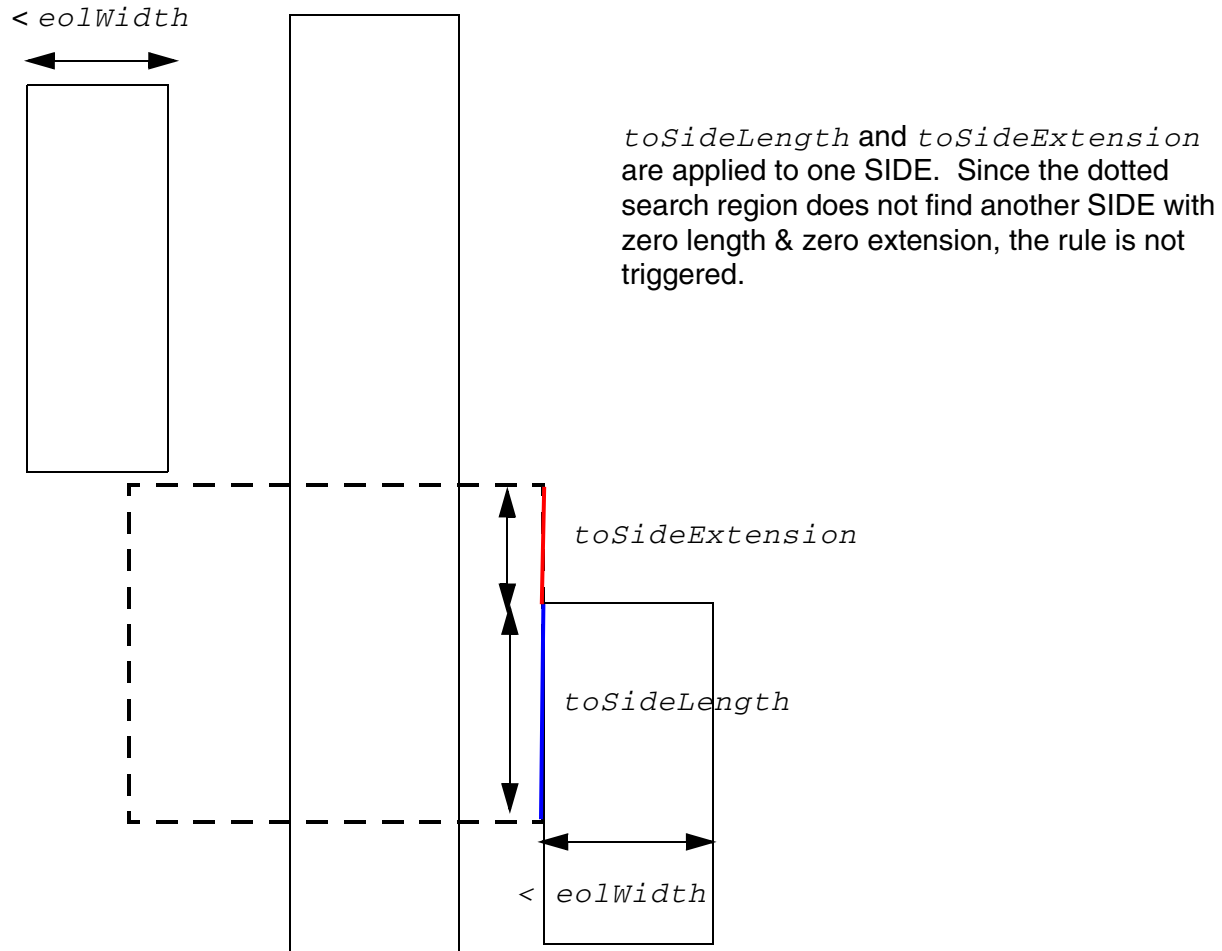


Illustration of SIDELENGTH *sideLength* TOSIDE *toSideLength*  
SIDEEXTENSION *sideExtension* TOSIDE *toSideExtension*

### **Right Way on Grid Only Rule**

The right way on grid only rule can be used to specify that the wires on the preferred routing direction of the layer must be on grid/track.

You can create a right way on grid only rule by using the following property definition:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_RIGHTWAYONGRIDONLY
    "RIGHTWAYONGRIDONLY [CHECKMASK]
    ; " ;
```

Where:

RIGHTWAYONGRIDONLY	Specifies that the wires on the preferred routing direction must be put on a grid/track.
CHECKMASK	Specifies that the mask/color of the wires on the preferred routing direction should be checked against the track color. For default-width wires, the wire and track color should be the same. If wide wires could block fewer tracks based on spacing by taking the reverse track color, the wire color should be the reverse of the track color.

### ***Two Wires Forbidden Spacing Rule***

Two wires forbidden spacing rules can be used to define a list of forbidden spacing ranges between two wires.

You can create a two wires forbidden spacing rule by using the following property definition:

```
PROPERTY LEF58_TWOWIRESFORBIDDENSPACING
    "TWOWIRESFORBIDDENSPACING [SAMEMASK | MASK maskNum]
    {minSpacing maxSpacing}...
    [HORIZONTAL|VERTICAL]
    { MINSPANLENGTH minSpanLength [EXACTSPANLENGTH]
      MAXSPANLENGTH maxSpanLength [EXACTSPANLENGTH]
    | FIRSTSPANLENGTH spanLength1 spanLength2
      [OTHERSPANLENGTH otherSpanLength1]
      SECONDSPANLENGTH spanLength3 spanLength4}
      [OTHERSPANLENGTH otherSpanLength2]
    PRL prl
    ; " ;
```

Where:

```
TWOWIRESFORBIDDENSPACING {minSpacing maxSpacing}...
    MINSPANLENGTH minSpanLength [EXACTSPANLENGTH]
    MAXSPANLENGTH maxSpanLength [EXACTSPANLENGTH]
    PRL prl
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that one or more set of forbidden spacing in the orthogonal direction of the preferred routing direction between two wires to be less than or equal to *minSpacing* and greater than or equal to *maxSpacing*. One of the wires must have a span length greater than or equal to *minSpanLength* or exactly equal to the given span length if EXACTSPANLENGTH is defined. The other wire must have a span length less than or equal to *maxSpanLength* or exactly equal to the given span length if EXACTSPANLENGTH is defined. The two wires must also have a parallel run length greater than *prl*.

*Type:* Float, specified in microns

FIRSTSPANLENGTH *spanLength1 spanLength2*  
SECONDSPANLENGTH *spanLength3 spanLength4*

Specifies that the span length of one wire must be greater than or equal to *spanLength1* and less than or equal to *spanLength2* and the other wire must be greater than or equal to *spanLength3* and less than or equal to *spanLength4* to trigger the rule.

*Type:* Float, specified in microns

HORIZONTAL | VERTICAL Specifies that the check applies only on the given direction on the spacing.

MASK *maskNum* Specifies that the forbidden spacing applies only on the given mask. *maskNum* must be a positive integer, and most applications support only the values 1, 2, or 3.

*Type:* Integer

OTHERSPANLENGTH *otherSpanLength*

Specifies that the rule applies only if the span length in the orthogonal direction is greater than *otherSpanLength*. OTHERSPANLENGTH should be defined along with either HORIZONTAL or VERTICAL.

*Type:* Float, specified in microns

SAMEMASK Specifies that the forbidden spacing rule applies only to two wires on the same mask. This rule is applied independent of whether there is a neighbor wire with a different mask between them.

### Two Wires Forbidden Spacing Example

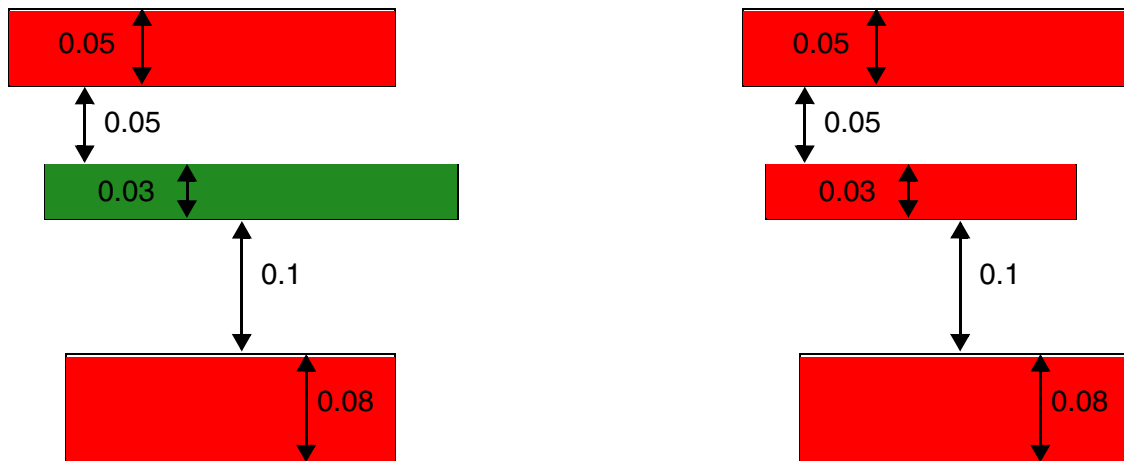
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- In the following example, the specified forbidden spacing rules are applied only in the horizontal direction:

```
DIRECTION HORIZONTAL ;  
PROPERTY LEF58_TWOWIRESFORBIDDENSPACING `  
    TWOWIRESFORBIDDENSPACING SAMEMASK 0.16 0.20  
    MINSPANLENGTH 0.05 EXACTSPANLENGTH  
    MAXSPANLENGTH 0.08 EXACTSPANLENGTH PRL 0 ; `` ;
```

**Figure 1-211 Illustration of TWOWIRESFORBIDDENSPACING**



a) Violation, the spacing between the top and bottom wires is 0.18, which is within the forbidden spacing of  $\geq 0.16$  and  $\leq 0.20$ . It is a violation with or without the middle 0.03 different-mask wire. If the middle wire was red, it would not be a violation.

b) Violation, the middle 0.03 same-mask wire does not fully cover the top and bottom wires. It is a violation based on the parallel run length on the right.

**Figure 1-212 Illustration of TWOWIRESFORBIDDENSPACING with OTHERSPANLENGTH**

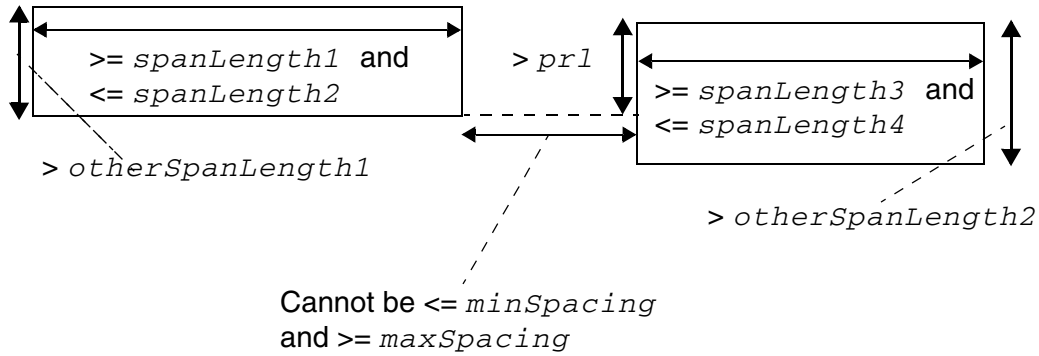


Illustration of TWOWIRESFORBIDDENSPACING *minSpacing*  
*maxSpacing*

```
FIRSTSPANLENGTH spanLength1 spanLength2
OTHERSPANLENGTH otherSpanLength1
SECONDSPANLENGTH spanLength3 spanLength4
OTHERSPANLENGTH otherSpanLength2
PRL prl
```

### ***Voltage Spacing Rule***

Voltage spacing rules can be used to define different-net spacing between objects on the metal layer and objects on the metal/cut layer above or below it according to different voltages.

You can create a voltage spacing rule by using the following property definition:

```
PROPERTY LEF58_VOLTAGESPACING
    "VOLTAGESPACING [TOCUT [ABOVE | BELOW]]
        {voltage spacing} ...
        ;..." ;
```

Where:

```
VOLTAGESPACING {voltage spacing} ...
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines different-net spacing among wires in the same routing layer in different voltages. Both the voltage and spacing values should be monotonically increasing. The spacing value is required for any net with a worst-case voltage between its neighbors, including negative minimum voltage greater than or equal to the voltage in the table. For example, if you have 1.5v and 2.0v rules, then the spacing between a net with a max 1.3v and a min 0.0v and a net with a max 1.4v and a min -0.5v should follow the 1.5v spacing since the worst-case voltage is  $\max(1.3v - (-0.5v) = 1.8v, 1.4v - 0.0v = 1.4v) = 1.8v$ .

TOCUT [ABOVE | BELOW]

Defines the spacing for wires in different voltages in routing layer to all cut vias in adjacent above and below cut layers. If ABOVE or BELOW is specified, the rule only applies to cuts in the above or below cut layer correspondingly.

*Type:* Float, *voltage* in volts and *spacing* in microns

### Examples

- The following example indicates that 0.1  $\mu\text{m}$  is the wire to wire spacing for voltage greater than or equal to 2.0V and less than 3.0V, 0.15  $\mu\text{m}$  is the wire to wire spacing for voltage greater than or equal to 3.0V and less than 4.0V, and 0.2  $\mu\text{m}$  is the wire to wire spacing for voltage greater than or equal to 4.0V:

```
VOLTAGESPACING
2.0      0.1
3.0      0.15
4.0      0.2 ;
```

- The following example indicates the same as above, except that the spacing is between wires in metal layer to all cut vias in the adjacent above and below cut layers:

```
VOLTAGESPACING TOCUT
2.0      0.1
3.0      0.15
4.0      0.2 ;
```

### Manufacturing Grid Rule

Manufacturing grid rules can be used to specify the manufacturing grid on a given layer.

You can create a manufacturing grid rule by using the following property definition:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_MANUFACTURINGGRID
    "MANUFACTURINGGRID value
    ; " ;
```

Where:

*MANUFACTURINGGRID value*

Specifies the manufacturing grid on the given layer that will override the library manufacturing grid value.

*Type:* Float, specified in microns

### ***Rectangle Only Rule***

Rectangle only rules can be used to indicate that a layer allows rectangular objects only.

You can create a rectangle only rule by using the following property definition:

```
PROPERTY LEF58_RECTONLY
    "RECTONLY [EXCEPTNONCOREPINS]
    ; " ;
```

Where:

**EXCEPTNONCOREPINS** Specifies that the rectangular object check requirement is exempted for a connection to the pins that are not in a standard cell (with **CLASS CORE**). In other words, the connection of a default-width wire to a wide I/O pad or block pin is allowed.

**RECTONLY** Specifies that the layer can only allow rectangular objects. In other words, rectilinear shapes with jogs or concave corners are illegal.  
*Type:* Float, specified in microns

### ***Region Rule***

You can create a region rule to define area-based rules on a routing layer.

You can use the following property definition:

```
PROPERTY LEF58_REGION
    "REGION regionLayerName BASEDLAYER routingLayerName
    ; " ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

`REGION regionLayerName BASEDLAYER routingLayerName`

Specifies a set of region- or area-based rules on a routing layer *routingLayerName*. *regionLayerName* is a masterslice layer with type `REGION`. Users can create a blockage on that masterslice layer on a design to indicate the areas of the region. In addition, `MACRO` may also have `OBS` on that masterslice layer to indicate that the areas of its instance should be subjected to these region-based rules. All the rules specified on this routing layer with `REGION` are applied to the corresponding areas of *regionLayerName* on *routingLayerName*. The areas outside *routingLayerName* honor the set of rules on the given routing layer without type `REGION`. If a rule check crosses the boundaries of a region, it is subjected to the rules for both inside and outside the region, with the tighter rule values of the two sets dominating.

The set of rules defined in the layer with type `REGION` should be very small as compared to the layer without type `REGION`. The rules defined in this layer with type `REGION` should be a replacement of the corresponding rule (having the same set of the constructs of a LEF statement or a complete replacement on a table like `SPACINGTABLE`) on the layer without type `REGION`. All the other rules defined on the layer without type `REGION` are also applicable on the layer with type `REGION`.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

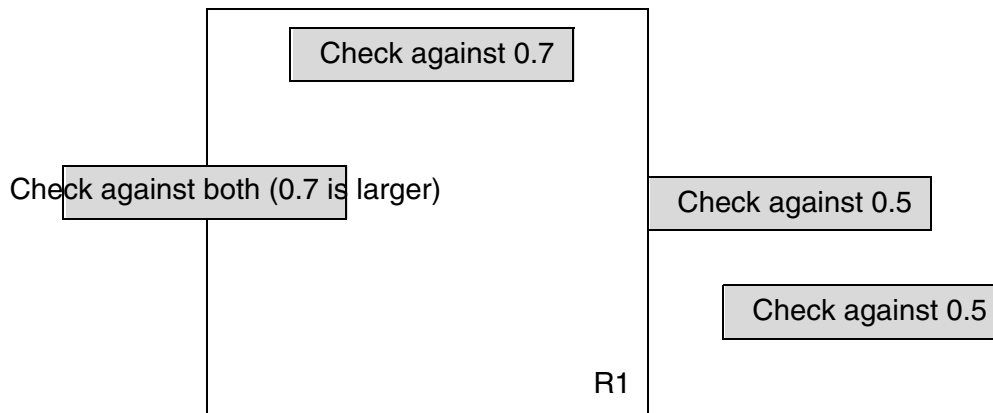
#### Region Rule Example

The following example means that a rectangular object overlapping region R1 should have spacing of 0.7 and a rectangular object outside region R1 should have spacing of 0.5. Non-rectangular object in region R1 would still follow 0.8 and any other rules on layer M1 (without REGION) would also be applied on region R1:

```
LAYER R1
    TYPE MASTERSLICE ;
    PROPERTY LEF58_TYPE "TYPE REGION ; " ;
END R1
...
LAYER M1
    TYPE ROUTING ;
    ...
    AREA 0.5 ;
    PROPERTY LEF58_AREA "AREA 0.8 EXCEPTRECTANGLE ; " ;
    ...
END M1

LAYER M1R1
    TYPE ROUTING ;
    PROPERTY LEF58_REGION "REGION R1 BASEDLAYER M1 ; " ;
    ...
    AREA 0.7 ;
    ...
END M1R1
```

**Figure 1-213 Illustration of Region Rule**



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### **Backside Rule**

A backside rule can be used to specify that the routing layer is used on the underside of the die.

You can create a backside rule by using the following property definition:

```
PROPERTY LEF58_BACKSIDE
    "BACKSIDE ;" ;
```

Where:

BACKSIDE                      Indicates that the routing layer is a backside routing layer.

The POLYROUTING, MIMCAP ([PROPERTY LEF58\\_TYPE](#)), and BACKSIDE keywords are mutually exclusive; you cannot specify them together.

#### **Perpendicular EOL Spacing Rule**

Perpendicular EOL spacing rules can be specified to allow a different spacing rule for perpendicular edges versus opposite edges.

You can create a perpendicular EOL spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING
    "SPACING eolSpace EOLPERPENDICULAR eolWidth perWidth ;" ;
```

Where:

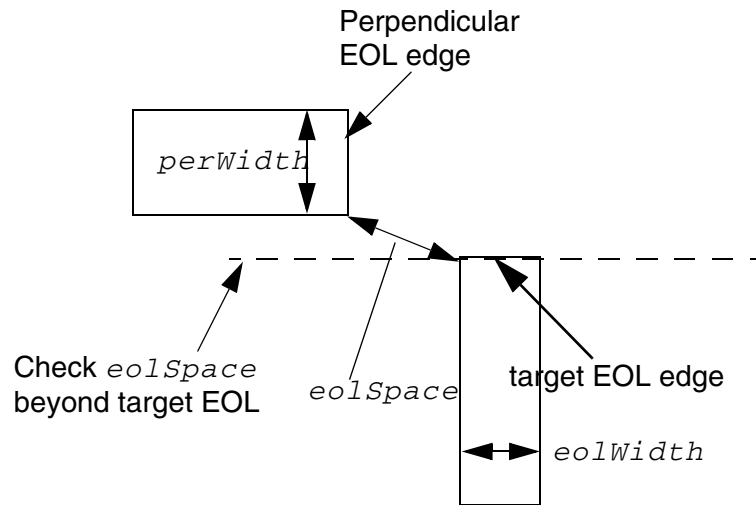
SPACING *eolSpace* EOLPERPENDICULAR *eolWidth perWidth*

Indicates that an EOL edge with a width that is less than *eolWidth* requires spacing greater than or equal to *eolSpace* beyond the EOL, to a perpendicular edge with a width that is less than *perWidth*. Typically, *eolSpace* is slightly larger than the minimum allowed spacing on this layer.  
*Type:* Float, specified in microns (for all values)

#### **Perpendicular EOL Spacing Rule Examples**

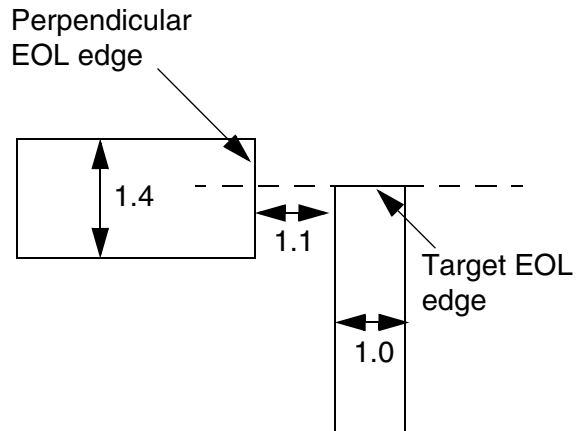


**Figure 1-214 Illustration of Perpendicular EOL Spacing Parameters**

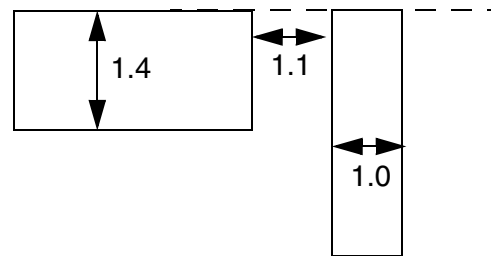


**Figure 1-215 Example of Perpendicular EOL Spacing Rule**

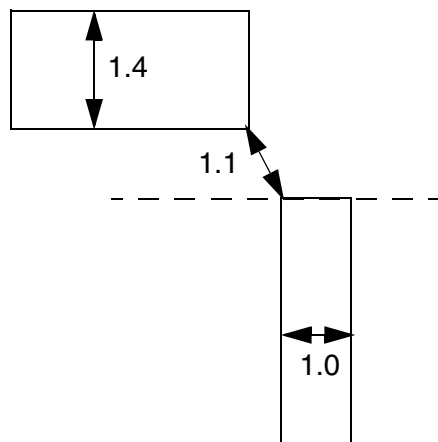
```
PROPERTY LEF58_SPACING "SPACING 1.2 EOLPERPENDICULAR 1.1 1.5 ;"
```



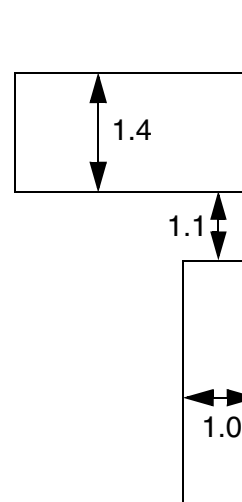
a) Violation. EOL is  $< 1.1$  wide, and has a perpendicular edge  $< 1.5$  wide that is  $< 1.2$  distance from EOL edge.



b) No violation. Perpendicular edge is not "seen" by EOL edge, so it is not checked.



c) Violation. EOL is  $< 1.1$  wide, and has a perpendicular edge  $< 1.5$  wide that is  $< 1.2$  distance from EOL edge.



d) No violation. Perpendicular edge is not "seen" by EOL edge, so it is not checked.

### Spacing Area Rule

Area spacing rules can be specified between objects having area less than the maximum area to any other objects in the layer.

You can create an area spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING  
    "SPACING minSpacing AREA maxArea ;" ;
```

Where:

*AREA maxArea*

Specifies the minimum spacing rule only applies between an object with area less than *maxArea* to any other objects in the layer.

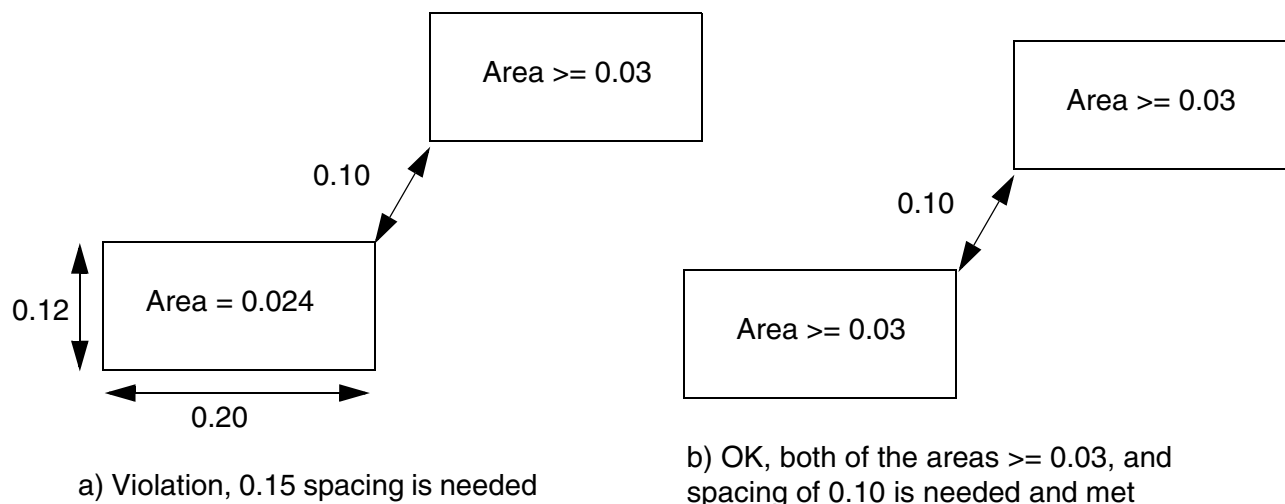
*Type:* Float, specified in microns squared

### Examples

The following spacing rule indicates that the minimum spacing of an object with area less than  $0.03 \mu\text{m}^2$  is  $0.15 \mu\text{m}$ , while the minimum spacing of another object in the layer is  $0.10 \mu\text{m}$ :

```
SPACING 0.10 ;  
PROPERTY LEF58_SPACING  
    "SPACING 0.15 AREA 0.03 ;" ;
```

**Figure 1-216 Illustration of Spacing Area Rule**



## Spacing Layer Rule

Spacing layer rules can be specified for shapes in a TRIMMETAL layer.

You can create a spacing layer rule by using the following property definition:

```
PROPERTY LEF58_SPACING
    "SPACING minSpacing LAYER trimLayer [TRIMMASK trimMaskNum]
    [MASK maskNum] EXCEPTOVERLAP overlapLength
    [TRIMENDSPACING trimEndSpacing]
    ;" ;
```

Where:

*SPACING minSpacing LAYER trimLayer EXCEPTOVERLAP overlapLength*

Specifies that the spacing to shapes in *trimLayer*, which must be a layer with TYPE TRIMMETAL, should be greater than or equal to *minSpacing*. However, there is an exemption. When a shape in the current layer overlaps with a *trimLayer* shape by a length that is greater than or equal to *overlapLength* as measured in the preferred routing direction in the current layer, the spacing check against that *trimLayer* shape is skipped. If *overlapLength* is a negative value and if the two shapes are within the  $\text{abs}(\text{overlapLength})$  value, the spacing rule is not applied. This is particularly meaningful when the trim metal shape and the metal shape are in a corner-to-corner case.

*Type:* Float, specified in microns

*MASK maskNum*

Specifies that spacing to shapes in *trimLayer*, applies only on the given mask of the current layer. *maskNum* must be a positive integer, and most applications only support values of 1, 2 or 3.

*TRIMENDSPACING trimEndSpacing*

Specifies that the metal spacing to the end edge, which is parallel to the preferred routing direction of the metal routing layer, of shapes in *trimLayer* should be greater than or equal to *trimEndSpacing*.

*Type:* Float, specified in microns

*TRIMMASK trimMaskNum*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

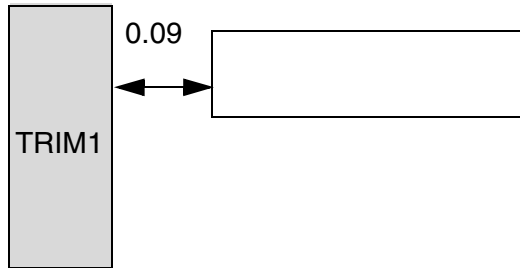
Specifies that the spacing for shapes in *trimLayer* applies only on the given mask of the *trimLayer* layer.

*trimMaskNum* must be a positive integer, and most applications only support values of 1, 2, or 3.

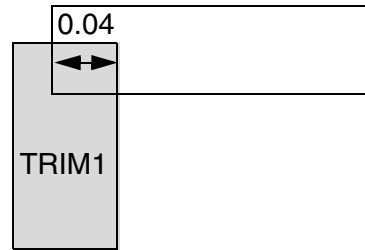
### Examples

The following spacing rule indicates that the spacing to shapes in `TRIMMETAL layer TRIM1` should be greater than or equal to 0.1  $\mu\text{m}$ , and the spacing to the end edges of shapes in the `TRIMMETAL layer TRIM1` should be greater than or equal to 0.12  $\mu\text{m}$ . When a shape in the current layer overlaps with a `TRIM1` shape by a length greater than or equal to 0.05 in the horizontal direction in the current layer, the spacing check against that `TRIM1` shape is skipped.

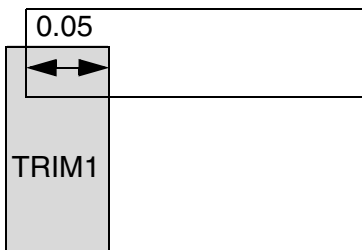
**Figure 1-217 Illustration of Spacing Layer Rule**



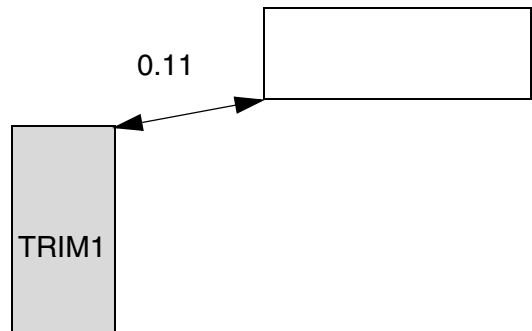
a) Violation, spacing to TRIM1 needs to be  $\geq 0.1$



b) Violation, horizontal (preferred routing direction) overlap must be  $\geq 0.05$  to be exempted



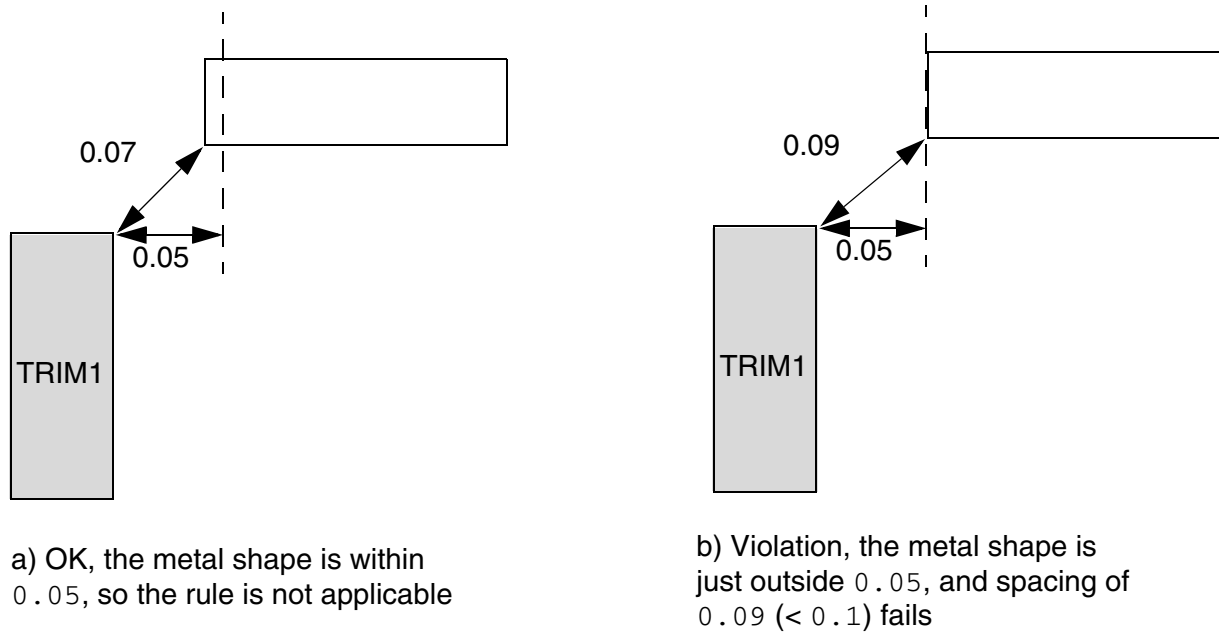
c) OK, overlap exemption condition is met



d) Violation, 0.12 spacing is needed to the end edge of the TRIM1 shape and is not met

```
DIRECTION HORIZONTAL ;
PROPERTY LEF58_SPACING "
    SPACING 0.1 LAYER TRIM1 EXCEPTOVERLAP 0.05
    TRIMENDSPACING 0.12;" ;
```

**Figure 1-218 Illustration of Spacing Layer Rule with Negative Overlap Length**



#### Example of

```
DIRECTION HORIZONTAL ;  
PROPERTY LEF58_SPACING "  
    SPACING 0.1 LAYER TRIM1 EXCEPTOVERLAP -0.05 ; " ;
```

### ***Samemask Spacing Rule***

Same mask spacing rule exists for double patterning technology, when two masks with different spacing requirements among them are applied on one routing layer. This adds a same mask spacing.

You can create same mask spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING  
    "SPACING minSpacing SAMEMASK ;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

**SAMEMASK** Specifies the minimum spacing rule only applies between objects on the same mask.

### Examples

- The following same mask spacing rule indicates that the minimum spacing of objects on the same mask in the routing layer is 0.15  $\mu\text{m}$  while the minimum spacing of objects on different masks is 0.10  $\mu\text{m}$ .

```
SPACING 0.10 ;
PROPERTY LEF58_SPACING
    "SPACING 0.15 SAMEMASK ; " ;
```

### ***Wrong Direction Spacing Rule***

Wrong direction spacing rule exists for minimum spacing, which is larger than the right direction spacing, between the long or side edges of two wires with direction perpendicular to the specified direction in `DIRECTION` on a Manhattan routing layer.

You can create wrong direction spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING
    "SPACING minSpacing [WRONGDIRECTION [NONEOL eolWidth]
        [PRL prl] [LENGTH length]]
    ;" ;
```

Where:

All other keywords are the same as the existing LEF routing layer `SPACING` syntax.

**LENGTH *length*** Specifies that the wrong direction spacing is switched to apply to the short/end edges if the length of both the wires is less than or equal to the specified *length*. If `PRL` is also defined, the common parallel run length will apply to short/end edges.  
*Type:* Float, specified in microns



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

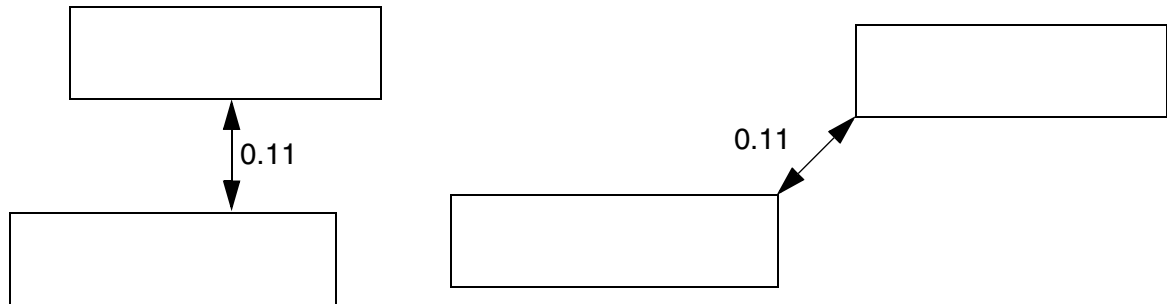
NONEOL <i>eolWidth</i>	<p>Specifies the minimum spacing between two edges having parallel run length greater than 0 with direction perpendicular to the specified direction in <code>DIRECTION</code> on a Manhattan routing layer. Those edges are not end-of-line edges with length less than the <i>eolWidth</i>.</p> <p>Type: Float, specified in microns</p>
PRL <i>prl</i>	<p>Specifies that the wrong direction spacing is only applied if long/side edges of two wires has common parallel run length greater than <i>prl</i>. If <i>prl</i> is a negative value, the two edges will be checked in <code>WITHIN</code> style.</p> <p>Type: Float, specified in microns</p>
WRONGDIRECTION	<p>Specifies the minimum spacing, which is larger than the right direction spacing, between two edges having parallel run length greater than 0 with direction perpendicular to the specified direction in <code>DIRECTION</code> on a Manhattan routing layer.</p>

### Examples

- The following example illustrates wrong direction spacing rule:

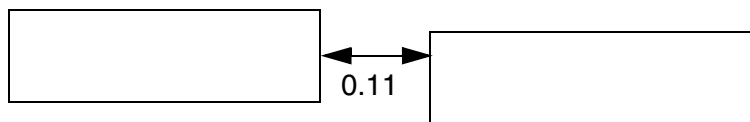
```
DIRECTION VERTICAL ;
SPACING 0.10 ;
PROPERTY LEF58_SPACING
    "SPACING 0.12 WRONGDIRECTION ; " ;
```

**Figure 1-219 Illustration of Spacing with WRONGDIRECTION**

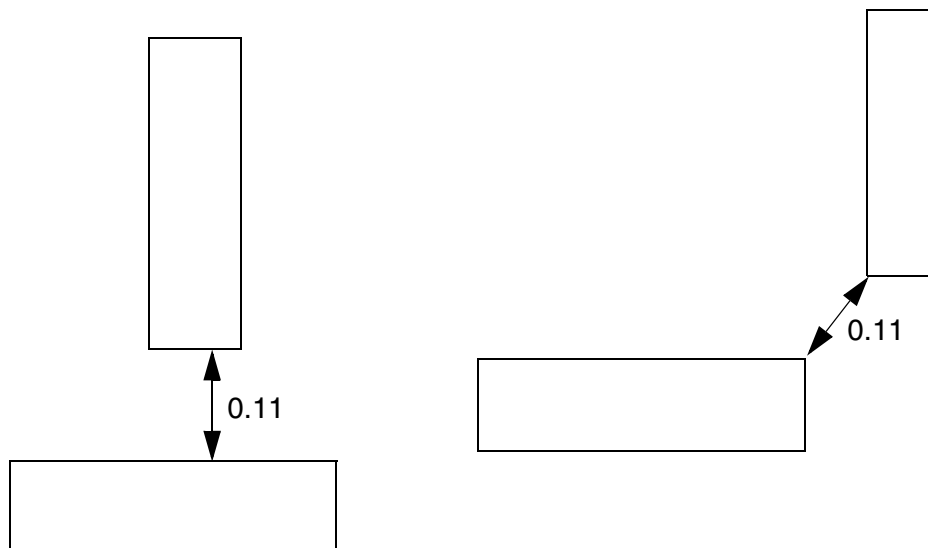


a) Violation, 0.12 spacing is needed between 2 horizontal edges

b) OK, the edges do not have common parallel run length > 0



c) OK, 0.10 spacing is needed between the short/end edges of horizontal wires and met



d) Violation, 0.12 spacing is needed between 2 horizontal edges independent of the wire direction

e) OK, 0.10 spacing is needed when parallel run length  $\leq 0$

## LEF/DEF 5.8 Language Reference

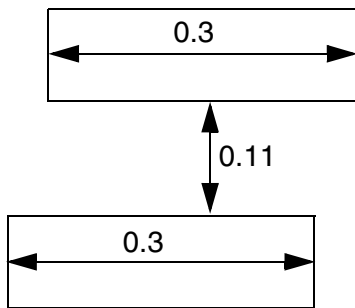
### LEF Syntax

- The following wrong direction spacing rule indicates that the long/side edges between two horizontal wires must have a spacing of 0.12  $\mu\text{m}$  if they are not EOL edges with length less than 0.15  $\mu\text{m}$  and they have common parallel run length greater than 0. Otherwise, 0.10  $\mu\text{m}$  is the minimum spacing.

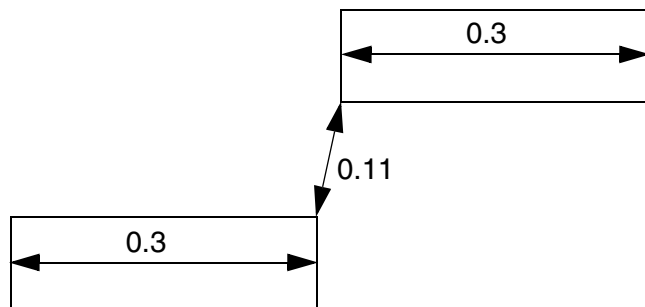
```
DIRECTION VERTICAL ;  
SPACING 0.10 ;  
PROPERTY LEF58_SPACING  
    "SPACING 0.12 WRONGDIRECTION NONEOL 0.15 ; " ;
```

**Figure 1-220 Illustration of WRONGDIRECTION with NONEOL**

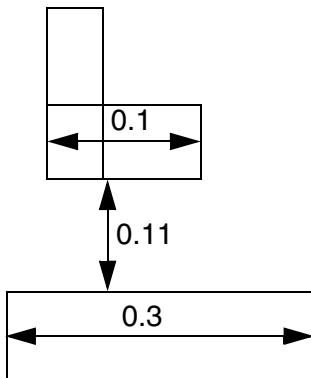
```
DIRECTION VERTICAL ;  
SPACING 0.10 ;  
PROPERTY LEF58_SPACING  
    "SPACING 0.12 WRONGDIRECTION NONEOL 0.15 ; " ;
```



a) Violation, 0.12 spacing is needed between 2 long/side edges of 2 horizontal wires



b) OK, the edges do not have common parallel run length > 0



c) OK, the top horizontal edge is an EOL edge, only 0.10 spacing is needed and met

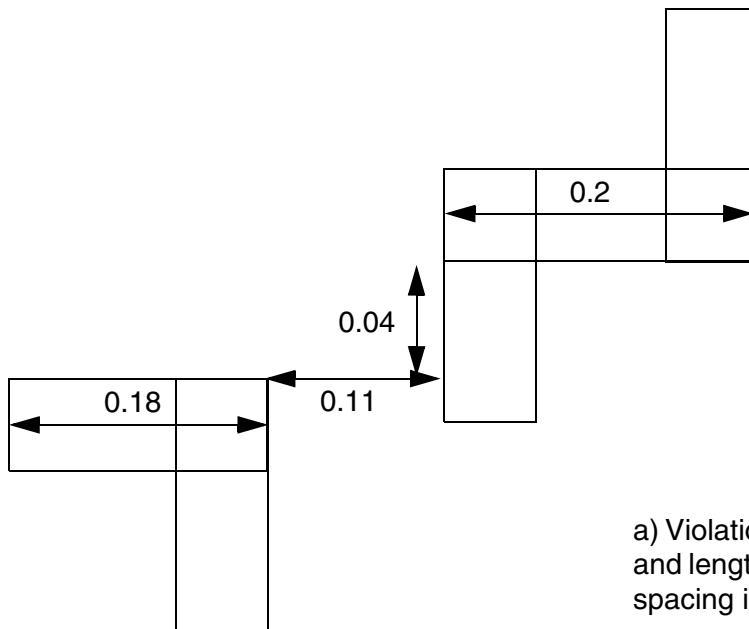
## LEF/DEF 5.8 Language Reference

### LEF Syntax

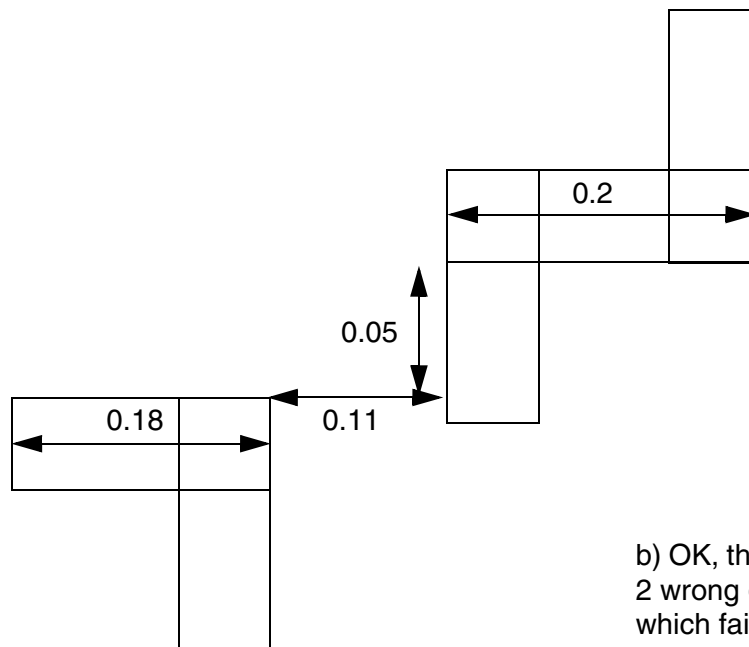
---

**Figure 1-221 Illustration of WRONGDIRECTION with PRL and LENGTH**

```
DIRECTION VERTICAL ;  
SPACING 0.10 ;  
PROPERTY LEF58_SPACING  
  "SPACING 0.12 WRONGDIRECTION  
    PRL -0.05 LENGTH 0.2 ; " ;
```



a) Violation, both parallel run length and length conditions are met, 0.12 spacing is needed

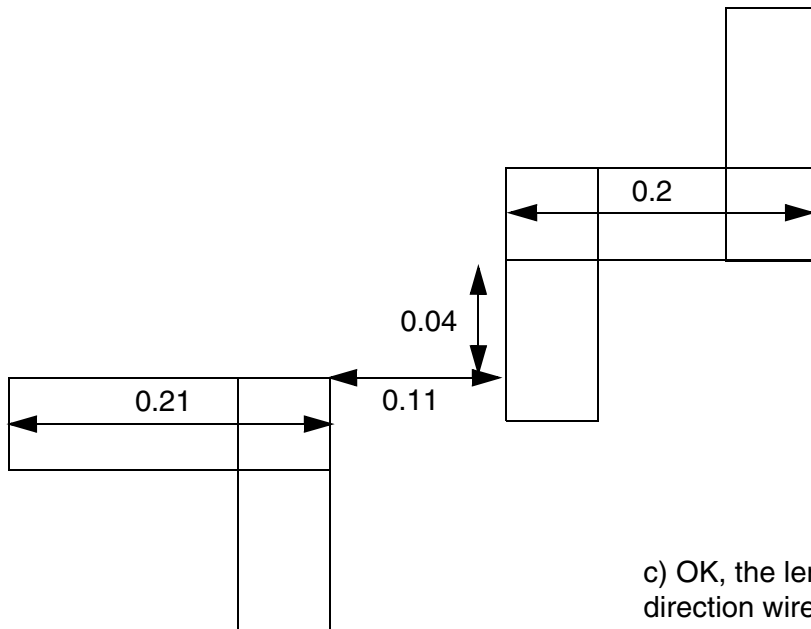


b) OK, the parallel run length of the 2 wrong direction wires is -0.05, which fails the PRL condition

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



c) OK, the length of the left wrong direction wire is 0.21 ( $> 0.20$ )

### **Notch Length Spacing Rule**

Notch length spacing rule exists for any notch with a notch length less than the minimum notch length with notch spacing of less than or equal to the minimum spacing.

You can create notch length spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING
    "SPACING minSpacing NOTCHLENGTH minNotchLength
    [EXCEPTWITHIN lowExcludeSpacing highExcludeSpacing]
    [WITHIN within SPANLENGTH sideOfNotchSpanLength
    | {WIDTH | CONCAVEENDS} sideOfNotchWidth
    | NOTCHWIDTH notchWidth]
    ;" ;
```

Where:

All other keywords are the same as the existing LEF routing layer SPACING syntax.

CONCAVEENDS *sideOfNotchWidth*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the minimum notch length spacing only applies if the width of both sides of the notch must be less than or equal to *sideOfNotchWidth*. In addition, one of the side edges with length less than *minNotchLength* must be between two concave corners at the ends, and the length of the opposite edge must be greater than or equal to *minNotchLength*.  
*Type*: Float, specified in microns

EXCEPTWITHIN *lowExcludeSpacing highExcludeSpacing*

Specifies that it is legal to have notch spacing greater than or equal to *lowExcludeSpacing* but less than *highExcludeSpacing*.  
*Type*: Float, specified in microns

NOTCHWIDTH *notchWidth*

Specifies that the notch length rule applies only if the width of the notch is less than or equal to *notchWidth*.  
*Type*: Float, specified in microns

WIDTH *sideOfNotchWidth*

Specifies that the minimum notch length spacing only applies if the width of at least one side of the notch is greater than or equal to the specified *sideOfNotchWidth*.

WITHIN *within* SPANLENGTH *sideOfNotchSpanLength*

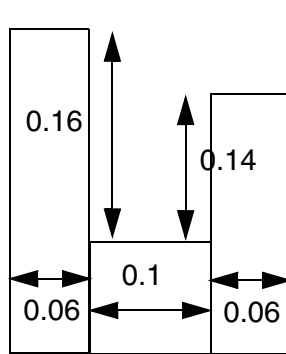
Specifies that if one of the side edges of the notch is within *within* distance and any portion of that edge has span length less than the *sideOfNotchSpanLength* and that portion has length greater than or equal to *minNotchLength* (this has a reverse definition of less than the *minNotchLength* without WITHIN), the notch spacing must be greater than or equal to the *minSpacing*. Otherwise, it would be a violation.  
*Type*: Float, specified in microns

### Notch Length Spacing Rule Examples

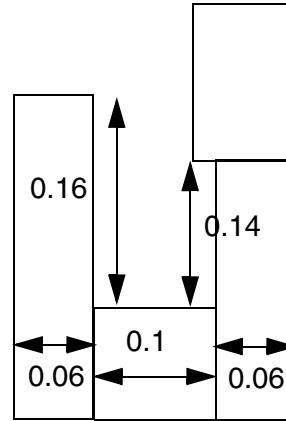
- The following spacing rule example shows notch length with CONCAVEENDS:

```
SPACING 0.10 ;  
PROPERTY LEF58_SPACING  
    "SPACING 0.12 NOTCHLENGTH 0.15 CONCAVEENDS 0.08 ; " ;
```

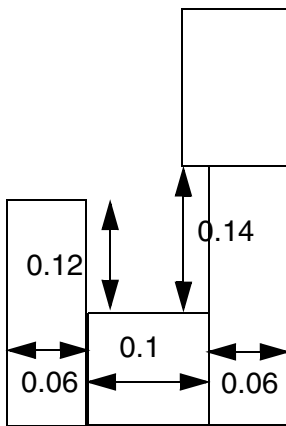
**Figure 1-222 Illustration of Spacing Rule with CONCAVEENDS**



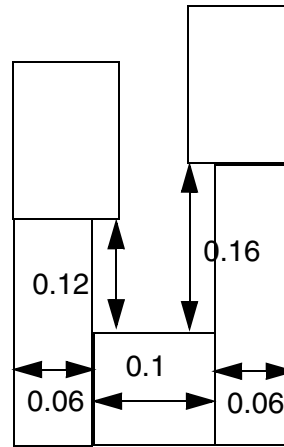
a) OK, the notch length of 0.14 is not between 2 concave corners. Without CONCAVEENDS, it is a violation.



b) Violation, both of the notch length conditions are met



c) OK, the length of the side edge without 2 concave corners is < 0.15

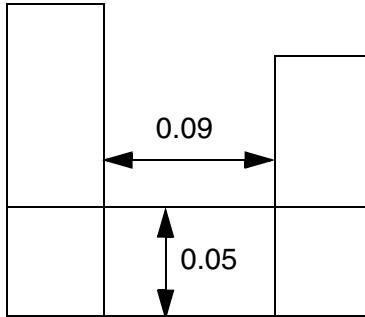


d) OK, this situation with concave corners on both of the side edges is fine

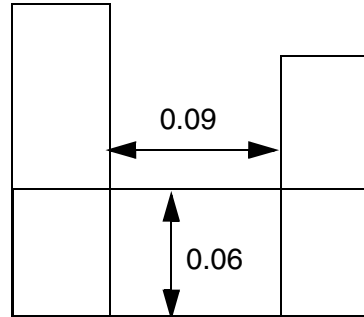
- The following spacing rule example shows notch length with EXCEPTWITHIN and NOTCHWIDTH:

```
PROPERTY LEF58_SPACING "  
    SPACING 0.1 NOTCHLENGTH 0  
    EXCEPTWITHIN 0.06 0.08 NOTCHWIDTH 0.05 ; " ;
```

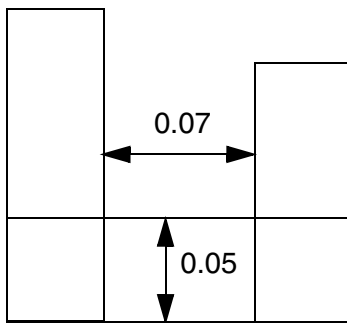
**Figure 1-223 Illustration of Spacing Rule with EXCEPTWITHIN and NOTCHWIDTH**



a) Violation, notch width of 0.05 ( $\leq 0.05$ ) and notch spacing of 0.09 ( $< 0.1$  and is not within the except within range)



b) OK, the notch width of 0.06 ( $> 0.05$ ) would not trigger



c) OK, notch spacing of 0.07 is within the except within range ( $\geq 0.06$  and  $< 0.08$ )



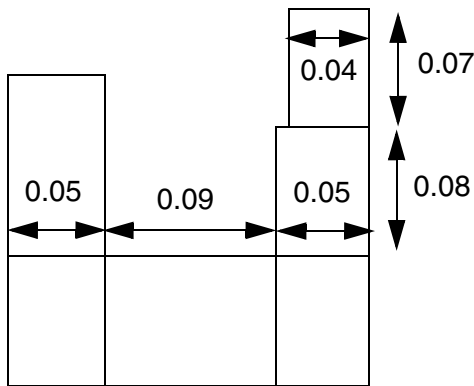
## LEF/DEF 5.8 Language Reference

### LEF Syntax

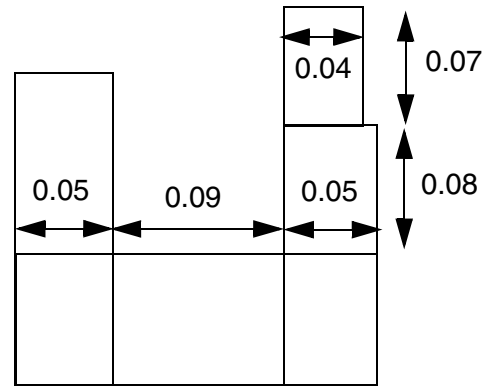
- The following spacing rule example shows notch length with `WITHIN` and `SPANLENGTH`:

```
SPACING 0.09 ;  
PROPERTY LEF58_SPACING  
    "SPACING 0.1 NOTCHLENGTH 0.06 WITHIN 0.2  
    SPANLENGTH 0.05 ; "
```

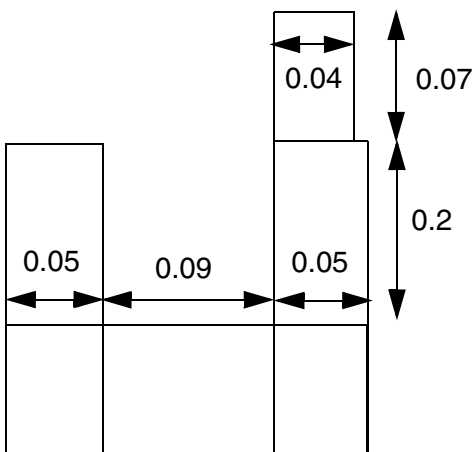
**Figure 1-224 Illustration of Spacing Rule with `WITHIN` and `SPANLENGTH`**



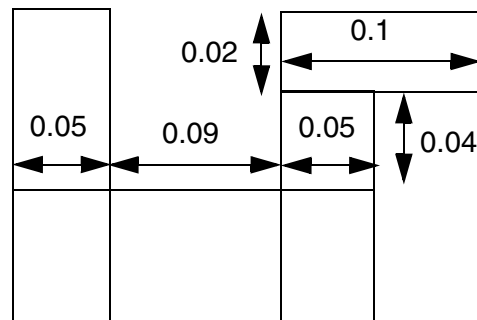
a) OK, only the bottom portion of right edge with width of 0.05 forming the notch is considered, and the top portion of 0.04 width/span length is ignored.



b) Violation, the top portion of the right side notch is within 0.2 and has span length of 0.04 ( $< 0.05$ ) with length of 0.07 ( $\geq 0.06$ )



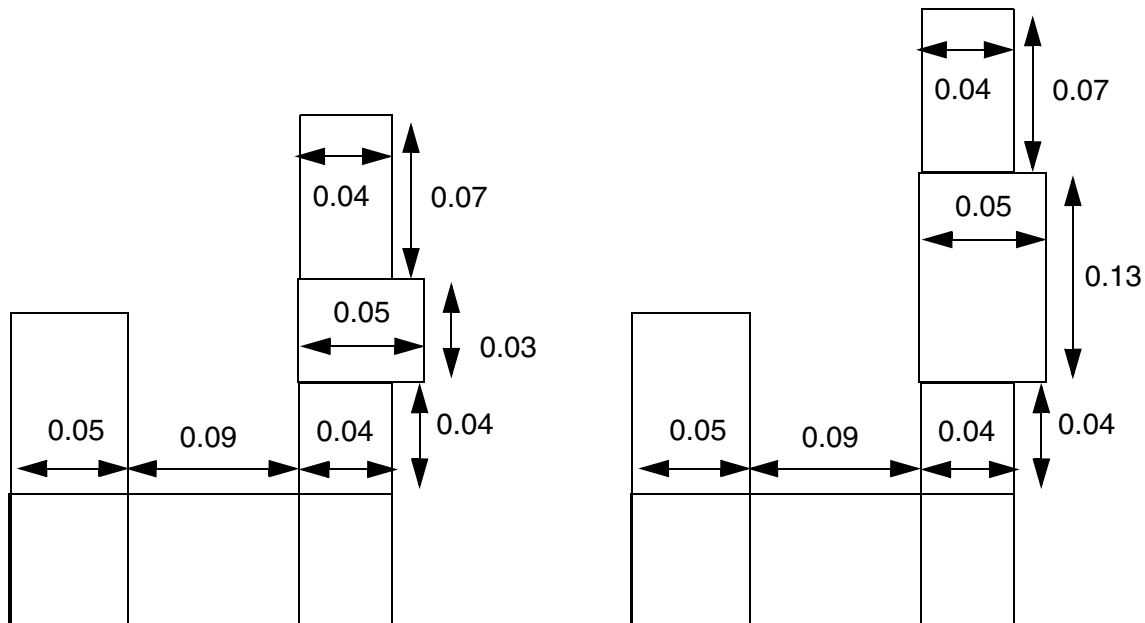
c) OK, the top portion of the right side notch is outside the length range  $> 0.2$



d) OK, the bottom portion of the right side notch has length of 0.04 ( $< 0.06$ ), and the top portion has span length  $\geq 0.05$  (width of 0.02 is irrelevant)

**Figure 1-225 Illustration of Spacing Rule with WITHIN and SPANLENGTH**

```
PROPERTY LEF58_SPACING
  "SPACING 0.1 NOTCHLENGTH 0.06 WITHIN 0.2
    SPANLENGTH 0.05 ; " ;
```



a) Violation, the top portion of the right side notch is within 0.2 and has span length of 0.04 (< 0.05) with length of 0.07 (>= 0.06)

b) OK, the top portion of the right side notch only has length of 0.03 (< 0.06) within 0.2

### ***Notch Span Spacing Rule***

Notch span spacing rule exists for any notch with span less than the specified span on both sides.

You can create notch span spacing rule by using the following property definition:

```
PROPERTY LEF58_SPACING
  "SPACING minSpacing NOTCHSPAN span NOTCHSPACING notchSpacing
    EXCEPTNOTCHLENGTH notchLength
  ; " ;
```

Where:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

All other keywords are the same as the existing LEF routing layer SPACING syntax.

```
SPACING minSpacing NOTCHSPAN span NOTCHSPACING notchSpacing  
EXCEPTNOTCHLENGTH notchLength
```

Specifies that any notch with span less than *span* on both sides and notch spacing less than *notchSpacing*, the portion of the notch length edge having common parallel run length greater than 0 on the opposite side must have a spacing greater than or equal to *minSpacing* to any neighbor wire with common parallel run length greater than 0. In addition, multiple consecutive U notches fulfilling the span and notch spacing conditions will be a violation, irrespective of how far the neighbor wires are. The rule does not apply if the minimum notch length is exactly equal to *notchLength* and the outer edges of the notches have neighbor wires that are greater than or equal to *minSpacing* away.

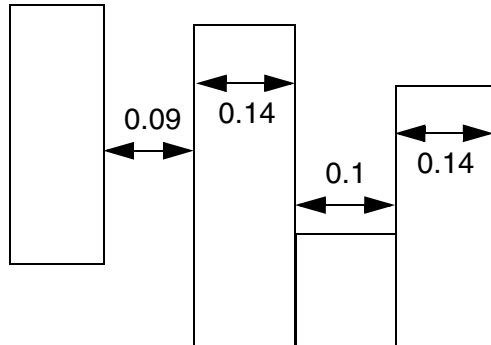
*Type:* Float, specified in microns

### Examples

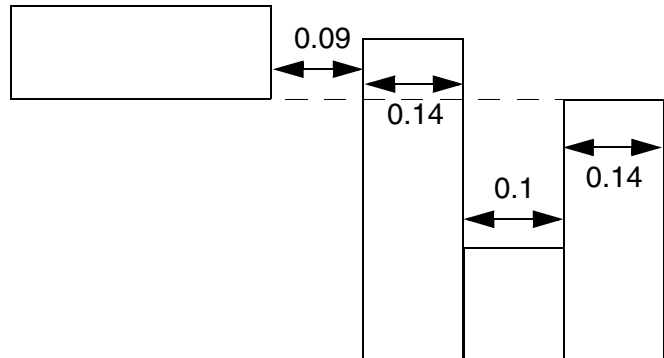
- The following example shows spacing rule with notch span:

```
SPACING 0.08 ;  
PROPERTY LEF58_SPACING  
    "SPACING 0.1 NOTCHSPAN 0.15 NOTCHSPACING 0.12  
    EXCEPTNOTCHLENGTH 0.13 ; " ;
```

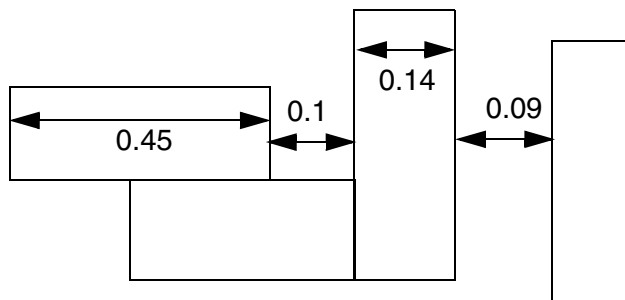
**Figure 1-226 Illustration of Spacing Rule with NOTCHSPAN**



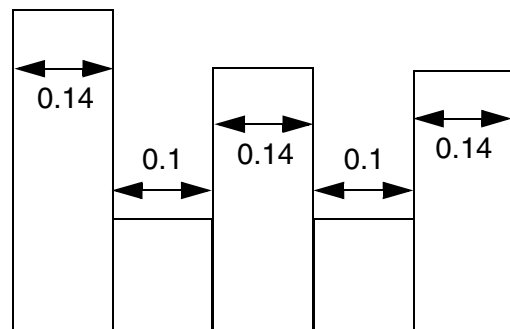
a) Violation, the notch span of 0.14 ( $< 0.15$ ) and notch spacing of 0.1 ( $< 0.12$ ) require spacing  $\geq 0.1$  to neighbor wire



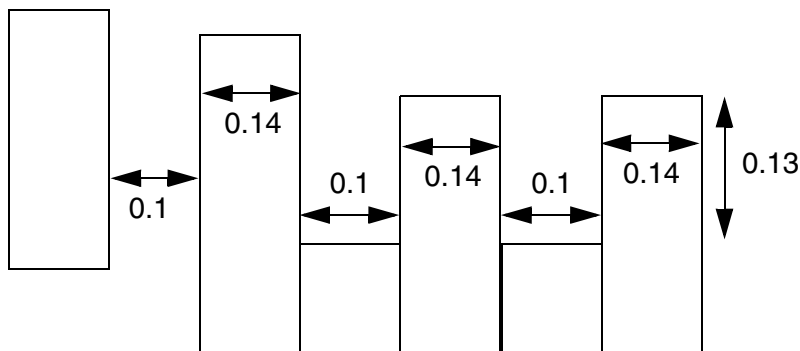
b) OK, the neighbor wire does not common parallel run length to the common portion of the notch length edge



c) OK, the left wire has span of 0.45 ( $\geq 0.15$ ), the rule does not apply



d) Violation, multiple consecutive notches is bad independent of existence of neighbor wires



e) OK, the notch length exception is fulfilled

### ***Minimum Area Rule***

In some cases, it is necessary to allow a smaller minimum area for "simple rectangles" and simple polygon shapes, but require a larger minimum area for complex polygons. This is done with multiple `AREA` statements.

You can define a minimum area rule that requires a larger area

1. When all the edges of the polygon are short
- or
2. If a minimum-sized rectangle cannot fit inside the polygon.

You can also combine the two into one statement, in which case the larger area is required if all the edges are short and a minimum-sized rectangle cannot fit inside the polygon.

You can create a minimum area rule by using the following property definition:

```
PROPERTY LEF58_AREA
    "AREA minArea
    [MASK maskNum]
    [EXCEPTMINWIDTH minWidth]
    | [EXCEPTEDGELENGTH {minEdgeLength maxEdgeLength | edgeLength}]
        [EXCEPTMINSIZE {minWidth minLength}...]
        [EXCEPTSTEP length1 length2]
    | RECTWIDTH rectWidth
    | EXCEPTRECTANGLE
    | LAYER trimLayer OVERLAP {1 | 2}
    ]
    ;..." ;
```

Where:

All the other keywords are the same as the existing LEF routing layer `AREA` syntax.

`EXCEPTMINWIDTH minWidth`

Specifies that the rule does not apply if the width of a wire is greater than or equal to the *minWidth*.

*Type:* Float, specified in microns

`EXCEPTEDGELENGTH minEdgeLength maxEdgeLength | edgeLength`

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

The *edgeLength* specifies that the minimum area rule applies for a given polygon, except if at least one edge-length is greater than or equal to *edgeLength*.

If both *minEdgeLength* and *maxEdgeLength* are specified instead, the minimum area rule does not apply if the entire edge length is less than *minEdgeLength* or at least one edge length is greater than or equal to the *maxEdgeLength*. In other words, the minimum area rule only applies if at least one edge length is greater than or equal to the *minEdgeLength* and the edge length is less than the *maxEdgeLength*.

*Type:* Float, specified in microns

EXCEPTMINSIZE {*minWidth minLength*}...

Indicates the *minArea* rule applies for a given polygon except if a minimum-sized rectangle of dimensions *minWidth minLength* can fit inside the polygon.

Multiple width/length pairs are allowed. The minimum area rule does not apply, if multiple width/length pairs are specified, and if any one of the given min-sized rectangles fit inside a given polygon.

*Type:* Float, specified in microns (for both values)

EXCEPTRECTANGLE

Specifies that the minimum area rule only applies to non-rectangular objects. It is illegal to define `AREA EXCEPTRECTANGLE` by itself, and it should be complemented by a separate `AREA` statement without any exception clauses.

EXCEPTSTEP *length1 length2*

Specifies that the minimum area rule does not apply for a polygon having an edge with length greater than or equal to *length1* with an adjacent edge with length less than *length2*.

*Type:* Float, specified in microns

LAYER *trimLayer* OVERLAP {1 | 2}

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the area rule applies only when:

- One line-end of a wire overlaps or touches with shapes in *trimLayer*, which must be a layer with TYPE TRIMMETAL, if OVERLAP is specified as 1.
- Both line-ends of a wire overlap or touch with shapes in *trimLayer*, if OVERLAP is specified as 2.

MASK *maskNum*

Specifies which mask the area rule is applied on. The *maskNum* must be a positive integer, and most applications only support values of 1, 2, or 3. If any one of the AREA statements uses MASK, all of the AREA statements must use MASK, and each mask must have at least one corresponding AREA statement.

Type: Integer

RECTWIDTH *rectWidth*

Specifies the minimum area of a rectangle with width less than or equal to *rectWidth* to be greater than or equal to *minArea*. For a given rectangle, find the smallest *rectWidth* that is greater than the width of the rectangle. If such an AREA statement can be found, the corresponding area rule should only be checked for the rectangle. If no such statement can be found, the rest of the AREA statements will apply to the rectangle.

Type: Float, specified in microns

### Minimum Area Rule Examples

- If you have the following AREA definition in your routing layer statement:

```
AREA 0.4 ;
```

All polygons on the layer must have a minimum area of 0.4  $\mu\text{m}$ .

- If the following minimum area rule exists:

```
PROPERTY LEF58_AREA "AREA 0.6 EXCEPTEDGELENGTH 0.5 ;" ;
```

All polygons on the layer must have a minimum area of 0.6  $\mu\text{m}$ , except if a polygon has at least one edge that is greater than or equal to 0.5  $\mu\text{m}$ .

- If the following minimum area rule exists:

```
PROPERTY LEF58_AREA  
"AREA 0.6 EXCEPTEDGELENGTH 0.5 EXCEPTMINSIZE 0.1 0.5 ;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Any polygon on the layer must have a minimum area of  $0.6 \mu\text{m}$  when neither of the following conditions hold:

- ❑ The polygon has at least one edge that is greater than or equal to  $0.5 \mu\text{m}$
- ❑ A rectangle of size  $0.1 \mu\text{m}$  by  $0.5 \mu\text{m}$  can fit inside the polygon

■ If the following minimum area rule exists:

```
AREA 0.1 ;  
PROPERTY LEF58 AREA  
    "AREA 0.15 EXCEPTMINWIDTH 0.05 ;" ;
```

All polygons with any width less than  $0.05 \mu\text{m}$  must have a minimum area of  $0.15 \mu\text{m}^2$ . Otherwise, a minimum area of  $0.1 \mu\text{m}^2$  is needed.

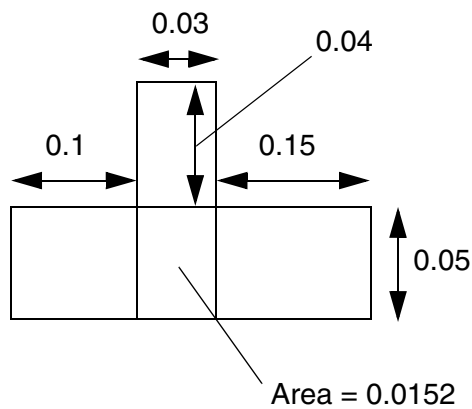
- The following minimum area rule indicates that any rectangles with width less than or equal to  $0.05 \mu\text{m}$  must have an area greater than or equal to  $0.08 \mu\text{m}^2$ , any rectangles with width greater than  $0.05 \mu\text{m}$  and less than or equal to  $0.1 \mu\text{m}$  must have an area greater than or equal to  $0.12 \mu\text{m}^2$  while any other objects must have an area greater than or equal to  $0.15 \mu\text{m}^2$ .

```
AREA 0.15 ;  
PROPERTY LEF58 AREA  
    "AREA 0.08 RECTWIDTH 0.05 ;  
    AREA 0.12 RECTWIDTH 0.1 ; " ;
```

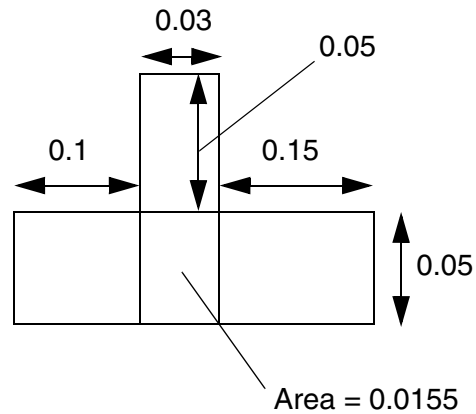


**Figure 1-227 Illustration of Area Rule with EXCEPTSTEP**

```
AREA 0.015 ;
PROPERTY LEF58_AREA
  "AREA 0.017 EXCEPTSTEP 0.12 0.05 ; " ;
```



a) OK, the edge of 0.15 ( $\geq 0.12$ ) has an adjacent edge of 0.04 ( $< 0.05$ ), and only 0.015 area should be followed and is met

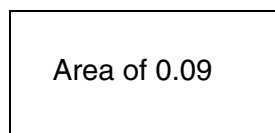


b) Violation, there is no edge  $\geq 0.12$  having an adjacent edge  $< 0.05$ , 0.017 area should be followed and is failed.

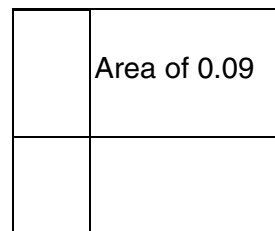
- The following minimum area rule indicates that any rectangles must have an area greater than or equal to  $0.08 \mu\text{m}^2$ , while any other polygons must have an area greater than or equal to  $0.10 \mu\text{m}^2$ :

```
AREA 0.08 ;
PROPERTY LEF58_AREA
  "AREA 0.10 EXCEPTRECTANGLE ;" ;
```

**Figure 1-228 Minimum Area Rule with EXCEPTRECTANGLE**



a) OK, a rectangle only requires minimum area of 0.08 and is met

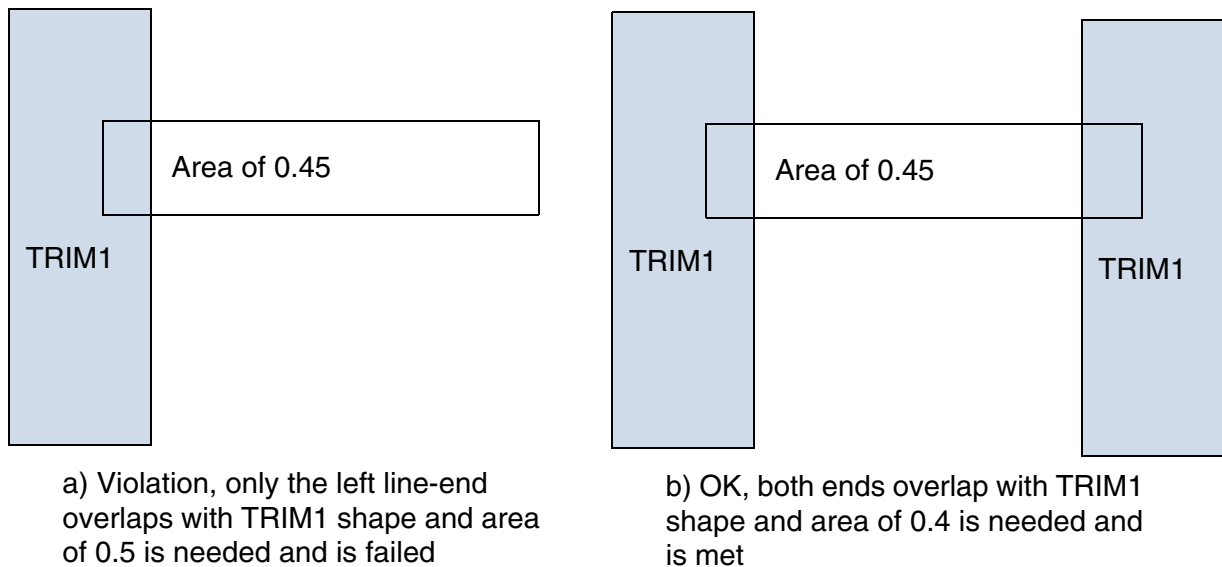


b) Violation, this is not a rectangle, and minimum area must be  $\geq 0.10$

- The following example illustrates minimum area rule for trim layer with overlap:

```
AREA 0.06 ;
PROPERTY LEF58_AREA "
    AREA 0.5 LAYER TRIM1 OVERLAP 1
    AREA 0.4 LAYER TRIM1 OVERLAP 2 ;" ;
```

**Figure 1-229 Minimum Area Rule for Trim Layer with Overlap**



### Corner Spacing Rule

You can create corner spacing rule to define spacing between convex/concave corners and any edges, depending on whether a convex or concave corner is defined.

You can create a corner spacing rule by using the following property definition:

```
PROPERTY LEF58_CORNERSPACING
    "CORNERSPACING
        {CONVEXCORNER [SAMEMASK]
            [CORNERONLY within]
            [EXCEPTEOL eolWidth]
            [EXCEPTJOGLLENGTH length [EDGELENGTH]
                [INCLUDELSHAPE]]]
        |CONCAVECORNER
            [MINLENGTH minLength] [EXCEPTNOTCH [notchLength]] }
    [EXCEPTSAMENET | EXCEPTSAMEMETAL]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
{WIDTH width SPACING spacing
| WIDTH width SPACING horizontalSpacing verticalSpacing}...
; " ;
```

Where:

CORNERONLY *within* Specifies that the corner spacing rule only applies to a neighbor corner, which is within spacing in the direction parallel to the specified direction in DIRECTION on a Manhattan routing layer and *within* in the perpendicular direction.

```
CORNERSPACING {CONVEXCORNER | CONCAVECORNER}
{WIDTH width SPACING spacing}...
```

Specifies the spacing, which is measured in MAXXY style, between a convex or concave corner and any edges depending on whether CONVEXCORNER or CONCAVECORNER keyword is defined. For convex corner cases, parallel run length to the neighbor wire must be less than or equal to 0 to trigger the rule. If the width of a wire containing the corner is greater than *width*, then the corresponding *spacing* is applied.

*Type:* Float, specified in microns

EXCEPTEOL *eolWidth* Specifies that the corner spacing rule does not apply to a corner connected to a EOL edge with width less than the *eolWidth*.

*Type:* Float, specified in microns

```
EXCEPTJOGLENGTH length [EDGELENGTH] [INCLUDELSHAPE]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the minimum default (right-way) spacing will be applied between a wrong way jog with minimum wrong way width and length less than *length* sandwiched by two right way wires with projected PRL less than 0 in wrong way direction in a 'Z' shape and a right way wire.

If INCLUDELSHAPE is specified, this minimum default spacing will also be applied between a wrong way jog with minimum wrong way width and length less than *length* connected to a right way wire in a 'L' shape and a right way wire.

If EDGELENGTH is specified, *length* is measured against the length of the edges in the wrong way direction of a wrong way jog in a 'Z' shape instead of the entire span length of the jog. If INCLUDELSHAPE is also specified to include an 'L' shape exemption, the longest edge length would be the same as the span length of the jog, and this EDGELENGTH construct would have no impact on it.

Type: Float, specified in microns

EXCEPTNOTCH [*notchLength*]

Specifies that the rule does not apply to a notch (U shape) that is a neighbor of the concave corner. *notchLength* specifies that the notch length must be less than the given value to be exempted from the rule. See [Figure 1-235](#) on page 435.

EXCEPTSAMENET | EXCEPTSAMEMETAL

Specifies that the rule does not apply to same-net or same-metal objects of the corner.

MINLENGTH *minLength*

Specifies that the corner spacing rule applies only if both edges of the concave corner have length  $\geq$  *minLength*.

Type: Float, specified in microns

SAMEMASK

Specifies that the corner spacing rule only applies to objects on the same mask.

WIDTH *width* SPACING *horizontalSpacing verticalSpacing*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the rule would require *horizontalSpacing* horizontally and *verticalSpacing* vertically from the concave corner in MAXXY style to avoid a neighbor wire. If EXCEPTNOTCH without a given *notchLength* is also defined, the notch length must be less than the minimum of *horizontalSpacing* and *verticalSpacing* to be exempted from the rule.

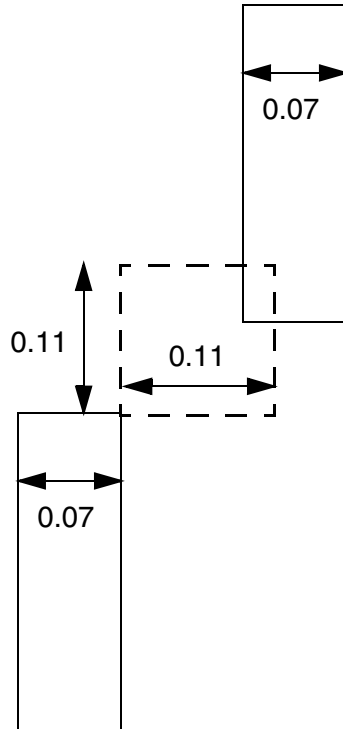
*Type:* Float, specified in microns

### Corner Spacing Rule Examples

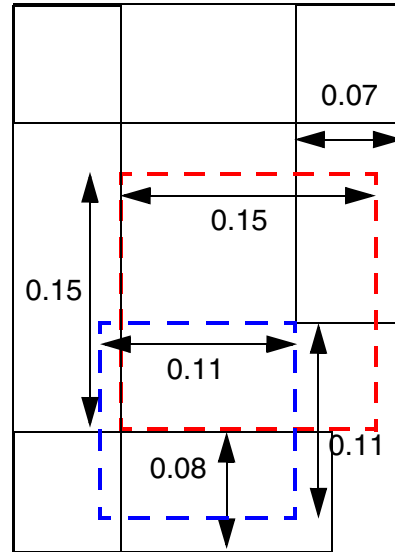
- The following example illustrates the corner spacing rule:

```
PROPERTY LEF58_CORNERSPACING
    "CORNERSPACING CONVEXCORNER EXCEPTEOL 0.090
WIDTH 0.000 SPACING 0.110 ... ;
    CORNERSPACING CONCAVECORNER EXCEPTNOTCH
WIDTH 0.000 SPACING 0.150 ; " ;
```

**Figure 1-230 Illustration of Corner Spacing Rule**

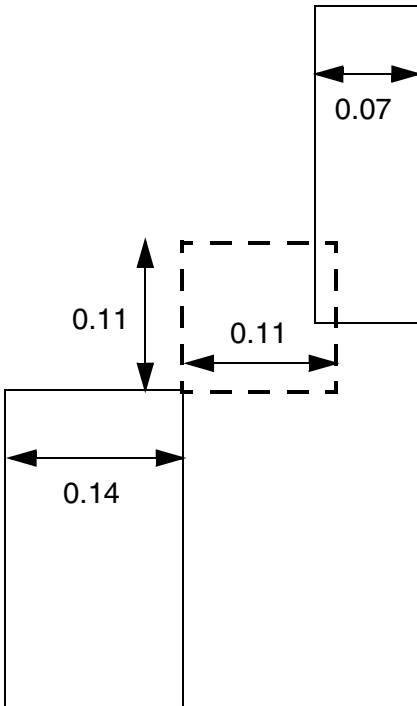


a) OK, end of line is exempted for convexcorner rule

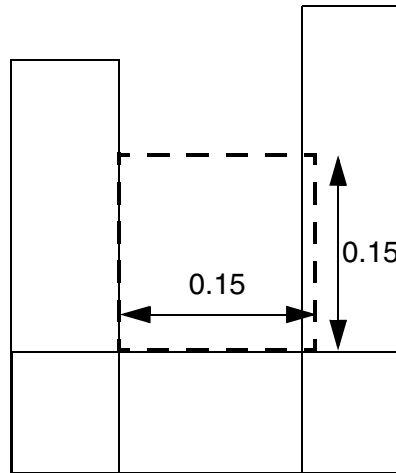


b) Violation, the 0.11 blue dotted line of convexcorner rule is exempted due to  $PRL > 0$ , but the 0.15 red dotted line of concavecorner rule still applies and fails

**Figure 1-231 Corner Spacing Rule**



c) Violation, one EOL edge would not exempt the rule.

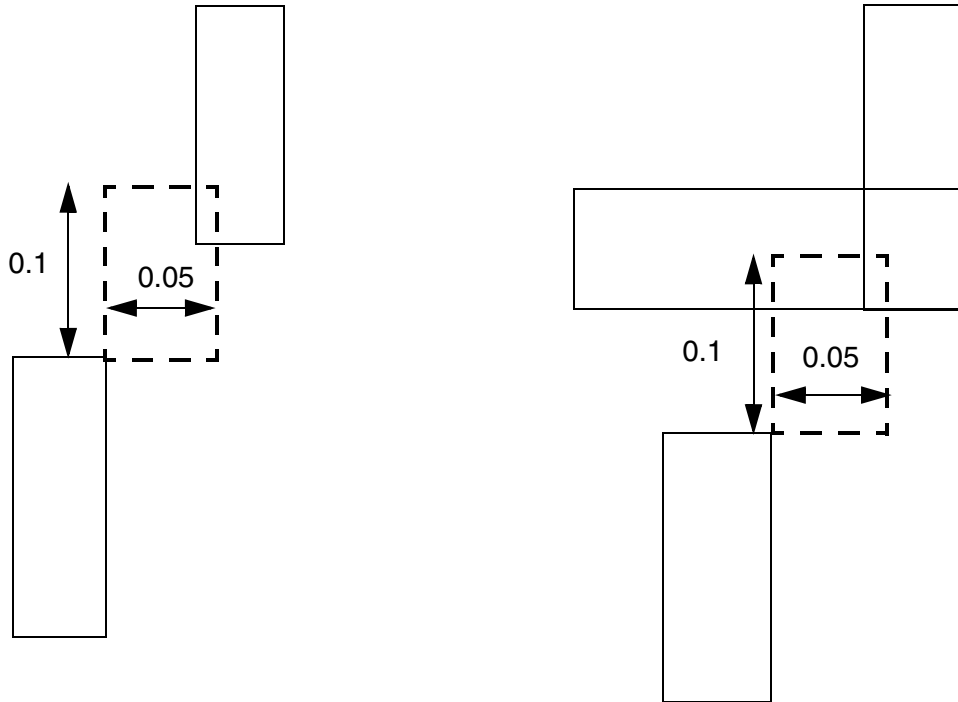


d) OK, notch is exempted for concave corner

- The following example illustrates corner spacing rule with CORNERONLY:

```
DIRECTION VERTICAL ;
PROPERTY LEF58_CORNERSPACING "
    CORNERSPACING CONVEXCORNER CORNERONLY 0.050
    WIDTH 0.000 SPACING 0.100 ; " ;
```

**Figure 1-232 Corner Spacing Rule with CORNERONLY**



a) Violation, find a neighbor corner within the search window.

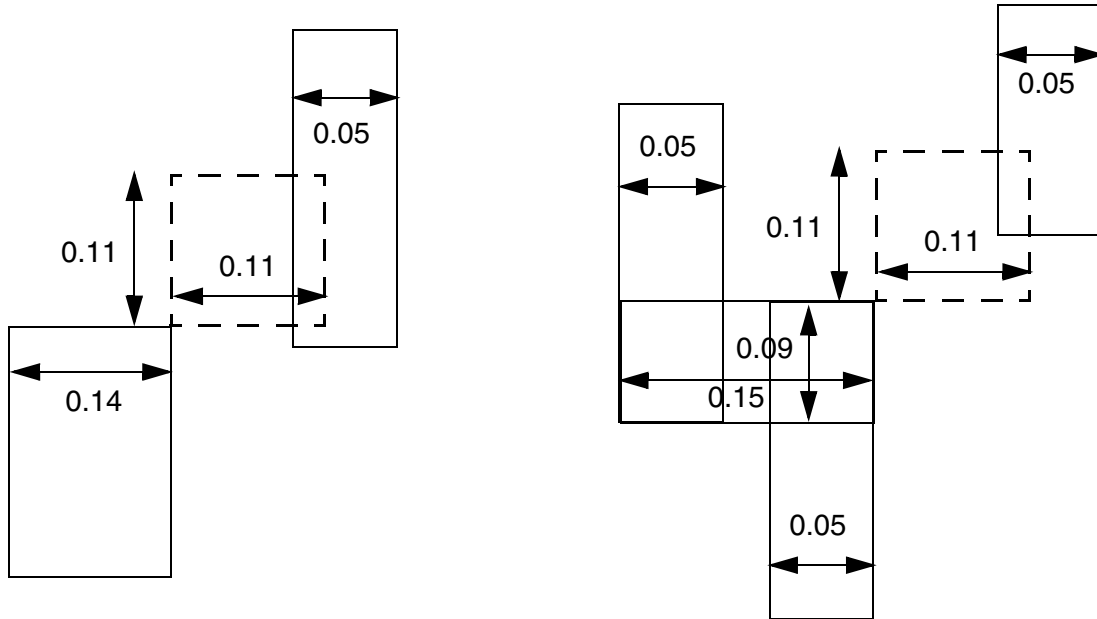
b) OK, no corner found.

■ The following example illustrates the corner spacing rule:

```
DIRECTION VERTICAL ;
PROPERTY LEF58_WIDHTABLE "
    WIDHTABLE 0.050 ... ;
    WIDHTABLE 0.090 ... WRONGDIRECTION ; " ;
PROPERTY LEF58_CORNERSPACING "
    CORNERSPACING CONVEXCORNER EXCEPTEOL 0.090
    EXCEPTJOGLENGTH 0.170 INCLUDELSHAPE
    WIDTH 0.000 SPACING 0.110 ... ;
    CORNERSPACING CONCAVECORNER MINLENGTH 0.05
    WIDTH 0.000 SPACING 0.150 ; " ;
```

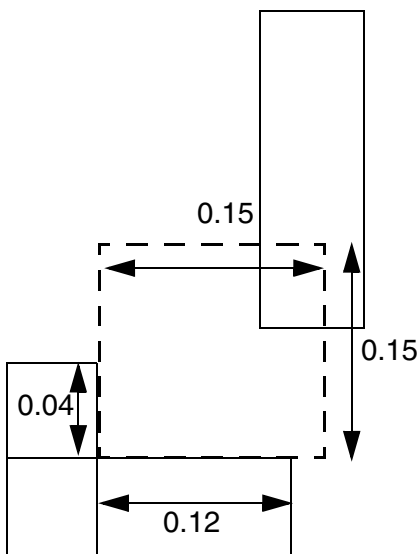


Figure 1-233 Illustration of Corner Spacing Rule



a) OK, the neighbor having PRL > 0 would not trigger the rule

b) OK, the rule is exempted between this 'Z' jog and a EOL edge.



c) OK, the vertical edge of the concave corner has length of 0.04 (< 0.05)

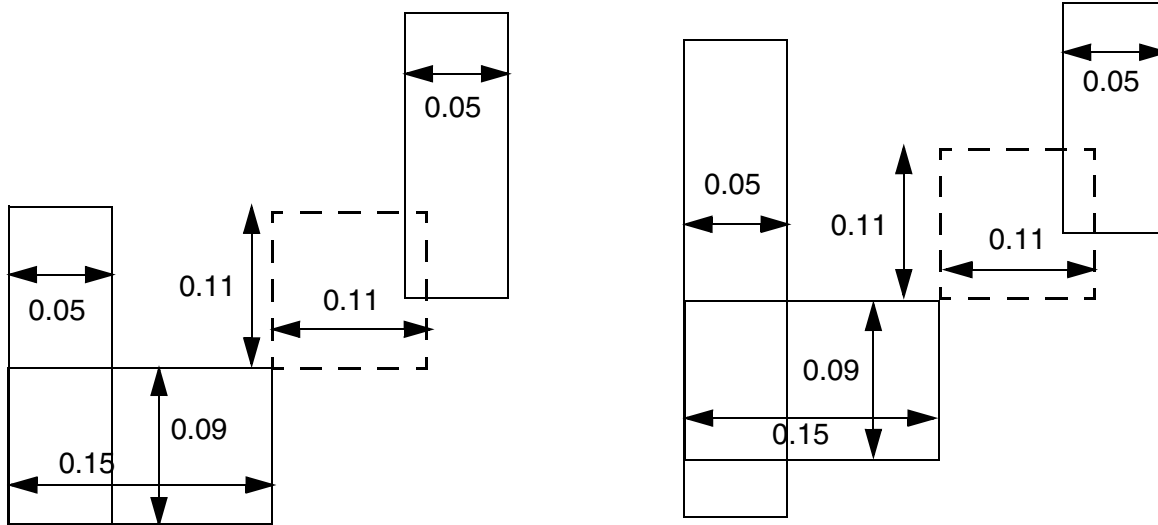
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- The following example illustrates the corner spacing rule:

```
DIRECTION VERTICAL ;  
PROPERTY LEF58_WIDHTABLE "  
    WIDHTABLE 0.050 ... ;  
    WIDHTABLE 0.090 ... WRONGDIRECTION ; "  
PROPERTY LEF58_CORNERSPACING "  
    CORNERSPACING CONVEXCORNER EXCEPTEOL 0.090  
    EXCEPTJOGLLENGTH 0.170  
    WIDTH 0.000 SPACING 0.110 ... ;
```

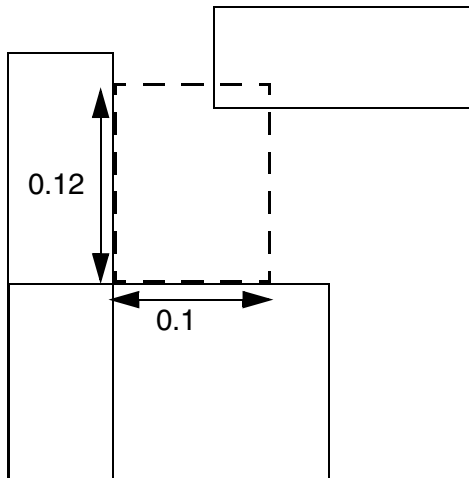
**Figure 1-234 Illustration of Corner Spacing Rule**



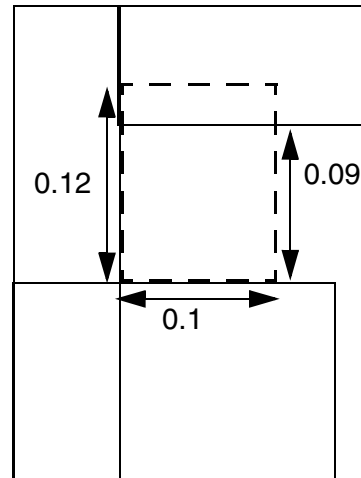
- The following example illustrates the corner spacing rule:

```
PROPERTY LEF58_CORNERSPACING "  
    CORNERSPACING CONCAVECORNER EXCEPTNOTCH 0.08  
    WIDTH 0.000 SPACING 0.100 0.120 ; "  
;
```

**Figure 1-235 Illustration of Corner Spacing Rule with EXCEPTNOTCH in CONCAVECORNER**



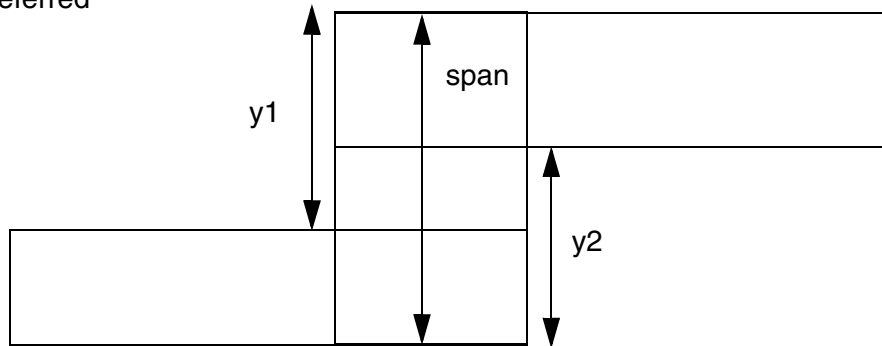
a) Violation, no neighbor could be inside the dotted line search window



b) Violation, notch length of 0.09 ( $\geq 0.08$ ). OK if notch length of 0.08 is omitted because 0.09 is less than minimum of (0.1, 0.12).

**Figure 1-236 Illustration of CORNERSPACING with EDGELENGTH in EXCEPTJOGLNGTH**

Horizontal is the preferred routing direction.



Both  $y1$  and  $y2$  must be  $< length$  to be qualified as a 'Z' shape for EXCEPTJOGLNGTH exemption. If EDGELENGTH is not specified, span must be  $< length$  to be qualified.

Illustration of EXCEPTJOGLNGTH  $length$  EDGELENGTH

### **Pitch Rule**

You can create a pitch rule to define non-uniform pitch or track on layers, whose preferred direction must be the same as the direction of the standard cell rows.

You can create a pitch rule by using the following property definition:

```
PROPERTY LEF58_PITCH
    "PITCH {distance | xDistance yDistance } [FIRSTLASTPITCH firstLastPitch]
    ; " ;
```

All other keywords are the same as the existing LEF routing layer PITCH syntax.

Where:

FIRSTLASTPITCH *firstLastPitch*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

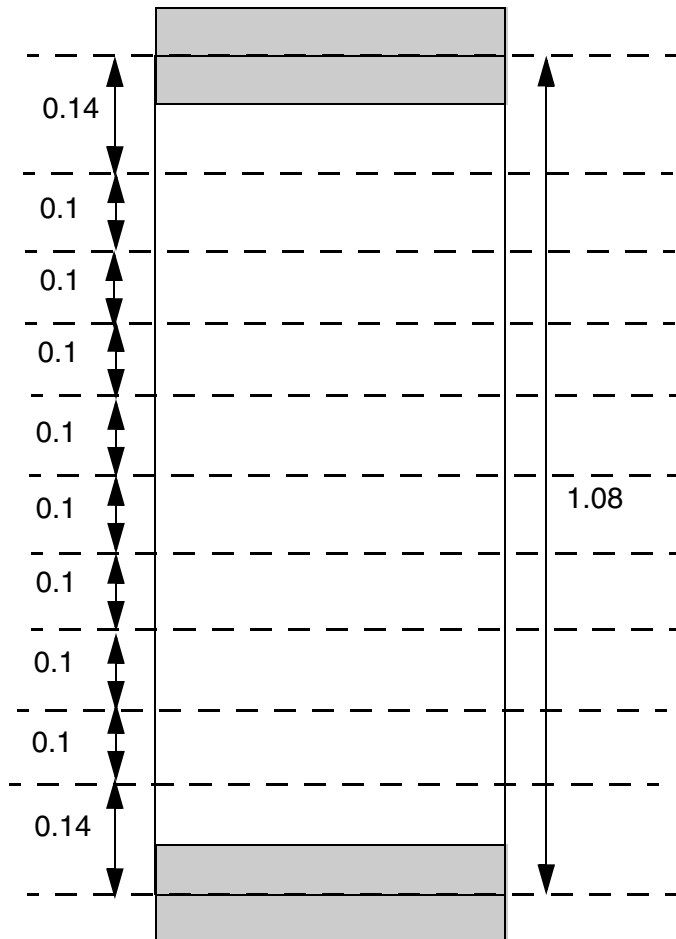
Specifies a non-uniform pitch/track on this layer, whose preferred direction must be in the same direction as that of the standard cell rows (normally horizontal). The routing tracks are repeated for every standard-cell row (see example below). The *firstLastPitch* defines the distance of the first and last tracks from the standard-cell row boundary and pitch value on this layer defines the distance of the intermediate tracks. In other words, subtracting the cell row height by 2 \* *firstLastPitch* should be divisible by pitch value on this layer. The pitch value of the property should be identical with the pitch value in the native/regular `PITCH` statement. If not, the pitch value of the property will be honored.  
*Type:* Floats, specified in microns

### Pitch Rule Examples

- The following example shows indicates routing tracks:

```
PROPERTY LEF58_PITCH STRING "  
    PITCH 0.1 FIRSTLASTPITCH 0.14 ; ";
```

**Figure 1-237 Example of Pitch Rule**



In this example, the cell height is 1.08. The dotted line represents the routing tracks. The first and last tracks are 0.14 from the cell row boundary while the intermediate tracks are 0.1 apart making  $2 * 0.14 + 8 * 0.1 = 1.08$ . The routing track definition would be repeated based on cell rows.

### ***Protrusion Width Rule***

You can use the protrusion width rule for a protrusion wire, connected to a wire, that has width greater than or equal to the specified width, or whose length is greater than or equal to the specified length.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

You can define a protrusion width rule by using the following property definition:

```
PROPERTY LEF58_PROTRUSIONWIDTH
    "PROTRUSIONWIDTH width1
        {LENGTH length WIDTH width2
        | {WIDTH width2
            {MINSIZE {minWidth minLength | minLength
                CUTCLASS className {FROMABOVE | FROMBELOW}}
            | MINLENGTH minLength
            [EXCEPTCUT cutDistance
                [FROMABOVE | FROMBELOW]]}...}
        ; " ;
```

Where:

All other keywords are same as the existing LEF routing layer PROTRUSIONWIDTH syntax.

EXCEPTCUT *cutDistance* [FROMABOVE | FROMBELOW]

Specifies that if there is a cut of any cut classes in above, below, or either cut layer and if FROMABOVE, FROMBELOW or none is specified less than or equal to the *cutDistance* from wire with width greater than or equal to the *width2*, then the rule does not apply.

Type: Float, specified in microns

MINSIZE *minWidth minLength*

Specifies that it is a violation if the area of protruded wire with width greater than or equal to *width1* outside of the wide wire of width greater than or equal to *width2* does not enclose a size of *minWidth minLength*. The rule does not apply if there is no such area. See [Figure 1-240](#) on page 442.

Type: Float, specified in microns

MINSIZE *minLength* CUTCLASS *className* {FROMABOVE | FROMBELOW}

Specifies that it is a violation if the area outside the wide wire of width greater than or equal to *width2* does not enclose a size of *width1 minLength*, and the protruded area contains one and only one cut with cut class of *className*, either on the above or below cut layer determined by ABOVE/BELOW. See [Figure 1-241](#) on page 443.

Type: Float, specified in microns

{WIDTH *width2* MINLENGTH *minLength*}...

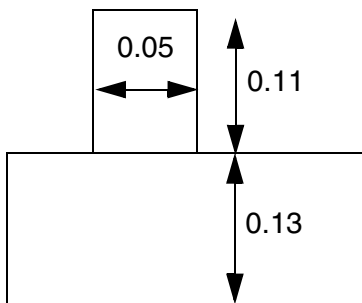
Specifies that when a protrusion wire is connected to a wire with width greater than or equal to *width2*, then it must have length greater than or equal to the *minLength* and width greater than or equal to *width1*.  
*Type*: Float, specified in microns

### Protrusion Width Rule Examples

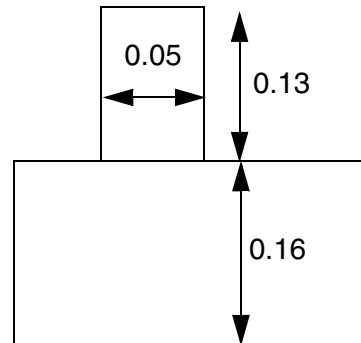
- The following protrusion width rule indicates that the width of the protrusion wire should be greater than or equal to the specified width of 0.05, and its length should be greater than or equal to the specified minimum lengths of 0.12 and 0.14 when it is connected to wide wire with width of 0.11 and 0.15.

```
PROPERTY LEF58_PROTRUSIONWIDTH  
  "PROTRUSIONWIDTH 0.05  
    WIDTH 0.11 MINLENGTH 0.12  
    WIDTH 0.15 MINLENGTH 0.14 ; " ;
```

**Figure 1-238 Illustration of Protrusionwidth Rule**



a) Violation, the width of the protrusion wire is fine, but it is too short.

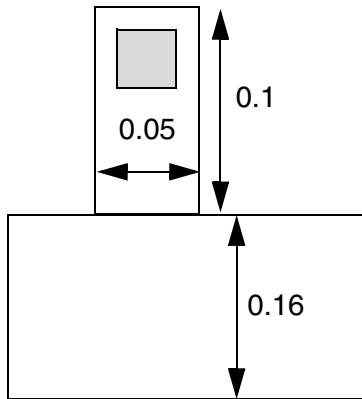


b) Violation, the length requirement varies by the width of the wide wire



**Figure 1-239 Illustration of Protrusionwidth Rule with EXCEPTCUT**

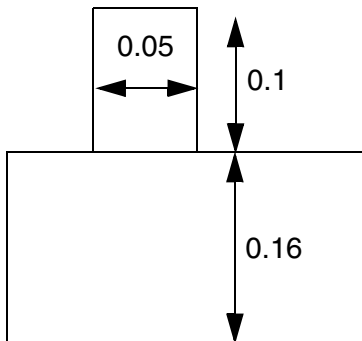
```
PROPERTY LEF58_PROTRUSIONWIDTH "  
    PROTRUSIONWIDTH 0.05  
    WIDTH 0.11 MINLENGTH 0.12 EXCEPTCUT 0.1 ; " ;
```



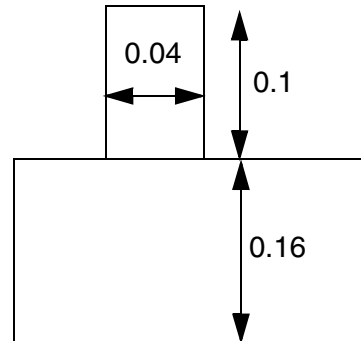
a) OK, cut exemption is kicked in.

**Figure 1-240 Illustration of Protrusionwidth Rule with MINSIZE**

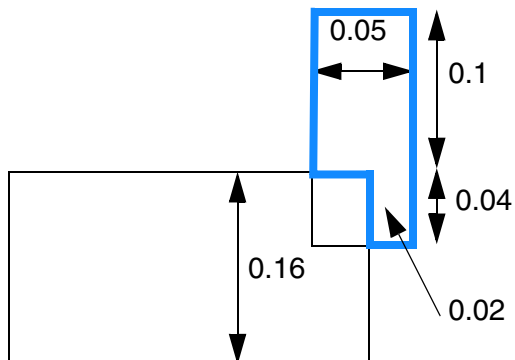
```
PROPERTY LEF58_PROTRUSIONWIDTH  
  "PROTRUSIONWIDTH 0.05  
    WIDTH 0.11 MINSIZE 0.02 0.12 ; " ;
```



a) Violation, the protruded length is < 0.12



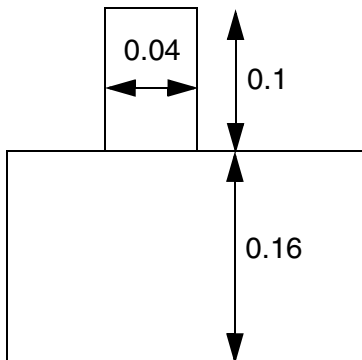
b) OK, there is no protruded area with width  $\geq 0.05$ .



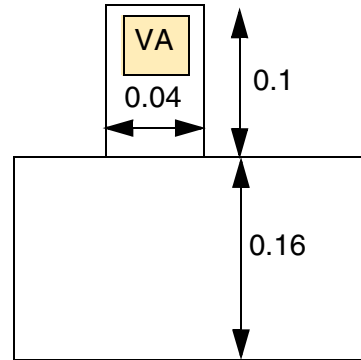
c) OK, the blue area is the protruded area with width  $\geq 0.05$ , and it fits size of 0.02 0.12

**Figure 1-241 Illustration of Protrusionwidth Rule with WIDTH and MINSIZE**

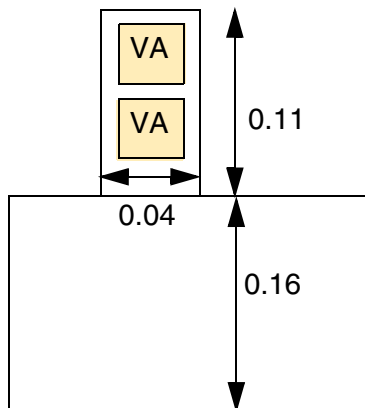
```
PROPERTY LEF58_PROTRUSIONWIDTH
  "PROTRUSIONWIDTH 0.05
    WIDTH 0.11 MINSIZE 0.12
    CUTCLASS VA FROMABOVE ; " ;
```



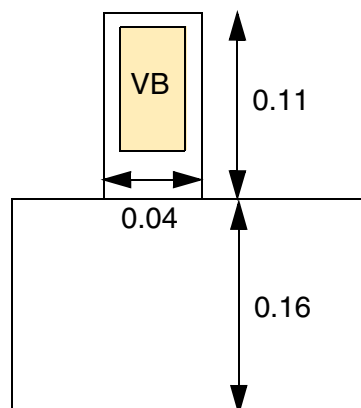
a) OK, the rule is only triggered with one and only one cut of VA.



b) Violation, only 1 cut of VA and the protruded area does not fit size of 0.05 0.12.



c) OK, there are more than 1 cut of VA in the protruded area.

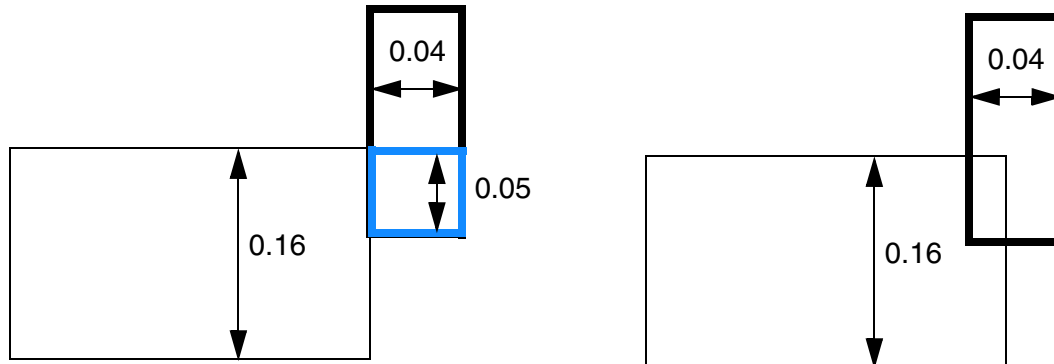


d) OK, there is no VA cut.

- The following protrusion width rule indicates that the rule will fail when the given value of LENGTH is 0, and if the protruded wire touches the wide wire, and the aligned portion has a width less than the specified first width:

```
PROPERTY LEF58_PROTRUSIONWIDTH
  "PROTRUSIONWIDTH 0.05 LENGTH 0 WIDTH 0.11 ; " ;
```

**Figure 1-242 Illustration of Protrusionwidth Rule with LENGTH and WIDTH**



a) OK, the blue touching portion has width  $\geq 0.05$ .

b) Violation, this is not a touching case.

### ***Enclosure Spacing Rule***

You can use the enclosure spacing rule to specify the spacing on an edge with enclosure less than the specified enclosure.

You can define enclosure spacing rule by using the following property definition:

```
PROPERTY LEF58_ENCLOSURESPACING
    "ENCLOSURESPACING
        [CUTCLASS cutClass [LONGEDGEONLY] ]
        [FROMABOVE | FROMBELOW]
        {ENCLOSURE enclosure SPACING spacing}...
    ; ... " ;
```

Where:

`CUTCLASS cutClass`

Specifies the spacing applies only to an edge with a cut of the given *cutClass*. If CUTCLASS is defined in cut layer, then CUTCLASS must be specified in the ENCLOSURESPACING statement.

You can specify individual rules with the CUTCLASS keyword for each cut class, if required.

```
ENCLOSURESPACING {ENCLOSURE enclosure SPACING spacing}
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the spacing on an edge with enclosure less than *enclosure* to be *spacing*. You can determine the portion of a cut having enclosure less than *enclosure*, extending the cut corner by *enclosure* in Euclidean style to intersect with the metal edge. The extended edge is checked against spacing.

*Type:* Float, specified in microns

FROMABOVE | FROMBELOW

Specifies the spacing applies only to a via cut from above or below the specified layer.

LONGEDGEONLY

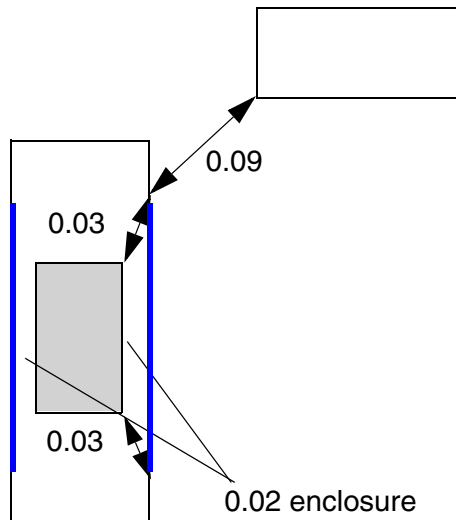
Specifies the spacing applies only to a metal edge containing a side/long edge of a rectangular cut with enclosure less than the *enclosure*.

### Enclosure Spacing Rule Examples

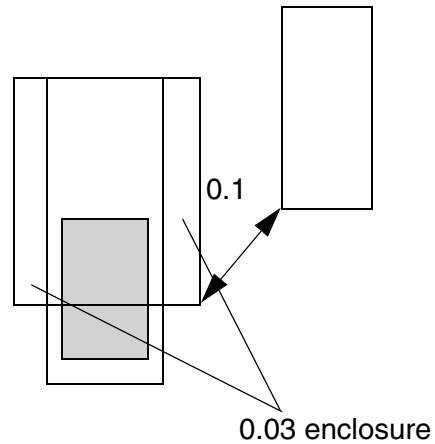
The following rule indicates enclosure spacing:

```
PROPERTY LEF58_ENCLOSURESPACING "  
ENCLOSURESPACING CUTCLASS VB LONGEDGEONLY  
ENCLOSURE 0.03 SPACING 0.1 ; " ;
```

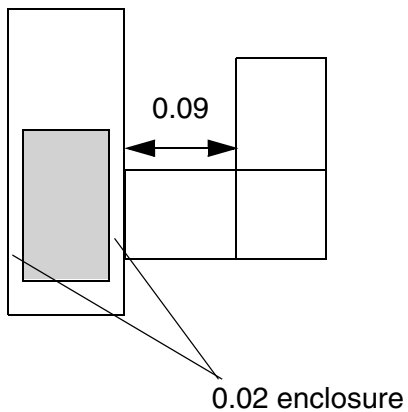
**Figure 1-243 Illustration of Enclosure Spacing Rule**



a) Violation, the cut corners need to extend by 0.03 to form the blue edges, where spacing of 0.1 is applied



b) OK, the metal spacing only applies to the portion of cut with enclosure < 0.03



c) Violation, the spacing applies to same-metal jog

#### ***Minimum Cut Rule***

Minimum cut rules exist for thin wires connected to a wide wire or pin.

You can define a minimum cut rule by using the following property definition:

```
PROPERTY LEF58_MINIMUMCUT
    "MINIMUMCUT {numCuts | {CUTCLASS className numCuts }... }
    WIDTH width [WITHIN cutDistance]
    [FROMABOVE | FROMBELOW]
    [LENGTH length WITHIN distance
    | AREA area [WITHIN distance]
    | SAMEMETALOVERLAP
    | FULLYENCLOSED]
    ; " ;
```

All other keywords are the same as the existing LEF routing layer `MINIMUMCUT` syntax.

Where:

```
AREA area [WITHIN distance]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Applies the minimum cut rule when the wide object has a width that is greater than *width*, and an area that is greater than *area*.

The area of a polygon is determined by a process in which the polygon is shrunk by an amount equal to  $width/2$ , then grown by an amount equal to  $width/2$ . The resulting polygon at the connection location is used for area comparison. If the connection is made at a thin wire (width less than and equal to *width*) that connects to the wide object, the biggest remaining neighbor should be checked individually with their own areas. If it is followed by a `WITHIN distance` syntax, the within distance is measured from the edges of the remaining neighbors.

If *width* is less than width of the default routing wire is used along with the `AREA` keyword, the minimum cut requirement on the routing vias can vary depending on the area of the routing wire on the layer. This should be used cautiously. A small area can result in longer routing run times, and more DRC violations.

`WITHIN` indicates that the rule applies for thin wires directly connected to a wide object, if the cuts on the thin wire are less than *distance* from the wide object. If `AREA` and `WITHIN` are defined, this rule only checks the thin wire connected to a wide wire; it does not check the wide wire itself. A separate `MINIMUMCUT numCuts WIDTH width` statement without `AREA` and `WITHIN` is required for any wide wire minimum cut rule.

*Type:* Float, specified in microns

**Note:** You can specify either `AREA WITHIN` or `LENGTH WITHIN` in a routing layer.

If `WITHIN cutDistance` is absent, only cuts belonging to the same via are considered as multiple cuts.

`CUTCLASS className numCuts`

Defines the minimum cut rule for a specific cut class *className* in the cut layers either above or below the current routing layer to be *numCuts*. Multiple `CUTCLASS` keywords can be defined to specify different *numCuts* for each of them. If `CUTCLASS` is defined in cut layer, then `CUTCLASS` must be specified in the `MINIMUMCUT` statement.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

FULLYENCLOSED	Specifies that all the cuts must be completely inside a wide wire with width greater than <i>minWidth</i> to be counted against the minimum cut requirement of <i>numCuts</i> .
SAMEMETALOVERLAP	Specifies that if the common metal on both the top and bottom layers containing the cut overlaps with the wide wire having a width greater than <i>minWidth</i> , then the rule still applies even if the cut itself does not overlap with the wide wire. See <a href="#">Figure 1-246</a> on page 452.

### Minimum Cut Rule Examples

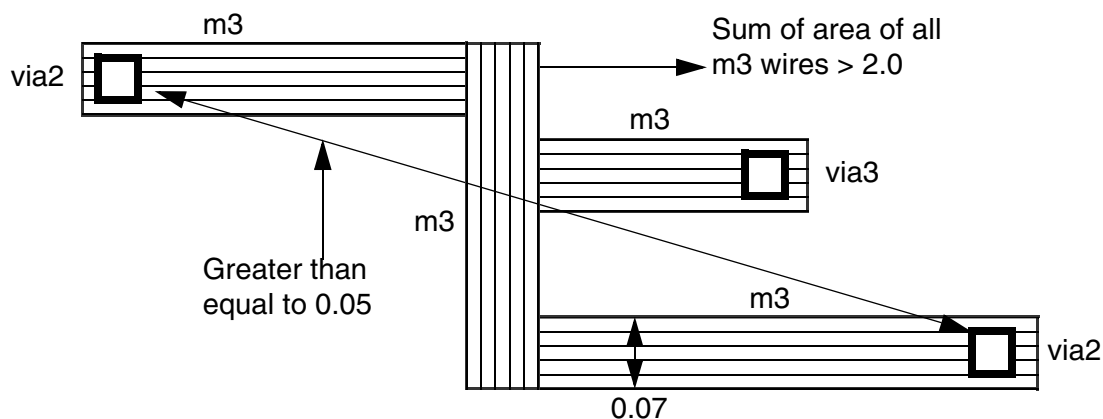
- The following minimum cut rule indicates that if an object has width greater than  $0.1\text{ }\mu\text{m}$ , either a VA via with minimum three cuts or a VB via with minimum two cuts are needed on that object:

```
PROPERTY LEF58_MINIMUMCUT
    "MINIMUMCUT CUTCLASS VA 3 CUTCLASS VB 2 WIDTH 0.1 ; " ;
```

- The following minimum cut rule indicates that vias with 2 cuts (placed at a distance less than  $0.05\text{ }\mu\text{m}$  apart) are required, if the wire has a width greater than  $0.09\text{ }\mu\text{m}$ , an area greater than  $2.0\text{ }\mu\text{m}^2$ , and *width* is less than the default routing width of  $0.10\text{ }\mu\text{m}$ .

```
PROPERTY LEF58_MINIMUMCUT
    "MINIMUMCUT 2 WIDTH 0.09 WITHIN 0.05 AREA 2.0 ;" ;
```

**Figure 1-244 Minimum Cut Rule Example**



Violation; all three vias should have two cuts. If *WITHIN*  $0.05$  is not specified, it is still a violation as two cuts are needed for all end points by definition.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

- The following minimum cut rules indicate that 3 via cuts are required, if a thin wire is connected to a wide wire with a width greater than  $0.24\text{ }\mu\text{m}$  and an area greater than  $1.6\text{ }\mu\text{m}^2$ , and the distance between the vias and the wide wire is less than  $3.0\text{ }\mu\text{m}$ .

```
PROPERTY LEF58_MINIMUMCUT
```

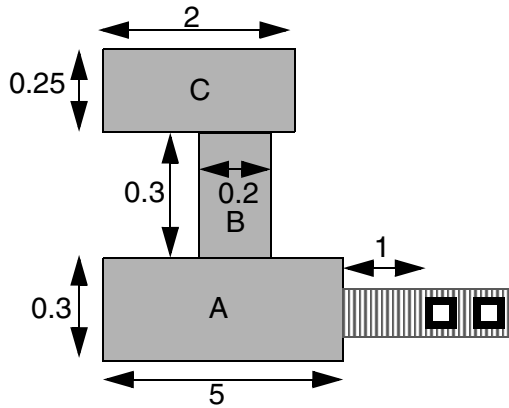
```
"MINIMUMCUT 2 WIDTH 0.24 ;
```

```
"MINIMUMCUT 3 WIDTH 0.40 ;
```

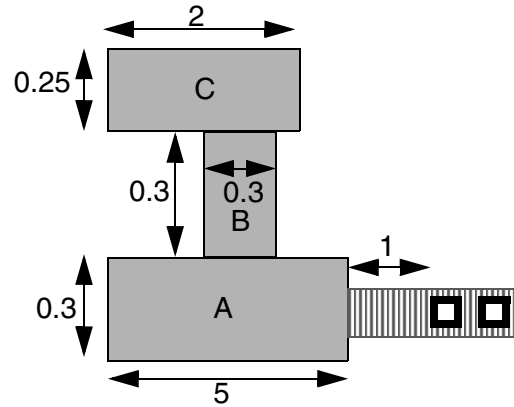
```
"MINIMUMCUT 2 WIDTH 0.24 AREA 0.3 WITHIN 10.0 ;
```

```
"MINIMUMCUT 3 WIDTH 0.24 AREA 1.6 WITHIN 3.0 ;" ;
```

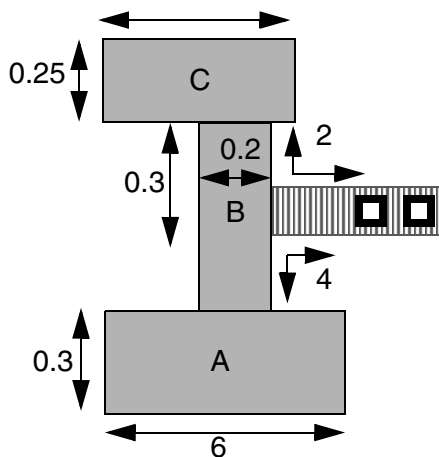
Figure 1-245 Minimum Cut Rule Example



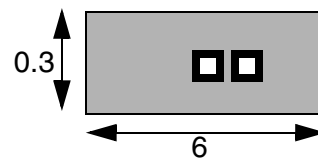
a) Okay; width of B < 0.24, only area of A, 1.5, would require two cuts.



b) Violation; sum of all three areas (A, B, and C) is 2.09, which would require three cuts.



c) Okay; the area of C only needs 2 cuts, and the area of A needs 3 cuts, but the cut is farther than the `WITHIN` distance of 3, and it also fulfills the 2 cuts requirement. Same situation occurs if a 2 cut via is directly connected to B instead of a wire.



d) Okay; `MINIMUMCUT 2`  
`WIDTH 0.24` rule applies. The `WITHIN` rules do not apply when direct via is dropped.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

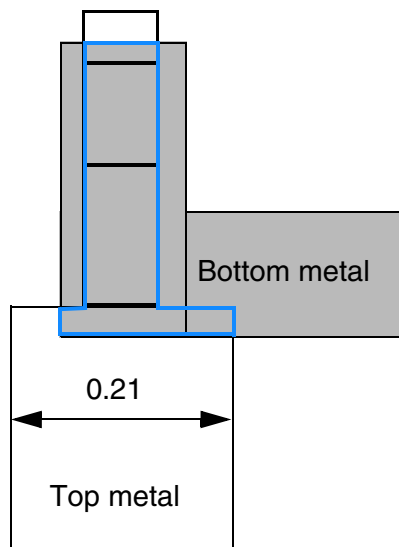
- The following minimum cut rule indicates that if an object has width greater than 0.1  $\mu\text{m}$ , either a VA via with minimum three cuts or a VB via with minimum two cuts are required on that object.

```
PROPERTY LEF58_MINIMUMCUT
    "MINIMUMCUT CUTCLASS VA 3 CUTCLASS VB 2 WIDTH 0.1 ; " ;
```

- The following minimum cut rule indicates that if a common metal, between top and bottom layers, overlaps with a wide wire having a width of 0.2  $\mu\text{m}$ , the rule still applies even if the cut itself does not overlap with the wide wire:

```
PROPERTY LEF58_MINIMUMCUT
    "MINMUMCUT 2 WIDTH 0.2 SAMEMETALOVERLAP ; " ;
```

**Figure 1-246 Illustration of Minimum Cut Rule with SAMEMETALOVERLAP**

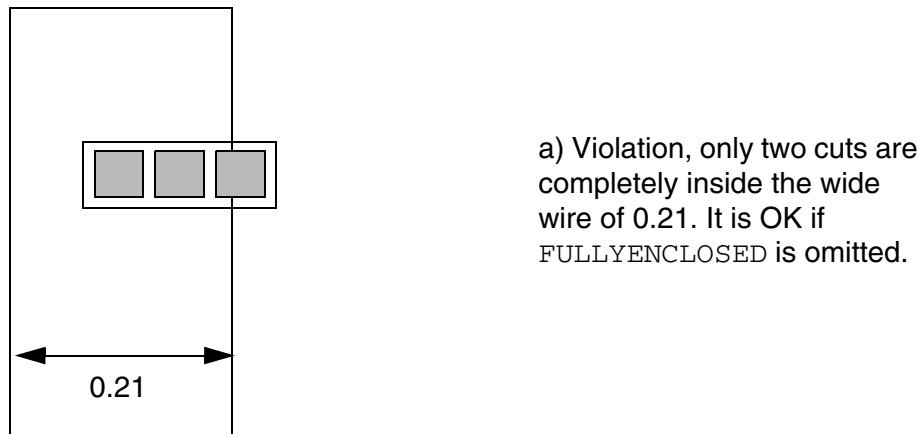


- a) Violation, the common metal on both top and bottom metal outlined in blue overlaps with the wide wire although the cut itself does not overlap with it.

- The following example is an illustration of the minimum cut rule with the **FULLYENCLOSED** keyword:

```
PROPERTY LEF58_MINMUMCUT "  
MINMUMCUT 3 WIDTH 0.2 FULLYENCLOSED ; "
```

**Figure 1-247 Illustration of Minimum Cut Rule with FULLYENCLOSED**



### ***Width Table Rules***

You can use width table rules to define all the allowable legal widths on the routing layer.

You can define a width table rule by using the following property definition:

```
PROPERTY LEF58_WIDHTHABLE  
"WIDHTHABLE {width}... [WRONGDIRECTION] [ORTHOGONAL]  
;" ;
```

Where:

ORTHOGONAL

Specifies that one of the right or wrong direction width between two inside corners of a rectilinear object must be greater than or equal to the first width value in the **WIDHTHABLE** in the corresponding direction, if **WIDHTHABLE WRONGDIRECTION** is specified. Otherwise, the width between the corners must be greater than or equal to the first width value in the **WIDHTHABLE** (for both directions) either vertically or horizontally.

*Type:* Float, specified in microns

**WIDHTHABLE** {width}...

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines all the allowable legal widths on the routing layer. All the given widths are exact width values, except for the last one, which is equal to or greater than the value.

The `WIDTH` syntax should be used to define the default routing width on the layer, which should match one of the values in the `WIDTHTABLE` statement. In case that the last value of `WIDTHTABLE` denotes the exact width also, the `MAXWIDTH` statement with the last value can be used to represent it.

A polygon will be fractured into rectangles, and only the shorter dimension of the rectangles will be checked against the allowable legal widths.

At the most, two `WIDTHTABLE` spacing tables can be defined, one with `WRONGDIRECTION` and the other without it.

*Type:* Float, specified in microns.

#### `WRONGDIRECTION`

Specifies that the allowable legal widths are for wires with direction perpendicular to the specified direction in `DIRECTION` on a Manhattan routing layer.

Note that using `WRONGDIRECTION` changes the interpretation of any wrong-way routing widths in the `DEF NETS` section. If `WRONGDIRECTION` is specified, then any wrong-way routing in the `DEF NETS` section will use the `WRONGDIRECTION` width for that layer unless the net or route has a `NONDEFAULTRULE` with a `WIDTH` greater than the `WRONGDIRECTION` width. But, the implicit default route-extension is still half of the preferred direction width.

Some older tools may not understand this behavior. Normally, they will still read/write and round-trip the `DEF` routing properly, but will not understand that the width is slightly larger for wrong-way routes. If these tools check wrong-way width, then the DRC rules may flag false violations. RC extraction with the wrong width will also flag errors, although wrong-way routes are generally short and the width difference is small, so the RC error is normally negligible.

- The following width table rule indicates that width must be 0.10  $\mu\text{m}$ , 0.15  $\mu\text{m}$ , 0.20  $\mu\text{m}$ , 0.25  $\mu\text{m}$ , 0.30  $\mu\text{m}$ , and greater than or equal to 0.40  $\mu\text{m}$ .

```
MAXWIDTH 0.40 ;          # To define legal width =, instead of >=, 0.40  $\mu\text{m}$ .  
PROPERTY LEF58_WIDTHTABLE
```

## LEF/DEF 5.8 Language Reference

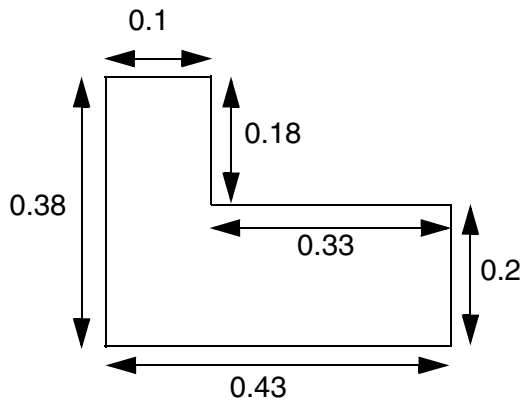
### LEF Syntax

---

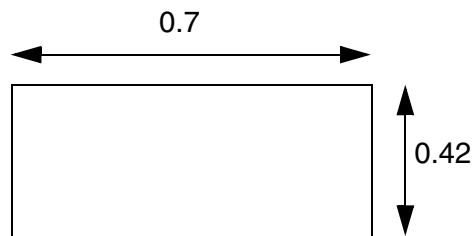
```
"WIDTHTABLE 0.10 0.15 0.20 0.25 0.30 0.40 ;" ;
```

**Figure 1-248 Illustration of WIDTHTABLE Rule**

```
PROPERTY LEF58_WIDTHTABLE "WIDTHTABLE 0.10 0.15 0.20 0.25 0.30 0.40 ;" ;
```



a) OK, only width of the fractured rectangles, 0.1 and 0.2, should be checked



b) OK, width  $\geq 0.40$ . Violation if MAXWIDTH 0.40 is also defined since it fails the max width check.

- The following width table rule indicates that width must be 0.05  $\mu\text{m}$ , 0.1  $\mu\text{m}$ , or greater than or equal to 0.15  $\mu\text{m}$  for horizontal wires.

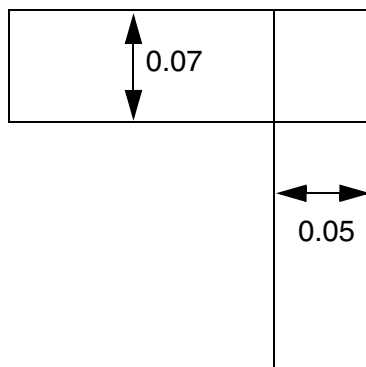
```
DIRECTION VERTICAL ;
```

```
PROPERTY LEF58_WIDTHTABLE
```

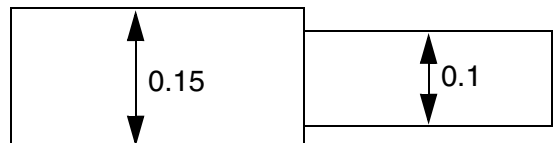
```
"WIDTHTABLE 0.05 0.1 0.15 WRONGDIRECTION ;" ;
```

**Figure 1-249 Illustration of Width Table Rule with ORTHOGONAL and WRONGDIRECTION**

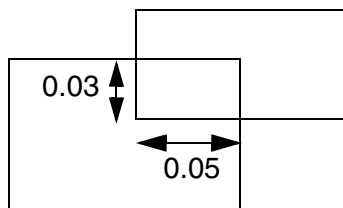
```
DIRECTION VERTICAL ;  
PROPERTY LEF58_WIDHTHABLE  
"WIDHTHABLE 0.05 0.07 0.1 0.12 0.15 ORTHOGONAL ;  
WIDHTHABLE 0.1 0.15 WRONGDIRECTION ; " ;
```



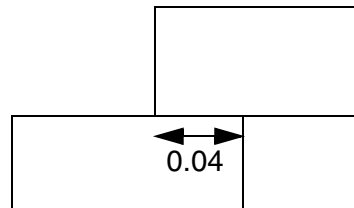
a) Violation, 0.07 is not a legal width of a horizontal wire



b) OK, both 0.15 and 0.1 match the wrong direction widths.



c) OK, the right direction width of 0.05 ( $\geq 0.05$ ) is met although the wrong way direction width fails



d) Violation, touching wires still subjects to the rule, and right direction width of 0.04 ( $< 0.05$ ) fails

### Width Rules

Width rules can be used to define the default routing width to use for all regular wiring on the layer.

You can define a width rule by using the following property definition:



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTY LEF58_WIDTH
    "WIDTH minWidth [WRONGDIRECTION]
    ;" ;
```

Where:

WIDTH is the same as the existing routing layer WIDTH syntax.

WRONGDIRECTION

Specifies the default routing width to use for all regular wiring with direction perpendicular to the specified direction in DIRECTION on a Manhattan routing layer.

Note that using WRONGDIRECTION changes the interpretation of any wrong-way routing widths in the DEF NETS section. If WRONGDIRECTION is specified, then any wrong-way routing in the DEF NETS section will use the WRONGDIRECTION width for that layer unless the net or route has a NONDEFAULTRULE with a WIDTH greater than the WRONGDIRECTION width. But, the implicit default route-extension is still half of the preferred direction width.

Some older tools may not understand this behavior. Normally, they will still read/write and round-trip the DEF routing properly, but will not understand that the width is slightly larger for wrong-way routes. If these tools check wrong-way width, then the DRC rules may flag false violations. RC extraction with the wrong width will also flag errors, although wrong-way routes are generally short and the width difference is small, so the RC error is normally negligible.

### Width Rule Examples

- The following width rule indicates that the default routing width of a vertical route is 0.1  $\mu\text{m}$ , while the default routing width of a horizontal route is 0.14  $\mu\text{m}$ :

```
DIRECTION VERITICAL;
WIDTH 0.1;
PROPERTY LEF58_WIDTH
    "WIDTH 0.14 WRONGDIRECTION ;" ;
```

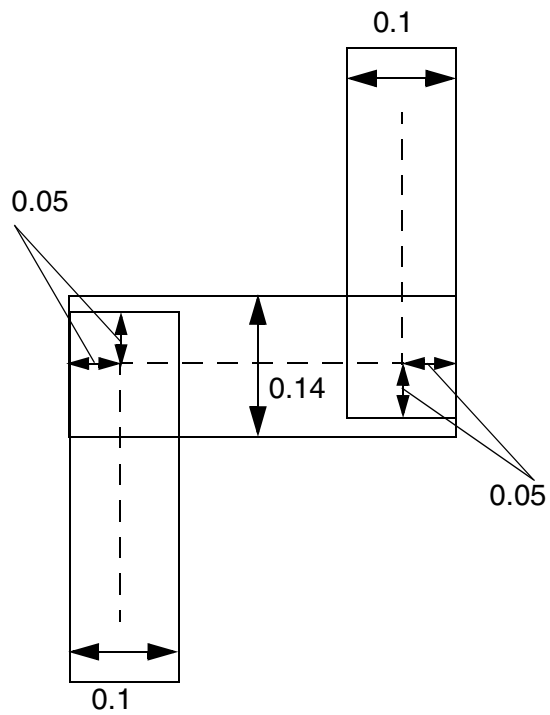
In the DEF, a vertical default route in the NETS section will have a width of 0.10  $\mu\text{m}$  with extension of 0.05  $\mu\text{m}$ , while a horizontal route will have a width of 0.14  $\mu\text{m}$  with extension of 0.05  $\mu\text{m}$ .

## LEF/DEF 5.8 Language Reference

### LEF Syntax

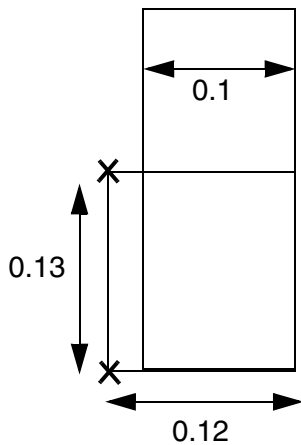
---

In more advanced nodes, wider widths are required for non-preferred direction. The following example shows route with wrong-way segment:

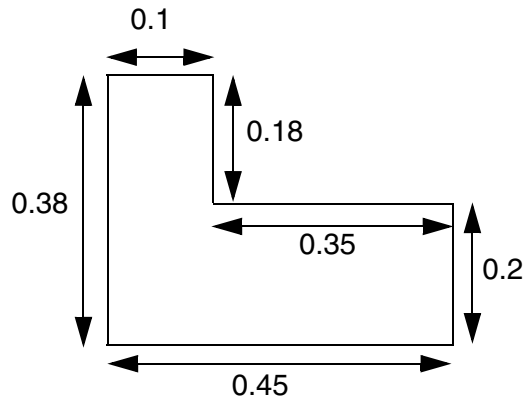


**Figure 1-250 Illustration of WIDTH Rule with WRONGDIRECTION**

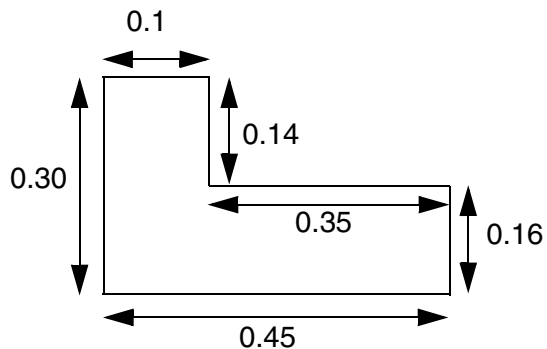
```
DIRECTION VERTICAL;  
WIDTH 0.1;  
PROPERTY LEF58_WIDTH "WIDTH 0.2 WRONGDIRECTION ;" ;
```



a) Violation. The 0.13 distance between the two convex corners marked with X is less than the wrong direction width of 0.2. It does not matter that 0.13 is larger or smaller than the 0.12 value in the other direction.



b) OK, the 0.18 vertical segment does not count as the wrong direction width. The wrong direction width along that edge is actually  $0.2 + 0.18 = 0.38$ .



c) Violation, the 0.16 vertical segment would fail the wrong direction width requirement of 0.2.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### **Minimum Step Rules**

Minimum step rules allow you to require a minimum adjacent edge length following edges that are less than the specified minimum step length. You can define at the most two `MINSTEP` statements.

You can define a minimum step rule by using the following property definition:

```
PROPERTY LEF58_MINSTEP
    "MINSTEP minStepLength
        [[INSIDECORNER | OUTSIDECORNER | STEP] [LENGTHSUM maxLength]]
        [MAXEDGES maxEdges]
        [ MINADJACENTLENGTH minAdjLength
            [CONVEXCORNER [EXCEPTWITHIN exceptWithin]
            |CONCAVECORNER
            |THREECONCAVECORNERS [CENTERWIDTH width]
            | minAdjLength2]
        | MINBETWEENLENGTH minBetweenLength
            [EXCEPTSAMECORNERS]
        | NOADJACENTEOL eolWidth ]
            [EXCEPTADJACENTLENGTH minAdjLength]
            | MINADJACENTLENGTH minAdjLength]
            [CONCAVECORNERS]
        | NOBETWEENEOL eolWidth]
    ] ;..." ;
```

Where:

All the other keywords are the same as the existing LEF routing layer `MINSTEP` syntax.

CONCAVECORNER	Specifies if a concave corner is between two convex corners, and if one of the length of the edges to form the conconcave corner is less than <i>minAdjLength</i> , then the other length must be greater than or equal to the <i>minStepLength</i> .  This has similar definition as CONVEXCORNER where the corner conditions are reversed.
CONCAVECORNERS	Specifies that the adjacent EOL minimum step rules only apply if both of the neighbor edges of the EOL have concave corners at the other end.
EXCEPTADJACENTLENGTH <i>minAdjLength</i>	Indicates that the adjacent EOL minimum step rule does not apply if the other neighbor edge of the minstep edge has length greater than or equal to <i>minAdjLength</i> . <i>Type: Float, specified in microns</i>

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### EXCEPTSAMECORNERS

Indicates that a *minBetweenLength* length is *not* required for an edge that has the same type of 90-degree corner at each end (that is, both corners are convex, or both are concave). See [Figure 1-253](#) on page 464 for an illustration of EXCEPTSAMECORNERS in a MINSTEP rule.

#### EXCEPTWITHIN *exceptWithin*

Specifies that if there is a neighbor object, same-net or different-net, is within *exceptWithin* measured in Euclidean distance from the convex corner or the two adjacent edges, the minstep rule does not apply.

Type: Float, specified in microns

#### MINADJACENTLENGTH *minAdjLength* [CONVEXCORNER | *minAdjLength2*]

Indicates that the edges adjacent to min-step edges that are less than *minStepLength* must be greater than or equal to *minAdjLength* in length in order to be allowed; otherwise, it is considered a violation.

If *minAdjLength2* is specified, then one adjacent edge must be greater than or equal to *minAdjLength* and the other adjacent edge must be greater than or equal to *minAdjLength2*. See **Minimum Step Rule Examples** on page 462.

Type: Float, specified in microns

The CONVEXCORNER keyword indicates that if a convex corner is between two concave corners, and if one of the length of the edges to form the convex corner is less than *minAdjLength*, then the other length must be greater than or equal to *minStepLength*.

#### MINADJACENTLENGTH *minAdjLength*

Specifies that the adjacent EOL minimum step rule applies only if the other neighbor edge forming a concave corner with the minimum step edge has length greater than *minAdjLength*. See [Figure 1-257](#) on page 466.

Type: Float, specified in microns

#### MINBETWEENLENGTH *minBetweenLength*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates that one of the edges between min-step edges that are less than *minStepLength* must be greater than or equal to *minBetweenLength* in length in order to be allowed; otherwise, it is considered a violation.

*Type:* Float, specified in microns

NOADJACENTEOL *eolWidth*

Indicates that the adjacent edges to min-step edges less than *minStepLength* must not be EOL edge with length less than *eolWidth* in order to be allowed; otherwise, it is considered a violation. In addition, MAXEDGES 1 should be defined along with NOADJACENTEOL.

*Type:* Float, specified in microns

NOBETWEENEOL *eolWidth*

Indicates that the edge between two minstep edges less than *minStepLength* must not be a EOL edge with length less than *eolWidth*. In addition, MAXEDGES 1 should be defined along with NOBETWEENEOL.

*Type:* Float, specified in microns

THREECONCAVECORNERS [CENTERWIDTH *width*]

Specifies that if a polygon has consecutive edges having three concave corners with two convex corners between them in a stair case, it is a violation if one of the edge touching the middle concave corner has length less than *minStepLength* and the other edge has length less than *minAdjLength*. If CENTERWIDTH is specified, it is only a violation if the width of both of the edges of the middle concave corner is less than or equal to *width*. See [Figure 1-262](#) on page 470.

*Type:* Float, specified in microns

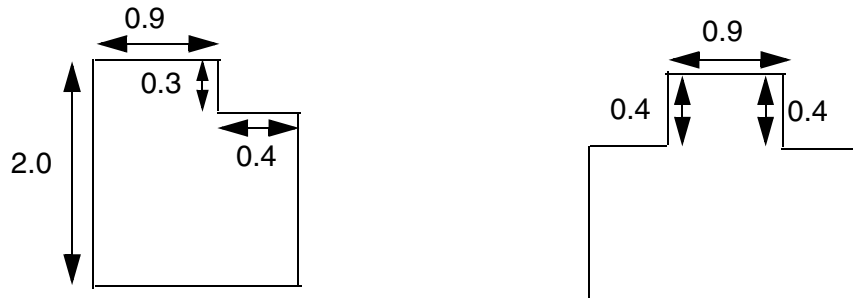
### Minimum Step Rule Examples

- The following minimum step rule indicates that a 0.9 μm EOL edge must not be between two minstep edges:

```
PROPERTY LEF58_MINSTEP
```

```
"MINSTEP 0.5 MAXEDGES 1 NOBETWEENEOL 1.0 ;" ;
```

**Figure 1-251 Illustration of MinStep Rule with NOBETWEENEOL**



a) OK, there is only one 0.3 minstep next to a 0.9 EOL, and having two consecutive minstep edges without a EOL between them does not trigger the rule

b) Violation, a 0.9 EOL is in between two minstep edges

- The following minimum step rule indicates that the rule applies if the non-EOL adjacent edge has a length that is greater than or equal to 0.8  $\mu\text{m}$ :

```
PROPERTY LEF58_MINSTEP
  "MINSTEP 0.5 MAXEDGES 1
    NOADJACENTEOL 1.0
    EXCEPTADJACENTLENGTH 0.8 ;" ;
```

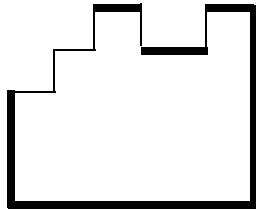
**Figure 1-252 Illustration of MinStep Rule with EXCEPTADJACENTLENGTH**



a) OK, the non-EOL adjacent edge has length of 0.8 ( $\geq 0.8$ )

b) Violation, both conditions on the neighbor edges fulfill

**Figure 1-253 Illustration of ExceptSameCorners Definition**



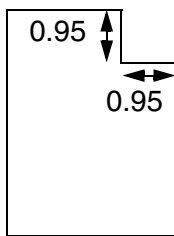
The `EXCEPTSAMECORNERS` definition of same-corners edges refers to any edge that ends in two convex or two concave 90-degree corners. All of the thick edges in the object above have same-corner edges, and would not be checked by this rule.. “Stair-step” edges are shown in thin lines.

- The following minimum step rule indicates that there can only be one edge less than 1.0  $\mu\text{m}$  in a row. The adjacent edges must be greater than or equal to 1.0  $\mu\text{m}$ .

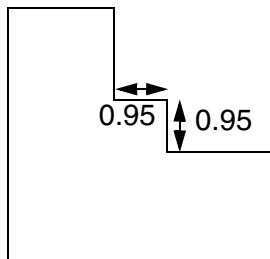
```
PROPERTY LEF58_MINSTEP  
    "MINSTEP 1.0 MAXEDGES 1 ;" ;
```

**Figure 1-254 MinStep Rule with MAXEDGES**

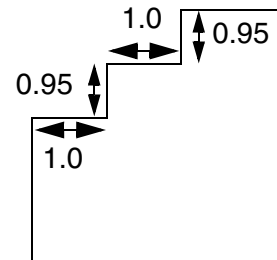
```
PROPERTY LEF58_MINSTEP "MINSTEP 1.0 MAXEDGES 1 ;" ;
```



a) Violation, more than 1 edges in a row that are < 1.0 in length.



b) Violation, more than 1 edges in a row that are < 1.0 in length.



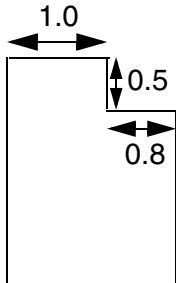
c) OK, only 1 edge in a row that is < 1.0 in length.

- The following minimum step rule indicates that an edge less than 0.6  $\mu\text{m}$  must have adjacent edges greater than or equal to 1.0  $\mu\text{m}$ .

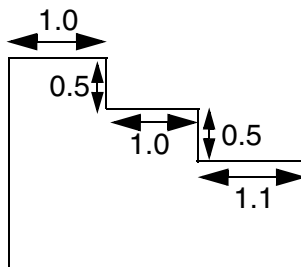
```
PROPERTY LEF58_MINSTEP  
    "MINSTEP 0.6 MAXEDGES 1 MINADJACENTLENGTH 1.0 ;" ;
```



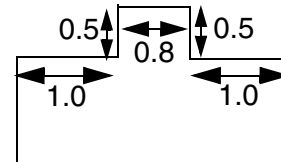
Figure 1-255 MinStep Rule with Adjacent Edge



a) Violation, an edge  $< 0.6$  long has adjacent edges  $< 1.0$ .



b) Okay, all edges  $< 0.6$  long have  $\geq 1.0$  adjacent edges.



c) Violation, the edges  $< 0.5$  have an adjacent edge  $< 1.0$ .

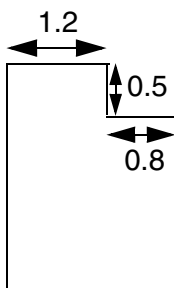
- The following minimum step rule indicates that no edge less than  $0.2 \mu\text{m}$ , and any edge less than  $0.6 \mu\text{m}$  must have one adjacent edge greater than or equal to  $1.0 \mu\text{m}$ , and the other adjacent edge greater than or equal to  $1.2 \mu\text{m}$ .

```
MINSTEP 0.2 MAXEDGES 0 ;
```

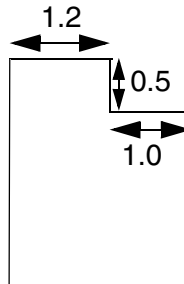
```
PROPERTY LEF58_MINSTEP
```

```
"MINSTEP 0.6 MAXEDGES 1 MINADJACENTLENGTH 1.0 1.2 ;" ;
```

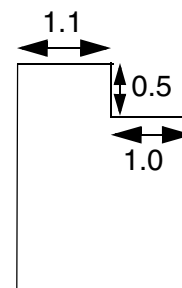
Figure 1-256 MinStep Rule with MINADJACENTLENGTH



a) Violation: an edge  $< 0.6$  long has adjacent edge  $< 1.0$ .



b) OK, an edge  $< 0.6$  long has adjacent edge  $\geq 1.0$ , and  $\geq 1.2$ .



c) Violation, an edge  $< 0.6$  long has adjacent edge  $\geq 1.0$ , but only 1.1 for the other edge ( $\geq 1.2$  required).

- The following rule illustrates that the minimum step rule does not apply if the non-EDO adjacent edge has length greater than the specified minimum adjacent length:

```
PROPERTY LEF58_MINSTEP "
```

```
MINSTEP 1.5 MAXEDGES 1
```

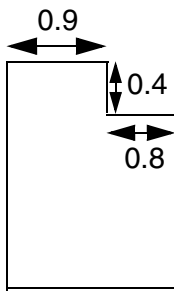
## LEF/DEF 5.8 Language Reference

### LEF Syntax

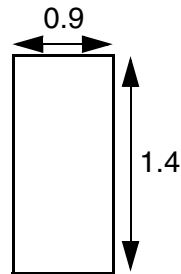
---

```
NOADJACENTEOL 1.0
MINADJACENTLENGTH 0.7 ; " ;
```

**Figure 1-257 MinStep Rule with MINADJACENTLENGTH in NOADJACENTEOL**



a) Violation, the non-EOL adjacent edge has length of 0.8 ( $> 0.7$ )

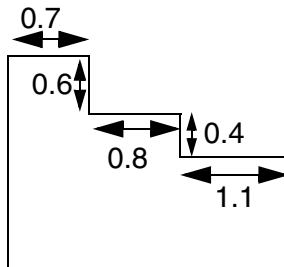


b) OK, the vertical min step edge does not form a concave corner with its neighbor

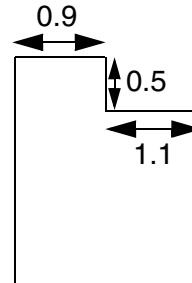
- The following minimum step rule illustrates CONVEXCORNER:

```
PROPERTY LEF58_MINSTEP
"MINSTEP 0.6 MAXEDGES 1 MINADJACENTLENGTH 1.0 CONVEXCORNER ;" ;
```

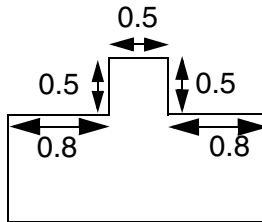
**Figure 1-258 MinStep Rule with CONVEXCORNER**



a) Violation, the convex corner of 0.4 and 0.8 edges is between 2 concave corners, and their lengths are smaller than 0.6 and 1.0 correspondingly



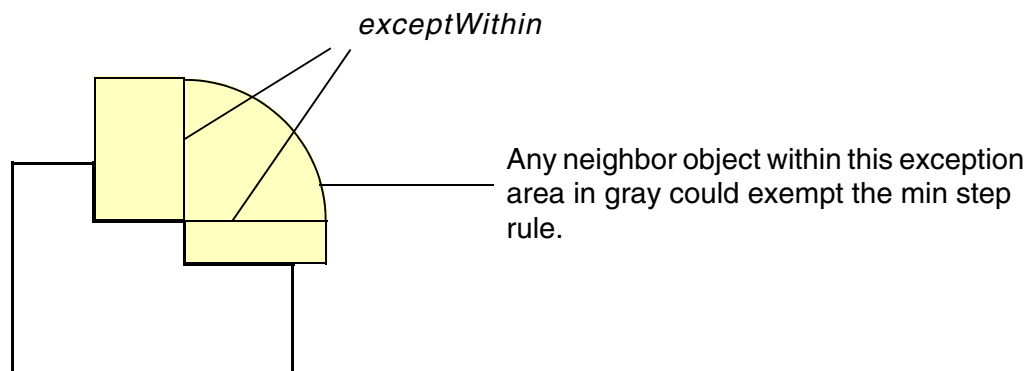
b) OK, the convex corner of 0.5 and 0.9 edges is not between 2 concave corners



c) OK, no convex corner is found between 2 concave corners

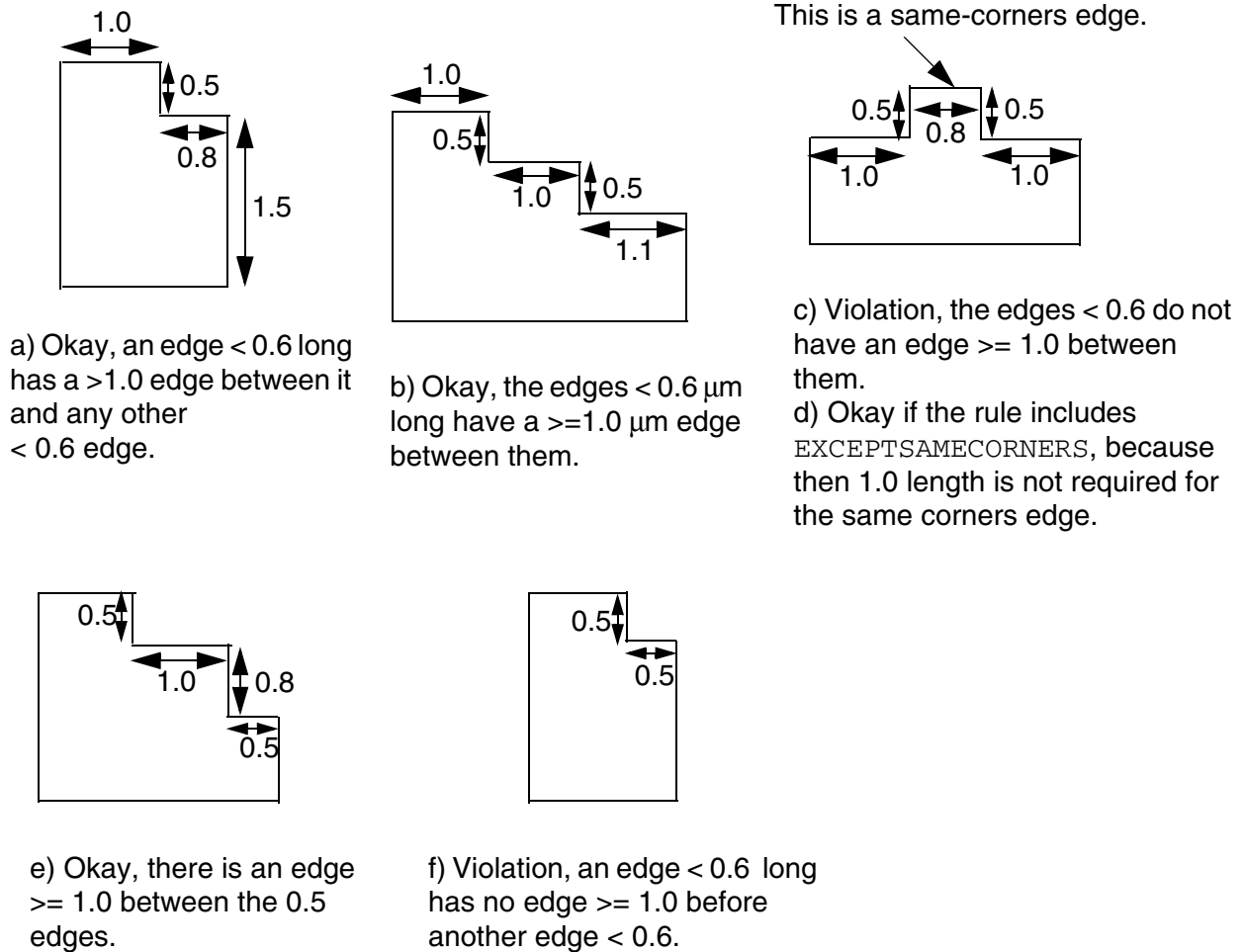
- The following figure defines `EXCEPTWITHIN` *exceptWithin* in `CONVEXCORNER`:

**Figure 1-259 Definition of `EXCEPTWITHIN` *exceptWithin* in `CONVEXCORNER`**



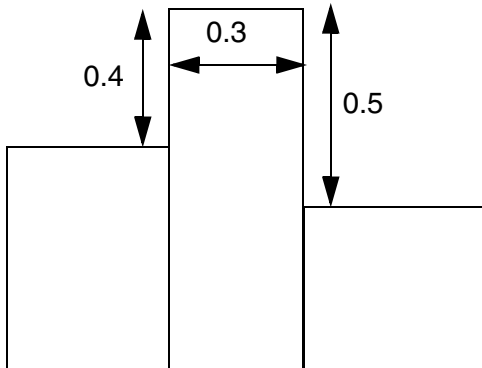
**Figure 1-260 MinStep Rule with EXCEPTSAMECORNERS**

```
PROPERTY LEF58_MINSTEP "MINSTEP 0.6 MAXEDGES 1 MINBETWEENLENGTH 1.0 ;" ;
```

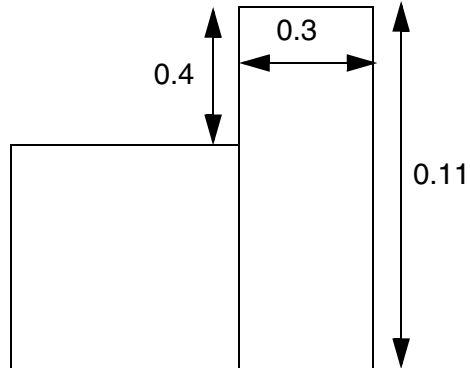


**Figure 1-261 MinStep Rule with CONCAVECORNERS**

```
PROPERTY LEF58_MINSTEP  
  "MINSTEP 0.5 MAXEDGES 1  
    NOADJACENTEOL 1.0 CONCAVECORNERS ; " ;
```



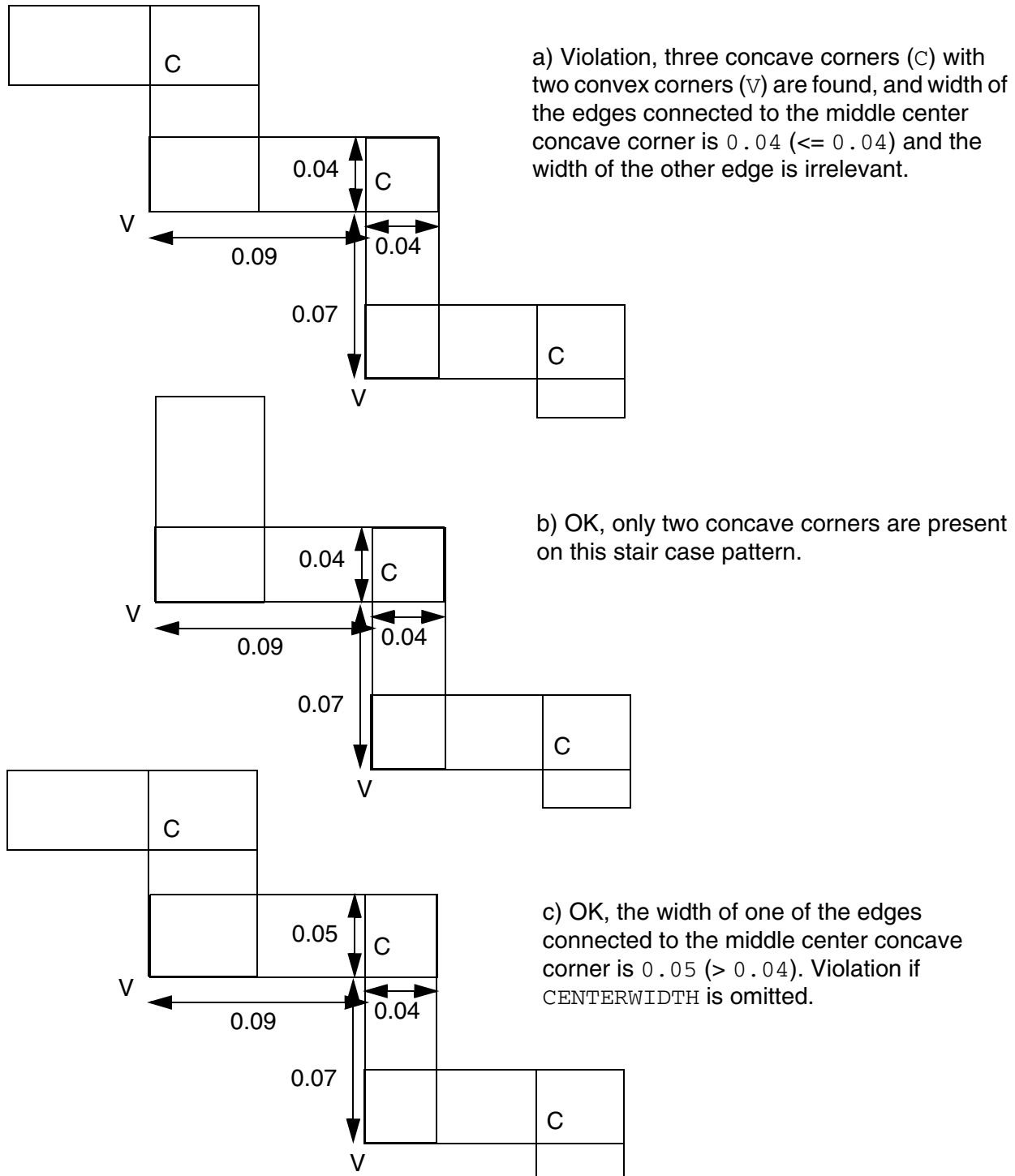
a) Violation, the 0.4 min step is next to 0.3 EOL while both 0.4 and 0.5 edges have concave corners on the other end



b) OK, the 0.11 edge does not have a concave corner on the other end. Violation if CONCAVECORNERS is omitted.

Figure 1-262 MinStep Rule with THREECONCAVECORNERS

```
PROPERTY LEF58_MINSTEP "  
  MINSTEP 0.1 MAXEDGES 1 MINADJACENTLENGTH 0.08  
  THREECONCAVECORNERS CENTERWIDTH 0.04; "
```



### ***Minimum Size Rule***

Minimum size rules allow you to specify the minimum width and length of a rectangle that must be able to fit somewhere within each polygon on a layer.

You can define a minimum size rule by using the following property definition:

```
PROPERTY LEF58_MINSIZE
    "MINSIZE [RECTONLY] minWidth minLength [minWidth minLength]...
    ] ;..." ;
```

Where:

All the other keywords are the same as the existing LEF routing layer `MINSIZE` syntax.

<code>RECTONLY</code>	Specifies that the minimum size rule applies only to rectangular wires.
-----------------------	-------------------------------------------------------------------------

### ***Jog to Jog Spacing Table Rule***

Jog to Jog spacing table rules can be used to define jog or protrusion spacing requirements.

You can define a jog to jog spacing table rule by using the following property definition:

```
PROPERTY LEF58_SPACINGTABLE
    "SPACINGTABLE JOGTOJOGSPACING jogToJogSpacing
    JOGWIDTH jogWidth SHORTJOGSPACING shortJogSpacing
    {WIDTH width PARALLEL parLength WITHIN parWithin
    [EXCEPTWITHIN lowExcludeSpacing highExcludeSpacing]
    LONGJOGSPACING longJogSpacing
    [SHORTJOGSPACING widthShortJogSpacing]} ...
    ; " ;
```

Where:

<code>EXCEPTWITHIN <i>lowExcludeSpacing highExcludeSpacing</i></code>	<p>Specifies that any wire that is greater than or equal to <i>lowExcludeSpacing</i> and less than <i>highExcludeSpacing</i> away from the wide wire should be ignored as if it does not exist.</p> <p>Type: Float, specified in microns</p>
-----------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
SPACINGTABLE JOGTOJOGSPACING jogToJogSpacing  
JOGWIDTH jogWidth SHORTJOGSPACING shortJogSpacing  
{WIDTH width PARALLEL parLength WITHIN parWithin  
LONGJOGSPACING longJogSpacing } ...
```

Specifies jog or protrusion spacing requirements as described below:

Find the last row that a wire with width greater than *width* has neighbor wires with a parallel run length greater than *parLength* that is less than *parWithin* distance away. If there is a jog or protrusion wire/edge in the wide wire or a neighbor wire, with continuous common projected length less than or equal to *jogWidth*, spacing to the opposite metal must be greater than or equal to *shortJogSpacing*. If the continuous common projected length is greater than *jogWidth*, then spacing to the opposite metal must be greater than or equal to *longJogSpacing*. If there are two or more jog or protrusion wires with continuous common projected length less than or equal to *jogWidth* and spacing on the jog or protrusion wires is less than *longJogSpacing*, (projected) spacing in the wire direction of the wide wire between any two jog or protrusion wires must be greater than or equal to *jogToJogSpacing*. All of the values in the table must be the same or increasing, and *longJogSpacing* must be larger than *shortJogSpacing*.

Type: Float, specified in microns

```
SHORTJOGSPACING widthShortJogSpacing
```

Specifies individual *widthShortJogSpacing* for the given width, which would override and has the same meaning as the global *shortJogSpacing*.

Type: Float, specified in microns

### Jog to Jog Spacing Table Rule Examples

- [Figure 1-263](#) on page 474 illustrates the following jog to jog spacing table rules:

```
PROPERTY LEF58_SPACINGTABLE "  
  "SPACINGTABLE JOGTOJOGSPACING 0.7  
    JOGWIDTH 0.2 SHORTJOGSPACING 0.08  
    WIDTH 0.4 PARALLEL 0.2 WITHIN 0.5  
    LONGJOGSPACING 0.13  
    WIDTH 0.8 PARALLEL 0.3 WITHIN 0.5
```



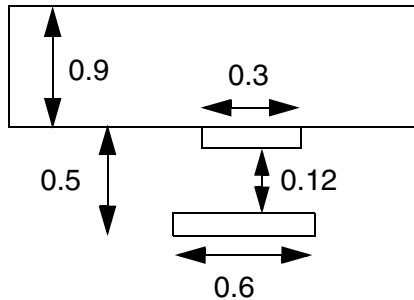
## LEF/DEF 5.8 Language Reference

### LEF Syntax

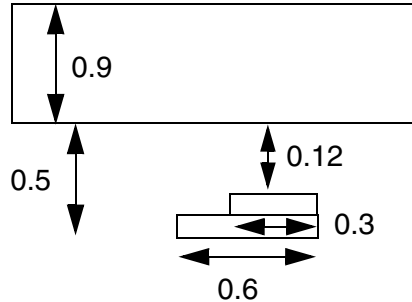
---

```
LONGJOGSPACING 0.15 ; " ;
```

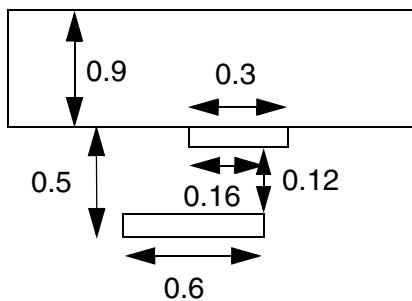
**Figure 1-263 Illustration of Spacing Table with JOGTOJOGSPACING**



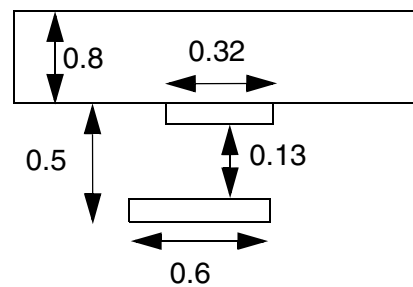
a) Violation, all the triggering conditions are met, and the spacing is 0.12 ( $< 0.15$ )



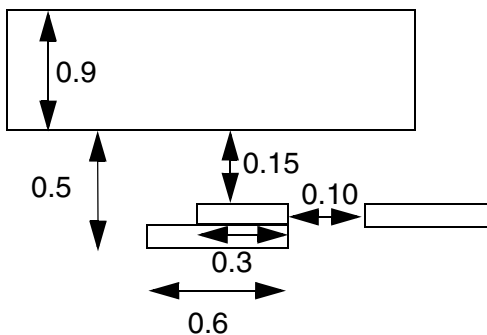
b) Violation, the jog/protrusion can be in the neighbor wire



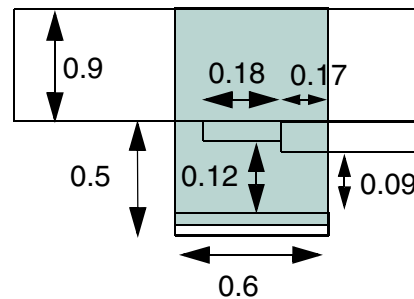
c) OK, the run length of the jog/protrusion is 0.16, spacing of 0.12  $\geq 0.08$



d) OK, the wide wire without the jog/protrusion must have width  $> 0.8$  to trigger the jog spacing of 0.15 while the jog spacing of 0.13 based on width  $> 0.4$  is met.



e) OK, the 0.15 jog spacing is applied only to the opposite metal, but not to the right side neighbor 0.10 away



f) Violation, the continuous run length with spacing  $< 0.15$  is 0.35 (0.18 + 0.17), which would require spacing of 0.15

## LEF/DEF 5.8 Language Reference

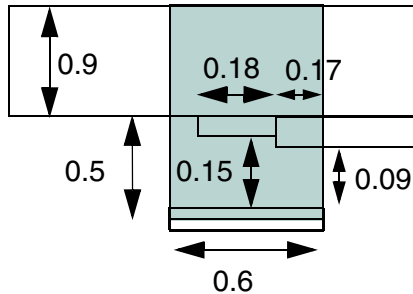
### LEF Syntax

---

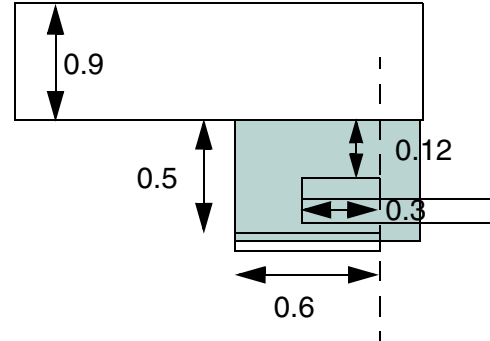
- Figure 1-264 on page 476 illustrates the following jog to jog spacing table rules:

```
PROPERTY LEF58_SPACINGTABLE "  
    "SPACINGTABLE JOGTOJOGSPACING 0.7  
        JOGWIDTH 0.2 SHORTJOGSPACING 0.08  
        WIDTH 0.8 PARALLEL 0.3 WITHIN 0.5  
        LONGJOGSPACING 0.15 ; " ;
```

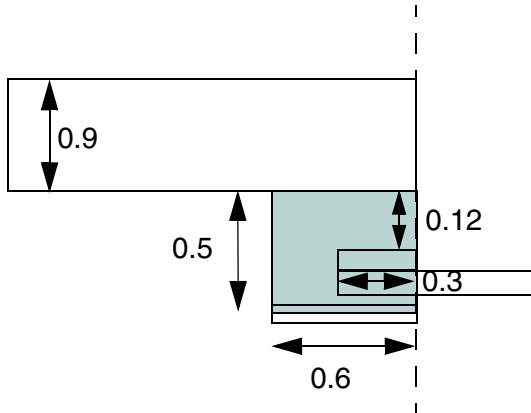
Figure 1-264 Illustration of Spacing Table with JOGTOJOGSPACING



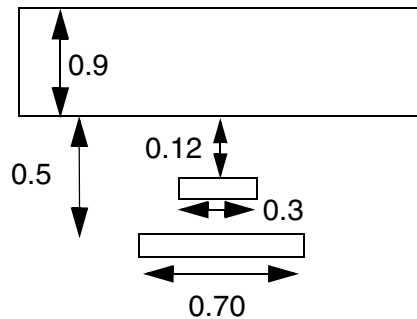
a) OK, only 0.17 segment is within 0.15, spacing of 0.08 is required and met



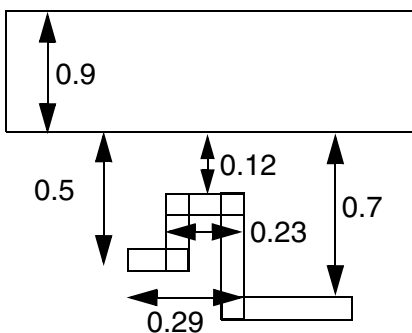
b) Violation, all neighbor wires in the green region are considered



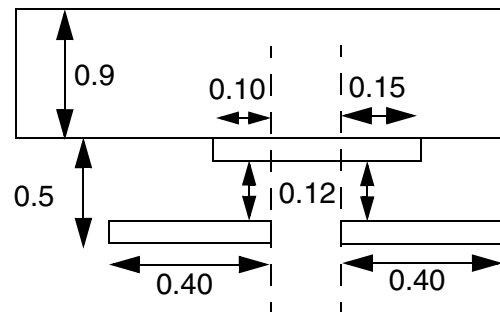
c) Violation, any neighbor wires with run length > 0.2 must have spacing  $\geq 0.15$



d) Violation, the neighbor wire does not need to be a jog



e) OK, the parallel run length is 0.29 within the 0.32 away, the rule is not checked



f) OK, the common projected run length of the jog/protrusion is 0.10 and 0.15 individually, 0.08 spacing is needed and met

## LEF/DEF 5.8 Language Reference

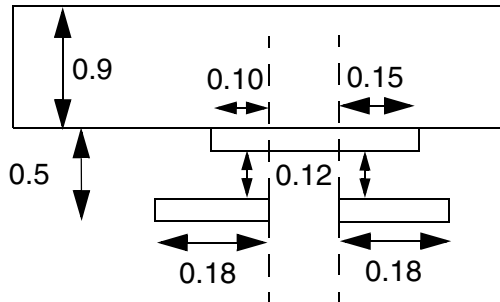
### LEF Syntax

---

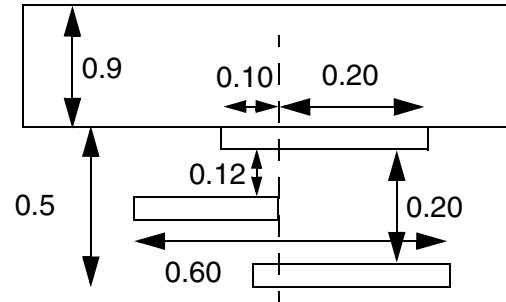
- Figure 1-265 on page 478 illustrates the following jog to jog spacing table rules:

```
PROPERTY LEF58_SPACINGTABLE "  
    "SPACINGTABLE JOGTOJOGSPACING 0.7  
    JOGWIDTH 0.2 SHORTJOGSPACING 0.08  
    WIDTH 0.8 PARALLEL 0.3 WITHIN 0.5  
    LONGJOGSPACING 0.15 ; " ;
```

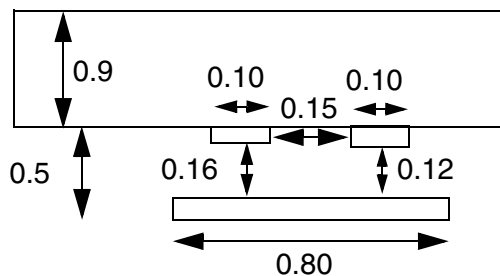
**Figure 1-265 Illustration of Spacing Table with JOGTOJOGSPACING**



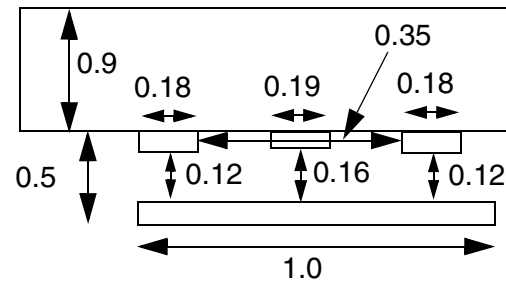
a) OK, parallel run length of 0.18 individually would not trigger the rule



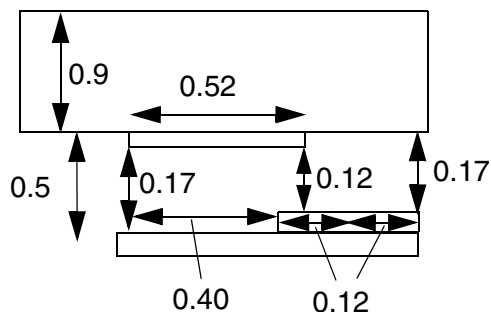
b) OK, only the 0.10 segment is within 0.15, spacing of 0.08 is required and met



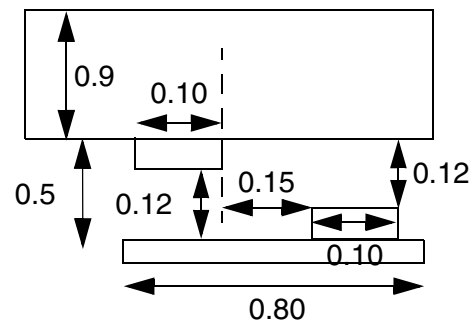
c) OK, the left jog/protrusion having a jog spacing of 0.16 ( $\geq 0.15$ ) is not a valid jog/protrusion for considering spacing among jog/protrusion wires



d) Violation, having a 'good' 0.19 jog/protrusion at the middle is irrelevant, the distance of 0.35 between the left and right jog/protrusion wires would still violate  $\geq 0.7$  jog to jog spacing.



e) OK, the top 0.52 jog/protrusion is broken into two segments for the rule, and 0.40 segment has spacing of 0.17 ( $\geq 0.15$ ), and the 0.12 segment has spacing of 0.12 ( $\geq 0.08$ ). The bottom 0.24 jog/protrusion is broken into two segments of 0.12 with spacing of 0.17 and 0.12 ( $\geq 0.08$ )



f) Violation, jog/protrusion wires on opposite wires still need to follow the 0.7 spacing

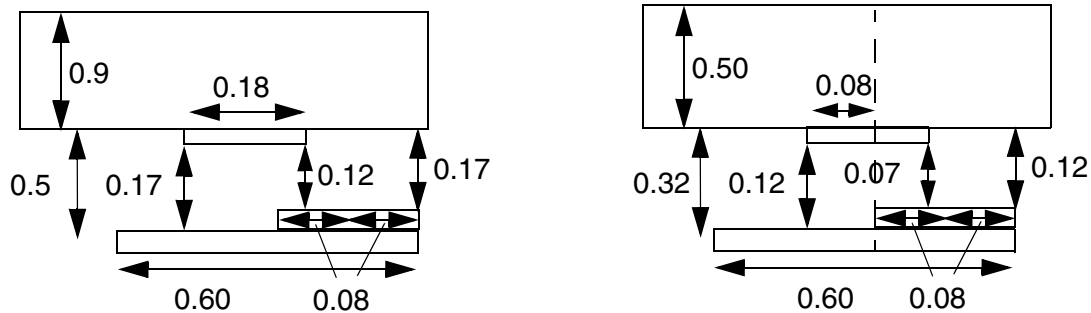
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- Figure 1-266 on page 479 illustrates the following jog to jog spacing table rules with JOGTOJOGSPACING 0.3:

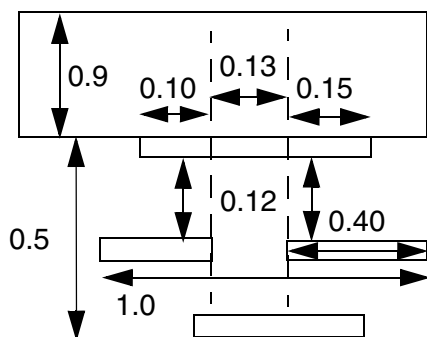
```
PROPERTY LEF58_SPACINGTABLE "  
    "SPACINGTABLE JOGTOJOGSPACING 0.7  
    JOGWIDTH 0.2 SHORTJOGSPACING 0.08  
    WIDTH 0.8 PARALLEL 0.3 WITHIN 0.5  
    LONGJOGSPACING 0.15 ; "
```

**Figure 1-266 Illustration of Spacing Table with JOGTOJOGSPACING**



a) OK, the jogs/protrusions are broken into two segments each. Eventually, there is only one common segment with length 0.08 and spacing < 0.15, jog to jog spacing of 0.7 is not checked.

b) Violation, the continuous run length with spacing < 0.13 is 0.24 (three segments of 0.08), which would require spacing of 0.13



c) Violation, 0.10 and 0.15 segments are considered individually, and would fail the jog to jog spacing of 0.7.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

- The following example indicates that the short jog spacing is 0.08  $\mu\text{m}$  for width greater than 0.4  $\mu\text{m}$ , 0.09  $\mu\text{m}$  for width greater than 0.8  $\mu\text{m}$ . Any neighbor wire greater than or equal to 0.04  $\mu\text{m}$  and less than 0.06  $\mu\text{m}$  away is ignored (for width greater than 0.8  $\mu\text{m}$  only).

```
PROPERTY LEF58_SPACINGTABLE "  
    "SPACINGTABLE JOGTOJOGSPACING 0.7  
        JOGWIDTH 0.2 SHORTJOGSPACING 0.08  
        WIDTH 0.4 PARALLEL 0.2 WITHIN 0.5  
            LONGJOGSPACING 0.13 SHORTJOGSPACING 0.08  
        WIDTH 0.8 PARALLEL 0.3 WITHIN 0.5  
            EXCEPTWITHIN 0.04 0.06  
            LONGJOGSPACING 0.15 SHORTJOGSPACING 0.09 ; " ;
```

### ***Direction Span Length Spacing Table Rule***

You can create a direction span length spacing table rule to create a table in which the spacing between two objects depends on the maximum span length of the two objects and their parallel run length.

You can define a direction span length spacing table using the following property definition:

```
PROPERTY LEF58_SPACINGTABLE  
    "SPACINGTABLE  
        DIRECTIONALSPANLENGTH [WRONGDIRECTION]  
        [SAMEMASK]  
        [EXCEPTJOGLNGTH length [EDGELENGTH] [INCLUDELSHAPE]]  
        [EXCEPTEOL eolWidth]  
        [EXACTSPANLENGTHSPACING spanLength1 TO spanLength2  
            [PRL prl] {exactSpacing}...]...  
        PRL {prl}...  
        {SPANLENGTH spanLength  
            [EXACTSPACING exactSpacing  
            | SPACINGTOMINSPAN spacingToMinSpan]  
            [EXACTSELFSPACING exactSelfSpacing]  
            [NOEXCEPTEOL|EOLSPACING eolspacing] {spacing}...}...  
    ; " ;
```

Where:

```
DIRECTIONALSPANLENGTH PRL {prl}...  
{ SPANLENGTH spanLength {spacing}...}...
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Creates a table in which spacing between two objects depends on the span length of the objects and their parallel run length. There must be exactly as many spacing values in each SPANLENGTH row as there are PRL columns. All of the values, *prl*, *spanLength* and *spacing* must be monotonically increasing. In other words, the *prl* values must be increasing from left to right, the *spanLength* values must be increasing from top to bottom. If *prl* value is negative, the spacing is measured in a MAXXY style. In other words, no neighbor can be within the search window with dimension of  $\text{abs}(\text{prl})$  and *spacing*. The spacing values must be the same or increasing from left to right, but could have any value relationship, from top to bottom in the table. To find the required spacing, locate the last row where the individual span length of the two objects is, greater than the table row span length, and the right-most column where the parallel run length of the two objects is, greater than the table column *prl*, the maximum of the corresponding spacing values is the required spacing. When parallel run length of the two objects is less than or equal to the first PRL value, then these rules do not apply, but some other rules will. In addition, spacing value can be zero, which can typically be defined in negative PRL, indicating that there is no constraint on that object, and spacing requirement of the other object is not checked. See [Figure 1-269](#) on page 487.

*Type:* Float, specified in microns

EOLSPACING *eolspacing*

Specifies a spacing value that should be applied when the EXCEPTEOL condition is met.

*Type:* Float, specified in microns

EXACTSELFSPACING *exactSelfSpacing*

Specifies that two wires with span length that belong to the same row could have exact spacing of *exactSelfSpacing*. This should not be used on the first row since it would have the same meaning of EXACTSPACING.

EXACTSPACING *exactSpacing*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that if the spacing of a wire with span length greater than specified span length of a row with `EXACTSPACING` and less than or equal to the specified span length of the next row is exactly *exactSpacing*, and it (or its) neighbor wire has a span length between the specified span length of the first two rows, then it is not a violation. The *exactSpacing* should be smaller than all of the spacing values on that row, except when the spacing value is zero for a certain PRL, which means that there is no constraint between wires belonging to that row to any other wires with parallel run length less than or equal to the specified value.

*Type:* Float, specified in microns

```
EXACTSPANLENGTHSPACING spanLength1 TO spanLength2  
[PRL prl] {exactSpacing}...
```

Specifies a list of exact spacing values in {*exactSpacing*}..., which are also allowed between a wire with exact span length of *spanLength1* and another wire with exact span length of *spanLength2* with parallel run length greater than *prl* or zero if PRL is not specified. This is in addition to spacing value of *spacing* in the table. In addition, this construct should not be used along with either `EXACTSPACING` or `EXACTSELFSPACING` in individual rows.

*Type:* Float, specified in microns

```
EXCEPTEOL eolWidth
```

Specifies that the directional span length spacing does not apply to EOL edges with width less than the *eolWidth*. Those EOL edges should follow the `SPACING ENDOFLINE` rules.

*Type:* Float, specified in microns

```
EXCEPTJOGLENGTH length [EDGELENGTH] [INCLUDELSHAPE]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies that the minimum default (right-way) spacing would be applied between a wrong way jog with minimum wrong way width and length less than *length* sandwiched by two right way wires with projected PRL less than 0 in wrong way direction in a 'Z' shape and a right way wire. See [Figure 1-267](#) on page 485.

If INCLUDELSHAPE is specified without EDGELENGTH, this minimum default spacing will also be applied between a wrong way jog with minimum wrong way width and length less than *length* connected to a right way wire in a 'L' shape and a right way wire.

If EDGELENGTH is specified, *length* is measured against the length of the edges in the wrong way direction of a wrong way jog in a 'Z' shape instead of the entire span length of the jog. If INCLUDELSHAPE is also specified to include an 'L' shape exemption, the longest edge length would be the same as the span length of the jog, and the EDGELENGTH construct would have no impact on it.

Type: Float, specified in microns

NOEXCEPTEOL

Specifies that EXCEPTEOL does not apply to the rows that have NOEXCEPTEOL keyword.

SAMEMASK

Specifies that the directional span length spacing rule only applies to objects on the same mask.

SPACINGTOMINSPAN *spacingToMinSpan*

Specifies that if the spacing of a wire with span length greater than specified span length of a row with SPACINGTOMINSPAN and less than or equal to specified span length of the next row is greater than or equal to *spacingToMinSpan*, and it (or it's) neighbor wire has a span length between the specified span length of the first two rows, then it is not a violation, and spacing requirement of the other object is not checked. When the spacing value is zero for a certain PRL, this indicates that there is no constraint between wires belonging to that row to any other wires with parallel run length less than or equal to the PRL value of the first non-zero spacing value. In other words, *spacingToMinSpan* will not override the zero spacing value in this case. See [Figure 1-270](#) on page 488.

Type: Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

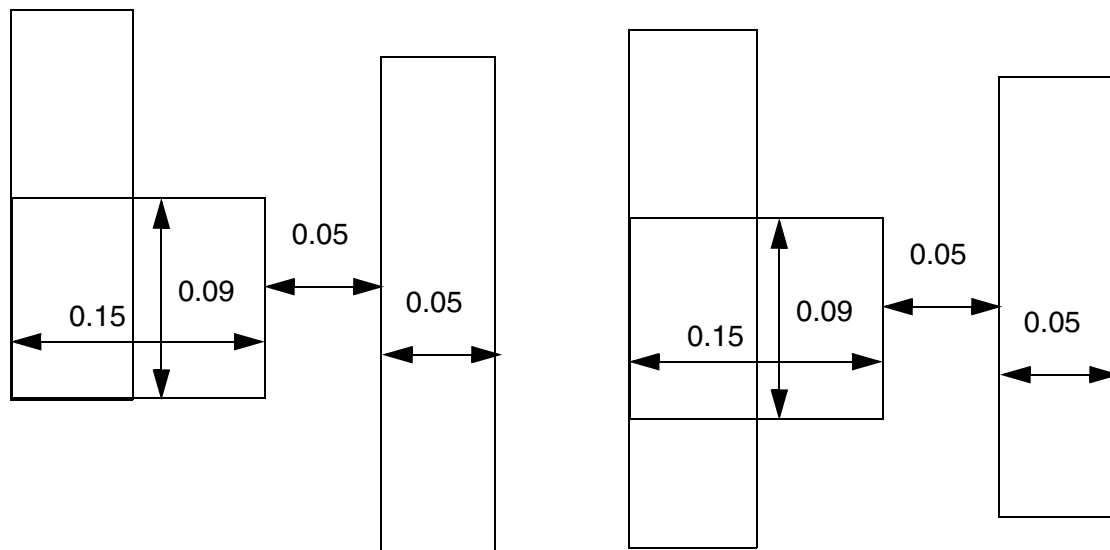
**WRONGDIRECTION** Specifies that the spacing is applied to wrong direction edges. In other words, the spacing is applied to the direction parallel to the specified direction in **DIRECTION** on a Manhattan routing layer. You must define two tables - one with **WRONGDIRECTION** and the other without **WRONGDIRECTION**. The spacing of the one without **WRONGDIRECTION** is applied to the direction perpendicular to the specified direction in **DIRECTION**.

### Direction Span Length Spacing Table Rule Examples

- The following example illustrates the following spacing table rules with direction span length:

```
DIRECTION VERTICAL ;
PROPERTY LEF58_WIDHTABLE
    "WIDHTABLE 0.050 ... ;
    WIDHTABLE 0.090 ... WRONGDIRECTION ; " ;
PROPERTY LEF58_SPACINGTABLE "
    SPACINGTABLE
        DIRECTIONALSPANLENGTH
            EXCEPTJOGLLENGTH 0.170
            PRL                                0.000 ...
            SPANLENGTH 0.000                  0.050 ...
            SPANLENGTH 0.140                  0.100 ... ;
```

**Figure 1-267 Illustration of Spacing Table Rule with DIRECTIONALSPANLENGTH**



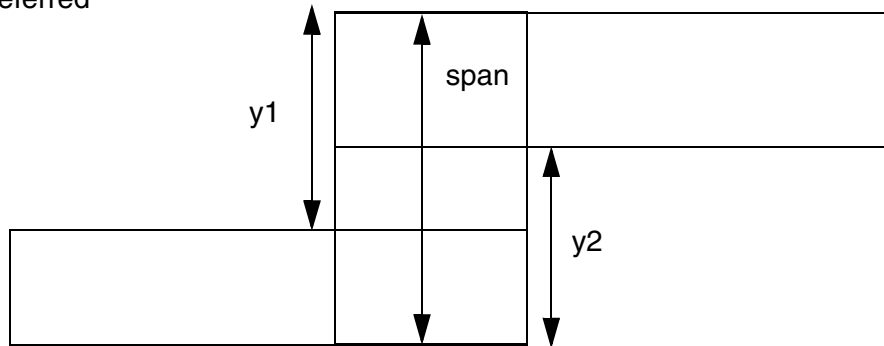
a) OK, `EXCEPTJOGLNGTH` applies to L shape configuration

b) Violation, `EXCEPTJOGLNGTH` does not apply to T shape configuration.

- The following example illustrates use of `EDGELENGTH`:

**Figure 1-268 Illustration of DIRECTIONALSPANLENGTH with EDGELENGTH in EXCEPTJOGLENGTH**

Horizontal is the preferred routing direction.



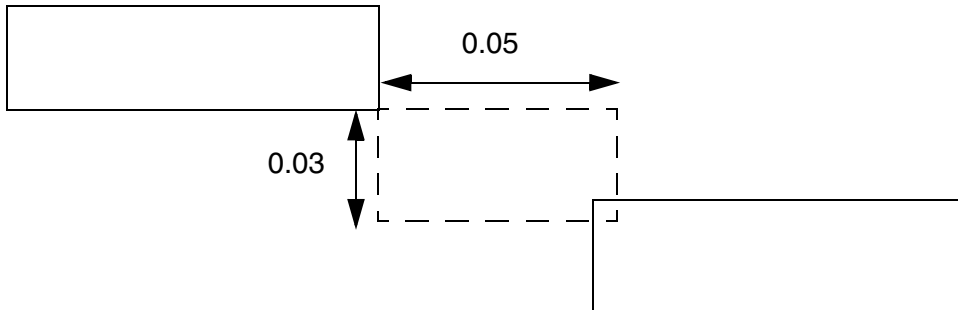
Both  $y1$  and  $y2$  must be  $< length$  to be qualified as a 'Z' shape for EXCEPTJOGLENGTH exemption. If EDGELENGTH is not specified, span must be  $< length$  to be qualified.

Illustration of EXCEPTJOGLENGTH *length* EDGELENGTH

- The following example illustrates direction spanlength with negative parallel run length:

```
DIRECTION HORIZONTAL ;
PROPERTY LEF58_SPACINGTABLE "
    SPACINGTABLE
        DIRECTIONALSPANLENGTH
            PRL                -0.050 ...
            SPANLENGTH         0.000    0.030 ... ; " ;
```

**Figure 1-269 Illustration of DIRECTIONALSPANLENGTH with PRL**



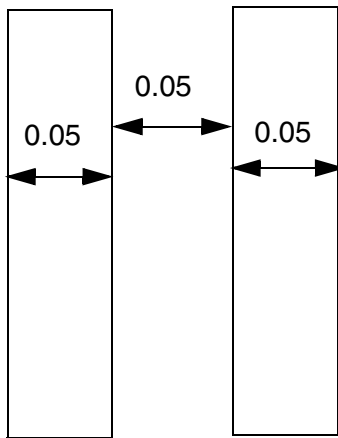
a) Violation, when PRL is negative, a search window of 0.03 x 0.05 is formed for neighbors (0.03 being the direction perpendicular to the preferred routing direction in the table without WRONGDIRECTION)

- The following example illustrates direction span length with spacing to minimum span:

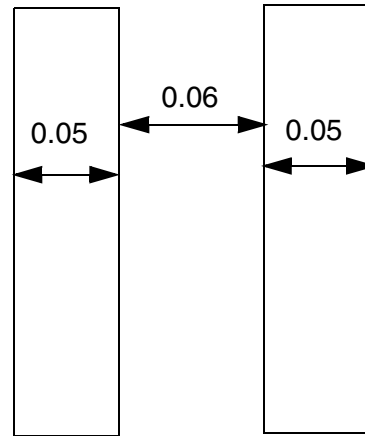
```

DIRECTION VERTICAL ;
PROPERTY LEF58_SPACINGTABLE "
    SPACINGTABLE
        DIRECTIONALSPANLENGTH
        PRL                                0.000 ...
        SPANLENGTH 0.000 EXACTSPACING 0.050 0.070 ...
        SPANLENGTH 0.090 SPACINGTOMINSPAN 0.060 0.080 ...
        SPANLENGTH 0.110                    0.090 ... ; " ;
    
```

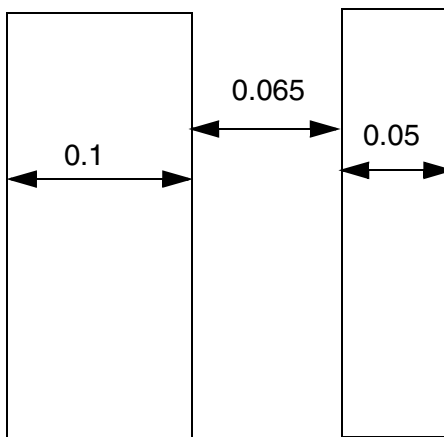
**Figure 1-270 Illustration of Spacing Table Rule with DIRECTIONALSPANLENGTH**



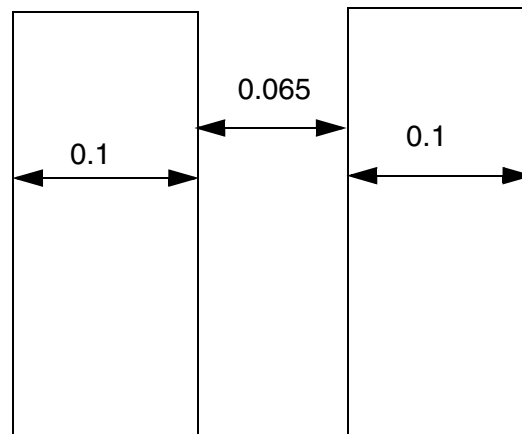
a) OK, exact spacing of 0.05 is fine.



b) Violation, if the spacing is not exactly 0.05, it needs to be  $\geq 0.07$



c) OK, spacing of span length of 0.1 to a span length of 0.05 between the first 2 rows ( $> 0$  &  $\leq 0.09$ ) must be  $\geq 0.06$ , which is fulfilled



d) Violation, spacing of span length of 0.1 of 2 wires must be  $\geq 0.08$

### ***Two-Widths Spacing Table Rule***

You can create a two-widths spacing table rule to create a table in which spacing between two objects depends on the widths of both the objects.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

You can define a two-widths spacing table rule by using the following property definition:

```
PROPERTY LEF58_SPACINGTABLE
    "SPACINGTABLE
        TWOWIDTHS [WRONGDIRECTION]
            [SAMEMASK | MASK maskNum]
            [EXCEPTEOL eolWidth]
            [FILLCONCAVECORNER fillTriangle]
            {WIDTH width [PRL runLength] {spacing} ...} ...
    ; " ;
```

Where:

All other keywords are the same as the existing LEF routing layer SPACINGTABLE syntax.

**EXCEPTEOL *eolWidth*** Specifies that the two widths table does not apply to EOL edges with width less than *eolWidth*. Those EOL edges should follow the SPACING ENDOFLINE rules instead.  
*Type:* Float, specified in microns

**FILLCONCAVECORNER *fillTriangle*** Specifies that any concave corners should be filled by a triangle with value of *fillTriangle* along the edges to form the corner before minimum spacing of the table is checked. This fill corner operation does not apply to any other spacing values. See [Figure 1-271](#) on page 490.  
*Type:* Float, specified in microns

**MASK *maskNum*** Specifies that the spacing rule applies only between two wires with the given mask. In addition, the spacing in the table entry is always applied between two wires of the corresponding width values even if there is another wire with any mask between them. *maskNum* must be a positive integer, and most applications only support the values 1, 2, or 3.  
*Type:* Integer

**SAMEMASK** Specifies that spacing applies only to the objects that belong to the same mask. The same mask objects recognize the first spacing table, but not the second one. The spacing in the second table is incorporated in the first one.  
*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

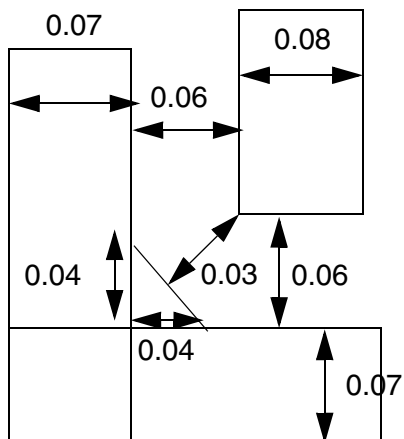
### LEF Syntax

**WRONGDIRECTION** Specifies the spacings of an edge with direction perpendicular to the specified direction in **DIRECTION** on a Manhattan routing layer to any other edges on that layer. This has a similar definition in **SPACINGTABLE PARALLELRUNLENGTH** spacing table.  
*Type:* Float, specified in microns

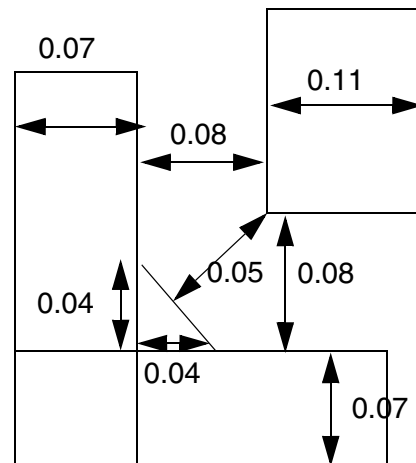
### Two-Widths Spacing Table Rule Examples

**Figure 1-271 Two-Widths Spacing Table Rule with FILLCONCAVECORNER**

```
PROPERTY LEF58_SPACINGTABLE
  "TWOWIDTHS FILLCONCAVECORNER 0.04
  WIDTH 0.06 0.08
  WIDTH 0.08 0.09 ; " ;
```



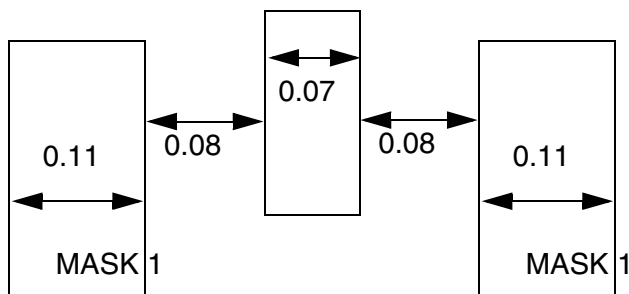
a) Violation, spacing is only 0.03 to the triangular fill with length of 0.04 along the edges of the concave corner.



b) OK, the corner fill does not apply to non-minimum spacing check

**Figure 1-272 Two-Widths Spacing Table Rule with MASK**

```
PROPERTY LEF58_SPACINGTABLE
    "TWOWIDTHS MASK 1
    WIDTH 0      0.04  0.06
    WIDTH 0.1    0.06  0.30 ; " ;
```



a) Violation between two 0.11 wires; having a middle wire with any mask is irrelevant.

### ***Parallel Span Length Spacing Table Rule***

Parallel span length spacing table rules can be used to define supplement spacing constraints based on span length of the objects.

You can define a parallel span length spacing table rule by using the following property definition:

```
PROPERTY LEF58_SPACINGTABLE
    "SPACINGTABLE
    [PARALLELRUNLENGTH [WRONGDIRECTION] [SAMEMASK]
    [EXCEPTEOL eolWidth]
    {length}...
    {WIDTH width
        [EXCEPTWITHIN lowExcludeSpacing highExcludeSpacing]
        {spacing}...}... ;
    [SPACINGTABLE
        INFLUENCE {WIDTH width WITHIN distance SPACING spacing} ... ;]
    |TWOWIDTHS {WIDTH width [PRL runLength] {spacing}...}... ;
    |PARALLELSPANLENGTH PRL runLength {SPANLENGTH spanLength {spacing}}...
    ;
    ; " ;
```

Where:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

All other keywords are the same as the existing LEF routing layer SPACINGTABLE syntax.

`EXCEPTEOL eolWidth` Specifies that the parallel run length table does not apply to EOL edges with width less than *eolWidth*. Those EOL edges should follow the SPACING ENDOFLINE rules instead.  
*Type*: Float, specified in microns

`EXCEPTWITHIN lowExcludeSpacing highExcludeSpacing`

Specifies that any wire that is at a distance greater than or equal to the *lowExcludeSpacing* and less than *highExcludeSpacing* away from the wide wire should be ignored, as if it does not exist. The conditions *lowExcludeSpacing* and *highExcludeSpacing* only apply to spacing if both of them are less than the specified *spacing*. In other words, spacing between the two wires must not be greater than 0 to less than *lowExcludeSpacing* and greater than or equal to *highExcludeSpacing* to less than *spacing*.

*Type*: Float, specified in microns

Note that at the most one such SAMEMASK spacing table and another parallel run length spacing table without the SAMEMASK keyword can be defined together.

The *length* can be negative that will act as extending the edges by  $\text{abs}(\text{length})$  in a WITHIN style and enforcing *spacing* in MAXXY style.

`PARALLELSPANLENGTH PRL runLength {SPANLENGTH spanLength {spacing}}`

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Creates a table in which spacing between two objects depends on the span length of both the objects that have a parallel run length greater than *runLength*. To find the required spacing, a NxN two-dimensional table is used that implicitly has the same span lengths for row and column headings. There must be exactly as many spacing values in each SPANLENGTH row as the number of SPANLENGTH rows. The *spanLength* values must increase from top to bottom in the table. The spacing values must be the same, or increase from left to right and from top to bottom across the table. Consider two objects with *spanLength1* and *spanLength2*. You need to find the last row where *spanLength1* is greater than the table row *spanLength*, and the right-most column where *spanLength2* is greater than the table column *spanLength*. The intersection of the matching row and column provides the required spacing.

The parallel run length is measured as a sum of lengths between objects. Turning corners may break up the parallel run length thus resulting in inaccurate calculations. These are supplement spacing constraints, in addition to the regular spacing based on wire width.

*Type:* Float, specified in microns (for all values)

SAMEMASK

Specifies that the spacings only apply to objects belonging to the same mask.

*Type:* Float, specified in microns

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### WRONGDIRECTION

Specifies the spacings of an edge with direction perpendicular to the specified direction in `DIRECTION` on a Manhattan routing layer to any other edges on that layer.

*Type:* Float, specified in microns

Note that at the most one such spacing table and another parallel run length spacing table without the `WRONGDIRECTION` keyword can be defined together. In addition, `SPACING` with `WRONGDIRECTION` keyword is also allowed on that layer.

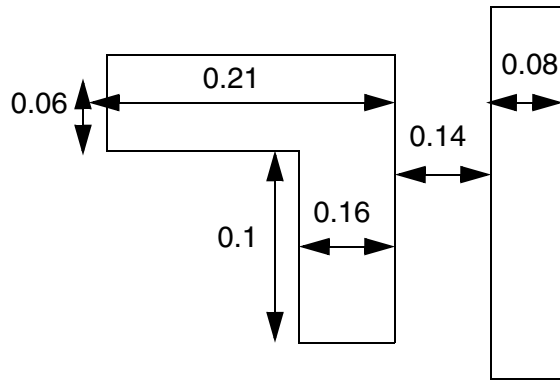
With the existence of `PARALLELRUNLENGTH` `WRONGDIRECTION`, there is a built-in condition for `PARALLELRUNLENGTH` for both with or without `WRONGDIRECTION` spacing tables. The edge length of an edge having width greater than *width* must be greater than the default wrong way minimum width (if not defined, use right way minimum width), in order for the spacing in the second and beyond columns to be applied. If edge length condition is not met, the spacing in the first column will be applied independent of parallel run length. This behavior will also work for a combination of a `PARALLELRUNLENGTH` `WRONGDIRECTION` and `TWOWIDTHS` spacing tables. See [Figure 1-277](#) on page 500.

### Spacing Table Rule Examples

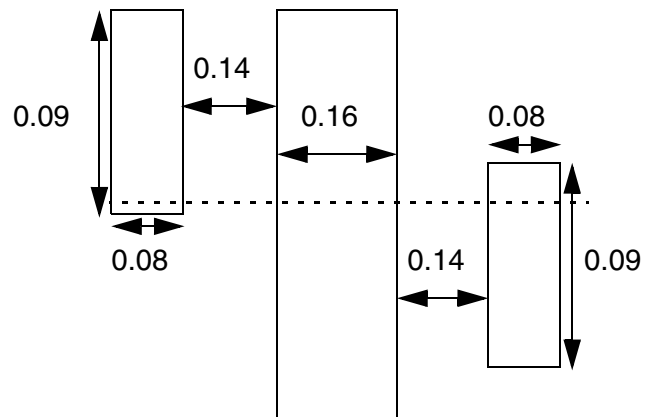
- [Figure 1-273](#) on page 495 illustrates the following spacing table rules with `SPANLENGTH` 0, 0.10, and 0.20:

```
PROPERTY LEF58_SPACINGTABLE
"SPACINGTABLE
PARALLELSPANLENGTH PRL 0.15
SPANLENGTH      0      0.10    0.15    0.20
SPANLENGTH      0.10   0.15    0.17    0.23
SPANLENGTH      0.20   0.20    0.23    0.25 ; " ;
```

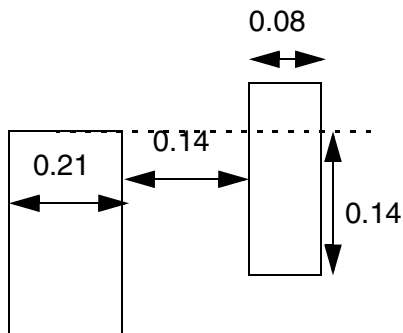
Figure 1-273 Spacing Table Rule Example



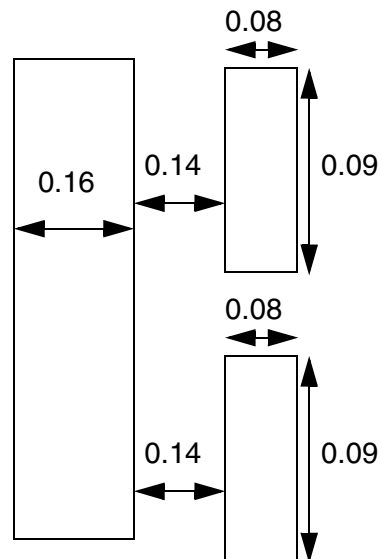
a) Violation, the continuous parallel run length of  $0.1 + 0.06 = 0.16 (> 0.15)$ , and top & bottom edges on the left have span length  $> 0.1$ , 0.15 spacing is needed



b) OK, parallel run length does not count for opposite sides



c) OK, parallel run length  $\leq 0.15$

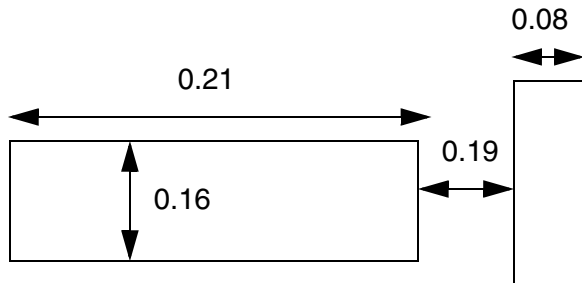


d) OK, parallel run length is not continuous, and each of the individual ones of  $0.09 \leq 0.15$  would not trigger the rule

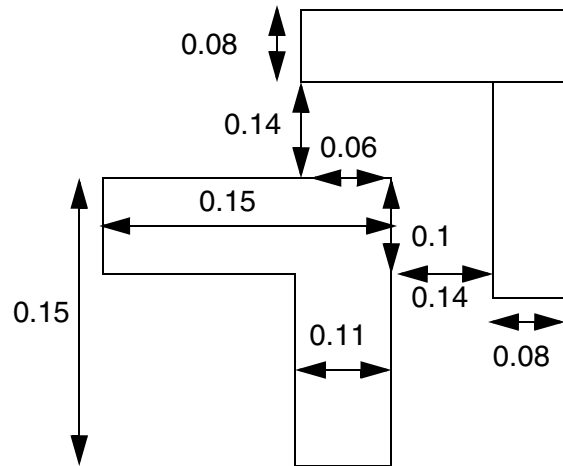
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



e) Violation, it can happen in EOL when the conditions are fulfilled. With span length of 0.21, 0.2 spacing is needed



f) OK, the top & side parallel run length is calculated separately, and each of them  $\leq 0.15$



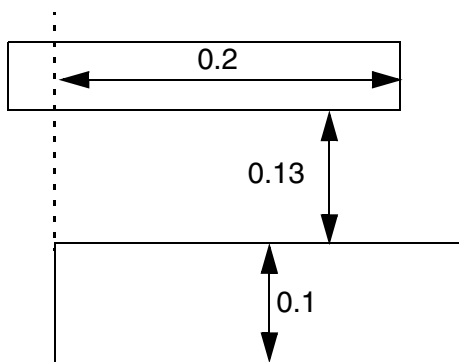
## LEF/DEF 5.8 Language Reference

### LEF Syntax

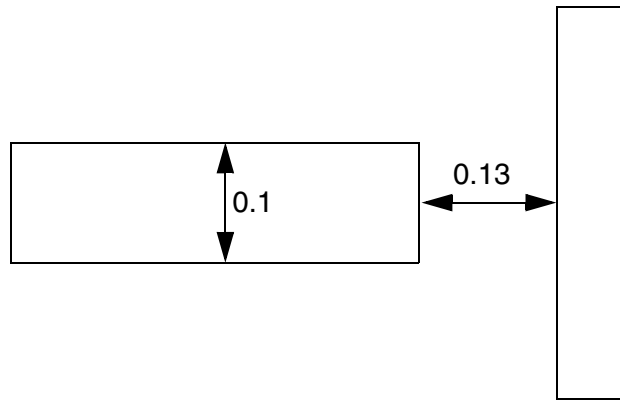
---

**Figure 1-274 Illustration of Spacing Table Rule with WRONGDIRECTION**

```
DIRECTION VERTICAL ;  
PROPERTY LEF58_SPACINGTABLE  
  "SPACINGTABLE  
    PARALLELRUNLENGTH WRONGDIRECTION 0.0 0.08  
    WIDTH 0.0 0.05 0.05  
    WIDTH 0.09 0.05 0.14 ; " ;
```



a) Violation, 0.14 spacing is needed



b) Violation, 0.14 spacing is also needed on the side

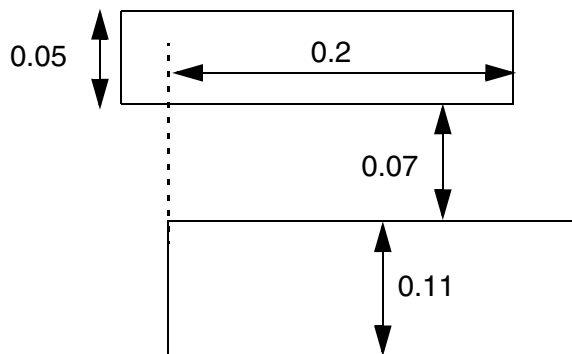
# LEF/DEF 5.8 Language Reference

## LEF Syntax

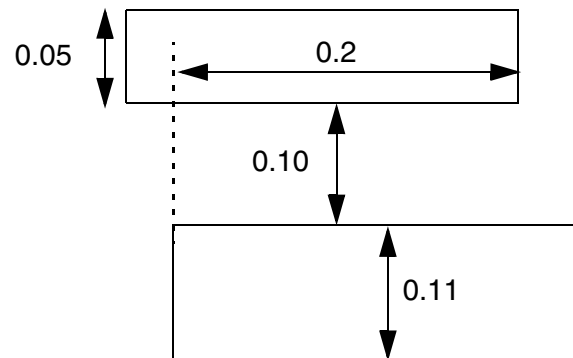
**Figure 1-275 Illustration of Spacing Table Rule with EXCEPTWITHIN**

```

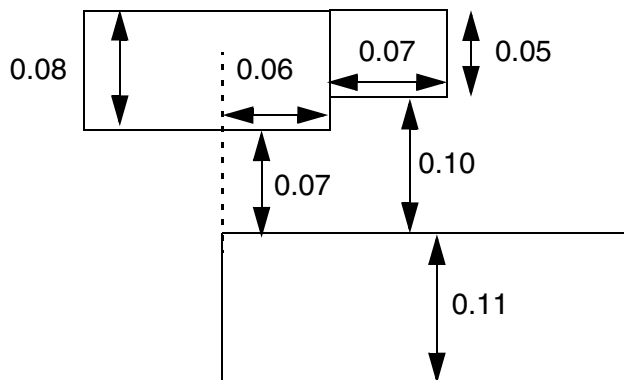
PROPERTY LEF58_SPACINGTABLE
"SPACINGTABLE
  PARALLELRUNLENGTH -0.041 -0.04 0.08
  WIDTH 0.0          0.05  0.06 0.06
  WIDTH 0.1 EXCEPTWITHIN 0.07 0.10 0.05 0.06 0.14 ; " ;
  
```



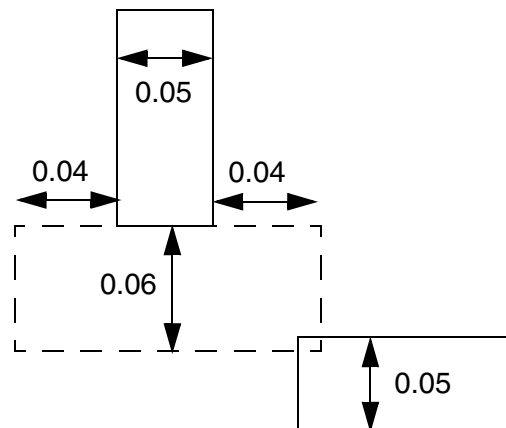
a) OK, the wire is within the exception range, and is fine.



b) Violation, the wire is barely outside the exception range, which needs spacing of 0.14.

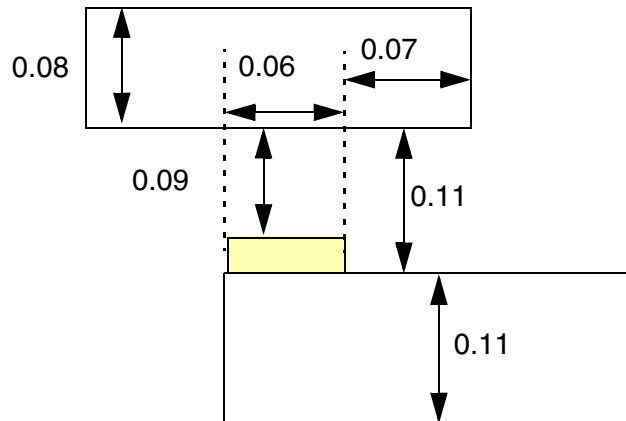


c) OK, PRL of the portion of wire outside the exception range is 0.07, which would not trigger the rule



d) Violation, the neighbor wire is within 0.04, and although the corner to corner spacing would be  $> 0.06$ , the spacing is checked in MAXXY style, which would fail.

**Figure 1-276 Illustration of Spacing Table Rule with EXCEPTWITHIN**

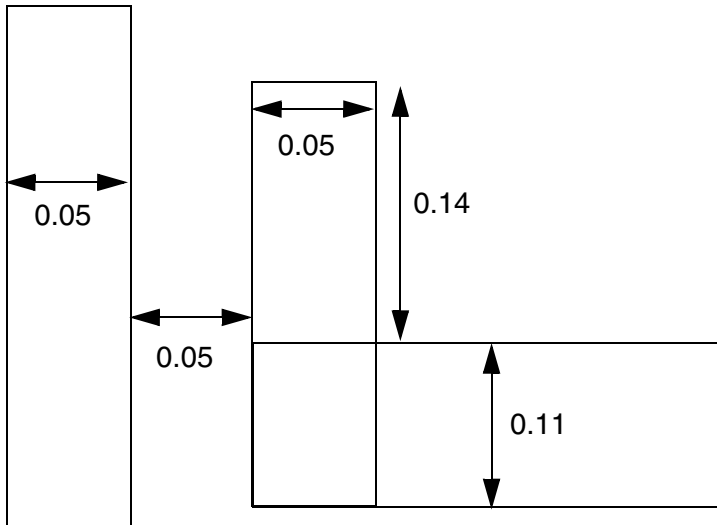


e) Violation, PRL measurement should be  $0.06 + 0.07 = 0.13$  ( $> 0.08$ ) because after the shrink & grow operation to determine the wide wire width, the yellow portion of the jog will be gone, and the spacing between the 2 objects will be 0.11 for the entire 0.13 length, which is a violation ( $< 0.14$ ).

■ The following example illustrates PARALLELRUNLENGTH with WRONGDIRECTION:

```
DIRECTION VERTICAL ;
PROPERTY LEF58_WIDTH
    "WIDTH 0.12 WRONGDIRECTION ;
SPACINGTABLE
    PARALLELRUNLENGTH      0.00      0.07
    WIDTH 0.0               0.05      0.05
    WIDTH 0.1               0.05      0.08 ; " ;
PROPERTY LEF58_SPACINGTABLE
    "SPACINGTABLE
        PARALLELRUNLENGTH WRONGDIRECTION ... ; " ;
```

**Figure 1-277 Illustration of PARALLELRUNLENGTH with WRONGDIRECTION**



a) OK, the entire left edge of the right wire is 0.25, but edge length with width > 0.1 is only 0.11 ( $\leq 0.12$ , default wrong direction minimum width). Hence, 0.05 spacing is needed and is met.

### ***EOL Extension Spacing Rule***

EOL extension spacing rule can be used to indicate that for a given width of an end-of-line certain extension should be applied to the EOL edge before checking for edge-to-edge spacing to any neighbor wires.

You can define a EOL extension spacing rule by using the following property definition:

```
PROPERTY LEF58_EOLEXTENSIONSPACING
    "EOLEXTENSIONSPACING spacing [SAMEMASK]
        [PARALLELONLY] [NONEOL]]
        [OTHERWIDTH otherWidth]
        {ENDOFFLINE eolWidth EXTENSION extension
            [ENDTOEND endToEndExtension]} ...
        [MINLENGTH minLength [TWO SIDES]]
    ;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

EOLEXTENSIONSPACING *spacing* {ENDOFLINE *eolWidth* EXTENSION  
*extension* [ENDTOEND *endToEndExtension*]} ...

Specifies that for a given width of an end-of-line, find the last row where the width is less than *eolWidth*, the corresponding extension should be applied to the EOL edge before checking for edge-to-edge spacing to any neighbor wires. The *eolWidth* values should be increased for each subsequent ENDOFLINE statement, and at the most three statements can be specified and supported.

*Type:* Float, specified in microns

ENDTOEND *endToEndExtension* specifies that *endToEndExtension* is applied if the neighbor wire is also an end-of-line, and extension is applied for an end-to-side situation.

*Type:* Float, specified in microns

MINLENGTH *minLength* [TWO SIDES]

Indicates that the rule does not apply if the end-of-line length is less than *minLength* along both sides.

*Type:* Float, specified in microns

TWO SIDES means that the rule only applies when the end-of-line length is greater than or equal to *minLength* along both the sides. In other words, the rule does not apply if the end-of-line length is less than *minLength* along any one side.

OTHERWIDTH *otherWidth*

Specifies that the rule applies only if the width of the neighbor wire is less than *otherWidth*. See [Figure 1-279](#) on page 504.

PARALLELONLY [NONEOL]

Specifies that the rule only applies if a facing neighbor parallel edge to the EOL edge is found. If NONEOL is specified, the neighbor parallel edge cannot be a EOL. See [Figure 1-280](#) on page 504.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

SAMEMASK

Specifies that the EOL extension spacing only applies to same-mask objects.

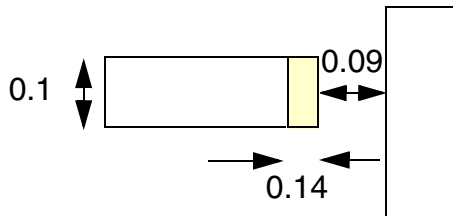
**Note:** Multiple EOLEXTENSIONSPACING statements with SAMEMASK can be defined.

### EOL Extension Spacing Rule Examples

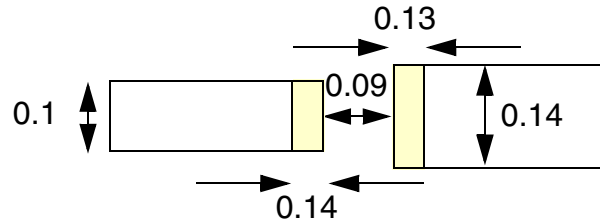
- The following EOL extension spacing rule indicates that an end-of-line edge with width less than 0.11  $\mu\text{m}$  should have 0.14  $\mu\text{m}$  extension, and an end-of-line edge with width less than 0.15  $\mu\text{m}$  and greater than or equal to 0.11  $\mu\text{m}$  should have 0.13  $\mu\text{m}$  extension in end-to-end situation or 0.12  $\mu\text{m}$  extension in end-to-side situation. Then 0.1  $\mu\text{m}$  edge to edge spacing must be enforced with proper extension on the end-of-line edge(s).

```
PROPERTY LEF58_EOLEXTENSIONSPACING
    "EOLEXTENSIONSPACING 0.1 ENDOFLINE 0.11 EXTENSION 0.14
    ENDOFLINE 0.15 EXTENSION 0.12 ENDTOEND 0.13 ;" ;
```

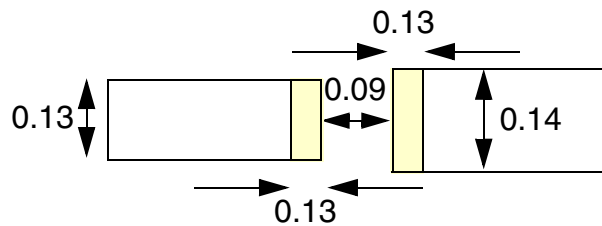
**Figure 1-278 EOL Extension Spacing Rule Example**



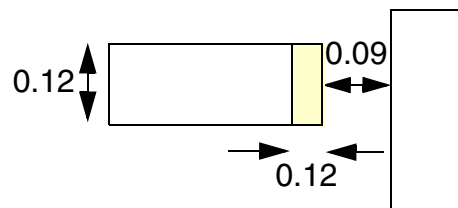
a) Violation, 0.14 extension is applied for width < 0.11, and 0.1 spacing is required after the extension



b) Violation, for end-to-end situation, 0.14 extension is applied for width < 0.11 on the left, 0.13 extension is applied for width  $\geq 0.11$ , and 0.1 spacing is required

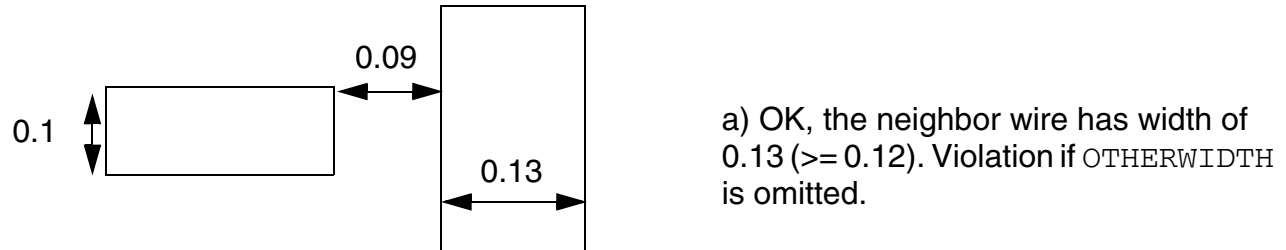


c) Violation, for end-to-end situation, 0.13 extension is applied for both wires with width  $\geq 0.11$ , and 0.1 spacing is required



d) Violation, 0.12 extension is applied for width  $\geq 0.11$  for end-to-side, and 0.1 spacing is required

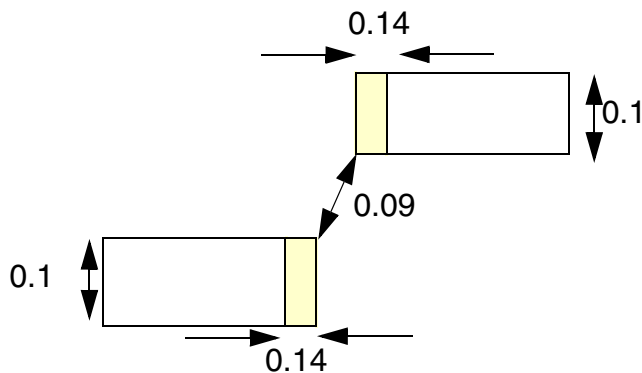
**Figure 1-279 EOL Extension Spacing Rule with OTHERWIDTH**



Example of PROPERTY LEF58\_EOLEXTENSIONSPACING "  
 EOLEXTENSIONSPACING 0.1  
 OTHERWIDTH 0.12  
 ENDOFLINE 0.11 EXTENSION 0.00 ; " ;

**Figure 1-280 EOL Extension Spacing Rule with NONEOL**

PROPERTY LEF58\_EOLEXTENSIONSPACING "  
 EOLEXTENSIONSPACING 0.1  
 PARALLELONLY NONEOL  
 ENDOFLINE 0.11 EXTENSION 0.14 ; " ;



a) OK, the neighbor edge must not be a EOL.  
 Violation if NONEOL is omitted.



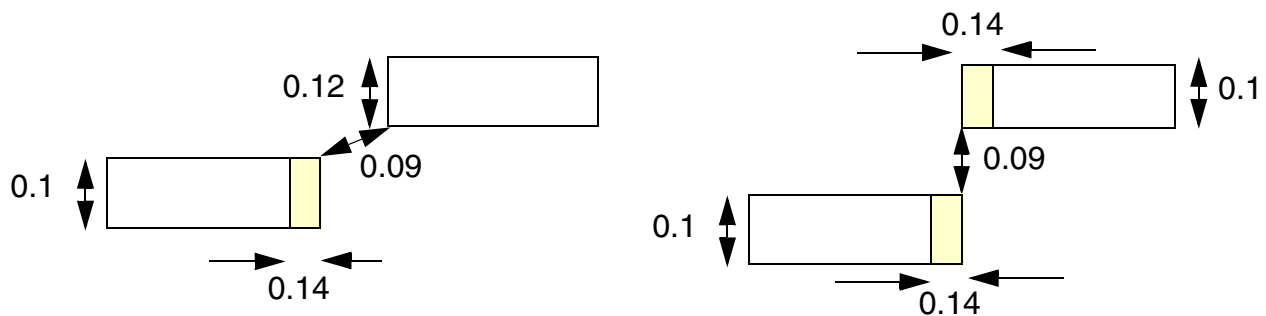
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- The following EOL extension spacing rule indicates that if a facing neighbor parallel edge to the EOL edge is found, then without `PARALLELONLY` it will be a violation:

```
PROPERTY LEF58_EOLEXTENSIONSPACING
  "EOLEXTENSIONSPACING 0.1 PARALLELONLY
  ENDOFLINE 0.11 EXTENSION 0.14 ; " ;
```

**Figure 1-281 EOL Extension Spacing Rule Example**



a) Violation, after extending 0.14, the EOL edge to a parallel (vertical) neighbor edge spacing is 0.09 ( $< 0.1$ )

b) OK, there is no facing parallel (vertical) neighbor edge. Violation if `PARALLELONLY` is omitted.

### AC Current Density Rule

The AC density rule can be used to specify how much AC current a wire on a layer of a certain width can handle at a certain frequency in units of milliamps per micron (mA/mm).

You can define a AC current density rule by using the following property definition:

```
PROPERTY LEF58_ACCURRENTDENSITY
  "ACCURRENTDENSITY {PEAK | AVERAGE | RMS}
  [TEMPPWL temp_1 multi_1 temp_2 multi_2 ...]
  [HOURSPWL hours_1 multi_1 hours_2 multi_2 ...]
  ; " ;
```

All other keywords are the same as the existing LEF routing layer `ACCURRENTDENSITY` syntax.

Note that the property statement is a compliment to the original syntax, which must be specified first.

Where:

```
HOURLPWL hours_1 multi_1 hours_2 multi_2 ...
```

Specifies a multiplier table based on the hours of operation. When the operation hour is greater than or equal to a certain hour in the table, the corresponding multiplier is used to scale the current by multiplying the multiplier with linear interpolation in the current calculation.

*Type:* Float

```
TEMPPWL temp_1 multi_1 temp_2 multi_2 ...
```

Specifies a multiplier table based on the temperature in Celsius. When the operation temperature is  $\geq$  certain temperature in the table, the corresponding multiplier is used to scale the current by multiplying the multiplier with linear interpolation in the current calculation.

*Type:* Float

## AC Current Density Rule Examples

The following AC current density rule indicates that the RMS table entry values will be defined for 100C and 10000 hours since the corresponding multipliers are 1.0 for those values. At 80C and 15000 hours, the multiplier is  $1.5 \times 0.8 = 1.2$ . Hence, the wires can carry 1.2x times the RMS table values.

```
PROPERTY LEF58_ACCURRENTDENSITY
    "ACCURRENTDENSITY RMS
        TEMPPWL 50 2.2 80 1.5 100 1.0 120 0.75
        HOURLPWL 5000 1.8 10000 1.0 15000 0.8 ; " ;
```

## Antenna Gate PWL Rule

The antenna gate PWL rule can be used to define a PWL (piece-wise linear) table on the given antenna model that is indexed by the real gate area, and returns an “effective gate-area” interpolated from the table.

You can define antenna gate PWL rule by using the following property definition:

```
PROPERTY LEF58_ANTENNAGATEPWL
    "ANTENNAGATEPWL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}
        ((gateArea1 effectiveGateArea1) (gateArea2 effectiveGateArea2) ... )
    ;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

**ANTENNAGATEPWL** Defines the PWL table that is indexed by the real gate area, and returns an "effective gate-area" interpolated from the table.

*(gateArea1 effectiveGateArea1) (gateArea2 effectiveGateArea2)*

The *effectiveGateArea* is used rather than the real gate area in the PAR equation. If the PWL table is not defined, then the real gate area is used.

*Type:* Float, specified in units of square microns.

### **Antenna Gate Plus Diffusion Rule**

The antenna gate plus diffusion rule can be used for representing protection provided by the diffusion area that is added to the gate-area value in the PAR equation on the given antenna model, which can be considered as "additional effective gate-area".

You can define the antenna gate plus diffusion rule by using the following property definition:

```
PROPERTY LEF58_ANTENNAGATEPLUSDIFF
    "ANTENNAGATEPLUSDIFF {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}
        {plusDiffFactor |
            PWL ((diffArea1 plusDiffProtect1) (diffArea1 plusDiffProtect2)
                ...)} ;" ;
```

All other keywords are the same as the existing LEF routing layer ANTENNAGATEPLUSDIFF syntax.

Where:

**ANTENNAGATEPLUSDIFF**

Indicates the antenna gate area plus diffusion area.

*plusDiffFactor | PWL ((diffArea1 plusDiffProtect1) (diffArea1 plusDiffProtect2) ...)*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates protection provided by the diffusion area that is added to the gate area value in the PAR equation, which can be considered as "additional effective gate-area". The PAR equation is as follows:

$$\text{PAR} = \{(\text{metalFactor} * \text{metal}) * \text{diffReducePWL}(\text{diff}) - (\text{minusDiffFactor} * \text{diff})\} / \{\text{gatePWL}(\text{gate}) + \text{plusDiffProtect}(\text{diff})\}$$

If *plusDiffFactor* value is specified, then:

$$\text{plusDiffProtect} = \text{plusDiffFactor} * \text{diffusion area}$$

If the PWL table is defined, then:

$$\text{plusDiffProtect} = \text{interpolated PWL value indexed by diffusion area.}$$

If PAR is greater than the *metalGateRatio* (from ANTENNAAREARATIO), and no diffusion is connected then there will be a violation.

If PAR is greater than *metalGateDiffRatio* (from ANTENNADIFFAREARATIO), then there will be a violation. This is checked even if the diffusion area is equal to 0 (that is, no diffusion is connected).

The *diffArea* and *plusDiffProtect* values are floats in units of square microns.

### Antenna Gate Plus Diffusion Rule Examples

The following rule shows antenna gate plus diffusion rule for metal1:

```
metal_area <= 100 * (funcGate(gate_area) + funcDiff(diff_area))
funcGate = 1 for gate_area < 1, and equals 2 for gate_area >= 1
funcDiff = 10 for diff_area < 2, and equals 50 for diff_area >= 2.
```

Then

```
100 >= metal_area / (funcGate(gate) + funcDiff(diff))
```

This fits the PAR formula above using *gatePWL(gate)* and *plusDiffProtect(diff)* values.

Then the LEF file will have the following:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
#Max-ratio = 100 for all diff-area values because the diff-protect value is
#accounted for inside the PAR equation. Note, there is no need for ANTENNAAREARATIO
# which is only checked for metal with no diff connected, while ANTENNADIFFAREARATIO
#is checked for diff and no diff connected.
```

```
ANTENNAMODEL OXIDE1 ;
```

```
ANTENNADIFFAREARATIO ( 0 100 ) ( 10000.0 100 ) ;
```

```
#This models funcGate(gate_area) above: 0 to 0.99 is 1, above 1.0 is 2
```

```
PROPERTY LEF58_ANTENNAGATEPWL OXIDE1
```

```
"ANTENNAGATEPWL ( ( 0 1 ) ( 0.99 1 ) ( 1.0 2 ) ( 10000.0 2 ) ) ;" ;
```

```
#This models funcDiff(diff_area) above: 0 to 1.99 is 10, above 1.0 is 50
```

```
PROPERTY LEF58_ANTENNAGATEPLUSDIFF OXIDE1
```

```
"ANTENNAGATEPLUSDIFF
```

```
( ( 0 10 ) ( 1.99 10 ) ( 2.0 50 ) ( 10000.0 50 ) ) ;" ;
```

### ***Routing Pitch***

The `PITCH` statements define the detail routing grid generated when you initialize a floorplan. The pitch for a given routing layer defines the distance between routing tracks in the preferred direction for that layer. The complete routing grid is the union of the tracks generated for each routing layer.

The spacing of the grid should be no less than line-to-via spacing in both the horizontal and vertical directions. Grid spacing less than line-to-via spacing can result in routing problems and can decrease the utilization results.

The grid should normally allow for diagonal vias. Via spacing on all layers included in the via definition in LEF determines whether or not diagonal vias can be used. The router is capable of avoiding violations between diagonal vias. If you allow diagonal vias, less time is needed for routing and the layout creates a smaller design.

## Library

### Defining Library Properties to Create 32/28 nm and Smaller Nodes Rules

You can include library properties in your LEF file to create 32/28 nm and smaller nodes rules that currently are not supported by existing LEF syntax. The properties are specified inside the PROPERTYDEFINITIONS statements.

- The property names used for these rules all start with LEF58\_.

All properties use the following syntax within the LEF PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
    LAYER propName STRING ["stringValue"] ;
END PROPERTYDEFINITIONS
```

The property definitions for the library properties are as follows:

```
PROPERTYDEFINITIONS
    LIBRARY LEF58_ANTENNAMAXCUMAREA STRING ;
    LIBRARY LEF58_CONSTRAINTLENGTH STRING ;
    LIBRARY LEF58_CELLEDGESPACINGTABLE STRING ;
    LIBRARY LEF58_FINFET STRING ;
    LIBRARY LEF58_IMPLANTGROUP STRING ;
    LIBRARY LEF58_LAYERMASKSHIFT STRING ;
    LIBRARY LEF58_MAXFLOATINGAREA STRING ;
    LIBRARY LEF58_MAXVIASTACK STRING ;
    LIBRARY LEF58_METALWIDTHTRACK STRING ;
    LIBRARY LEF58_METALWIDTHVIAMAP STRING ;
    LIBRARY LEF58_OALAYERMAP STRING ;
    LIBRARY LEF58_PGVIATRACK STRING ;
    LIBRARY LEF58_STACKVIALAYERRULE STRING ;
    LIBRARY LEF58_STACKVIARULE STRING ;
    LIBRARY LEF58_TRIMMETALTRACK STRING ;
END PROPERTYDEFINITIONS
```

### **Antenna Maximum Cumulative Area Rule**

You can create an antenna maximum cumulative area rule to specify the maximum cumulative area of all layers to gate, which is not connected to the diffusion diode.

You can define an antenna maximum cumulative rule by using the following PROPERTYDEFINITIONS statement:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTYDEFINITIONS
LIBRARY LEF58 ANTENNAMAXCUMAREA STRING
    "ANTENNAMAXCUMAREA value
        ;" ;
END PROPERTYDEFINITIONS
```

Where:

ANTENNAMAXCUMAREA *value*

Specifies the maximum possible cumulative area of all layers, which is not connected to the diffusion diode, to be less than or equal to *value*. This would include floating metal that does not necessarily connect to a gate yet. The gate connection is done through a metal layer above the floating metal.

*Type:* Float, specified in microns

### **Constraint Length Rule**

You can create a constraint length rule to define the horizontal and vertical length of the concatenated constraint area. You can define a constraint length rule by using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 CONSTRAINTLENGTH STRING
"CONSTRAINTLENGTH length {MAX|MIN} [HORIZONTAL|VERTICAL]
    CONSTRAINTAREATYPE typeName
        ;" ;
END PROPERTYDEFINITIONS
```

Where:

CONSTRAINTLENGTH *length* {MAX|MIN}  
CONSTRAINTAREATYPE *typeName*

Specifies the horizontal and vertical length of the concatenated constraint area with type *typeName* defined in CONSTRAINTAREATYPE in MACRO must be less than *length* in MAX or greater than *length* in MIN.

*Type:* Float, specified in microns

HORIZONTAL | VERTICAL Specifies that the constraint length rule applies only on the given direction of either HORIZONTAL or VERTICAL.

### ***Cell Edge Spacing Table Rule***

You can create a cell edge spacing table rule to define a spacing table between two macro edges having different edge types.

This cell edge spacing table must be defined before reading in the `MACRO` definition with defined `EDGETYPE` in cell LEF files.

You can define a cell edge spacing table rule by using the following `PROPERTYDEFINITIONS` statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58_CELLEDGESPACINGTABLE STRING
    "CELLEDGESPACINGTABLE [NODEFAULT]
        {EDGETYPE {edgeType1 | DEFAULT}
        {edgeType2 | DEFAULT} [EXCEPTABUTTED]
            [OPTIONAL | SOFT | EXACT] spacing} ...
        ;..." ;
END PROPERTYDEFINITIONS
```

Where:

```
CELLEDGESPACINGTABLE
{EDGETYPE {edgeType1 | DEFAULT}
{edgeType2 | DEFAULT} spacing}...
```

Defines a spacing table between a macro edge with type *edgeType1* and another macro edge with type *edgeType2*. The `DEFAULT` keyword represents macro edge without a type. If there is no spacing defined between two types, it indicates that there is no spacing constraint (or 0.0  $\mu\text{m}$ ) between them.  
*Type*: Float, specified in microns

`EXACT` Specifies that it is a violation only when the spacing is exactly equal to *spacing*.

`EXCEPTABUTTED` Specifies that two abutted macros of the given edge types are also allowed.

`NODEFAULT` Specifies that `DEFAULT` should not be defined in the spacing table, and all the edges of any standard cell macros (with a `SITE` that is `CLASS CORE`) must have `EDGETYPE` property defined on both sides.

`OPTIONAL` Specifies that the corresponding spacing constraints are optional. Users can use certain placement option to indicate whether those constraints should be observed or not.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

SOFT	Specifies that the corresponding spacing constraints are soft. This means that the tool has the freedom to honor the constraint with certain efforts. If the constraint will disturb the placement quality too much, the tool can choose to ignore the constraint.
------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### Cell Edge Spacing Table Rule Example

■ The following rule indicates that:

- ❑ 0.1  $\mu\text{m}$  is the spacing between a macro edge with type `GROUP1` and a macro edge with type `GROUP2`. Abutting the two macros is also allowed.
- ❑ 0.2  $\mu\text{m}$  is the optional spacing between two macro edges with type `GROUP1`; a certain placement option controls whether or not it is honored.
- ❑ 0.3  $\mu\text{m}$  is the spacing between a macro edge with type `GROUP2` and a macro edge without a type.
- ❑ 0.4  $\mu\text{m}$  is the soft spacing between two macro edges with type `GROUP2`, which would be ignored if placement quality is impacted too much.
- ❑ 0.5  $\mu\text{m}$  spacing is not allowed between a macro edge with type `GROUP1` and a macro edge without a type.
- ❑ The rest of the combination of edges, `DEFAULT` to `DEFAULT`, do not have any spacing constraint, or 0.0  $\mu\text{m}$ .

```
PROPERTYDEFINITIONS
  LIBRARY LEF58_CELLEDGESPACINGTABLE STRING
    "CELLEDGESPACINGTABLE
      EDGETYPE GROUP1 GROUP2 EXCEPTABUTTED 0.1
      EDGETYPE GROUP1 GROUP1 OPTIONAL 0.2
      EDGETYPE GROUP2 DEFAULT 0.3
      EDGETYPE GROUP2 GROUP2 SOFT 0.4; ";
      EDGETYPE GROUP1 DEFAULT EXACT 0.5; ";
    END PROPERTYDEFINITIONS
```

### Finfet Rule

You can create a finfet rule to specify the finfet pitch to be pitch.

You can define a finfet rule by using the following `PROPERTYDEFINITIONS` statement:

```
PROPERTYDEFINITIONS
  LIBRARY LEF58_FINFET STRING
    "FINFET
      PITCH pitch [OFFSET offset] {HORIZONTAL|VERTICAL}
        [NOCORECELL]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
    ;" ;  
END PROPERTYDEFINITIONS
```

Where:

```
FINFET  
    PITCH pitch [OFFSET offset] {HORIZONTAL | VERTICAL}
```

Specifies the FinFET pitch to be *pitch*, which starts at the *offset*, if specified, or zero from the origin of the design. The **HORIZONTAL/VERTICAL** keywords mean that the FinFET pitch is used to define the legal Y/X values of the placement grid that all cells, blocks, and IOs must align to (specifically, the origin of every cell must be aligned to the legal Y/X values), in order to guarantee all the FinFETs inside are aligned properly. The X/Y value of the placement grid is derived from the standard cell site width and only applies to standard cells. The cell row height should typically be multiple of the FinFET pitch.

*Type:* Floats, specified in microns

```
NOCORECELL
```

Specifies that the FinFet grid does not apply to standard cells with **MACRO** with **CLASS CORE**.

### Finfet Rule Examples

The following example indicates that the FinFET y pitch is 0.108  $\mu\text{m}$ :

```
PROPERTYDEFINITIONS  
LIBRARY LEF58_FINFET STRING "  
    FINFET  
        PITCH 0.108 HORIZONTAL ; "  
END PROPERTYDEFINITIONS
```

### Implant Group Rule

You can create an implant group rule to specify rules that apply for all the shapes on an implant layer.

You can define an implant group rule by using the following **PROPERTYDEFINITIONS** statement:

```
PROPERTYDEFINITIONS  
LIBRARY LEF58_IMPLANTGROUP STRING  
    "IMPLANTGROUP groupName  
        LAYER {layerName}... BASEDLAYER basedLayerName
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
    ;" ;  
END PROPERTYDEFINITIONS
```

Where:

```
IMPLANTGROUP groupName  
    LAYER {layerName}... BASEDLAYER basedLayerName
```

Specifies that the combined shapes on all of the given layers in *layerName*, which must be an implant layer, must follow all of the rules on *basedLayerName*.

### Implant Group Rule Example

```
PROPERTYDEFINITIONS  
LIBRARY LEF58 IMPLANTGROUP STRING "  
    IMPLANTGROUP GROUP1 LAYER VT1 VT2 BASEDLAYER VT1 ;  
    IMPLANTGROUP GROUP2 LAYER VT1 VT3 BASEDLAYER VT1 ; "  
END PROPERTYDEFINITIONS
```

In this example, the combined shapes on VT1 and VT2 and the combined shapes on VT1 and VT3 must follow the rules on VT1.

### Layer Mask Shift Rule

You can create a layer mask shift rule to specify that mask could be shifted on the given layers.

You can define a layer mask shift rule by using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS  
LIBRARY LEF58 LAYERMASKSHIFT STRING  
"LAYERMASKSHIFT layer1 [layer2 ...]  
;" ;  
END PROPERTYDEFINITIONS
```

Where:

```
LAYERMASKSHIFT layer1 [layer2 ...]
```

Specifies that mask could be shifted on the given layers. It should be defined and used along with FIXEDMASK in the library section or on individual MACRO. If LAYERMASKSHIFT is not specified, mask-shifting is not allowed on any layers.  
*Type:* String

### **Layer Mask Shift Rule Example**

The following example indicates that mask-shifting is allowed on LEF layer M1 and VIA1 but not on other layers:

```
PROPERTYDEFINITIONS
LIBRARY LEF58_LAYERMASKSHIFT STRING "
    LAYERMASKSHIFT M1 VIA1 ; " ;
END PROPERTYDEFINITIONS
```

### **Maximum Floating Area Rule**

Maximum area rules exist for floating metal shapes that are not connected to a diffusion or polysilicon gate. Similar to process antenna rules, maximum floating area rules apply only to the current layer and any lower layers (that is, all layers that have been fabricated up to the layer of interest). Maximum floating area rules can be used to avoid shorts between floating and non-floating metal wires that are caused by arcing due to charge buildup during processing steps.

You can create a global maximum floating area rule by using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58_MAXFLOATINGAREA STRING
"MAXFLOATINGAREA maxArea
    {SINGLELAYER | CONNECTED | ALLCONNECTED} minRoutingLayer maxRoutingLayer
    [[LAYERS minRoutingLayer maxRoutingLayer]
        SPACING minSpacing [PARSPACING minParSpacing minParallelLength ...]] ...
;" ;
END PROPERTYDEFINITIONS
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

`ALLCONNECTED minRoutingLayer maxRoutingLayer`

Indicates that the rule applies to the connected area of floating metal shapes connected together on each layer between *minRoutingLayer* and *maxRoutingLayer* inclusive. The total connected area on the current and lower layers must be less than or equal to *maxArea*, or meet the minimum spacing to other grounded shapes on this layer and all lower-connected layer shapes. The names you specify for *minRoutingLayer* and *maxRoutingLayer* must be two previously defined LEF routing layers.

`CONNECTED minRoutingLayer maxRoutingLayer`

Indicates that the rule applies to the area of floating metal shapes connected together on each layer between *minRoutingLayer* and *maxRoutingLayer* inclusive. The connected area on a current layer must be less than or equal to *maxArea*, or meet the minimum spacing to other grounded shapes on this layer and all lower connected layer shapes. The names you specify for *minRoutingLayer* and *maxRoutingLayer* must be two previously defined LEF routing layers.

`LAYERS minRoutingLayer maxRoutingLayer`

Indicates that layers between *minRoutingLayer* and *maxRoutingLayer* inclusive must meet the specified SPACING rules. The names you specify for *minRoutingLayer* and *maxRoutingLayer* must be previously defined LEF routing layers, and can be the same layer.

`MAXFLOATINGAREA maxArea`

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Indicates that floating metal shapes must have a total area that is less than or equal to *maxArea*.

*Type:* Float, specified in units of microns squared

Floating metal is defined as metal on the current layer that cannot trace a path to a diffusion connection or polysilicon gate, using only same-layer or lower-layer metal connections.

Grounded metal is defined as metal that can connect to a diffusion connection or polysilicon gate using only same-layer or lower-layer connections.

**Note:** The MAXFLOATINGAREA rule depends on the existing LEF MACRO PIN ANTENNADIFFAREA and ANTENNAGATEAREA statements to indicate which pins are connected to diffusion or polysilicon gates.

You can have two MAXFLOATINGAREA statements: one for SINGLELAYER, and one for CONNECTED or ALLCONNECTED.

`PARSPACING minParSpacing minParallelLength ...`

Indicates that floating metal that is greater than or equal to *minParSpacing* from grounded metal must have greater than or equal to *minParallelLength* at the minimum spacing distance. If more than one pair of values is given, the smallest spacing value that matches is used. (See Example 2.)

*Type:* Float, specified in microns (for both values)

The *minParSpacing* values must be defined in decreasing value, and be smaller than `SPACING minSpacing`. The intent of the PARSPACING rule is to spread out the charge build up on floating shapes to reduce the chance of spark; therefore, smaller spacing values require larger parallel lengths in order to be legal.

If you define PARSPACING for the same layer more than once (due to overlapping LAYERS layer ranges) the last PARSPACING values overwrite the previous values.

`SINGLELAYER minRoutingLayer maxRoutingLayer`

## LEF/DEF 5.8 Language Reference

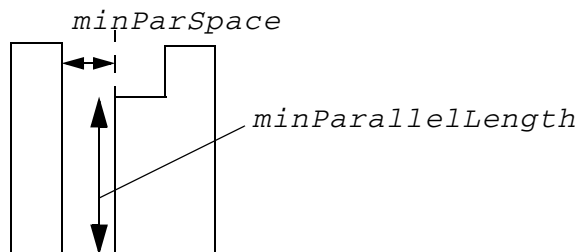
### LEF Syntax

---

Indicates the rule applies to each individual floating metal shape on any routing layer between *minRoutingLayer* and *maxRoutingLayer* inclusive. Each shape must have an area that is less than or equal to *maxArea*, or meet the minimum spacing to other grounded shapes. The names you specify for *minRoutingLayer* and *maxRoutingLayer* must be two previously defined LEF routing layers.

`SPACING minSpacing` Indicates that if the current layer floating metal has an area that is greater than *maxArea*, the floating metal on the current layer (and all connected lower layers with `CONNECTED` or `ALLCONNECTED`) must be greater than or equal to *minSpacing* distance from grounded metal for all layers that have `MAXFLOATINGAREA` rules. If you define `SPACING` for the same layer more than once (due to overlapping `LAYERS` layer ranges) the last `SPACING` value overwrites the previous values.  
*Type:* Float, specified in microns

Definition of *minSpace* and *minParallelLength* for `MAXFLOATINGAREA` rule.



### Maximum Floating Area Rule Examples

#### ■ Example 1

Assume the following rule exists:

```
PROPERTYDEFINITIONS
  LIBRARY LEF58_MAXFLOATINGAREA STRING
    "MAXFLOATINGAREA 1000 SINGLELAYER m1 m6 SPACING 1.0 ;" ;
END PROPERTYDEFINITIONS
```

Every shape on layers *m1* to *m6* with maximum floating area greater than 1000  $\mu\text{m}^2$  must have spacing of greater than or equal to 1.0  $\mu\text{m}$  to any grounded metal.

#### ■ Example 2

Assume the following rule exists:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTYDEFINITIONS
  LIBRARY LEF58 MAXFLOATINGAREA STRING
    "MAXFLOATINGAREA 1000 CONNECTED m1 m6
      LAYERS m1 m1 SPACING 1.0 PARSPACING 0.5 0.8 0.2 2.0
      LAYERS m2 m6 SPACING 0.6 PARSPACING 0.3 0.9 ;" ;
END PROPERTYDEFINITIONS
```

For layer *m1*, any floating metal must be either greater than or equal to 1.0  $\mu\text{m}$  distance from grounded metal, or:

- ❑ If it is greater than or equal to 0.5  $\mu\text{m}$  distance away, there must be at least 0.8  $\mu\text{m}$  of parallel length at the minimum spacing.
- ❑ If it is greater than or equal to 0.2  $\mu\text{m}$  distance away, there must be at least 2.0  $\mu\text{m}$  of parallel length at the minimum spacing.

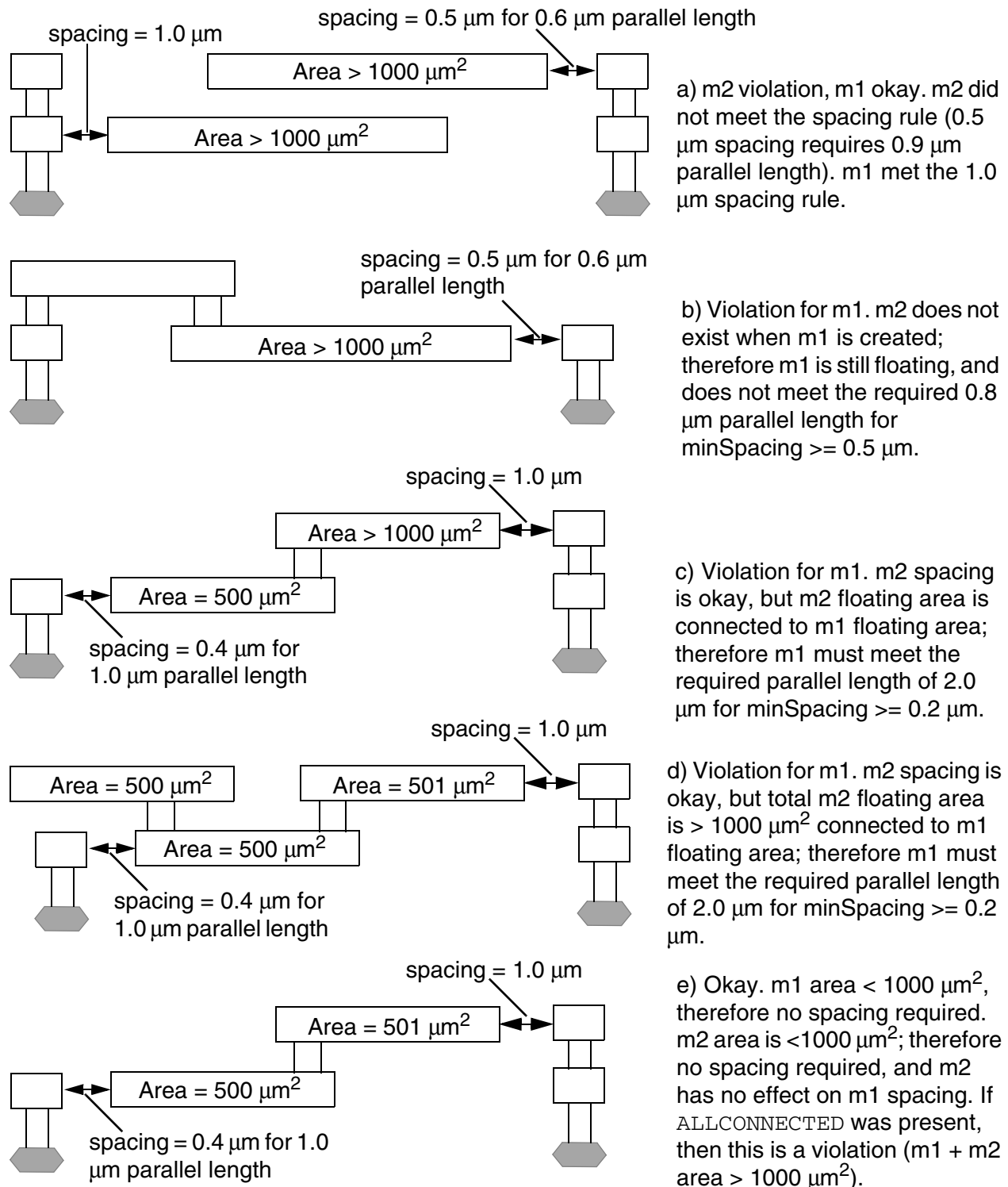
Spacing that is less than 0.2  $\mu\text{m}$  is not allowed.

Any floating *m2* through *m6* shapes with area that is greater than 1000  $\mu\text{m}^2$ , must either be greater than or equal to 0.6  $\mu\text{m}$  distance from grounded metal, or they must be greater than or equal to 0.3  $\mu\text{m}$  distance away with greater than or equal to 0.9  $\mu\text{m}$  of parallel length.

See [Figure 1-282](#) on page 521 for different layout examples using this rule.



Figure 1-282 Maximum Floating Area Rule



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### **Maximum Via Stack Rule**

You can create a maximum via stack rule to require a series of stacked vias to all be multi-cut vias. A via is considered to be in a stack with other vias if the cuts of all the vias partially overlap (the boolean AND of the cut layer shapes from every via in the stack is not empty). A multi-cut via interrupts the stack, unless the NOSINGLE keyword is specified.

If multiple MAXVIASTACK statements are defined, all of them must be fulfilled individually.

You can define a maximum via stack rule using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58_MAXVIASTACK STRING
"MAXVIASTACK maxStack [NOSINGLE] [MINMULTICUT numCut]
    [WITHIN {LOWCUTSIZE | {cutLayerName within}...}]
    [EXCEPTPGNET]
    [EXCEPTAREA individualArea sumArea]
    [RANGE bottomLayer topLayer]
    [[NOVIALIST {viaName}...]...
    |[NOCUTCLASSLIST {cutLayer cutClassName}...]...]
;" ;
END PROPERTYDEFINITIONS
```

Where:

EXCEPTAREA *individualArea sumArea*

Specifies that the rule does not apply if the metal shape containing the stacked vias has an area greater than or equal to *individualArea* on any layers except the bottom most and top most metal layer of the stacked vias or the union sum of those metal areas is greater than or equal to *sumArea*.  
*Type*: Float, specified in microns squared

EXCEPTPGNET

Specifies that the rule is exempted for power or ground net.

MAXSTACK *maxStack*

Specifies the maximum number of single-cut vias that are allowed on top of each other (that is, in one continuous stack).  
*Type*: Integer

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`MINMULTICUT` *numCut* Specifies the minimum multi-cut number to be greater than or equal to *numCut* in order to be a multi-cut exemption to a stacked via violation with number of vias greater than or equal to *maxStack*. If `NOSINGLE` is specified, all of the vias must have a cut number greater than or equal to *numCut* to pass the rule. By default, without `MINMULTICUT`, the multi-cut number is 2. Therefore, *numCut* must be greater than 2.

*Type:* Integer

`[NOCUTCLASSLIST {cutLayer cutClassName}...]...`

Specifies that the given cut classes, which belong to the given *cutClassName* on the given cut layer *cutLayer*, are not allowed for stacking although the number of cut classes specified in `NOCUTCLASSLIST` is less than or equal to *maxStack*.

*Type:* String

`[NOVIALIST {viaName}...]...`

Specifies that the given list of vias in `{viaName}...` is not allowed for stacking although the number of cut vias specified in `NOVIALIST` is less than or equal to *maxStack*. The via list should always start from a lowest cut layer and consecutively move up the cut layers, and they should be inside the metal layers in *RANGE* if defined. Wildcard asterisk (\*) is supported, which would map to all of the LEF predefined vias that match the wildcard. If it is used alone to represent all of the vias on a certain cut layer, the via name on previous or following cut layers would have enough information to indicate what layer the asterisk is referring to.

*Type:* String

`RANGE` *bottomLayer topLayer*

Specifies a range of routing layers for which the maximum stacked via rule applies. If you do not specify a range, the *maxStack* value applies for all routing layers. The *bottomLayer* and *topLayer* values are routing layer names. The specified *topLayer* layer must be above the layer specified for *bottomLayer*.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

NOSINGLE	Indicates that any single-cut via in a stack that is larger than <i>maxStack</i> is a violation, and multi-cut vias do not interrupt a stack. Therefore, any stack larger than <i>maxStack</i> must consist of all multi-cut vias.
WITHIN {LOWCUTSIZE   { <i>cutLayerName within</i> }...}	<p>Specifies that an object is considered as a stacked via if the upper adjacent layer cut is within the minimum cut size of multiple CUTCLASS definitions or <i>minWidth</i> value in WIDTH on the lower cut layer in LOWCUTSIZE or within <i>within</i> distance of a cut on the lower layer in <i>cutLayerName</i>. Touching the within search window is considered as stacked.</p> <p>Type: Float, specified in microns</p>

## Maximum Via Stack Rule Examples

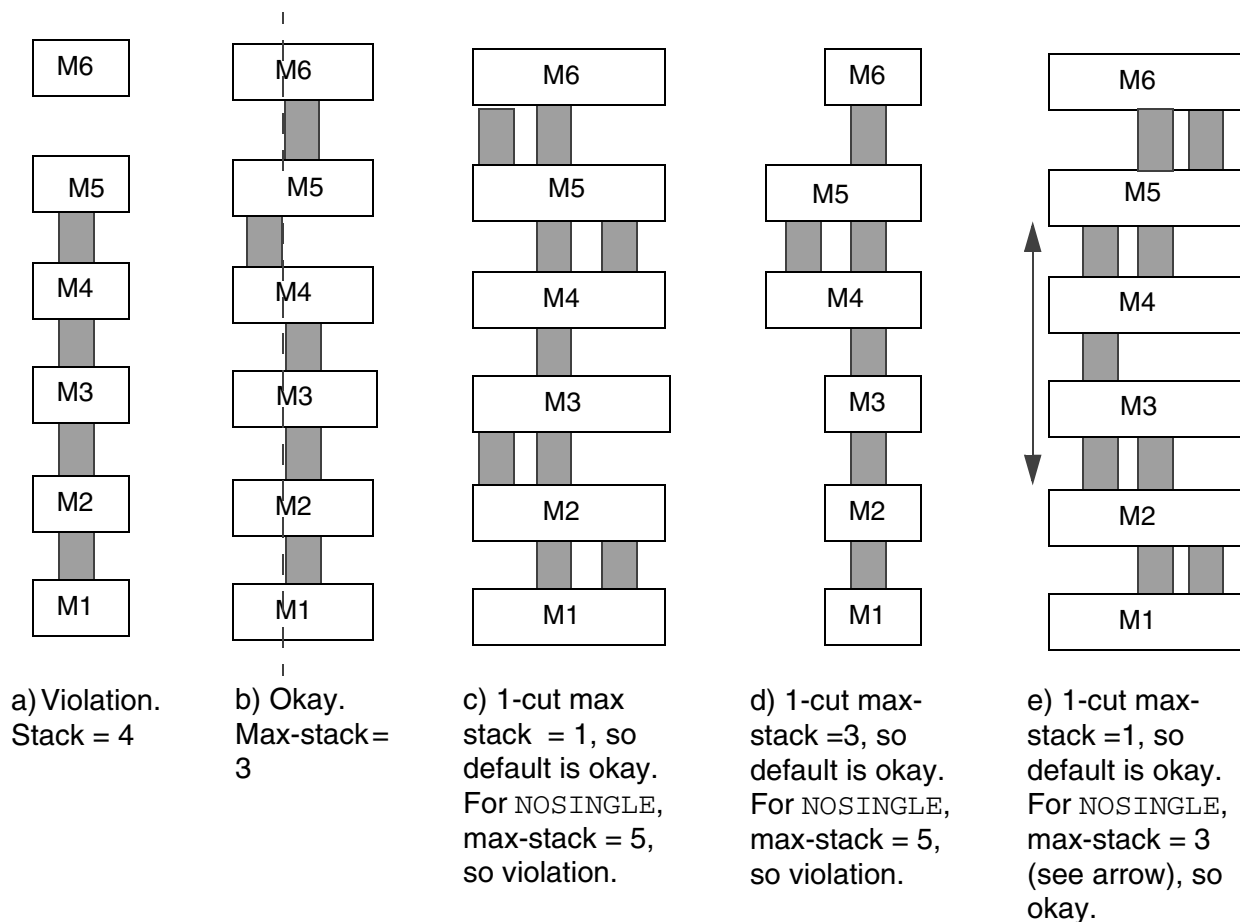
- If the following rule exists:

```
PROPERTYDEFINITIONS
  LIBRARY LEF58_MAXVIASTACK STRING
    "MAXVIASTACK 3 RANGE m1 m6 ;" ;
END PROPERTYDEFINITIONS
```

Only three single-cut vias can be stacked between layers *m1* and *m6*. See [Figure 1-283](#) on page 525 for different layout examples using this rule.

**Figure 1-283 Max Via Stack Rule Examples**

```
Library LEF58_MAXVIASTACK STRING "MAXVIASTACK 3 RANGE m1 m6 ;" ;
Library LEF58_MAXVIASTACK STRING "MAXVIASTACK 3 NOSINGLE RANGE
m1 m6 ;" ;
```



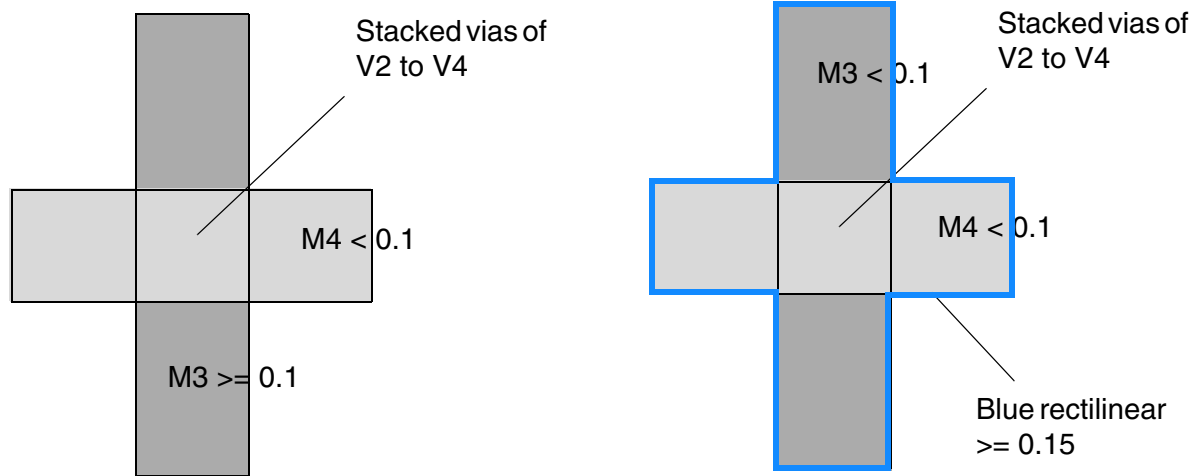
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- The following rule applies if metal shapes containing the stacked vias have an area less than the individual area of 0.1 on any layer:

```
PROPERTYDEFINITIONS
  LIBRARY LEF58_MAXVIASTACK STRING
    "MAXVIASTACK 3 EXCEPTAREA 0.1 0.15
    RANGE M2 M6 ; " ;
```

**Figure 1-284 Illustration of Max Via Stack Rule with EXCEPTAREA**



a) OK, M3 has an area  $\geq 0.1$ . Violation if EXCEPTAREA is omitted.

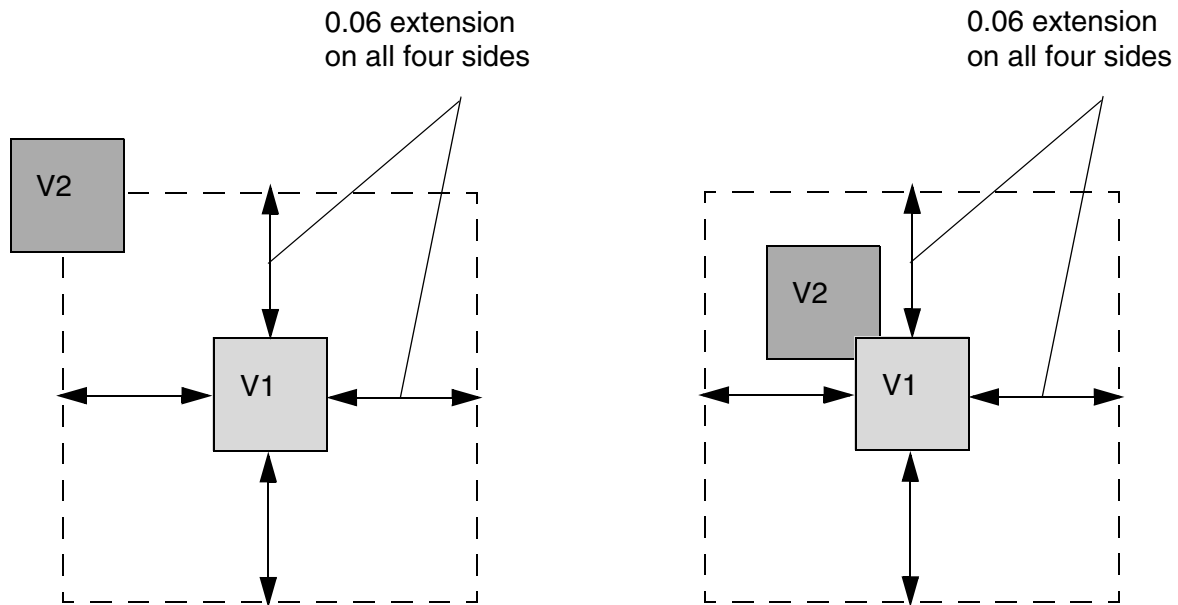
b) OK, the area of the blue rectilinear object  $\geq 0.15$ . Violation if EXCEPTAREA is omitted.

Metal area of M2 (bottom most layer) or M5 (top most layer) of the stacked vias is irrelevant in EXCEPTAREA.

- The following property rule illustrates WITHIN LOWCUTSIZE with V1 layer:

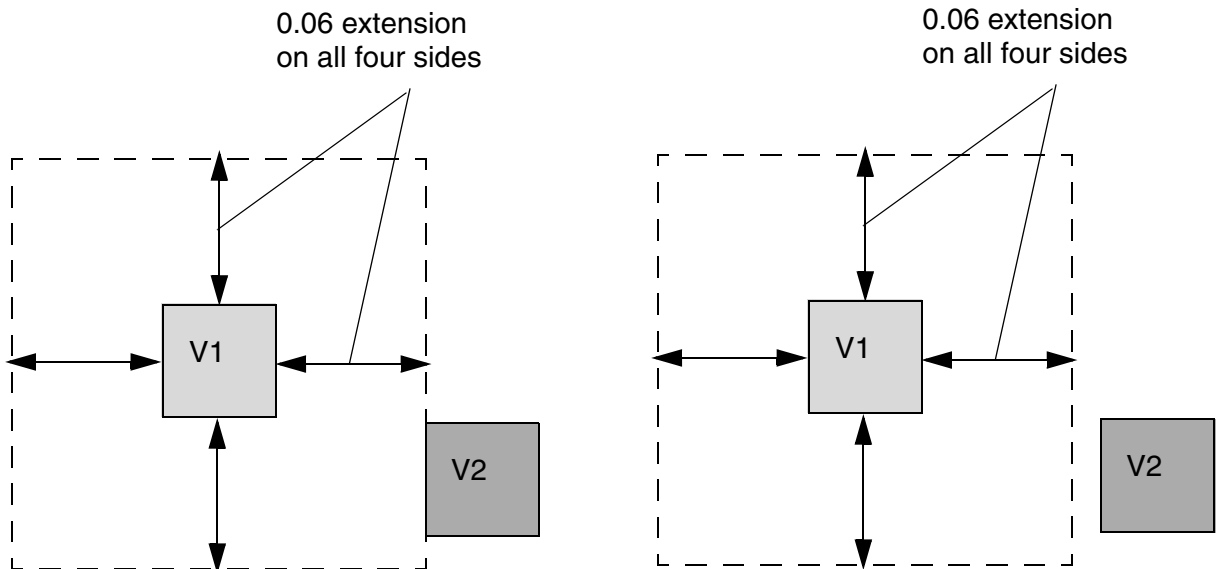
```
PROPERTY LEF58_CUTCLASS
  "CUTCLASS ... WIDTH 0.06 ... ;
  CUTCLASS ... WIDTH 0.12 ... ; "
```

**Figure 1-285 Illustration of Max Via Stack Rule with WITHIN LOWCUTSIZE**



a) Consider as stacked vias as long as V2 is within the dotted line search window of V1

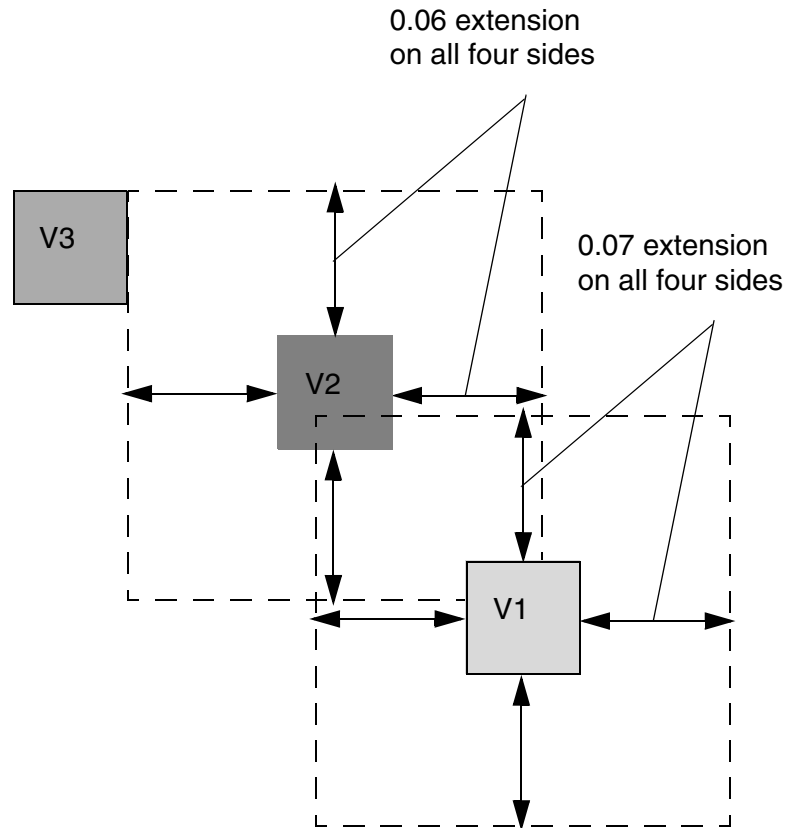
b) Consider as stacked vias if the cuts directly overlap



c) Consider as stacked vias when V2 is touching the dotted line search window of V1

d) Not considered as stacked vias since V2 is outside the dotted line search window of V1.

**Figure 1-286 Illustration of Max Via Stack Rule with WITHIN**



a) Consider as stacked vias from V1 to V3. V1 & V2 have their own specified within values and V3 touches V2 within search window

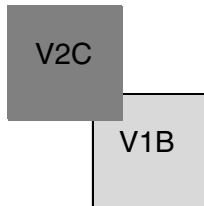
Illustration of WITHIN V1 0.07 V2 0.06



- The following property rule illustrates the max via stack rule with NOVIALIST:

```
PROPERTYDEFINITIONS
    LIBRARY LEF58_MAXVIASTACK STRING `
        MAXVIASTACK 3 NOVIALIST V1B V2C ; ` ;
```

**Figure 1-287 Illustration of Max Via Stack Rule with NOVIALIST**

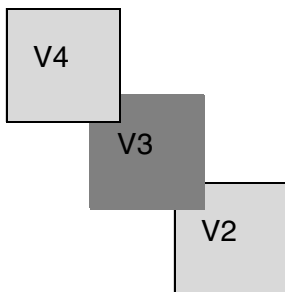


a) Violation, although three stacked vias are allowed, specific V1B and V2C combination is not allowed to be stacked

- The following property rule illustrates multiple max via stack rules:

```
PROPERTYDEFINITIONS
    LIBRARY LEF58_MAXVIASTACK STRING `
        MAXVIASTACK 2 RANGE M1 M5 ;
        MAXVIASTACK 3 RANGE M3 M7 ; ` ;
```

**Figure 1-288 Illustration of Multiple Max Via Stack Rules**



a) Violation, the second statement passes, but failing the first statement alone is still a violation

### ***Metal Width Track Rule***

You can create a metal width track rule to define the preferred routing direction for tracks on a specified layer.

You can define a metal width track rule by using the following `PROPERTYDEFINITIONS` statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 METALWIDTHTRACK STRING
`METALWIDTHTRACK routingLayer COREOFFSET offset
    {WIDTH width PITCH pitch [REPEAT repeat]}...
;` ;
END PROPERTYDEFINITIONS
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where:

```
METALWIDTHTRACK routingLayer COREOFFSET offset
{WIDTH width PITCH pitch [REPEAT repeat]}
```

Specifies the preferred routing direction tracks on the layer *routingLayer*. *offset* specifies the starting coordinate of the track with respect to the origin of the core. The *width* and *pitch* set of values define the width and pitch of the subsequent tracks. *repeat* specifies how many times the given width and pitch should be repeated. Then, the entire sets of width and pitch would be repeated. Multiple METALWIDTHTRACK statements can be defined for different routing layers. For example:

```
METALWIDTHTRACK M1 ...
METALWIDTHTRACK M2 ...
```

However, one routing layer can have only one such statement. So, it would be illegal to have the following:

```
METALWIDTHTRACK M1 ...
METALWIDTHTRACK M1 ...
```

The METALWIDTHTRACK property is typically used to define a non-uniform track on a certain layer, which depend on the cell objects on the corresponding layer. In advanced nodes, all of the cell objects are typically required on the track on lower layers. Horizontal, non-uniform tracks may be created such that instances could be placed on any row such that the cell objects are still ensured on the track. Therefore, this library property is likely to be defined in a cell LEF or side LEF file, instead of in a tech LEF file.

*Type:* Float, specified in microns

### Metal Width Track Rule Example

- The following example is an illustration of the metal track width rule:

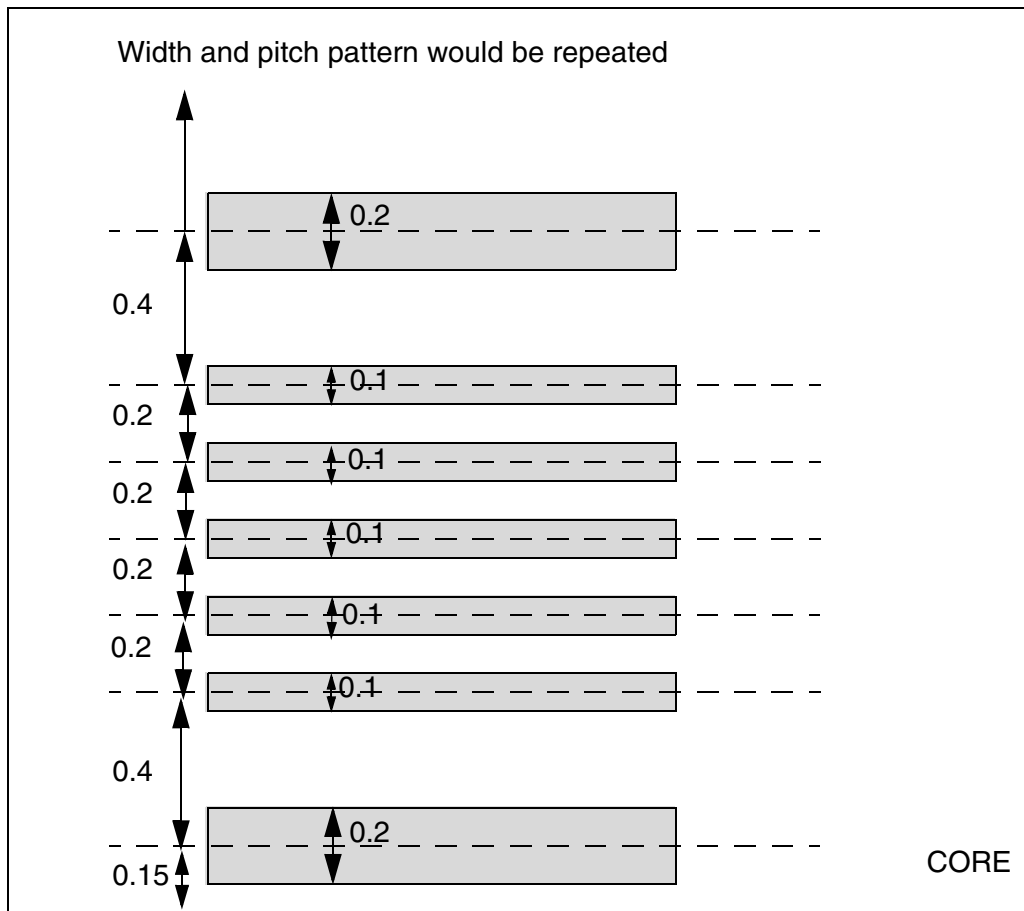
```
LAYER M1
    TYPE ROUTING ;
    DIRECTION HORIZONTAL ;
    ...
END M1
...
LIBRARY LEF58_METALWIDTHTRACK STRING
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

```
"METALWIDTHTRACK M1 COREOFFSET 0.15
    WIDTH 0.2 PITCH 0.4
    WIDTH 0.1 PITCH 0.2 REPEAT 4
    WIDTH 0.1 PITCH 0.4 ; " ;
END PROPERTYDEFINITIONS
```

**Figure 1-289 Illustration of Metal Track Width Rule**



### ***Metal Width Via Map Rule***

You can create a metal width via map rule to specify the vias to be used based on the given width values of the below and above metal layers.

You can define a metal width via map rule by using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58_METALWIDTHVIAMAP STRING
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
"METALWIDTHVIAMAP
  {VIA viaLayer
    {belowLayerWidth aboveLayerWidth
      |belowLayerLowWidth belowLayerHighWidth
        aboveLayerLowWidth aboveLayerHighWidth}
    viaName [PGVIA]}...
  ;...";
END PROPERTYDEFINITIONS
```

Where:

```
METALWIDTHVIAMAP
  {VIA viaLayer belowLayerWidth aboveLayerWidth viaName}...
```

Specifies the vias to be used based on the given width values of the below and above metal layers. *viaName* is used on cut layer *viaLayer* for below and above metal widths of *belowLayerWidth* and *aboveLayerWidth*.

*Type:* Float, specified in microns

PGVIA

Specifies that the given via is for PG net usage only. For a given below and above metal width combination, you can do one of the following:

- Define only one via to be used for both signal and PG nets. This via should be defined without PGVIA.
- Define multiple vias of which one is without PGVIA. In this case, the via without PGVIA would be used for signal nets, and the rest of the vias with PGVIA would be used for PG nets.

```
VIA viaLayer belowLayerLowWidth belowLayerHighWidth
    aboveLayerLowWidth aboveLayerHighWidth viaName...
```

Specifies the vias to be used based on the given width range values of the below and above metal layers. That is the below and above metal width should be greater than or equal to *belowLayerLowWidth* and *aboveLayerLowWidth*, respectively, and less than or equal to *belowLayerHighWidth* and *aboveLayerHighWidth*, respectively, to use *viaName*.

*Type:* Float, specified in microns

### Metal Width Via Map Rule Examples

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

- The following example indicates that V1\_SMALL should be used on cut layer V1 for below metal width of 0.05 and above metal width of 0.08 wires while V1\_LARGE should be used for below metal width of 0.07 and above metal width of 0.08 wires:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 METALWIDTHVIAMAP STRING "
    METALWIDTHVIAMAP VIA V1 0.05 0.08 V1_SMALL
    VIA V1 0.07 0.08 V1_LARGE; " ;
END PROPERTYDEFINITIONS
```

### ***OpenAccess Layer Map Rule***

You can create an OpenAccess layer map rule to define the equivalent OpenAccess layer name for a LEF layer.

You can define an OpenAccess layer map rule by using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 OALAYERMAP STRING
    "OALAYERMAP oaLayer
        LAYER layer [MASK maskNum]
        ;..." ;
END PROPERTYDEFINITIONS
```

Where:

```
OALAYERMAP oaLayer LAYER layer [MASK maskNum]
```

Specifies the equivalent OpenAccess layer name *oaLayer* of LEF layer *layer*. If MASK is specified only on a multi-patterning layer with MASK, only *maskNum* shapes would be mapped to the given OpenAccess layer name. If MASK is specified, there would be multiple such properties for each of the possible masks.

### **OpenAccess Layer Map Rule Examples**

- The following example indicates that the LEF layer M1 is mapped to OpenAccess layer Metal1:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 OALAYERMAP STRING "
    OALAYERMAP Metal1 LAYER M1 ; " ;
END PROPERTYDEFINITIONS
```

- The following example indicates that the LEF MASK 1 shapes on M1 would go to OpenAccess layer Metal1A while MASK 2 shapes on M1 would go to OpenAccess layer Metal1B:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
PROPERTYDEFINITIONS
LIBRARY LEF58 OALAYERMAP STRING `
    OALAYERMAP MetallA LAYER M1 MASK 1 ;
    OALAYERMAP MetallB LAYER M1 MASK 2 ; ` ;
END PROPERTYDEFINITIONS
```

### **PG Via Track Rule**

You can use a PG via track rule to define vertical tracks for aligning all of the power and ground cell vias on a specified layer of a cell macro with the `ALIGNPGVIATOTRACK` property.

You can define a PG via track rule by using the the following `PROPERTYDEFINITIONS` statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 PGVIATRACK STRING
    `PGVIATRACK cutLayer COREOFFSET offset PITCH pitch; ` ;
END PROPERTYDEFINITIONS
```

Where:

`PGVIATRACK cutLayer COREOFFSET offset PITCH pitch`

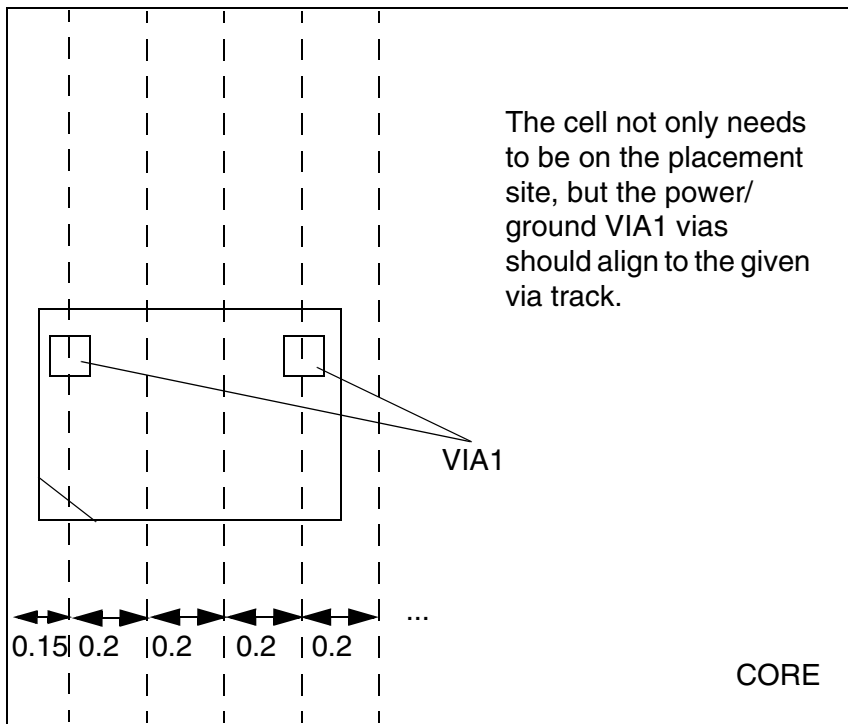
Defines vertical tracks for aligning all the power and ground cell vias on the layer *cutLayer* of a cell macro with the `ALIGNPGVIATOTRACK` property. *offset* specifies the starting coordinate of the track with respect to the origin of the core. *pitch* specifies the pitch of the tracks. All the power and ground cell vias should be in multiples of *pitch* such that aligning any one of them to the track is sufficient.  
*Type*: Float, specified in microns

### **PG Via Track Rule Example**

The following example is an illustration of the PG via track rule:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 PGVIATRACK STRING `
    PGVIATRACK VIA1 COREOFFSET 0.15
    PITCH 0.2 ; ` ;
END PROPERTYDEFINITIONS
```

**Figure 1-290 Illustration of the PG Via Track Rule**



### **Stack Via Layer Rule**

You can use a stack via layer rule to define stack via rules on individual layers. These rules are combined to form a stack via rule.

You can define a stack via layer rule by using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 STACKVIALAYERERRULE STRING
  "STACKVIALAYERERRULE ruleName
    LAYER cutLayerName [CUTCLASS className]
      ROWCOL numCutRows numCutCols
        [XPITCH xPitch [MAXXPITCH maxXPitch]]
        [YPITCH yPitch [MAXYPITCH maxYPitch]]
        [BELOWMINLENGTH belowMinLength]
        [ABOVEMINLENGTH aboveMinLength]
        [MAXCELEXTENSION extensionPitch]
        [OFFGRIDCUT]
        [MAXXLINEEND maxLeftExtensiion maxRightExtension]
        [MAXYLINEEND maxBottomExtensiion maxTopExtension]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`END PROPERTYDEFINITIONS`

Where:

`ABOVEMINLENGTH` *aboveMinLength*

Specifies the minimum length of the default width wires on the above metal layer. This construct must be specified with `XPITCH` and/or `YPITCH`.

*Type:* Float, specified in microns

`BELOWMINLENGTH` *belowMinLength*

Specifies the minimum length of the default width wires on the below metal layer. This construct must be specified with `XPITCH` and/or `YPITCH`.

*Type:* Float, specified in microns

`MAXCELLEXTENSION` *extensionPitch*

Specifies the maximum number of pitch *extensionPitch* extended from the cell boundary of the pin along both directions from which the stacked vias must be enclosed. By default, the stacked vias must be inside the cell boundary if this construct is not specified. This construct must be specified with `XPITCH` and/or `YPITCH`.

*Type:* Integer

`MAXXLINEEND` *maxLeftExtension* *maxRightExtension*

Forms a boundary box of the pin shapes to which the stacked vias connect and expands to the left and right horizontally by *maxLeftExtension* and *maxRightExtension*, respectively. All of the via cuts on this layer must be enclosed by this expanded boundary box. If negative values are given, it would shrink the boundary box in the given direction. This construct should be mutually exclusive to `MAXCELLEXTENSION`.

*Type:* Float, specified in microns

`MAXXPITCH` *maxXPitch*



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the maximum number of pitch by which all the vertical minimum-default-width metal wires of the stacked vias could be apart. The pitch must be smaller than or equal to the given value.

*Type:* Integer

MAXYLINEEND *maxBottomExtension maxTopExtension*

Forms a boundary box of the pin shapes to which the stacked vias connect and expands to the bottom and top vertically by *maxBottomExtension* and *maxTopExtension*, respectively. All of the via cuts on this layer must be enclosed by this expanded boundary box. If negative values are given, it would shrink the boundary box in the given direction. This construct should be mutually exclusive to MAXCELLEXTENSION.

*Type:* Float, specified in microns

MAXYPITCH *maxYPitch*

Specifies the maximum number of pitch by which all the horizontal minimum-default-width metal wires of the stacked vias could be apart. The pitch must be smaller than or equal to the given value.

*Type:* Integer

OFFGRIDCUT

Specifies that the cuts could be placed off grid to block fewer routing and the below metal shape would become an L shape. It is typically applied on a transition cut layer in which the cut size is larger than the default wire width on the below metal.

STACKVIALAYERERRULE *ruleName*

LAYER *cutLayerName* [CUTCLASS *className*]

ROWCOL *numCutRows numCutCols*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies a stacked via rule on cut layer *cutLayerName* with *className* cuts. If CUTCLASS is not specified, the cut class with the smallest cut size would be used or the table lookup cut class would be applied based on the wire width if METALWIDTHVIAMAP is defined. These stacked via rules would be combined in STACKVIARULE to build stacked vias on top of some special instance pins. ROWCOL specifies the number of cut rows and columns of a multiple-cuts via of cut class *className* to use on the pin.

*Type:* String and Integer

Note that this rule is only for via creation. It is difficult to re-determine the cut dimension of a generated via, and the checker does not check for the correctness of the generated vias.

XPITCH *xPitch*

Specifies that all the individual cuts must be on the grid on both the top and bottom metal layers. The cuts are *xPitch* horizontally. There would be multiple vertical minimum-default-width metal wires, and each one of them would connect separately to a cut. In the case of a transition cut layer, the cut size would be larger than the default wire width on the below metal layer. It is typical to apply the OFFGRIDCUT construct such that the below metal shape becomes an L shape to block fewer tracks. If the common metal between two adjacent cut layers is vertical, *xPitch* on these two cut layers must be identical if they are defined. If neither XPITCH nor YPITCH is specified, a typical via with one large metal shape each on both the top and bottom layers would be created.

*Type:* Integer

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

YPITCH *yPitch*

Specifies that all the individual cuts must be on the grid on both the top and bottom metal layers. The cuts are *yPitch* vertically. There would be multiple horizontal minimum-default-width metal wires, and each one of them would connect separately to a cut. In the case of a transition cut layer, the cut size would be larger than the default wire width on the below metal layer. It is typical to apply the OFFGRIDCUT construct such that the below metal shape becomes an L shape to block fewer tracks. If the common metal between two adjacent cut layers is horizontal, *yPitch* on these two cut layers must be identical if they are defined. If neither XPITCH nor YPITCH is specified, a typical via with one large metal shape each on both the top and bottom layers would be created.

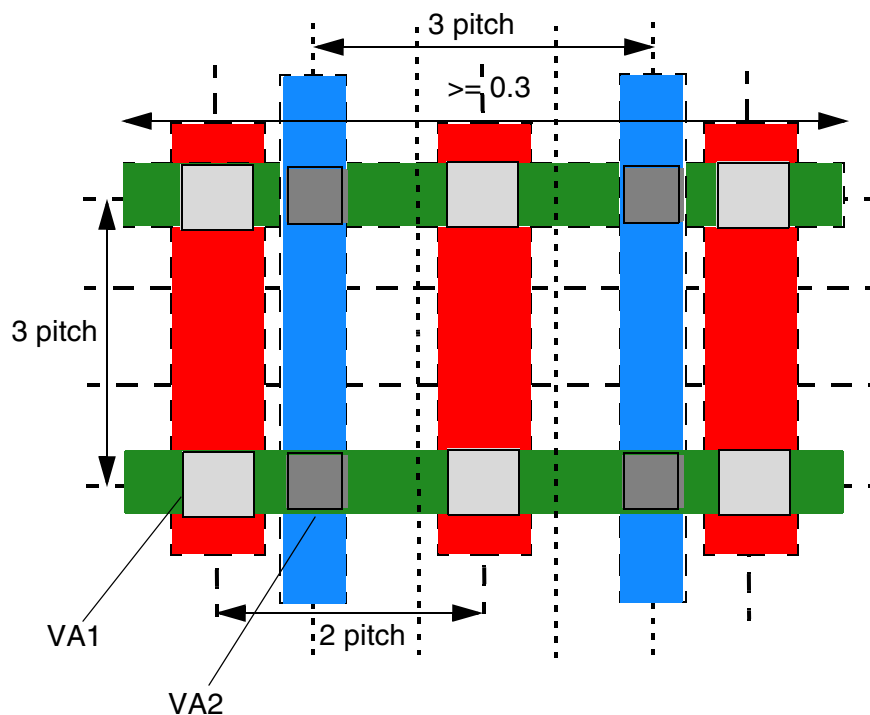
*Type:* Integer

## Stack Via Layer Rule Examples

- The following example is an illustration of the stack via layer rule:

```
PROPERTY LEF58_STACKVIALAYERRULE "  
    STACKVIALAYERRULE SVLR1 LAYER VIA1 CUTCLASS VA1  
        ROWCOL 2 3 XPITCH 2 YPITCH 3 ;  
    STACKVIALAYERRULE SVLR2 LAYER VIA2 CUTCLASS VA2  
        ROWCOL 2 2 XPITCH 2 YPITCH 3  
        BELOWMINLENGTH 0.3 ; "  
PROPERTY LEF58_STACKVIARULE "  
    STACKVIARULE SVR1 SVLR1 SVLR2 ; "
```

**Figure 1-291 Illustration of Stack Via Layer Rules**



VIA1 would be one via with six cuts, three vertical red metal shapes and two horizontal green metal shapes.

VIA2 would be one via with four cuts, two vertical blue metal shapes and two horizontal green metal shapes.

Alternatively, multiple single-cut vias and multiple wires or patches could be used to represent them.

Figure 1-292 Illustration of MAXCELLEXTENSION *extensionPitch*

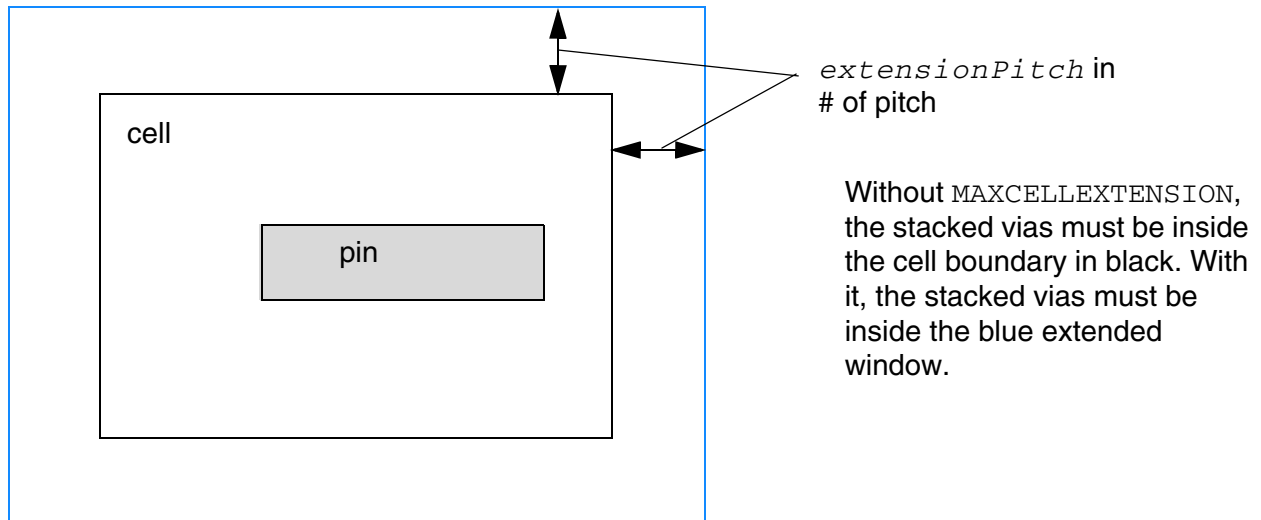
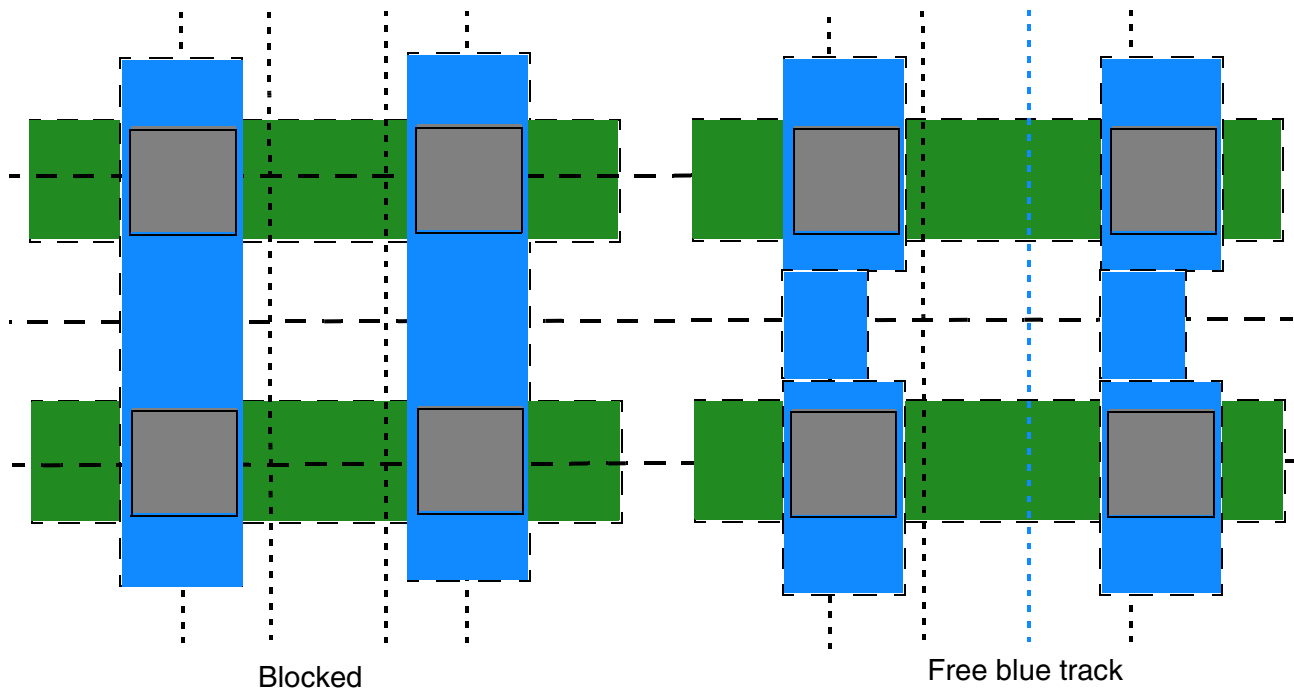


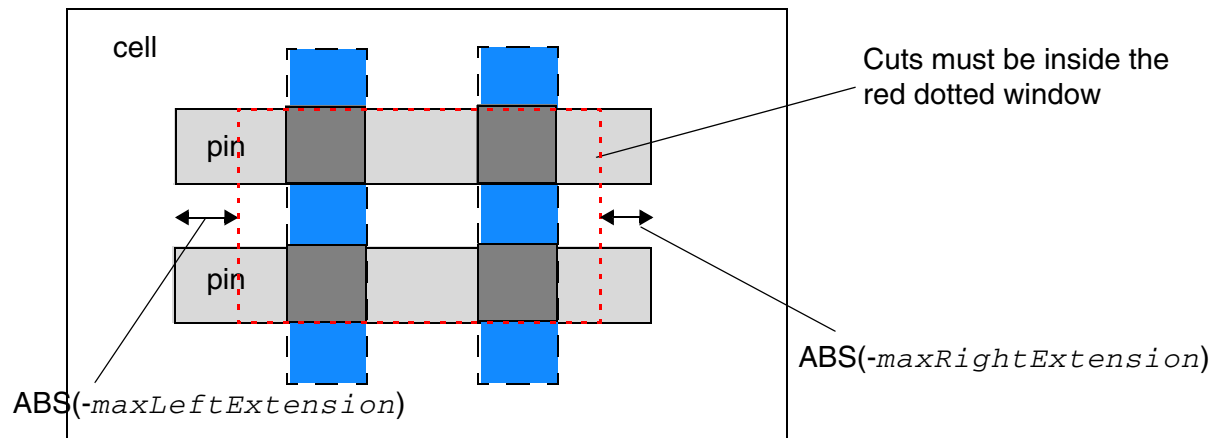
Figure 1-293 Illustration of OFFGRIDCUT



Without OFFGRIDCUT: Each blue wire  
could block three tracks individually.

With OFFGRIDCUT: Each blue wire can  
block only two tracks individually.

**Figure 1-294 Illustration of MAXXLINEEND with Negative Values for *maxLeftExtension* and *maxRightExtension***



### Stack Via Rule

You can create a stack via rule to specify a list of stack via rules for individual layers.

You can define a stack via rule by using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58_STACKVIARULE STRING
"STACKVIARULE_ruleName {stackLayerRuleName}...[EMFACTOR emFactor]
; " ;
END PROPERTYDEFINITIONS
```

Where:

**EMFACTOR *emFactor*** Specifies an EM-effective factor to indicate how well this stack via rule could prevent EM violations. The higher the specified number, the more effective is the rule in preventing EM violations.

*Type:* Float

**STACKVIARULE *ruleName* {*stackLayerRuleName*}...**

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies a stacked via rule, which consists of a list of stack via rules on individual layers in the given *stackLayerRuleName*. This list of stack via rules are the rule names specified with STACKVIALAYERRULE.

*stackLayerRuleName* should correspond to individual layer rules on consecutive layers. It is illegal to define multiple *stackLayerRuleName* on the same layer.

STACKVIALAYERRULE statements should be defined prior to STACKVIARULE in the tech LEF file.

Type: String

### Trim Metal Track Rule

You can create a trim metal track rule to define the trim metal tracks on a trim layer.

You can define a trim metal track rule by using the following PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
LIBRARY LEF58 TRIMMETALTRACK STRING
"TRIMMETALTRACK trimLayer [ONTRACK] [MASK maskNum]
    COREOFFSET offset PITCH pitch
    ;" ;
END PROPERTYDEFINITIONS
```

Where:

ONTRACK

Specifies the trim metal tracks are a hard constraint. It means that all of the inserted trims must be on a track and having an off-grid trim would be a violation by itself. This hard constraint does not apply to pre-defined trims in MACRO in the cell LEF. In addition, if one TRIMMETALTRACK statement defines ONTRACK on a *trimLayer*, all of the statements on that trim layer must also have ONTRACK.

```
TRIMMETALTRACK trimLayer [MASK maskNum] COREOFFSET offset
PITCH pitch
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the trim metal tracks on layer *trimLayer*. The direction of the tracks should always be the orthogonal direction of the preferred routing direction of the metal layer defined in TRIMMEDMETAL on the layer *trimLayer* that the trim metal trims. MASK must be defined if the layer *trimLayer* has multiple masks defined in MASK, and *maskNum* defines the track on the given mask. *offset* specifies the starting coordinate of the track with respect to the origin of the core with repeatable tracks that are *pitch* apart. Multiple statements could be defined on the same mask and/or different masks for different sets of tracks. Typically, the trim metal tracks depend on the cell objects, and this library property is likely to be defined in a cell LEF or side LEF file, instead of in a tech LEF file.

*Type*: Float, specified in microns

### Trim Metal Track Rule Example

```
LAYER TM1
    TYPE MASTERSLICE ;
    MASK 2 ;
    PROPERTY LEF58_TYPE "TYPE TRIMMETAL ; " ;
    PROPERTY LEF58_TYPE "TYPE TRIMMEDMETAL M1 ; " ;
    ...
END TM1

...

LAYER M1
    TYPE ROUTING ;
    DIRECTION VERTICAL ;
    ...
END M1

...

LIBRARY LEF58_TRIMMETALTRACK STRING "
    TRIMMETALTRACK TM1 MASK 1 COREOFFSET 0.01 PITCH 0.2 ;
    TRIMMETALTRACK TM1 MASK 1 COREOFFSET 0.12 PITCH 0.2 ;
    TRIMMETALTRACK TM1 MASK 2 COREOFFSET 0.08 PITCH 0.2 ;
    TRIMMETALTRACK TM1 MASK 2 COREOFFSET 0.19 PITCH 0.2 ;
END PROPERTYDEFINITIONS
```

The above example means that the trim metal layer of TM1 has two sets of horizontal tracks on MASK 1 with starting *y* locations of 0.01 and 0.12 with respect to the origin of the core with repeatable pitch of 0.2, and it has another two sets of horizontal tracks on MASK 2 with starting *y* locations of 0.08 and 0.19 with respect to the origin of the core with repeatable pitch of 0.2.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## Macro

MACRO *macroName*

```
[CLASS
  { COVER [BUMP]
  | RING
  | BLOCK [BLACKBOX | SOFT]
  | PAD [INPUT | OUTPUT | INOUT | POWER | SPACER | AREAIO]
  | CORE [FEEDTHRU | TIEHIGH | TIELOW | SPACER | ANTENNACELL | WELLTAP]
  | ENDCAP {PRE | POST | TOPLEFT | TOPRIGHT | BOTTOMLEFT | BOTTOMRIGHT}
  }
;]
[FIXEDMASK ;]
[FOREIGN foreignCellName [pt [orient]] ;] ...
[ORIGIN pt ;]
[EEQ macroName ;]
[SIZE width BY height ;]
[SYMMETRY {X | Y | R90} ... ;]
[SITE siteName [sitePattern] ;] ...
[PIN statement] ...
[OBS statement] ...
[DENSITY statement] ...
[PROPERTY propName propVal ;] ...
[PROPERTY LEF58 ACCESSAREA
  "ACCESSAREA {LAYER layerName RECT pt pt [CUTCLASS className]
  [EXCEPTEXTRACUT]}...
  ; " ;]
[PROPERTY LEF58 ALIGNPGVIATOTRACK
  "ALIGNPGVIATOTRACK
  ; " ;]
[PROPERTY LEF58 CLASS
  "CLASS
  {COVER [BUMP | FILL]
  | RING
  | BLOCK [BLACKBOX | SOFT]
  | PAD [INPUT | OUTPUT | INOUT | POWER | SPACER | AREAIO]
  | CORE [FEEDTHRU | TIEHIGH | TIELOW | SPACER
    | ANTENNACELL | WELLTAP]
  | ENDCAP {PRE | POST | TOPLEFT | TOPRIGHT | BOTTOMLEFT
    | BOTTOMRIGHT}
    | TOPEDGE | BOTTOMEDGE | LEFTEDGE | RIGHTEDGE
    | LEFTEVENSITEEDGE | LEFTODDSITEEDGE
    | RIGHTEVENSITEEDGE | RIGHTODDSITEEDGE
    | LEFTTOPEDGE | RIGHTTOPEDGE
    | LEFTBOTTOMEDGE | RIGHTBOTTOMEDGE
    | LEFTTOPEVENSITEEDGE | LEFTTOPODDSITEEDGE
    | RIGHTTOPEVENSITEEDGE | RIGHTTOPODDSITEEDGE
    | LEFTBOTTOPEVENSITEEDGE
    | LEFTBOTTOMODDSITEEDGE
    | RIGHTBOTTOPEVENSITEEDGE
    | RIGHTBOTTOMODDSITEEDGE
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```

| LEFTTOPCORNER | RIGHTTOPCORNER
| LEFTBOTTOMCORNER | RIGHTBOTTOMCORNER
| LEFTTOPEVENSITECORNER | LEFTTOPODDSITECORNER
| RIGHTTOPEVENSITECORNER
| RIGHTTOPODDSITECORNER
| LEFTBOTTOPEVENSITECORNER
| LEFTBOTTOMODDSITECORNER
| RIGHTBOTTOPEVENSITECORNER
| RIGHTBOTTOMODDSITECORNER
| LEFTEDGETOPBORDER
| LEFTEDGEBOTTOMBORDER
| RIGHTEDGETOPBORDER
| RIGHTEDGEBOTTOMBORDER
| LEFTTOPEDGE NEIGHBOR
| RIGHTTOPEDGE NEIGHBOR
| LEFTBOTTOMEDGE NEIGHBOR
| RIGHTBOTTOMEDGE NEIGHBOR}

}
; " ;]
[PROPERTY LEF58 CONSTRAINTAREATYPE
  "CONSTRAINTAREATYPE typeName {RECT pt pt}...
  ;..." ;]
[PROPERTY LEF58 EDGETYPE
  "EDGETYPE {RIGHT | LEFT | TOP | BOTTOM}
  edgeType | BOTHSOURCE | SOURCEDRAIN | DRAINSOURCE | BOTHDRAIN
  | BOTHFLOATING
  [CELLROW cellRow | HALFROW halfRow | RANGE xLow xHigh]
  ;..." ;]
[PROPERTY LEF58 FOREIGN
  "FOREIGN foreignCellName [pt [orient]]
  [COMPORIENT foreignOrientName {compOrient}...]...
  ;... " ;]
[PROPERTY LEF58 OBSPARTIAL
  "OBSPARTIAL {LAYER layerName
  WIDTH width SPACING spacing RECT pt pt}...
  ; " ;]
[PROPERTY LEF58 OBSPACING
  "OBSPACING {MIN | spacing} [LAYER layerName]...
  ; " ;]

```

END *macroName*

Defines macros in the design.

**Note:** The keywords must be specified in the given order. For example if ORIGIN and SITE are both defined, ORIGIN must be specified first.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### CLASS

Specifies the macro type. If you do not specify `CLASS`, the macro is considered a `CORE` macro, and a warning prints when the LEF file is read in. You can specify macros of the following types:

**COVER** Macro with data that is fixed to the floorplan and cannot change, such as power routing (ring pins) around the core. The placers understand that `CLASS COVER` cells have no active devices (such as diffusion or polysilicon), so the `MACRO SIZE` statement does not affect the placers, and you do not need an artificial `OVERLAP` layer. However, any pin or obstruction geometry in the `COVER` cells can affect the pin access checks done by the placers.

A cover macro can be of the following sub-class:

**BUMP**—A physical-only cell that has bump geometries and pins. Typically a bump cell has geometries only on the top-most “bump” metal layer, although it might contain a via and pin to the metal layer below.

**RING** Large macro that has an internal power mesh, and only exposes power-pin shapes that form a ring along the macro boundary. When power stripes are added across the macro, they connect to each side of the ring-pin but do not go inside the ring. The `CLASS RING` macro can also be used for power-switch cells that are abutted together to form a power-ring around a power-domain. In that case, their power-pins have the same effect of interrupting power stripes as the ring-pins in a single block `RING` macro.

**BLOCK** Predefined macro used in hierarchical design.

A block macro can have one of the following sub-classes:

**BLACKBOX**—A block that sometimes only contains a `SIZE` statement that estimates its total area. A blackbox can optionally contain pins, but in many cases, the pin names are taken from a Verilog description and do not need to match the LEF `MACRO` pin names.

**SOFT**—A cell that also contains a version of the sub-block that is not fully implemented. Normally, a soft block LEF can still have certain parts of it modified (for example, the aspect ratio, or pin locations) because the sub-block is not yet fully implemented. Any changes should be passed to the sub-block implementation. In contrast, a `BLACKBOX` has no sub-block implementation available.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**PAD** I/O pad. A pad can be one of the following types: `INPUT`, `OUTPUT`, `INOUT`, `POWER`, or `SPACER`, for I/O rows; `INPUT`, `OUTPUT`, `INOUT`, or `POWER`, for I/O corner pads; `AREAIO` for area I/O driver cells that do not have the bump built in as part of the macro (and therefore require routing to a `CLASS COVER BUMP` macro for a connection to the IC package).

For an example of a macro pad cell, see [Example 1-22](#) on page 549.

**CORE** A standard cell used in the core area. `CORE` macros should always contain a `SITE` definition so that standard cell placers can correctly align the `CORE` macro to the standard cell rows.

A core macro can be one of the following types:

**FEEDTHRU**—Used for connecting to another cell.

**TIEHIGH,TIELOW**—Used for connecting unused I/O terminals to the power or ground bus. The software does not rely on this sub-class. A tie-cell has to be `CLASS CORE`, but the software does not consider the sub-class to determine its type. The dotlib representation of the cell's output pin is considered, and based on the function on that pin, it is determined whether it is a tiehigh, or tielow.

**SPACER**—Sometimes called a filler cell, this cell is used to fill in space between regular core cells. The `SPACER` sub-class needs to be cells with no logic-pins. Thus even with the sub-class defined, a cell will not be considered `SPACER` (also called `FILLER`) unless it has no logic/signal pins. A filler can only have Power and Ground pins. The instances of these cells will be marked by the insertion command to be of type 'Physical'.

**ANTENNACELL**—Used for solving process antenna violations. This cell has a single input to a diode to bleed off charge that builds up during manufacturing.

**WELLTAP**—Standard cell that connects N and P diffusion wells to the correct power or ground wire. The `WELLTAP` cells provide a tap for the N and P wells to the power/ground wires.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**ENDCAP**      A macro placed at the ends of core rows (to connect with power wiring).

If the library includes only one corner I/O macro, then appropriate **SYMMETRY** must be included in its macro description. An **ENDCAP** macro can be one of the following types:

**PRE**—A left-end macro

**POST**—A right-end macro

**TOPLEFT**—A top left I/O corner cell

**TOPRIGHT**—A top right I/O corner cell

**BOTTOMLEFT** —A bottom left I/O corner cell

**BOTTOMRIGHT**—A bottom right I/O corner cell

The **ENDCAP** sub-class is required. The **PRE** and **POST** are **CORE** area cells, whereas the other four are **PAD CLASS**.

### Example 1-22 Macro Pad Cell

The following example defines a power pad cell that illustrates when to use the **CLASS CORE** keywords on power ports. For the **VDD** pin, there are two ports: one to connect to the interior core power ring, and one to complete the I/O power ring. Figure 1-1 on page 6 illustrates this pad cell.

```
MACRO PAD_0
  CLASS PAD ;
  FOREIGN PAD_0 0.000 0.000 ;
  ORIGIN 0.000 0.000 ;
  SIZE 100.000 BY 300.000 ;
  SYMMETRY X Y R90 ;
  SITE PAD_SITE ;

# Define pin VDD with SHAPE ABUTMENT because there are no obstructions
# to block a straight connection to the pad rings. The port without
# CLASS CORE is used for completing the I/O power ring.

  PIN VDD
    DIRECTION INOUT ;
    USE POWER ;
    SHAPE ABUTMENT ;
    PORT
      LAYER metal2 ;
      RECT 0.000 250.000 100.000 260.000 ;
      LAYER metal3 ;
      RECT 0.000 250.000 100.000 260.000 ;
    END
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

# Define VDD port with PORT CLASS CORE to indicate that the port connects  
# to the core area instead of to the pad ring.

```
PORT
    CLASS CORE ;
    LAYER metal2 ;
        RECT 0.000 290.000 100.000 300.000 ;
    LAYER metal3 ;
        RECT 0.000 290.000 100.000 300.000 ;
END
END VDD
```

# Define pins VCC and GND with SHAPE FEEDTHRU because these pins  
# cannot make a straight connection to the pad rings due to obstructions.

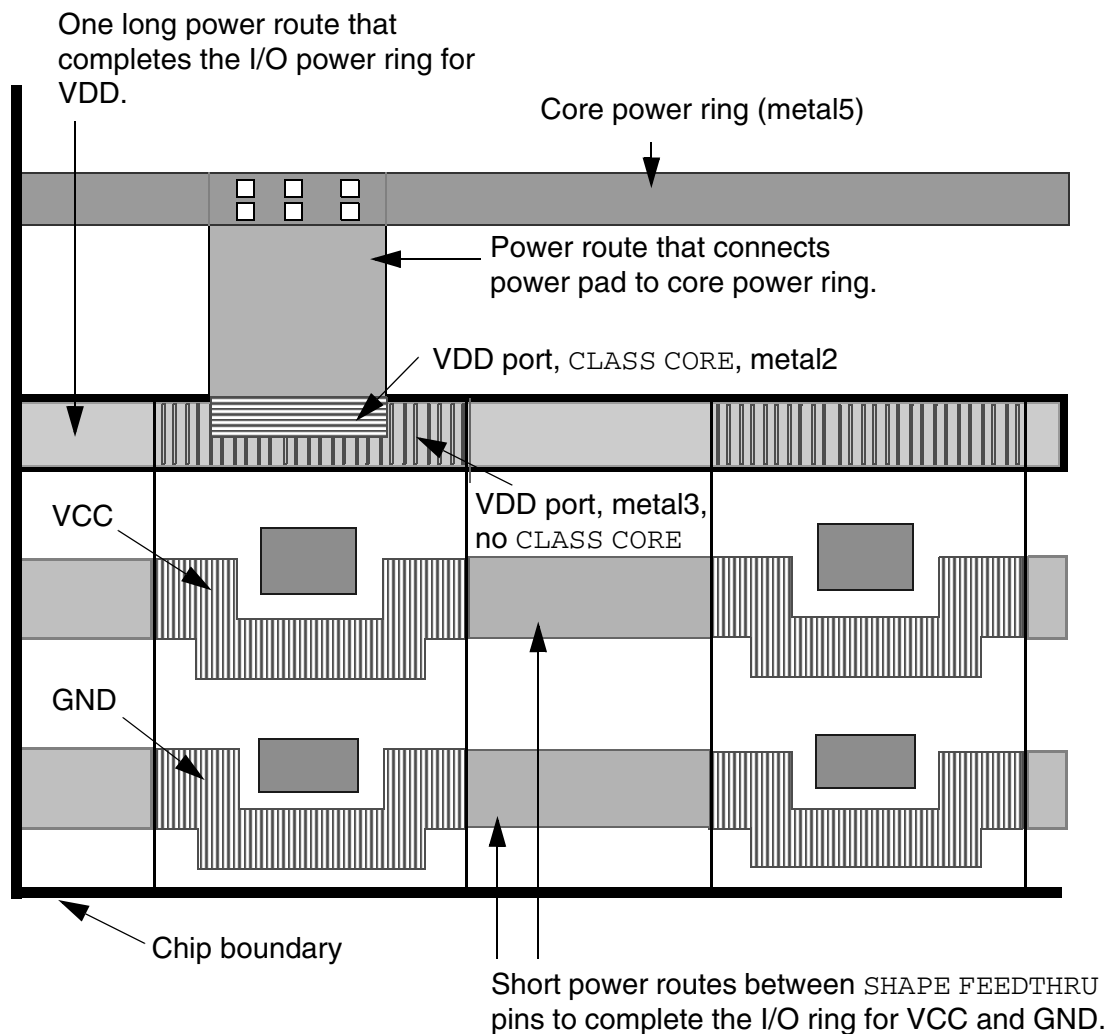
```
PIN VCC
    DIRECTION INOUT ;
    USE POWER ;
    SHAPE FEEDTHRU ;
    PORT
        LAYER metal2 ;
            RECT 0.000 150.000 20.000 160.000 ;
            RECT 20.000 145.000 80.000 155.000 ;
            RECT 80.000 150.000 100.000 160.000 ;
        LAYER metal3 ;
            RECT 0.000 150.000 20.000 160.000 ;
            RECT 20.000 145.000 80.000 155.000 ;
            RECT 80.000 150.000 100.000 160.000 ;
    END
END VCC
PIN GND
    DIRECTION INOUT ;
    USE GROUND ;
    SHAPE FEEDTHRU ;
    PORT
        LAYER metal2 ;
            RECT 0.000 50.000 20.000 60.000 ;
            RECT 80.000 50.000 100.000 60.000 ;
    END
END GND
OBS
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

```
LAYER metal1 ;  
  RECT 0.000 0.000 100.000 300.000 ;  
LAYER metal2 ;  
  RECT 25.000 50.000 75.000 60.000 ;  
  RECT 30.500 157.000 70.500 167.000 ;  
  
END  
END PAD_0
```

**Figure 1-295 Power Pad Cell**



#### DENSITY statement

Specifies the metal density for large macros.

The **DENSITY** rectangles on a layer should not overlap, and should cover the entire area of the macro. You can choose the size of the rectangles based on the uniformity of the

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

density of the block. If the density is uniform, a single rectangle can be used. If the density is not very uniform, the size of the rectangles can be specified to be 10 to 20 percent of the density window size, so that any error due to non-uniform density inside each rectangle area is small.

For example, if the metal density rule is for a 100  $\mu\text{m}$  x 100  $\mu\text{m}$  window, the density rectangles can be 10x10  $\mu\text{m}$  squares. Any non-uniformity will have little impact on the density calculation accuracy.

If two adjacent rectangles have the same or similar density, they can be merged into one larger rectangle, with one average density value. The choice between accuracy and abstraction is left to the abstract generator.

The DENSITY syntax is defined as follows:

```
[DENSITY
  {LAYER layerName ;
    {RECT x1 y1 x2 y2 densityValue ;} ...
  } ...
END] ...
```

*densityValue*      Specifies the density for the rectangle, as a percentage. For example, 50.0 indicates that the rectangle has a density of 50 percent on *layerName*.  
*Type:* Float  
*Value:* 0 to 100

*layerName*            Specifies the layer on which to create the rectangle.

*x1 y1 x2 y2*          Specifies the coordinates of a rectangle.  
*Type:* Float, specified in microns

### Example 1-23 Macro Density

The following statement specifies the density for macro `testMacro`:

```
MACRO testMacro
  CLASS ...
  PIN ...
  OBS ...
  DENSITY
    LAYER metall ;
    RECT 0 0 100 100 45.5 ; #rect from (0,0) to (100,100), density of 45.5%
    RECT 100 0 200 100 42.2 ; #rect from (100,0) to (200, 100), density of 42.2%
  END
  ...
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

END testMacro

EEQ *macroName* Specifies that the macro being defined should be electrically equivalent to the previously defined *macroName*. EEQ macros include devices such as OR-gates or inverters that have several implementations with different shapes, geometries, and orientations.

Electrically equivalent macros have the following requirements:

- Corresponding pins must have corresponding functionality.
- Pins must be defined in the same order.
- For each group of corresponding pins (one from each macro), pin function and electrical characteristics must be the same.
- The EEQ *macroName* specified must refer to a previously defined macro. If the EEQ *macroName* referenced is already electrically equivalent to other model macros, all referenced macros are considered electrically equivalent.

FIXEDMASK Indicates that the specified macro does not allow mask-shifting. All the LEF PIN MASK assignments must be kept fixed and cannot be shifted to a different mask to optimize routing density. All the LEF PIN shapes should have MASK assignments, if FIXEDMASK statement is present.

For example,

```
MACRO my_block
    CLASS BLOCK ;
    FIXEDMASK ;
    ...
```

FOREIGN *foreignCellName* [*pt* [*orient*]]

Specifies the foreign (GDSII) structure name to use when placing an instance of the macro. The optional *pt* coordinate specifies the macro origin (lower left corner when the macro is in north orientation) offset from the foreign origin. The FOREIGN statement has a default offset value of 0 0, if *pt* is not specified.

The optional *orient* value specifies the orientation of the foreign cell when the macro is in north orientation. The default *orient* value is N (North).

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### Example 1-24 Foreign Statements

The following examples show two variations of the `FOREIGN` statement. The negative offset specifies that the GDSII structure should be above and to the right of the macro lower left corner.

```
MACRO ABC ...
FOREIGN ABC -2 -3 ;
```

The positive offset specifies that the GDSII structure should be below and to the left of the macro lower left corner.

```
MACRO EFG ...
FOREIGN EFG 2 3 ;
```

<code>MACRO <i>macroName</i></code>	Specifies the name of the library macro.
<code>OBS <i>statement</i></code>	Defines obstructions on the macro. Obstruction geometries are specified using layer geometries syntax. See <a href="#">“Macro Obstruction Statement”</a> on page 588 for syntax information.
<code>ORIGIN <i>pt</i></code>	Specifies how to find the origin of the macro to align with a <code>DEF COMPONENT</code> placement point. If there is no <code>ORIGIN</code> statement, the <code>DEF</code> placement point for a North-oriented macro is aligned with 0, 0 in the macro. If <code>ORIGIN</code> is given in the macro, the macro is shifted by the <code>ORIGIN</code> x, y values first, before aligning with the <code>DEF</code> placement point. For example, if the <code>ORIGIN</code> is 0, -1, then macro geometry at 0, 1 are shifted to 0, 0, and then aligned to the <code>DEF</code> placement point.
<code>PIN <i>statement</i></code>	Defines pins for the macro. See <a href="#">“Macro Pin Statement”</a> on page 591 for syntax information.
<code>PROPERTY <i>propName propName</i></code>	Specifies a numerical or string value for a macro property defined in the <code>PROPERTYDEFINITIONS</code> statement. The <i>propName</i> you specify must match the <i>propName</i> listed in the <code>PROPERTYDEFINITIONS</code> statement.
<code>SITE <i>siteName</i> [<i>sitePattern</i>]</code>	

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the site associated with the macro. Normal row-based standard cells only have a single `SITE siteName` statement, without a *sitePattern*. The *sitePattern* syntax indicates that the cell is a gate-array cell, rather than a row-based standard cell. Gate-array standard cells can have multiple `SITE` statements, each with a *sitePattern*.

The *sitePattern* syntax is defined as follows:

```
[xOrigin yOrigin siteOrient [stepPattern]]
```

*xOrigin yOrigin* Specifies the origin of the site inside the macro.

*Type:* Float, specified in microns

*siteOrient* Specifies the orientation of the site at that location.

*Value:* N, S, E, W, FN, FS, FE, or FW

**Note:** Legal placement locations for macros with site patterns must match the site pattern inside the macro to the site pattern in the design rows.

If the site is repeated, you can specify a *stepPattern* that defines the repeating pattern. The *stepPattern* syntax is defined as follows:

```
[DO xCount BY yCount STEP xStep yStep]
```

*xCount yCount* Specifies the number of sites to add in the x and y directions. You must specify values that are greater than or equal to 0 (zero).

*Type:* Integer

*xStep yStep* Specifies the spacing between sites in the x and y directions.

*Type:* Float, specified in microns

### Example 1-25 Macro Site

The following statement defines a macro that uses the sites created in [Example 1-37](#) on page 610:

```
MACRO myTest
  CLASS CORE ;
  SIZE 10.0 BY 14.0 ;    #Uses 2 F and 1 L site, is F + L wide, and double height
  SYMMETRY X ;          #Can flip about the X axis
  SITE Fsite 0 0 N ;    #The lower left Fsite at 0,0
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

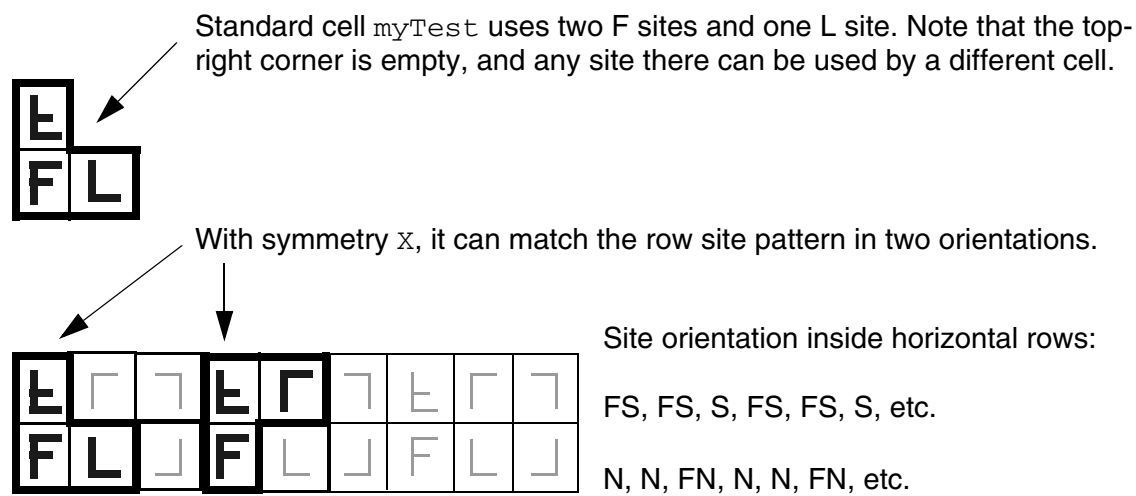
```

SITE Fsite 0 7.0 FS ; #The flipped south Fsite above the first Fsite at 0,7
SITE Lsite 4.0 0 N ; #The Lsite to the right of the first Fsite at 4,0
...
PIN ... ;
END myTest

```

Figure 1-296 on page 556 illustrates the placement results of this definition.

**Figure 1-296**



The following statement includes the gate-array site pattern syntax. It uses two F sites in a row with N (North) orientation.

```

MACRO myTest
  CLASS CORE ;
  SIZE 8.0 BY 7.0 ; #Width = 2 * Fsite width, height = Fsite height
  SITE Fsite 0 0 N DO 2 BY 1 STEP 4.0 0 ; #Xstep = 4.0 = Fsite width
  ...
END myTest

```

This definition produces a cell with the sites shown in Figure 1-297 on page 556.

**Figure 1-297**



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`SIZE width BY height`

Specifies a placement bounding rectangle, in microns, for the macro. The bounding rectangle always stretches from (0, 0) to the point defined by `SIZE`. For example, given `SIZE 10 BY 40`, the bounding rectangle reaches from (0, 0) after adjustment due to the `ORIGIN` statement, to (100, 400).

Placers assume the placement bounding rectangle cannot overlap placement bounding rectangles of other macros, unless `OBS OVERLAP` shapes are used to create a non-rectangular area.

After placement, a `DEF COMPONENTS` placement *pt* indicates where the lower-left corner of the placement bounding rectangle is placed after any possible rotations or flips. The bounding rectangle width and height should be a multiple of the placement grid to allow for abutting cells.

For blocks, the placement bounding rectangle typically contains all pin and blockage geometries, but this is not required. For example, typical standard cells have pins that lie outside the bounding rectangle, such as power pins that are shared with cells in the next row above them.

`SYMMETRY {X | Y | R90}`

Specifies which macro orientations should be attempted by the placer before matching to the site of the underlying rows. In general, most standard cell macros should have symmetry `X Y`. N (North) is always a legal candidate. For each type of symmetry defined, additional orientations become legal candidates. For more information on defining symmetry, see [“Defining Symmetry”](#) on page 578.

Possible orientations include:

X	N and FS orientations should be tried.
Y	N and FN orientations should be tried.
X Y	N, FN, FS, and S orientations should all be tried.
R90	Specify this value only for non-standard cells.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**Note:** If you do not specify a `SYMMETRY` statement, only N orientation is tried.

For corner I/O pads, if the library includes `BOTTOMLEFT`, `BOTTOMRIGHT`, `TOPLEFT`, and `TOPRIGHT` I/O corner cells, then they are placed in North orientation (no flipping). However, if the library includes only one type of corner I/O, then `SYMMETRY` in x and y are required to create the rows for all four of them.

### Defining Macro Properties to Create 32/28 nm and Smaller Nodes Rules

You can include macro layer properties in your LEF file to create 32/28 nm and smaller nodes rules that currently are not supported by existing LEF syntax. The properties are specified inside the Macro statements where they can be seen with other rules.

Before you can reference them, properties must be defined at the beginning of the LEF file in the PROPERTYDEFINITIONS statement, immediately before the first Macro statement.

- Properties belong to the MACRO/PIN object and have a type of STRING.
- The property names used for these rules all start with LEF58\_.

All properties use the following syntax within the LEF PROPERTYDEFINITIONS statement:

```
PROPERTYDEFINITIONS
    MACRO propName STRING ["stringValue"] ;
    PIN propName STRING ["stringValue"] ;
END PROPERTYDEFINITIONS
```

The property definitions for the macro properties are as follows:

```
PROPERTYDEFINITIONS
    MACRO LEF58_ACCESSAREA STRING ;
    MACRO LEF58_ALIGNPGVIATOTRACK STRING ;
    MACRO LEF58_CLASS STRING ;
    MACRO LEF58_CONSTRAINTAREATYPE STRING ;
    MACRO LEF58_EDGETYPE STRING ;
    MACRO LEF58_FOREIGN STRING ;
    MACRO LEF58_OBSPARTIAL STRING ;
    MACRO LEF58_OBSPACING STRING ;
END PROPERTYDEFINITIONS
```

### Access Area Rule

The access area rule can be used on the macro to specify a via access area on the specified layer.

You can specify the access area rule by using the following property definition:

```
PROPERTY LEF58_ACCESSAREA
    "ACCESSAREA {LAYER layerName RECT pt pt [CUTCLASS className]
    [EXCEPTEXTRACUT]}...
    ; " ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Where :

`ACCESSAREA {LAYER layerName RECT pt pt }...`

Specifies a via access area on *layerName*, which must be a cut layer. All the cuts of a via connected to any pin shape on the bottom metal layer that overlaps the given area in `RECT` must be inside the given area. If there are multiple via access areas overlapping a pin shape, the entire size of the cuts of a via connected to the pin need to be inside one of the areas. Different via access areas should not overlap or be abutted. For a `MUSTJOINALLPORTS` pin, the given area must overlap with all of the pin shapes on the bottom metal layer. Then, the cuts on each of the individual pin shapes must be inside the given area and must be connected by a same-metal wire on the above metal layer as well. This construct can only be defined on standard cells with `MACRO CLASS` of `CORE`.

*Type:* Float, specified in microns

`CUTCLASS className`

Specifies the via access area for the given *className*, which is connected to the pin shapes. Without `CUTCLASS`, the via access area would be applied to vias of any cut class. Multiple via access areas could be defined for different cut classes. If a cut class does not have a corresponding via access area defined, the vias of that cut class could be connected to the pins without any restriction.

*Type:* String

`EXCEPTEXTRACUT`

Specifies that the via access area is not applied to vias with multiple cuts connected to a single pin shape. If a via pillar with multiple wires on the above metal layer of the pin shape layer is used, this via access area does not need to be honored. For example, if a via pillar with 2x2 cuts is connected to a `MUSTJOINALLPORTS` with two pin shapes, this via access area could be ignored.

### Area Access Rule Examples

- The following example means that the cuts of `VIA1` of `VA` must be inside `1.0 1.0 2.0 2.0` and the cuts of `VIA1` of the other cut class must be inside `1.5 1.5 2.2 2.2` when connecting to a `M1` (below metal layer) pin shape that overlaps with the given area.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

```
PROPERTY LEF58_ACCESSAREA "
ACCESSAREA LAYER VIA1 RECT 1.0 1.0 2.0 2.0 CUTCLASS VA
LAYER VIA1 RECT 1.5 1.5 2.2 2.2 ; " ;
```

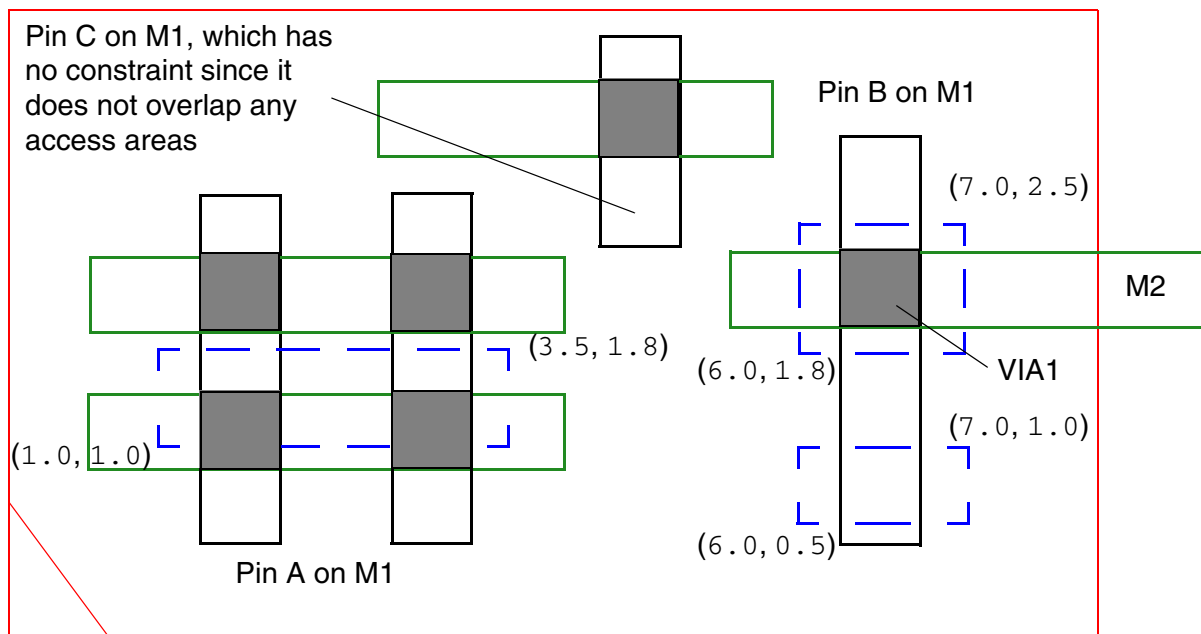
- The following example means that the cuts of VIA1 of VA must be inside 1.0 1.0 2.0 2.0 when connect to a M1 (below metal layer) pin shape that overlaps with the given area and the cuts of VIA1 of the other cut class does not have any restriction.

```
PROPERTY LEF58_ACCESSAREA "
ACCESSAREA LAYER VIA1 RECT 1.0 1.0 2.0 2.0 CUTCLASS VA ; " ;
```

- The diagram below illustrates the following access area rule:

```
PROPERTY LEF58_ACCESSAREA "
ACCESSAREA LAYER VIA1 RECT 1.0 1.0 3.5 1.8
EXCEPTEXTRACUT
LAYER VIA1 RECT 6.0 0.5 7.0 1.0
LAYER VIA1 RECT 6.0 1.8 7.0 2.5 ; " ;
```

**Figure 1-298 Illustration of the Access Area Rule**



OK because multiple cuts are connected to a pin shape on pin A while the cut of pin B is within the given top blue dotted area, which is one of the two areas overlapping with the pin shape. Violation if EXCEPTEXTRACUT is omitted.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### ***Align PG Via to Track Rule***

The align PG via to track attribute can be used on the macro to align the power or ground cell vias to the given tracks defined in the PG via track rule in PGTRACKVIA.

You can specify the align PG via to track attribute by using the following property definition:

```
PROPERTY LEF58_ALIGNPGVIATOTRACK
    "ALIGNPGVIATOTRACK
        ; ... " ;
```

Where :

ALIGNPGVIATOTRACK	Specifies that all the power or ground cell vias of this macro should be aligned to the tracks defined in PGVIATRACK in the library property in a tech LEF file.
-------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### ***Class Rule***

You can use the class rule to define a macro as a special cover macro for metal filling purpose.

You can create class rule by using the following property definition:

```
PROPERTY LEF58_CLASS
    "CLASS
        {COVER [BUMP | FILL]
        | RING
        | BLOCK [BLACKBOX | SOFT]
        | PAD [INPUT | OUTPUT | INOUT | POWER | SPACER | AREAIO]
        | CORE [FEEDTHRU | TIEHIGH | TIELOW | SPACER | ANTENNACELL
            | WELLTAP]
        | ENDCAP {PRE | POST | TOPLEFT | TOPRIGHT | BOTTOMLEFT
            | BOTTOMRIGHT
            | TOPEDGE | BOTTOMEDGE | LEFTEDGE | RIGHTEDGE
            | LEFTEVENSITEEDGE | LEFTODDSITEEDGE
            | RIGHTEVENSITEEDGE | RIGHTODDSITEEDGE
            | LEFTTOPEDGE | RIGHTTOPEDGE
            | LEFTBOTTOMEDGE | RIGHTBOTTOMEDGE
            | LEFTTOPEVENSITEEDGE | LEFTTOPODDSITEEDGE
            | RIGHTTOPEVENSITEEDGE | RIGHTTOPODDSITEEDGE
            | LEFTBOTTOPEVENSITEEDGE | LEFTBOTTOMODDSITEEDGE
            | RIGHTBOTTOPEVENSITEEDGE | RIGHTBOTTOMODDSITEEDGE
            | LEFTTOPCORNER | RIGHTTOPCORNER
            | LEFTBOTTOMCORNER | RIGHTBOTTOMCORNER
            | LEFTTOPEVENSITECORNER | LEFTTOPODDSITECORNER
            | RIGHTTOPEVENSITECORNER | RIGHTTOPODDSITECORNER
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
    | LEFTBOTTOOMEVENSITECORNER
    | LEFTBOTTOMODDSITECORNER
    | RIGHTBOTTOOMEVENSITECORNER
    | RIGHTBOTTOMODDSITECORNER
    | LEFTEDGETOPBORDER
    | LEFTEDGEBOTTOMBORDER
    | RIGHTEDGETOPBORDER
    | RIGHTEDGEBOTTOMBORDER
    | LEFTTOPEEDGE NEIGHBOR
    | RIGHTTOPEEDGE NEIGHBOR
    | LEFTBOTTOMEDGE NEIGHBOR
    | RIGHTBOTTOMEDGE NEIGHBOR }
; " ;
```

Where :

All other keywords are same as the existing macro CLASS syntax.

**FILL** Specifies that the macro is a special cover macro for metal filling purpose.

LEFTEDGETOPBORDER | LEFTEDGEBOTTOMBORDER | RIGHTEDGETOPBORDER |  
RIGHTEDGEBOTTOMBORDER

Specifies a left or right edge endcap cell, which is right next to the LEFTTOPEEDGE/LEFTBOTTOMEDGE or RIGHTTOPEEDGE/RIGHTBOTTOMEDGE cell.

LEFTEVENSITEEDGE | LEFTODDSITEEDGE  
| RIGHTEVENSITEEDGE | RIGHTODDSITEEDGE  
| LEFTTOPEVENSITEEDGE | LEFTTOPODDSITEEDGE  
| RIGHTTOPEVENSITEEDGE | RIGHTTOPODDSITEEDGE  
| LEFTBOTTOOMEVENSITEEDGE | LEFTBOTTOMODDSITEEDGE  
| RIGHTBOTTOOMEVENSITEEDGE | RIGHTBOTTOMODDSITEEDGE  
| LEFTTOPEVENSITECORNER | LEFTTOPODDSITECORNER  
| RIGHTTOPEVENSITECORNER | RIGHTTOPODDSITECORNER  
| LEFTBOTTOOMEVENSITECORNER  
| LEFTBOTTOMODDSITECORNER  
| RIGHTBOTTOOMEVENSITECORNER  
| RIGHTBOTTOMODDSITECORNER

Specifies that the macro with \*EVENSITE\* or \*ODDSITE\* endcap type has even or odd site. These are used to always form even number of sites horizontally, including the endcap cells.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

LEFTTOPEDGE  
LEFTTOPEDGE | RIGHTTOPEDGE | LEFTBOTTOMEDGE |  
LEFTBOTTOMEDGE

Specifies a top or bottom edge endcap cell, which is right next to the LEFTTOPEDGE/LEFTBOTTOMEDGE or RIGHTTOPEDGE/RIGHTBOTTOMEDGE cell. When all four types of \*NEIGHBOR are defined, they are placed in the R0 or MX orientation. If only LEFTTOPEDGE and LEFTBOTTOMEDGE are defined, they can be used by y-flipping for RIGHTTOPEDGE and RIGHTBOTTOMEDGE. However, LEFTTOPEDGE and RIGHTTOPEDGE cannot be x-flipped to replace LEFTBOTTOMEDGE and LEFTBOTTOMEDGE because the power/ground pins cannot be matched.

TOPEDGE | BOTTOMEDGE | LEFTEDGE | RIGHTEDGE  
| LEFTTOPEDGE | RIGHTTOPEDGE  
| LEFTBOTTOMEDGE | RIGHTBOTTOMEDGE  
| LEFTTOPCORNER | RIGHTTOPCORNER  
| LEFTBOTTOMCORNER | RIGHTBOTTOMCORNER}

Specifies that the macro has nwell resided on certain edge or corner of the endcap cells. For example, the TOPEDGE keyword indicates that the endcap cell has n-well on the top edge, while LEFTTOPEDGE indicates that nwell exists on both top and left edges. The LEFTTOPCORNER keyword indicates that nwell only exists on the top left corner.

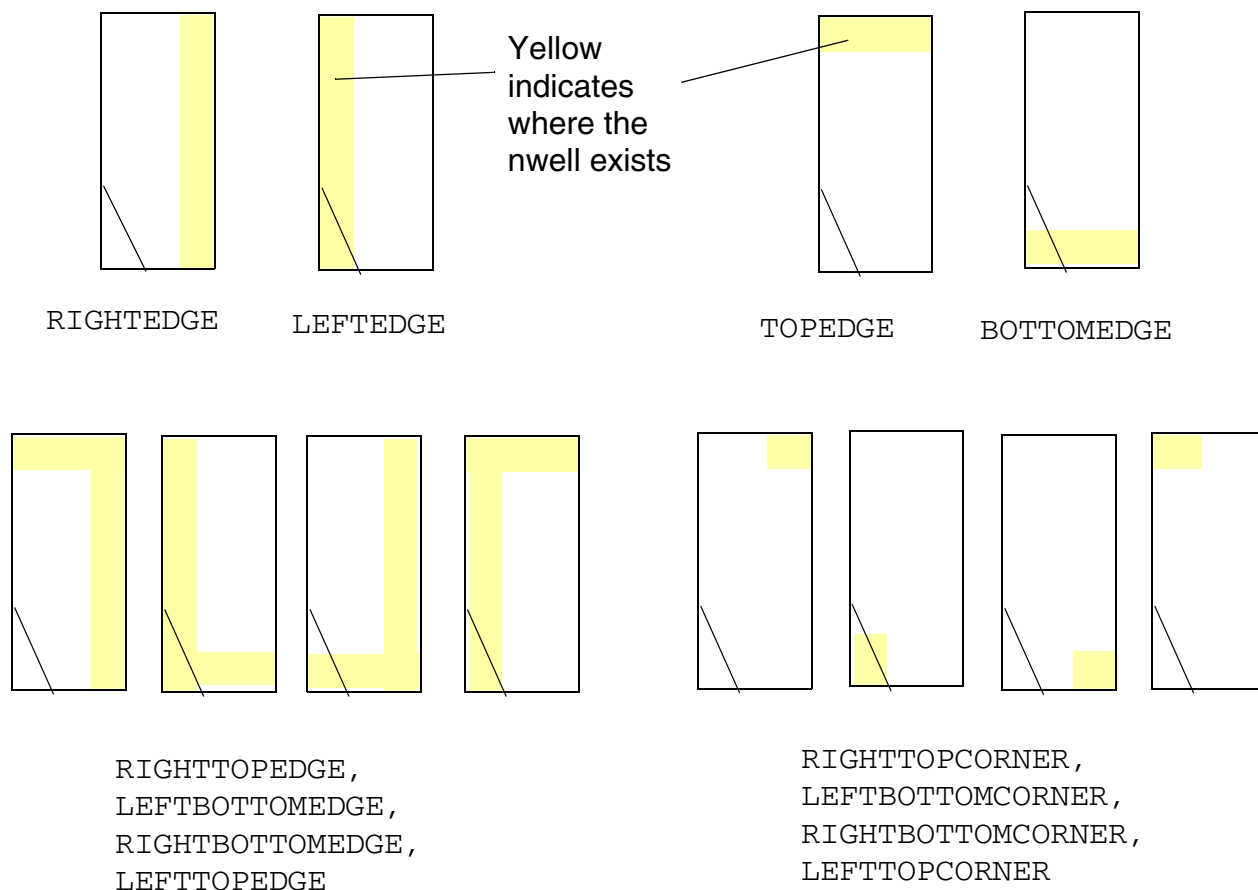
### Class Rule Examples

- The following example indicates that macro A is a cover macro for metal filling purpose:

```
MACRO A
    PROPERTY LEF58_CLASS "CLASS COVER FILL ; " ;
    ...
END A
```

- The following example shows four different set of configurations for class rule:

**Figure 1-299 Illustration of Class Rule**



A library can have cells in all of these configurations, but not likely nor necessarily. At minimum, one configuration from each of the 4 sets must be specified. It is expected that the available configuration from TOPEDGE or BOTTOMEDGE would have a list of cells of different size (similar to fillers), in order to reduce number of cells used to fill up the row.

- The following example shows the use of RIGHTEDGE (RE), TOPEDGE (TE), LEFTTOPEDGE (LTE) and LEFTTOPCORNER (LTC) endcap cells:
  - ❑ RIGHTEDGE in N and FS orientation is used in start of rows
  - ❑ RIGHTEDGE in FN and S orientation is used in end of rows
  - ❑ TOPEDGE in N orientation is used in bottom row of the core or top row right above a block macro (the first row should have N orientation; otherwise, the bottom FS row would be wasted)

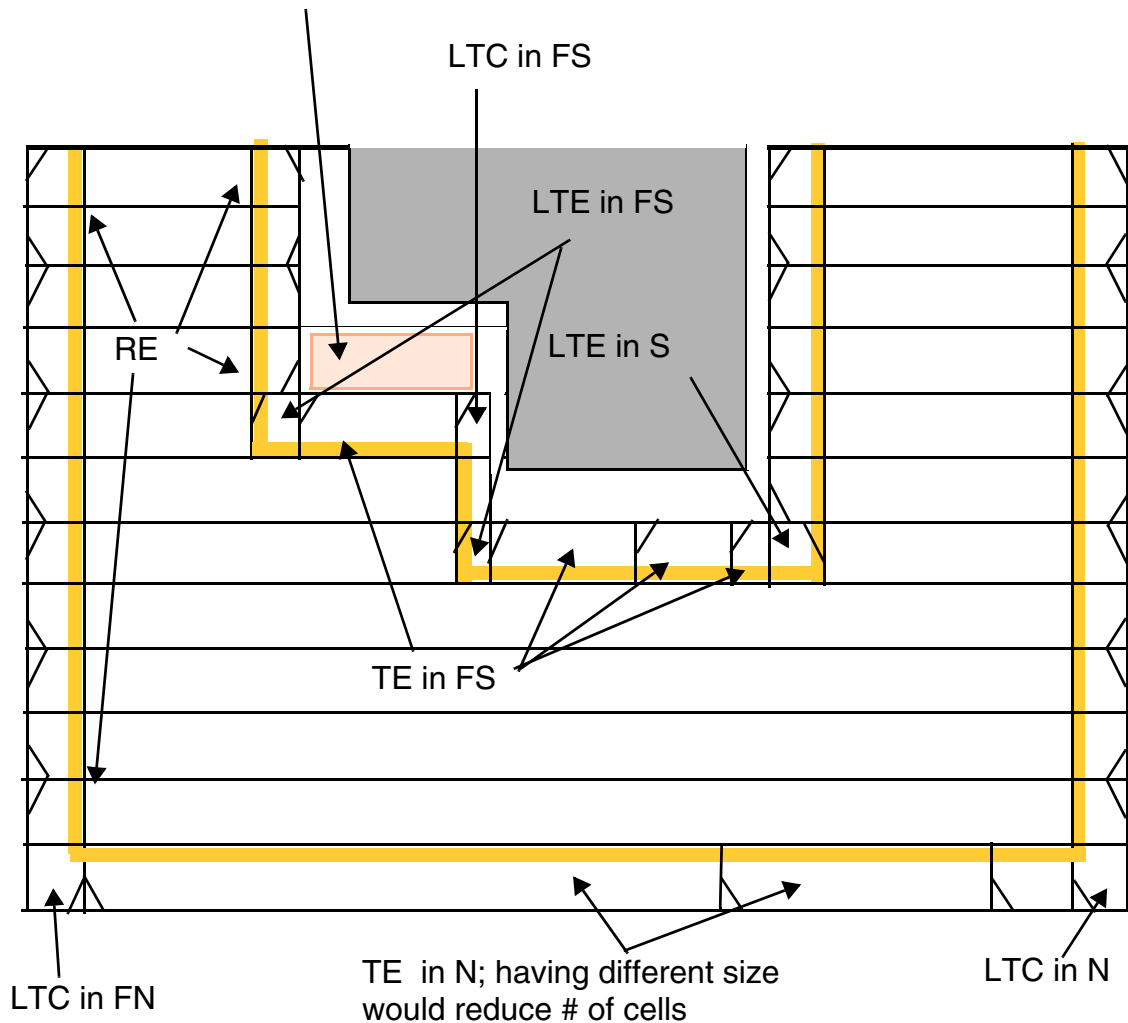
## LEF/DEF 5.8 Language Reference

### LEF Syntax

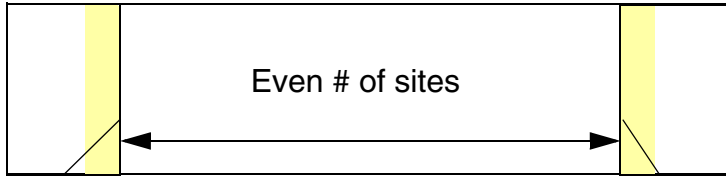
- ❑ TOPEDGE in FS orientation is used in top row of the core or bottom row right below a block macro
- ❑ LEFTTOPEDGE is used at the 4 outer corners in N (left-bottom corner), FN (right-bottom), S (right-top) and FS (left-top) orientation
- ❑ LEFTTOPCORNER is used at the 4 inner corners similarly. Hence, all the cells must have SYMMETRY X Y.

**Figure 1-300 Illustration of Class Rule**

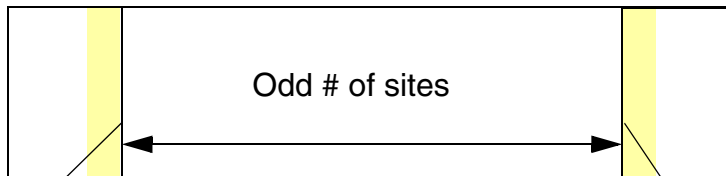
Blockage: this wasted row could be avoided by defining BOTTOMEDGE, 2 edge cell and corner with BOTTOM favor



**Figure 1-301 Illustration of Class Rule**



The above endcap cells must have `LEFTEVENSITEEDGE CLASS`



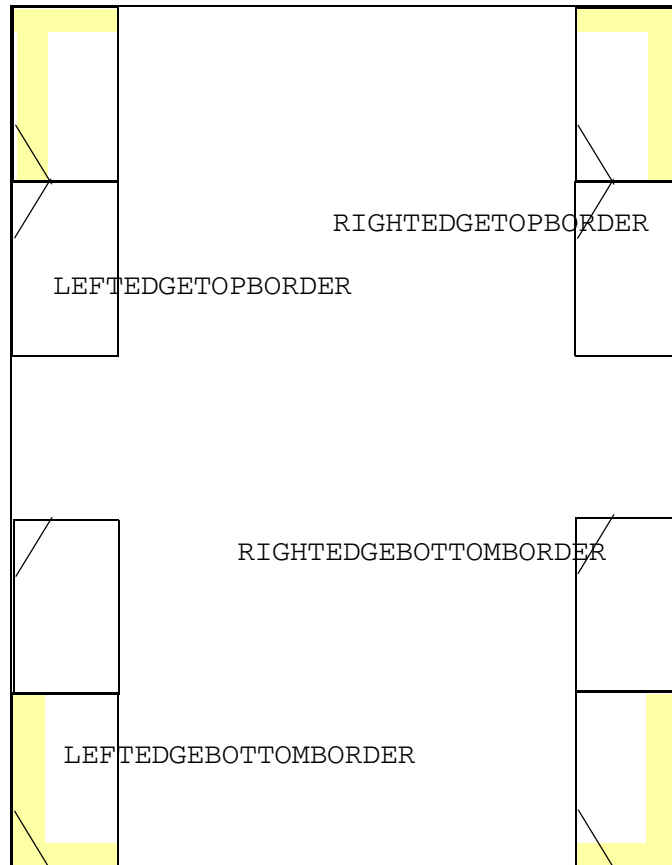
One of the endcap cells must have `LEFTODDSITEEDGE CLASS` & another must have `LEFTEVENSITEEDGE CLASS` such that the # of sites including the endcap cells would be even

## LEF/DEF 5.8 Language Reference

### LEF Syntax

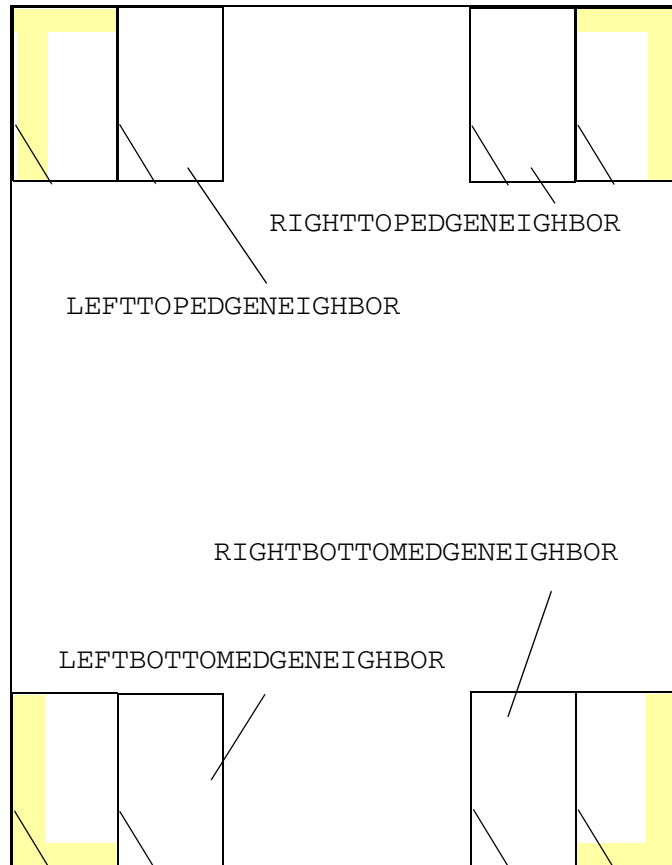
---

**Figure 1-302 Illustration of** LEFTEDGETOPBORDER, LEFTEDGEBOTTOMBORDER, RIGHTEDGETOPBORDER **and** RIGHTEDGEBOTTOMBORDER





**Figure 1-303 Illustration of LEFTTOPEDGE, RIGHTTOPEDGE, LEFTBOTTOMEDGE and RIGHTBOTTOMEDGE**



### ***Constraint Area Type Rule***

The constraint area type rule can be used to define a list of constraint rectangular areas.

You can create a constraint area type rule by using the following property definition:

```
PROPERTY LEF58_CONSTRAINTAREATYPE  
    ""CONSTRAINTAREATYPE typeName {RECT pt pt}...  
    ;...";
```

Where:

```
CONSTRAINTAREATYPE typeName {RECT pt pt}...
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines a list of constraint rectangular areas in `RECT` with respect to the cell origin with type name, *typeName*. Then, rules such as length limit, could be defined on those areas in separate LEF constructs, such as `CONSTRAINTLENGTH`.

*Type*: Float, specified in microns

### Edge Type Rule

The edge type rule can be used to define edge types for right and left edges.

You can create an edge type rule by using the following property definition:

```
PROPERTY LEF58_EDGEType
    "EDGEType {RIGHT | LEFT | TOP | BOTTOM}
        edgeType | BOTHSOURCE | SOURCEDRAIN | DRAINSOURCE | BOTHDRAIN
        | BOTHFLOATING
        [CELLROW cellRow | HALFROW halfRow | RANGE xLow xHigh]
    ;..." ;
```

Where :

CELLROW *cellRow*

Defines which cell row, *cellRow*, the *edgeType* is defined on for multiple height cells, which can only be defined on LEFT or RIGHT edges. The legal value of *cellRow* is from 1 to n, where n is number of cell height of the cell. By default, the *edgeType* is applied to the entire edge covering all cell rows. For example, 1 will apply *edgeType* to the bottom most cell row only.

```
EDGEType { RIGHT | LEFT | TOP | BOTTOM } edgeType
| BOTHSOURCE | SOURCEDRAIN | DRAINSOURCE | BOTHDRAIN | BOTHFLOATING
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines an edge type, *edgeType*, on the specified edge (RIGHT, LEFT, TOP, or BOTTOM) such that a spacing can be defined in CELLEDGESPACINGTABLE by referring to the types. For a macro, you can specify multiple statements on one edge.

BOTHSOURCE, SOURCEDRAIN, DRAINSOURCE, BOTHDRAIN, and BOTHFLOATING are special keywords that can be defined only on the LEFT or RIGHT edges:

- BOTHSOURCE indicates that the entire portion of that edge is a source.
- BOTHDRAIN indicates that the entire portion of that edge is a drain.
- BOTHFLOATING indicates that the entire portion of that edge is floating.
- SOURCEDRAIN indicates that the top portion is a source and the bottom portion is a drain.
- DRAINSOURCE indicates that the top portion is a drain and the bottom portion is a source.

Edges of those special types may or may not have corresponding spacing defined in CELLEDGESPACINGTABLE. They are used for ensuring a common edge type be defined in different cell libraries such that spacing can be defined in some combinations on those edge types in CELLEDGESPACINGTABLE.

See [Figure 1-305](#) on page 573.

HALFROW *halfRow*

Defines the edge type per half row, which can be defined only on the LEFT or RIGHT edges. For a single height cell, the *halfRow* value of 1 would apply the given edge type on the bottom half of the cell while a value of 2 would apply the given edge type on the top half of the cell. Multiple height cells would have the same representation.

*Type:* Integer

RANGE *xLow xHigh*

Defines a partial range of an edge that the type should be labeled. This can only be defined on TOP or BOTTOM edges.

*Type:* Float, specified in microns

You can define multiple `EDGETYPE` statements on an edge, including single height cells, such that different type of constraints can be defined on an edge.

### Edge Type Rule Examples

- The following statement indicates that edge type `GROUP1` contains the right edge of `MACRO1` and `GROUP2` contains the left edge of `MACRO1` and right edge of `MACRO2`:

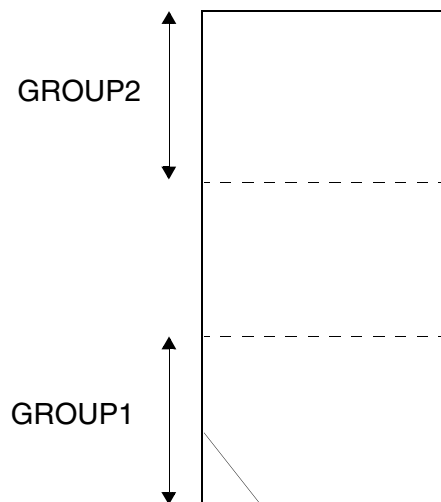
```
MACRO MACRO1
...
PROPERTY LEF58_EDGETYPE "
    EDGETYPE RIGHT GROUP1 ;
    EDGETYPE LEFT GROUP2 ; " ;
END MACRO1

MACRO MACRO2
...
PROPERTY LEF58_EDGETYPE "
    EDGETYPE RIGHT GROUP2 ; " ;
END MACRO2
```

- The following example indicates that `GROUP1` and `GROUP2` are defined on the first and last (third) row on the left edge for the triple height cell:

```
PROPERTY LEF58_EDGETYPE "
    EDGETYPE LEFT GROUP1 CELLROW 1 ;
    EDGETYPE LEFT GROUP2 CELLROW 3 ; " ;
```

**Figure 1-304 Illustration of Macro with Cell Row**



## LEF/DEF 5.8 Language Reference

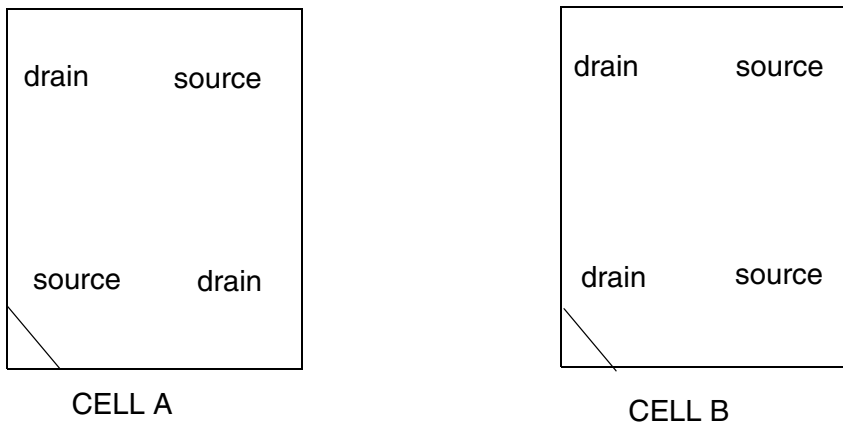
### LEF Syntax

---

- The following example illustrates the edgetype rule with special edge type keywords:

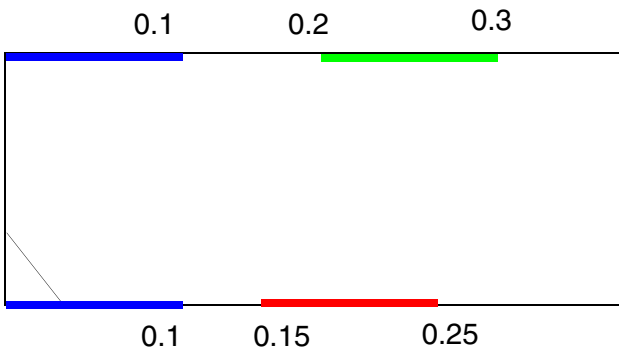
```
MACRO A
...
  PROPERTY LEF58_EDGETYPE
    "EDGETYPE LEFT DRAINSOURCE ;
    EDGETYPE RIGHT SOURCEDRAIN ; " ;
...
END A
MACRO B
...
  PROPERTY LEF58_EDGETYPE "
    EDGETYPE LEFT BOTHDRAIN ;
    EDGETYPE RIGHT BOTHSOURCE ; " ;
...
END B
```

**Figure 1-305 Illustration of Macro with special edge type keywords**



**Figure 1-306 Illustration of Macro with Range**

```
MACRO A
...
PROPERTY LEF58_EDGETYPE "
    EDGETYPE TOP GROUP1 RANGE 0 0.1 ;
    EDGETYPE BOTTOM GROUP1 RANGE 0 0.1 ;
    EDGETYPE TOP GROUP2 RANGE 0.2 0.3 ;
    EDGETYPE BOTTOM GROUP3 RANGE 0.15 0.25 ; " ;
...
END A
```



Blue line indicates GROUP1 edge  
Green line indicates GROUP2 edge  
Red line indicates GROUP3 edge

### ***FOREIGN Rule***

You can use the foreign rule to indicate that the foreign structure of the specified foreign orient should be used if a component of the macro matches one of the given component orientations.

You can create foreign rule by using the following property definition:

```
PROPERTY LEF58_FOREIGN
    "FOREIGN foreignCellName [pt [orient]]
        [COMPORIENT foreignOrientName {compOrient}...]...
    ;... " ;
```

Where :

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

All other keywords are same as the existing macro `FOREIGN` syntax.

```
COMPORIENT foreignOrientName {compOrient}...
```

Specifies that the foreign structure of *foreignOrientName* should be used if a component of the macro matches one of the given component orientations specified using *compOrient*. Otherwise, all the other components will use *foreignCellName*.  
*Values:* N, S, E, W, FN, FS, FE, or FN

### FOREIGN Rule Examples

The following example indicates that the foreign structure of foreign orient `EFG` should be applied if the instance is placed with orientation of `S` or `FS`, and all the other instances use the foreign orient `ABC`.

```
PROPERTY LEF58_FOREIGN
    "FOREIGN ABC COMPORIENT EFG S FS ; " ;
```

### OBS Partial Rule

You can use the OBS partial rule to indicate a partially routing blockage.

You can create a OBS partial rule by using the following property definition:

```
PROPERTY LEF58_OBSPARTIAL
    "OBSPARTIAL {LAYER layerName
    WIDTH width SPACING spacing RECT pt pt}...
    ; " ;
```

Where :

```
OBSPARTIAL {LAYER layerName
WIDTH width SPACING spacing RECT pt pt }...
```

Specifies a partially routing blockage. Any wires in the given `RECT` area on *layerName* must have minimum width of *width* and spacing of *spacing*. In addition, wires cannot change layer within that area.  
*Type:* Float, specified in microns

### OBS Partial Rule Examples

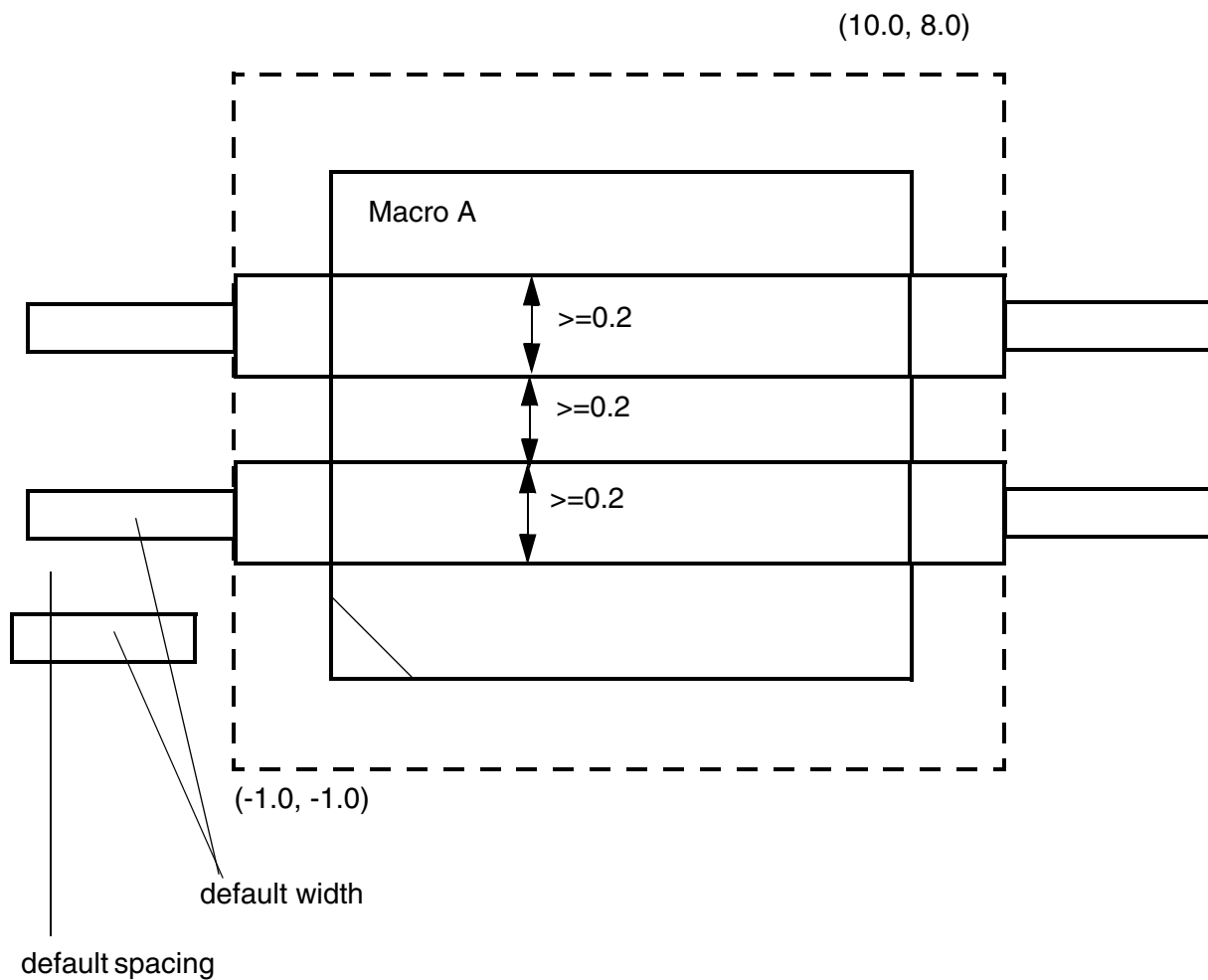
## LEF/DEF 5.8 Language Reference

### LEF Syntax

- The following example in

```
MACRO A
...
PROPERTY LEF58_OBSPARTIAL
  "OBSPARTIAL LAYER M2 WIDTH 0.2 SPACING 0.2
    RECT -1.0 -1.0 10.0 8.0 ; " ;
```

**Figure 1-307 Illustration of OBS Partial Rule**



All of the wires in the dotted area on layer M2 must have minimum width of 0.2 and spacing of 0.2.



### ***OBS Spacing Rule***

You can use the OBS spacing rule to specify spacing for the OBS of the macro.

You can create a OBS spacing rule by using the following property definition:

```
PROPERTY LEF58_OBSSPACING
    "OBSSPACING {MIN | spacing} [LAYER layer]...
    ; " ;
```

Where :

<code>LAYER <i>layer</i></code>	Specifies that the given spacing would be applied on OBS of this macro only on the given <i>layer</i> .
---------------------------------	---------------------------------------------------------------------------------------------------------

<code>OBSSPACING {MIN   <i>spacing</i>}</code>	Specifies that OBS of this macro would either use min spacing if MIN is given or the given <i>spacing</i> value. This is equivalent to using OBS SPACING value for all the OBS shapes in this LEF MACRO. If LAYER is specified, the given spacing is applied on OBS of this macro only on the given <i>layer</i> . If LAYER is not specified, the spacing value applies to all the OBS shapes. Note that this construct must be used with caution because the defined spacing would always be used even on a neighbor wide wire, which may require a larger spacing. <i>Type:</i> Float, specified in microns
------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### **OBS Spacing Rule Examples**

- The following example indicates that all of the OBS on M3 would use 0.2 um while the rest of OBS uses min spacing.

```
PROPERTY LEF58_OBSSPACING `
    OBSSPACING MIN ;
    OBSSPACING 0.2 LAYER M3 ; ` ;
```

- The following example indicates that all of the OBS on M1 and M2 of this macro would use min spacing while M3 OBS would use 0.2um.

```
PROPERTY LEF58_OBSSPACING `
    OBSSPACING MIN LAYER M1 LAYER M2 ;
    OBSSPACING 0.2 LAYER M3 ;` ;
```

### ***Defining Cover Macros***

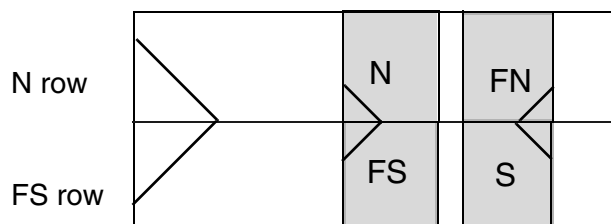
If you define a cover macro with its actual size, some place-and-route tools cannot place the rest of the cells in your design because it uses the cell boundary to check for overlaps. You can resolve this in two ways:

- The easiest way to support a cover macro is to define the cover macro with a small size, for example, 1 by 1.
- If you want to define the cover macro with its actual size, create an overlap layer with the non-routing `LAYER TYPE OVERLAP` statement. You define this overlap layer (cover macro) with the macro obstruction (`OBS`) statement.

### ***Defining Symmetry***

Symmetry statements specify legal orientations for sites and macros. [Figure 1-308](#) on page 578 illustrates the normal orientations for single-height, flipped and abutted rows with standard cells and sites.

**Figure 1-308 Normal Orientations for Single-Height Rows**

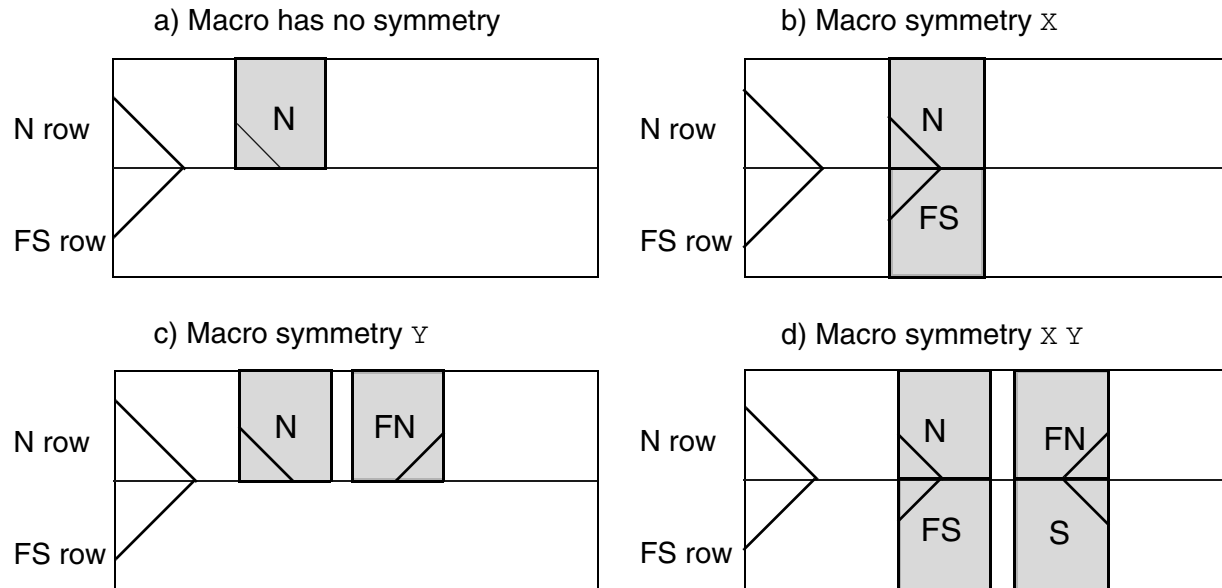


The following examples describe typical combinations of orientations for standard cells. Applications typically create only N (or FS for flipped) row orientations for horizontal standard cell rows; therefore, the examples describe these two rows.

### **Example 1-26 Single-Height Cells**

Single-height cells for flipped and abutted rows should have `SITE` symmetry `Y` and `MACRO` symmetry `X Y`. These specifications allow N and FN macros in N rows, and FS and S macros in FS rows, see [Figure 1-309](#) on page 579. These symmetries work with flipped and abutted rows, as well as rows that are not flipped and abutted, so if the rows are all N orientation, the cells all have N or FN orientation. The extra `MACRO` symmetry of `X` is not required in this case, but causes no problems.

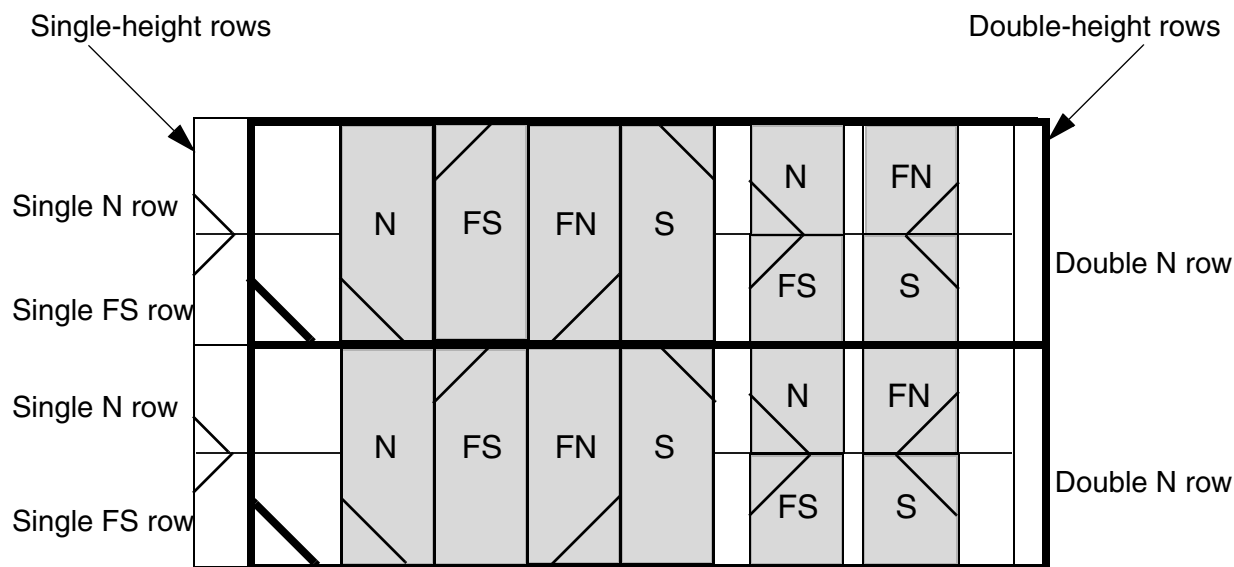
**Figure 1-309 Legal Placements for Row Sites with Symmetry Y**



### Example 1-27 Double-Height Cells

Double-height cells that are intended to align with flipped and abutted single-height rows should have `SITE symmetry X Y` and `MACRO symmetry X Y`. These symmetries allow all four cell orientations (N, FN, FS, and S) to fit inside the double-height row (see [Figure 1-310](#) on page 580). Usually, double-height rows are just N orientation rows that are abutted and aligned with a pair of single-height flipped and abutted rows.

**Figure 1-310 Legal Placements for Single-Height Row Sites with Symmetry Y and Double-Height Row Sites with Symmetry X Y**

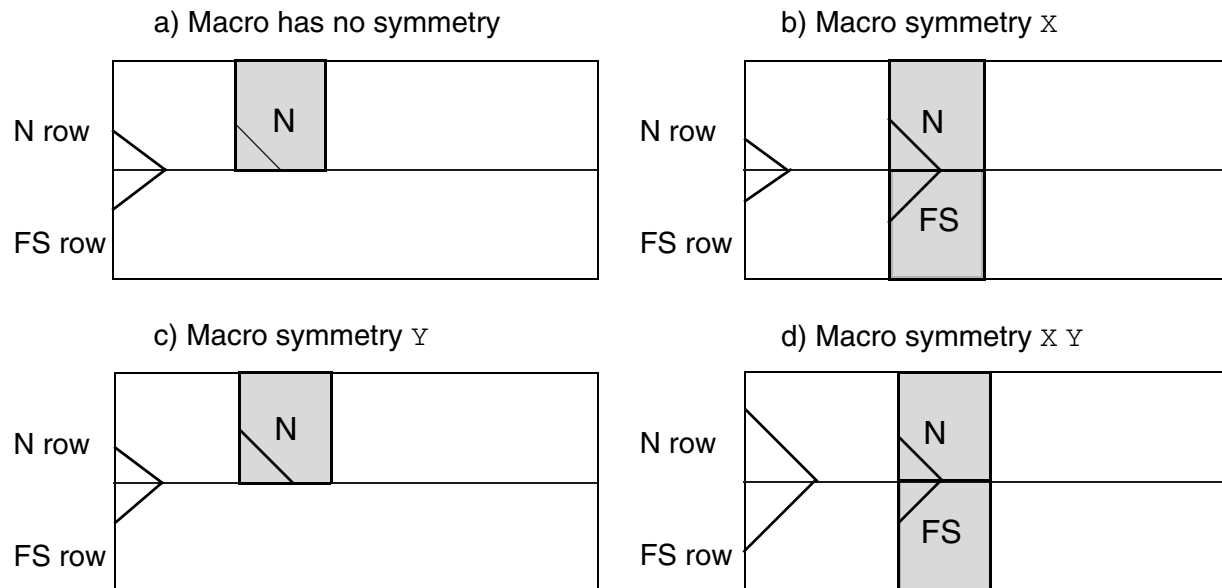


**Note:** The single-height rows are shifted slightly to the left of the double-height rows in the above figure for illustration purposes. In a real design, they should be aligned.

### Example 1-28 Special Orientations

Some single-height cells have special orientation needs. For example, the design requires flipped and abutted rows, but only N and FS orientations are allowed because of the special layout of well taps on the right side of a group of cells that borrow from the left side of the next cell. That is, you cannot place an N and FN cell against each other in one row because only N cells are allowed in an N row. In this case, the `SITE` symmetry should not be defined, and the `MACRO` symmetry should be X. A `MACRO` symmetry of X Y can also be defined because the Y-flipped macros (FN and S orientations) do not match the N or FS rows. See [Figure 1-311](#) on page 581 for the different combinations when the `SITE` has no symmetry.

**Figure 1-311 Legal Placements for Row Sites with No Symmetry**

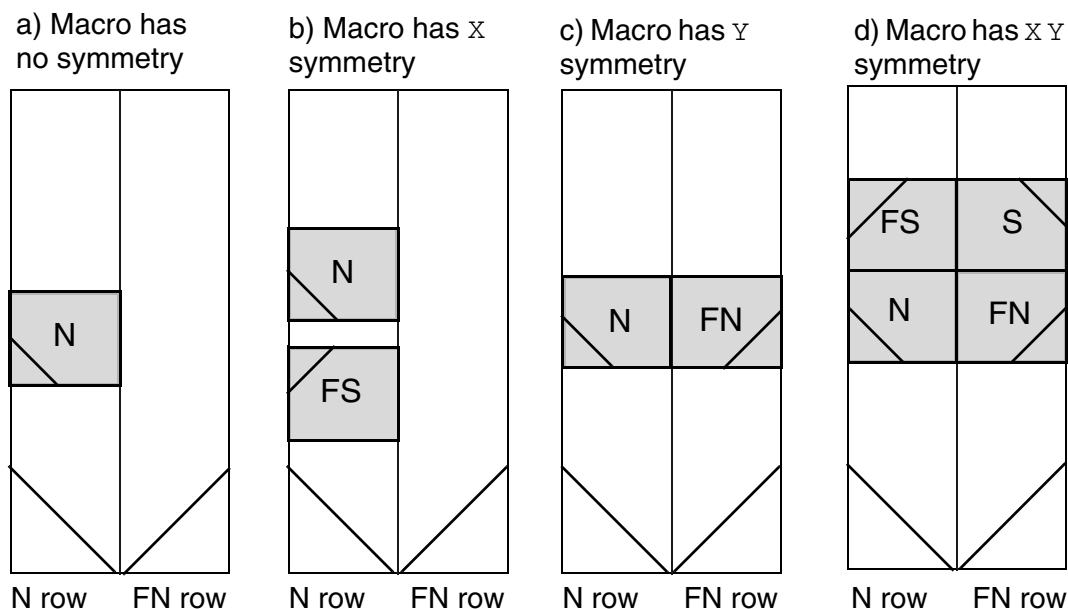


### Example 1-29 Vertical Rows

Vertical rows use N or FN row and site orientations. The flipped, abutted vertical row orientation is N and FN, rather than the horizontal row orientation of N and FS. Otherwise, the meaning of the site symmetries and macro symmetries is the same as those for horizontal rows.

Single-height sites are normally given symmetry  $\times$ , and single-height cells are normally given symmetry  $\times \gamma$ . Example d in [Figure 1-312](#) on page 582 shows the legal placement for a site with symmetry  $\times$ , and the typical standard cell `MACRO` symmetry  $\times \gamma$ .

**Figure 1-312 Legal Placements for Vertical Row Sites With Symmetry X**



## Layer Geometries

```
{ LAYER layerName
    [EXCEPTPGNET]
    [SPACING minSpacing | DESIGNRULEWIDTH value] ;
    [WIDTH width ;]
    { PATH [MASK maskNum] pt ... ;
      | PATH [MASK maskNum] ITERATE pt ... stepPattern ;
      | RECT [MASK maskNum] pt pt ;
      | RECT [MASK maskNum] ITERATE pt pt stepPattern ;
      | POLYGON [MASK maskNum] pt pt pt pt ... ;
      | POLYGON [MASK maskNum] ITERATE pt pt pt pt ... stepPattern ;
    } ...
    | VIA [MASK viaMaskNum] pt viaName ;
    | VIA ITERATE [MASK viaMaskNum] pt viaName stepPattern ;
  } ...
```

Used in the macro obstruction (OBS) and pin port (PIN) statements to define layer geometries in the design.

DESIGNRULEWIDTH *value*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the effective design rule width. If specified, the obstruction or pin is treated as a shape of this width for all spacing checks. If you specify `DESIGNRULEWIDTH`, you cannot specify the `SPACING` argument. As a lot of spacing rules in advanced nodes no longer just rely on wire width, `DESIGNRULEWIDTH` is not allowed for 20nm and below nodes.

*Type:* Float, specified in microns

`EXCEPTPGNET`

Indicates that the obstruction shapes block signal routing, but do *not* block power or ground routing. This can be used to block signal routes that might cause noise, but allow connections to power and ground pins.

`ITERATE`

Creates an array of the `PATH`, `RECT`, `POLYGON`, or `VIA` geometry, as specified by the given step pattern. `ITERATE` specifications simplify the definitions of cover macros. The syntax for *stepPattern* is defined as follows:

`DO numX BY numY STEP spaceX spaceY`

*numX* Specifies the number of columns of points.

*numY* Specifies the number of rows of points.

*spaceX spaceY* Specifies the spacing, in distance units, between the columns and rows of points.

`LAYER layerName`

Specifies the layer on which to place the geometry.

**Note:** For macro obstructions, you can specify cut, implant, or overlap layers.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`MASK maskNum`

Specifies which mask from double- or triple-patterning to use for this shape. The *maskNum* variable must be a positive integer. Most applications only support values of 1, 2, or 3.

Shapes without any defined mask have no mask set (they are considered uncolored). The uncolored PIN shapes can be assigned to an arbitrary mask as long as they do not have a spacing conflict with neighbor objects. The meaning of uncolored OBS shapes depends on the cell. For standard cell MACROs (with a `SITE` that is `CLASS CORE`), the uncolored OBS shapes are considered to be real metal shapes that can be assigned to any mask as long as no mask spacing conflicts occur. For other MACRO types, uncolored OBS shapes are assumed to be abstractions that may be any mask, so other shapes must be spaced far enough away to avoid a violation to any mask shape at that location.

`MASK viaMaskNum`

Specifies which mask for double- or triple-patterning lithography to be applied to via shapes on each layer.

The *viaMaskNum* is a hex-encoded 3 digit value of the form:

`<topMaskNum><cutMaskNum><bottomMaskNum>`

For example, `MASK 113` means the top metal and cut layer *maskNum* is 1, and the bottom metal layer *maskNum* is 3. A value of 0 means the shape on that layer has no mask assignment (is uncolored), so `013` means the top layer is uncolored. If either the first or second digit is missing, they are assumed to be 0, so `013` and `13` means the same thing. Most applications only support *maskNum* values of 0, 1, 2, or 3 for double or triple patterning.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

The *topMaskNum* and *bottomMaskNum* variables specify which mask the corresponding metal shape belongs to. The via-master metal mask values have no effect. For the cut-layer, the *cutMaskNum* defines the mask for the bottommost, and then the leftmost cut. For multi-cut vias, the via-instance cut masks are derived from the via-master cut mask values. The via-master must have a mask defined for all of the cut shapes and every via-master cut mask is "shifted" (1 to 2, 2 to 1 for two mask layers, and 1 to 2, 2 to 3, 3 to 1 for three mask layers) so the lower-left cut matches the *cutMaskNum* value. See [Example 1-31](#) on page 587 .

Similarly, for the metal layer, the *topMaskNum/bottomMaskNum* would define the mask for the bottommost, then leftmost metal shape. For multiple disjoint metal shapes, the via-instance metal masks are derived from the via-master metal mask values. The via-master must have a mask defined for all of the metal shapes, and every via-master metal mask is "shifted" (1->2, 2->1 for two mask layers, 1->2, 2->3, 3->1 for three mask layers) so the lower-left cut matches the *topMaskNum/bottomMaskNum* value.

Shapes without any defined mask that need to be assigned, can be assigned to an arbitrary choice of mask by applications.

PATH *pt*

Creates a path between the specified points, such as *pt1 pt2 pt3*. The path automatically extends the length by half of the current *width* on both endpoints to form a rectangle. (A previous *WIDTH* statement is required.) The line between each pair of points must be parallel to the x or y axis (45-degree angles are not allowed).

You can also specify a path with a single coordinate, in which case a square whose side is equal to the current *width* is placed with its center at *pt*.

POLYGON *pt pt pt pt*

Specifies a sequence of at least three points to generate a polygon geometry. Every polygon edge must be parallel to the x or y axis, or at a 45-degree angle. Each *POLYGON* statement defines a polygon generated by connecting each successive point, and then by connecting the first and last points.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

<code>RECT <i>pt pt</i></code>	Specifies a rectangle, where the two points specified are opposite corners of the rectangle. There is no functional difference between a geometry specified using <code>PATH</code> and a geometry specified using <code>RECT</code> .
<code>SPACING <i>minSpacing</i></code>	<p>Specifies the minimum spacing allowed between this particular <code>OBS</code> and any other shape. While the syntax is shared for both <code>OBS</code> and <code>PIN</code>, it is only intended to be used with <code>OBS</code> shapes. The <i>minSpacing</i> value overrides all other normal <code>LAYER</code>-based spacing rules, including wide-wire spacing rules, end-of-line rules, parallel run-length rules, etc. An <code>OBS</code> with <code>SPACING</code> is not “seen” by any other DRC check, except the simple check for <i>minSpacing</i> to any other routing shape on the same layer.</p> <p>One common application is to put an <code>OBS SPACING 0</code> shape on top of some <code>PIN</code> shapes to restrict the access of a router to other parts of the <code>PIN</code> without the <code>OBS</code> shape. This is sometimes needed for cells with large drive strengths to avoid electromigration problems by restricting the router to connect only to the middle of the output pin.</p> <p>The <i>minSpacing</i> value cannot be larger than the maximum spacing defined in the <code>SPACING</code> or <code>SPACINGTABLE</code> for that layer. Tools may change larger values to the maximum spacing value with a warning.</p>
<code>VIA <i>pt viaName</i></code>	Specifies the via to place, and the placement location.
<code>WIDTH <i>width</i></code>	Specifies the width that the <code>PATH</code> statements use. If you do not specify <i>width</i> , the default width for that layer is used. When you specify a width, that width remains in effect until the next <code>WIDTH</code> or <code>LAYER</code> statement. When another <code>LAYER</code> statement is given, the <code>WIDTH</code> is automatically reset to the default width for that layer.

### Example 1-30 Layer Geometries

The following example shows how to define a set of geometries, first by using `ITERATE` statements, then by using individual `PATH`, `VIA` and `RECT` statements.

The following two sets of statements are equivalent:

```
PATH ITERATE 532.0 534 1999.2 534
DO 1 BY 2 STEP 0 1446 ;
VIA ITERATE 470.4 475 VIABIGPOWER12
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
DO 2 BY 2 STEP 1590.4 1565 ;
RECT ITERATE 24.1 1.5 43.5 16.5
DO 2 BY 1 STEP 20.0 0 ;
PATH 532.0 534 1999.2 534 ;
PATH 532.0 1980 1999.2 1980 ;
VIA 470.4 475 VIABIGPOWER12 ;
VIA 2060.8 475 VIABIGPOWER12;
VIA 470.4 2040 VIABIGPOWER12;
VIA 2060.8 2040 VIABIGPOWER12;
RECT 24.1 1.5 43.5 16.5 ;
RECT 44.1 1.5 63.5 16.5 ;
```

#### Example 1-31 Layer Geometries - multi-mask patterns

The following example shows how to use multi-mask patterns:

```
LAYER M1 ;
    RECT MASK 2 10 10 11 11 ;
LAYER M2 ;
    RECT 10 10 11 11 ;
VIA 5 5 VIA1_1 ;
VIA MASK 031 15 15 VIA1_2 ;
```

This indicates that the:

- M1 rect shape belongs to MASK 2
- M2 rect shape has no mask set
- VIA1\_1 via has no mask set (all the metal and cut shapes have no mask)
- VIA1\_2 via will have:
  - ❑ No mask set for the top metal shape (*topMaskNum* is 0 in the 031 value)
  - ❑ MASK 1 for the bottom metal shape (*botMaskNum* is 1 in the 031 value)
  - ❑ The bottommost, and then the leftmost cut of the via-instance is MASK 3. The mask for the other cuts of the via-instance are derived from the via-master by "shifting" the via-master cut masks to match. So if the via-master's bottomleft cut is MASK 1, then the via-master cuts on MASK 1 become MASK 3 for the via-instance, and similarly cuts on 2 to 1, and cuts on 3 to 2. See [Figure 1-313](#) on page 588.

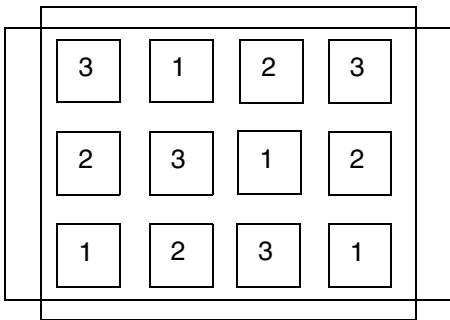
```
LAYER M1 ;
    RECT MASK 2 10 10 11 11 ;
LAYER M2 ;
```

## LEF/DEF 5.8 Language Reference

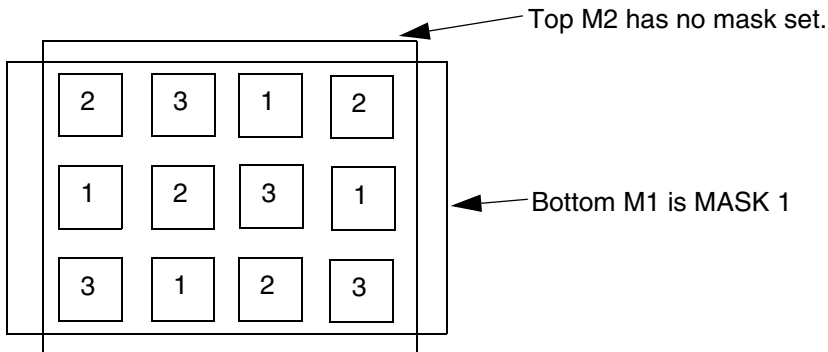
### LEF Syntax

```
RECT 10 10 11 11 ;
VIA 5 5 VIA1_1 ;
VIA MASK 031 15 15 VIA1_2 ;
```

**Figure 1-313 Via-master multi-mask patterns**



Via-master cut masks for VIA1\_2.



Masks for via-instance: VIA MASK 031 15 15 VIA1\_2 ;  
 Bottom M1 is MASK 1. Top M2 has no mask set. Lower-left cut is MASK 3, and other via-instance cut shape masks are derived from via-master cut-masks as shown above by "shifting" the via-master masks to match: 1->3, 2->1, 3->2.

## Macro Obstruction Statement

```
[OBS
{ LAYER layerName
  [EXCEPTPGNET]
  [SPACING minSpacing | DESIGNRULEWIDTH value] ;
  [WIDTH width ;]
  { PATH [MASK maskNum] pt ... ;
    | PATH [MASK maskNum] ITERATE pt ... stepPattern ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
| RECT [MASK maskNum] pt pt ;  
| RECT [MASK maskNum] ITERATE pt pt stepPattern ;  
| POLYGON [MASK maskNum] pt pt pt pt ... ;  
| POLYGON [MASK maskNum] ITERATE pt pt pt pt ... stepPattern ;  
} ...  
| VIA [MASK maskNum] pt viaName ;  
| VIA [MASK maskNum] ITERATE pt viaName stepPattern ;  
} ...
```

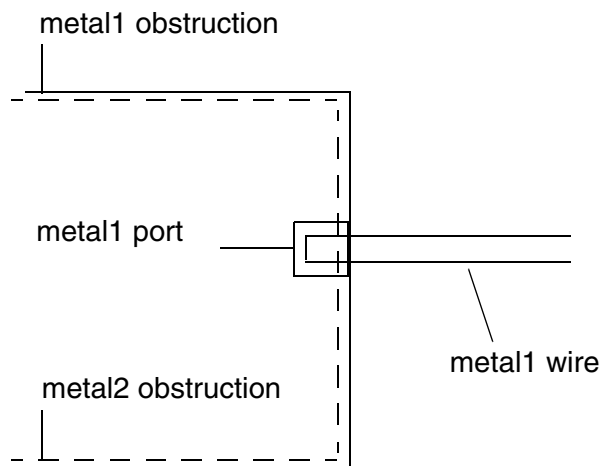
END]

Defines a set of obstructions (also called blockages) on the macro. You specify obstruction geometries using the layer geometry syntax. See “[Layer Geometries](#)” on page 582 for syntax information.

Normally, obstructions block routing, except for when a pin port overlaps an obstruction (a port geometry overrules an obstruction). For example, you can define a large rectangle for a *metal1* obstruction and have *metal1* port in the middle of the obstruction. The port can still be accessed by a via, if the via is entirely inside the port.

In [Figure 1-314](#) on page 589, the router can only access the *metal1* port from the right. If the *metal2* obstruction did not exist, the router could connect to the port with a *metal12* via, as long as the *metal1* part of the via fit entirely inside the *metal1* port.

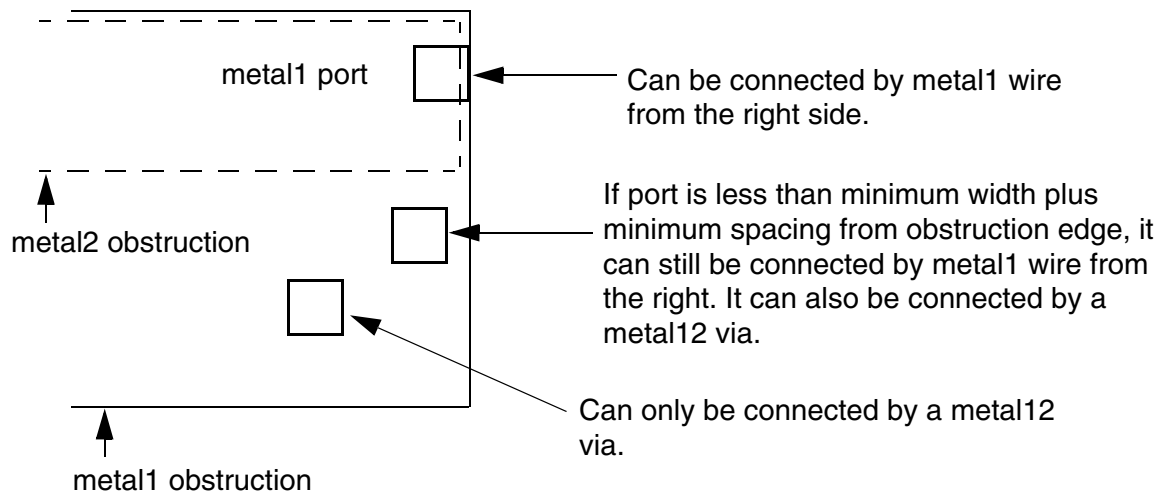
**Figure 1-314**



Routing can also connect to such a port on the same layer if the routing does not cross any obstruction by more than a distance of the total of minimum width plus minimum spacing before reaching the pin. This is because the port geometry is known to be “real,” and any obstruction less than a distance of minimum width plus minimum spacing away from the port is not a real obstruction. If the pin is more than minimum width plus minimum spacing away

from the obstruction edge, the router can only route to the pin from the layer above or below using a via (see [Figure 1-315](#) on page 590).

**Figure 1-315**



If a port is on the edge of the obstruction, a wire can be routed to the port without violations. Pins that are partially covered with obstructions or in apparent violation with nearby obstructions can limit routing options. Even though the violations are not real, the router assumes they are. In these cases, extend each obstruction to cover the pin. The router then accesses the pin as described above.

### ***Benefits of Combining Obstructions***

Significant routing time can be saved if obstructions are simplified. Especially in *metal1*, construct obstructions so that free tracks on the layer are accessible to the router. If most of the routing resource is obstructed, simplify the obstruction modeling by combining small obstructions into a single large obstruction. For example, use the bounding box of all *metal1* objects in the cell, rather than many small obstructions, as the bounding box of the obstruction.

You must be sure to model via obstructions over the rest of the cell properly. A single, large *cut12* obstruction over the rest of the cell can do this in some cases, as when *metal1* resource exists within the cell outside the power buses.

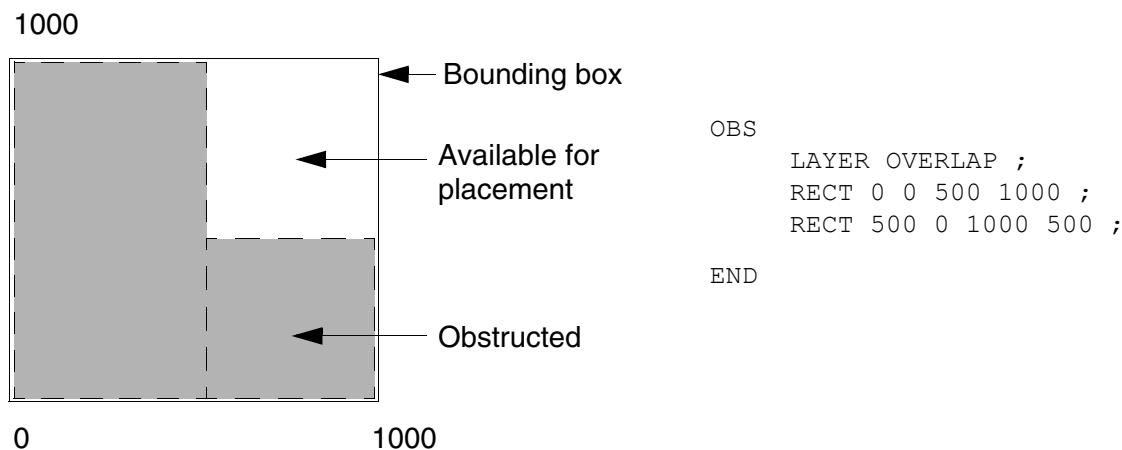
### Rectilinear Blocks

Normally, footprint descriptions in LEF are rectangular. However, it is possible to describe rectilinear footprints using an overlap layer. The overlap layer is defined specifically for this purpose and does not contain any routing.

Describe a rectilinear footprint by setting the `SIZE` of the macro as a whole to a rectangular bounding box, then defining obstructions within the bounding box on the overlap layer. The obstructions on the overlap layer indicate areas within the bounding box which no other macro should overlap. The obstructions should completely cover the rectilinear shape of the macro, but not the portion of the bounding box that might overlap with other macros during placement.

**Note:** Specify the overlaps for the macro using the `OBS` statement. To do this, specify a layer of type `OVERLAP` and then give the overlap geometries, as shown in [Figure 1-316](#) on page 591.

**Figure 1-316**



### Macro Pin Statement

```
[PIN pinName
  [TAPERRULE ruleName ;]
  [DIRECTION {INPUT | OUTPUT [TRISTATE] | INOUT | FEEDTHRU} ;]
  [USE { SIGNAL | ANALOG | POWER | GROUND | CLOCK } ;]
  [NETEXPR "netExprPropName defaultNetName" ;]
  [SUPPLYSENSITIVITY powerPinName ;]
  [GROUNDSENSITIVITY groundPinName ;]
  [SHAPE {ABUTMENT | RING | FEEDTHRU} ;]
  [MUSTJOIN pinName ;]
  {PORT
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[CLASS {NONE | CORE | BUMP} ;]
{layerGeometries} ...
END} ...
[PROPERTY propName propVal ;] ...
[PROPERTY LEF58 MUSTJOINALLPORTS
  "MUSTJOINALLPORTS
  ; " ;]
[PROPERTY LEF58 VIAINMUSTJOIN
  "VIAINMUSTJOIN
  ; " ;]
[PROPERTY LEF58 VIAINPINONLY
  "VIAINPINONLY
  ; " ;]
[ANTENNAPARTIALMETALAREA value [LAYER layerName] ;] ...
[ANTENNAPARTIALMETALSIDEAREA value [LAYER layerName] ;] ...
[ANTENNAPARTIALCUTAREA value [LAYER layerName] ;] ...
[ANTENNADIFFAREA value [LAYER layerName] ;] ...
[ANTENNAMODEL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4} ;] ...
[ANTENNAGATEAREA value [LAYER layerName] ;] ...
[ANTENNAMAXAREACAR value LAYER layerName ;] ...
[ANTENNAMAXSIDEAREACAR value LAYER layerName ;] ...
[ANTENNAMAXCUTCAR value LAYER layerName ;] ...

END pinName]
```

Defines pins for the macro. PIN statements must be included in the LEF specification for each macro. All pins, including VDD and VSS, must be specified. The first pin listed becomes the first pin in the database. List the pins in the following order:

- Netlist pins, including inout pins, output pins, and input pins
- Power and ground pins
- Mustjoin pins

**ANTENNADIFFAREA** *value* [LAYER *layerName*]

Specifies the diffusion (diode) area, in micron-squared units, to which the pin is connected on a layer. If you do not specify a layer name, the value applies to all layers. For more information on process antenna calculation, see [Appendix C, "Calculating and Fixing Process Antenna Violations."](#)

**ANTENNAGATEAREA** *value* [LAYER *layerName*]

Specifies the gate area, in micron-squared units, to which the pin is connected on a layer. If you do not specify a layer name, the value applies to all layers. For more information on process antenna calculation, see [Appendix C, "Calculating and Fixing Process Antenna Violations."](#)



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**ANTENNAMAXAREACAR** *value* LAYER *layerName*

For hierarchical process antenna effect calculation, specifies the maximum cumulative area ratio value on the specified *layerName*, using the metal area at or below the current pin layer, excluding the pin area itself. This is used to calculate the actual cumulative antenna ratio on the pin layer, or the layer above it.

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

**ANTENNAMAXCUTCAR** *value* LAYER *layerName*

For hierarchical process antenna effect calculation, specifies the maximum cumulative antenna ratio value on the specified *layerName*, using the cut area at or below the current pin layer, excluding the pin area itself. This is used to calculate the actual cumulative antenna ratio for the cuts above the pin layer.

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

**ANTENNAMAXSIDEAREACAR** *value* LAYER *layerName*

For hierarchical process antenna effect calculation, specifies the maximum cumulative antenna ratio value on the specified *layerName*, using the metal side wall area at or below the current pin layer, excluding the pin area itself. This is used to calculate the actual cumulative antenna ratio on the pin layer or the layer above it.

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

**ANTENNAMODEL** {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}

Specifies the oxide model for the pin. If you specify an **ANTENNAMODEL** statement, the value affects all **ANTENNAGATEAREA** and **ANTENNA\*CAR** statements for the pin that follow it until you specify another **ANTENNAMODEL** statement. The **ANTENNAMODEL** statement does not affect **ANTENNAPARTIAL\*AREA** and **ANTENNADIFFAREA** statements because they refer to the total metal, cut, or diffusion area connected to the pin, and do not vary with each oxide model.

**Default:** OXIDE1, for a new PIN statement

Because LEF is often used incrementally, if an **ANTENNA** statement occurs twice for the same oxide model, the last value specified is used.

For most standard cells, there is only one value for the **ANTENNAPARTIAL\*AREA** and **ANTENNADIFFAREA** values per pin; however, for a block with six routing layers, it is possible to have six different **ANTENNAPARTIAL\*AREA** values and six different **ANTENNAPINDIFFAREA** values per pin. It is also possible to have six different **ANTENNAPINGATEAREA** and **ANTENNAPINMAX\*CAR** values for each oxide model on each pin.

### Example 1-32 Pin Antenna Model

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

The following example describes oxide model information for pins IN1 and IN2.

```
MACRO GATE1
  PIN IN1
    ANTENNADIFFAREA 1.0 ;           #not affected by ANTENNAMODEL
    ...
    ANTENNAMODELOXIDE OXIDE1 ;      #OXIDE1 not required, but is good
                                     #practice
    ANTENNAGATEAREA 1.0 ;           #OXIDE1 gate area
    ANTENNAMAXAREACAR 50.0 LAYER m1 ; #metal1 CAR value
    ...
    ANTENNAMODEL OXIDE2 ;           #OXIDE2 starts here
    ANTENNAGATEAREA 3.0 ;           #OXIDE2 gate area
    ...
  PIN IN2
    ANTENNADIFFAREA 2.0 ;           #not affected by ANTENNAMODEL
    ANTENNAPARTIALMETALAREA 2.0 LAYER m1 ;
    ...
    #no OXIDE1 specified for this pin
    ANTENNAMODEL OXIDE2 ;
    ANTENNAGATEAREA 1.0 ;
    ...
```

**ANTENNAPARTIALCUTAREA** *value* [LAYER *layerName*]

Specifies the partial cut area above the current pin layer and inside the macro cell on the layer. For a hierarchical design, only the cut layer above the I/O pin layer is needed for partial antenna ratio calculation. If you do not specify a layer name, the value applies to all layers.

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

**ANTENNAPARTIALMETALAREA** *value* [LAYER *layerName*]

Specifies the partial metal area connected directly to the I/O pin and the inside of the macro cell on the layer. For a hierarchical design, only the same metal layer as the I/O pin, or the layer above it, is needed for partial antenna ratio calculation. If you do not specify a layer name, the value applies to all layers.

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

**Note:** Metal area is calculated by adding the pin’s geometric metal area and the ANTENNAPARTIALMETALAREA value.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

ANTENNAPARTIALMETALSIDEAREA *value* [LAYER *layerName*]

Specifies the partial metal side wall area connected directly to the I/O pin and the inside of the macro cell on the layer. For a hierarchical design, only the same metal layer as the I/O pin or the layer above is needed for partial antenna ratio calculation. If you do not specify a layer name, the value applies to all layers.

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

DIRECTION {INPUT | OUTPUT [TRISTATE] | INOUT | FEEDTHRU}

Specifies the pin type. Most current tools do not usually use this keyword. Typically, pin directions are defined by timing library data, and not from LEF.

**Default:** INPUT

**Value:** Specify one of the following:

INPUT	Pin that accepts signals coming into the cell.
OUTPUT [TRISTATE]	Pin that drives signals out of the cell. The optional TRISTATE argument indicates tristate output pins for ECL designs.
INOUT	Pin that can accept signals going either in or out of the cell.
FEEDTHRU	Pin that goes completely across the cell.

GROUNDSENSITIVITY *groundPinName*

Specifies that if this pin is connected to a tie-low connection (such as 1'b0 in Verilog), it should connect to the same net to which *groundPinName* is connected.

*groundPinName* must match a pin on this macro that has a USE GROUND attribute.

The ground pin definition can follow later in this MACRO statement; it does not have to be defined before this pin definition. For an example, see [Example 1-33](#) on page 596.

**Note:** GROUNDSENSITIVITY is useful only when there is more than one ground pin in the macro. By default, if there is only one USE GROUND pin, then the tie-low connections are already implicitly defined (that is, tie-low connections are connected to the same net as the one ground pin).

MUSTJOIN *pinName*

Specifies the name of another pin in the cell that must be connected with the pin being defined. MUSTJOIN pins provide connectivity that must be made by the router. In the LEF file, each pin referred to must be defined before the referring pin. The remaining MUSTJOIN pins in the set do not need to be defined contiguously.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**Note:** `MUSTJOIN` pin names are never written to the DEF file; they are only used by routers to add extra connection points during routing.

`MUSTJOIN` pins have the following restrictions:

- A set of `MUSTJOIN` pins cannot have more than one schematic pin.
- Nonschematic `MUSTJOIN` pins must be defined after all other pins.

Schematic and nonschematic `MUSTJOIN` pins are handled in slightly different ways. For schematic `MUSTJOIN` pins, the pins are added to the pin set for the (unique) net associated with the ring for each component instance of the macro. The net is routed in the usual manner, and routing data for the `MUSTJOIN` pins are included in routing data for the net.

The mustjoin routing is not necessarily performed before the rest of the net. Timing relations should not be given for `MUSTJOIN` pins, and internal mustjoin routing is modeled as lumped capacitance at the schematic pin.

Nonschematic `MUSTJOIN` pin sets get routed in the usual manner. However, when the DEF file is outputted, routing data is reported in the `NETS` section of the file as follows:

```
MUSTJOIN compName pinName + regularWiring ;
```

Here, *compName* is the component and *pinName* is an arbitrary pin in the set. You can also use the preceding to input prewiring for the `MUSTJOIN` pin, using `FIXED` or `COVER`.

```
NETEXPR "netExprPropName defaultNetName"
```

Specifies a net expression property name (such as `power1` or `power2`) and a default net name. If *netExprPropName* matches a net expression property in the netlist (such as in Verilog, VHDL, or OpenAccess), then the property is evaluated, and the software identifies a net to which to connect this pin. If this property does not exist, *defaultNetName* is used for the net name.

*netExprPropName* must be a simple identifier in order to be compatible with other languages, such as Verilog and CDL. Therefore, it can only contain alphanumeric characters, and the first character cannot be a number. For example, `power2` is a legal name, but `2power` is not. You cannot use characters such as `$` and `!`. The *defaultName* can be any legal DEF net name.

### Example 1-33 Net Expression and Supply Sensitivity

The following statement defines sensitivity and net expression values for four pins on the macro `myMac`:

```
MACRO myMac
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
...
PIN in1
    ...
    SUPPLYSENSITIVITY vddpin1 ; #If in1 is 1'b1, use net connected to vddpin1.
                                #Note that no GROUNDSENSITIVITY is needed
                                #because only one ground pin exists.
                                #Therefore, 1'b0 implicitly means net from
                                #pin gndpin.
    ...
END in1

PIN vddpin1
    ...
    NETEXPR "power1 VDD1" ; #If power1 net expression is defined in the
                            #netlist, use it to find the net connection. If
                            #not, use net VDD1.
    ...
END vddpin1
PIN vddpin2
    ...
    NETEXPR "power2 VDD2" ; #If power2 net expression is defined in the
                            #netlist, use it to find the net connection.If
                            #not, use net VDD2.
    ...
END vddpin2
PIN gndpin
    ...
    NETEXPR "gnd1 GND" ; #If gnd1 net expression is defined in the
                        #netlist, use it to find the net connection. If
                        #not, use net GND.
    ...
END gndpin
...
END myMac

PIN pinName
```

Specifies the name for the library pin.

#### PORT

Starts a pin port statement that defines a collection of geometries that are electrically equivalent points (strongly connected). A pin can have multiple ports. Each **PORT** of the same **PIN** is considered weakly connected to the other **PORTs**, and should already be connected inside the **MACRO** (often through a resistive path).

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Strongly connected shapes (that is, multiple shapes of one `PORT`) indicate that a signal router is allowed to connect to one shape of the `PORT`, and continue routing from another shape of the same `PORT`.

Weakly connected shapes (that is, separate `PORTS` of the same `PIN`) are assumed to be connected through resistive paths inside the `MACRO` that should not be used by routers. The signal router should connect to one or the other `PORT`, but not both.

Power routers should connect to every `PORT` statement, if there is more than one for a given `PIN`. For example, if a block has several `PORTS` on the boundary for the `VSS` `PIN`, each `PORT` should be connected by the power router.

The syntax for describing pin port statements is defined as follows:

```
{PORT
  [CLASS {NONE | CORE | BUMP} ;]
  {layerGeometries} ...
END} ...
```

```
CLASS {NONE | CORE | BUMP}
```

Specifies the port type.

**Default:** NONE

A port can be one of the following:

**BUMP**—Specifies the port is a bump connection point. A bump port should only be connected by routing to a bump (normally a `MACRO CLASS COVER BUMP` cell).

**CORE**—Specifies the port is a core ring connection point. A core port is used only on power and ground I/O drivers used around the periphery. The core port indicates which power or ground port to connect to a core ring for the chip (inside the I/O pads).

**NONE**—Specifies the port is a default port that is connected by normal “default” routing. `NONE` is the default value if no `PORT CLASS` statement is specified.

```
layerGeometries
```

Defines port geometries for the pin. You specify port geometries using layer geometries syntax. See “[Layer Geometries](#)” on page 582 for syntax information.

```
PROPERTY propName propVal
```

Specifies a numerical or string value for a pin property defined in the `PROPERTYDEFINITIONS` statement. The `propName` you specify must match the `propName` listed in the `PROPERTYDEFINITIONS` statement.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### SHAPE

Specifies a pin with special connection requirements because of its shape.

*Value:* Specify one of the following:

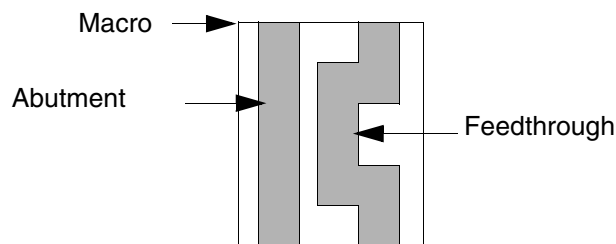
ABUTMENT	Pin that goes straight through cells with a regular shape and connects to pins on adjoining cells without routing.
RING	Pin on a large block that forms a ring around the block to allow connection to any point on the ring. Cover macro special pins also typically have shape RING.
FEEDTHRU	Pin with an irregular shape with a jog or neck within the cell.

Figure 1-317 on page 599 shows an example of an abutment and a feedthrough pin.

**Note:** When you define feedthrough and abutment pins for use with power routing, you must do the following:

- Feedthrough pin widths must be the same on both edges and consistent with the routing width used with the power route commands.
- Feedthrough pin centers on both edges must align for successful routing.
- Power pins in fork shapes must be represented in two ports and be defined as a feedthrough shape. In most other cases, power pin geometries do not represent more than one port.
- An abutment pin must have at least one geometric rectangle with layer and width consistent with the values specified in the power route commands.

**Figure 1-317**



#### SUPPLYSENSITIVITY *powerPinName*

Specifies that if this pin is connected to a tie-high connection (such as 1'b1 in Verilog), it should connect to the same net to which *powerPinName* is connected.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

*powerPinName* must match a pin on this macro that has a `USE POWER` attribute. The power pin definition can follow later in this `MACRO` statement; it does not have to be defined before this pin definition. For an example, see [Example 1-33](#) on page 596.

**Note:** `SUPPLYSENSITIVITY` is useful only when there is more than one power pin in the macro. By default, if there is only one `USE POWER` pin, then the tie-high connections are already implicitly defined (that is, tie-high connections are connected to the same net as the one power pin).

`TAPERRULE ruleName`

Specifies the nondefault rule to use when tapering wires to the pin.

`USE {ANALOG | CLOCK | GROUND | POWER | SIGNAL}`

Specifies how the pin is used. Pin use is required for timing analysis.

**Default:** `SIGNAL`

**Value:** Specify one of the following:

<code>ANALOG</code>	Pin is used for analog connectivity.
<code>CLOCK</code>	Pin is used for clock net connectivity.
<code>GROUND</code>	Pin is used for connectivity to the chip-level ground distribution network.
<code>POWER</code>	Pin is used for connectivity to the chip-level power distribution network.
<code>SIGNAL</code>	Pin is used for regular net connectivity.

### ***Must Join All Ports Rule***

You can use the must join all ports rule to specify that all ports must be connected individually.

You can create a must join all ports rule by using the following property definition:

```
PROPERTY LEF58_MUSTJOINALLPORTS
    "MUSTJOINALLPORTS
    ; " ;
```

Where:



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### MUSTJOINALLPORTS

Specifies that all of the ports must be connected individually. If there are multiple shapes per ports, at least one of the shapes of the port needs to be connected. However, tools typically would connect to only one of the shapes.

For example:

```
PIN A
...
PROPERTY LEF58_MUSTJOINALLPORTS "
    MUSTJOINALLPORTS ; " ;
PORT
    LAYER M1 ;
        RECT 1 1 2 2 ;
        RECT 4 4 5 5 ;
    END
PORT
    LAYER M2 ;
        RECT 2 2 3 3 ;
    END
...
END A
```

means that the router must connect to at least one of the two M1 shapes and to the M2 shape individually. However, tools typically would connect to exactly one of the two M1 shapes and to the M2 shape individually.

#### ***Via In Must Join Rule***

You can create a via in must join rule by using the following property definition:

```
PROPERTY LEF58_VIAINMUSTJOIN
    "VIAINMUSTJOIN
    ; " ;
```

Where:

#### VIAINMUSTJOIN

Specifies that two stacked vias must be used to connect to the must-join pins and all of the cuts of these two stacked vias must be within the boundary box of the must-join pins. For example, if the must-join pins have multiple M1 shapes, the cuts of the stacked vias to M3 must be within the boundary box formed by the M1 pin shapes.

### ***Via In Pin Only Rule***

You can use the via in pin only rule to specify that vias must be dropped inside the original pin shapes to connect to the pin.

You can create a via in pin only rule by using the following property definition:

```
PROPERTY LEF58_VIAINPINONLY
    "VIAINPINONLY
; "
```

Where:

VIAINPINONLY

Specifies that at least one cut of a via must be dropped inside the original pin shapes to connect to the pin, and a planar connection to the pin is not allowed. In some advanced nodes, the pin shapes can be extended for metal alignment purpose. For a single-cut via, it means that the cut would be in the original pin shapes. For a multiple-cut via, particularly with the via pillar having multiple fingers, only one cut needs to be in the original pin shapes and the rest of the via pillar structure can be outside the original pin shapes.

### **Manufacturing Grid**

```
[MANUFACTURINGGRID value ;]
```

Defines the manufacturing grid for the design. The manufacturing grid is used for geometry alignment. When specified, shapes and cells are placed in locations that snap to the manufacturing grid.

*value*

Specifies the value for the manufacturing grid, in microns. *value* must be a positive number.

*Type:* Float

### **Maximum Via Stack**

```
[MAXVIASTACK value [RANGE bottomLayer topLayer] ;]
```

Specifies the maximum number of single-cut stacked vias that are allowed on top of each other (that is, in one continuous stack). A via is considered to be in a stack with another via if the cut of the first via overlaps any part of the cut of the second via. A double-cut or larger via interrupts the stack. For example, a via stack consisting of single *via12*, single *via23*, double-

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

cut *via34*, and single *via45* has a single-cut stack of height 2 for *via12* and *via23*, and a single-cut stack of height 1 for *via45* because the full stack is broken up by double-cut *via34*.

The `MAXVIASTACK` statement should follow the `LAYER` statements in the LEF file; however, it is not attached to any particular layer. You can specify only one `MAXVIASTACK` statement in a LEF file.

`RANGE bottomLayer topLayer`

Specifies a range of layers for which the maximum stacked via rule applies. If you do not specify a range, the value applies for all layers.

`value`

Specifies the maximum allowed number of single-cut stacked vias.

*Type:* Integer

#### Example 1-34 Maximum Via Stack Statement

The following `MAXVIASTACK` statement specifies that only four stacked vias are allowed on top of each other. This rule applies to all layers.

```
LAYER metal9
...
END LAYER
```

```
MAXVIASTACK 4 ;
```

If you specify the following statement instead, the stacked via limit applies only to layers *metal1* through *metal7*.

```
MAXVIASTACK 4 RANGE m1 m7 ;
```

#### Nondefault Rule

```
[NONDEFAULTRULE ruleName
  [HARDSPACING ;]
  {LAYER layerName
    WIDTH width ;
    [DIAGWIDTH diagWidth ;]
    [SPACING minSpacing ;]
    [WIREEXTENSION value ;]
  END layerName} ...
  [VIA viaStatement] ...
  [USEVIA viaName ;] ...
  [USEVIARULE viaRuleName ;] ...
  [MINCUTS cutLayerName numCuts ;] ...
  [PROPERTY propName propValue ;] ...
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[PROPERTY LEF58 USEVIACUTCLASS
"USEVIACUTCLASS cutLayerName className
  [ROWCOL numCutRows numCutCols]
  ;... " ;]
END ruleName]
```

Defines the wiring width, design rule spacing, and via size for regular (signal) nets. You do not need to define cut layers for the nondefault rule.

Some tools have limits on the total number of nondefault rules they can store. This limit can be as low as 30; however most tools that support 90 nanometer rules (that is, LEF 5.5 and newer) can handle at least 255.

**Note:** Use the `VIA` statement to define vias for nondefault wiring.

DIAGWIDTH *diagWidth*

Specifies the diagonal width for *layerName*, when 45-degree routing is used.

*Default:* The minimum width value (`WIDTH minWidth`)

*Type:* Float, specified in microns

HARDSPACING

Specifies that any spacing values that exceed the LEF `LAYER` spacing requirements are “hard” rules instead of “soft” rules. By default, routers treat extra spacing requirements as soft rules that are high cost to violate, but not real spacing violations. However, in certain situations, the extra spacing should be treated as a hard, or real, spacing violation, such as when the route will be modified with a post-process that replaces some of the extra space with metal.

LAYER *layerName* ... END *layerName*

Specifies the layer for the various width and spacing values. This layer must be a routing layer. Every routing layer must have a `WIDTH` keyword and value specified. All other keywords are optional.

MINCUTS *cutLayerName* *numCuts*

Specifies the minimum number of cuts allowed for any via using the specified cut layer. Routers should only use vias (generated or predefined fixed vias) that have at least *numCuts* cuts in the via.

*Type:* (*numCuts*) Positive integer

NONDEFAULTRULE *ruleName*

Specifies a name for the new routing rule. The name `DEFAULT` is reserved for the default routing rule used by most nets. The default routing rule is constructed automatically from the LEF `LAYER` statement `WIDTH`, `DIAGWIDTH`, `SPACING`, and `WIREEXTENSION` values, from the LEF `VIA` statement (any vias with the `DEFAULT` keyword), and from the LEF

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**VIARULE** statement (any via rules with the **DEFAULT** keyword). If you specify **DEFAULT** for *ruleName*, the automatic creation is overridden, and the default routing rule is defined directly from this rule definition.

**PROPERTY** *propName propName* *propValue*

Specifies a numerical or string value for a nondefault rule property defined in the **PROPERTYDEFINITIONS** statement. The *propName* you specify must match the *propName* listed in the **PROPERTYDEFINITIONS** statement.

### Use Via Cut Class Rule

You can use the use via cut class only rule to specify the cut class to be used with the routing rule.

You can create a use via cut class rule by using the following property definition:

```
PROPERTY LEF58_USEVIACUTCLASS
    USEVIACUTCLASS cutLayerName className
        [ROWCOL numCutRows numCutCols]
        ;... " ;
```

Where:

```
USEVIACUTCLASS cutLayerName className
    [ROWCOL numCutRows numCutCols]
```

Specifies *className* cuts on *cutLayerName*, which must be a cut layer, to be used with this routing rule. **ROWCOL** specifies the number of cut rows and columns of a multiple-cut via of the cut class *className* to be used with this routing rule. By default, a single cut via or the given cut number defined in **MINCUTS** is used.

**SPACING** *minSpacing*

Specifies the recommended minimum spacing for *layerName* of routes using this **NONDEFAULTRULE** to other geometries. If the spacing is given, it must be at least as large as the foundry minimum spacing rules defined in the **LAYER** definitions. Routers should attempt to meet this recommended spacing rule; however, the spacing rule can be relaxed to the foundry spacing rules along some parts of the wire if the routing is very congested, or if it is difficult to reach a pin.

Adding extra space to a nondefault rule allows a designer to reduce cross-coupling capacitance and noise, but a clean route with no actual foundry spacing violations will still be allowed, unless the **HARDSPACING** statement is specified.

*Type:* Float, specified in microns

**USEVIA** *viaName*

Specifies a previously defined via from the LEF **VIA** statement, or a previously defined **NONDEFAULTRULE** via to use with this routing rule.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

USEVIARULE *viaRuleName*

Specifies a previously defined VIARULE GENERATE rule to use with this routing rule. You cannot specify a rule from a VIARULE without a GENERATE keyword.

VIA *viaStatement*

Defines a new via. You define nondefault vias using the same syntax as default vias. For syntax information, see “Via” on page 616. All vias, default and nondefault, must have unique via names. If you define more than one via for a rule, the router chooses which via to use.

**Note:** Defining a new via is no longer recommended, and is likely to become obsolete. Instead, vias should be predefined in a LEF VIA statement, then added to the nondefault rule using the USEVIA keyword.

WIDTH *width*

Specifies the required minimum width for *layerName*.

*Type:* Float, specified in microns

WIREEXTENSION *value*

Specifies the distance by which wires are extended at vias. Enter 0 (zero) to specify no extension. Values other than 0 must be greater than or equal to half of the routing width for the layer, as defined in the nondefault rule.

*Default:* Wires are extended half of the routing width

*Type:* Float, specified in microns

**Note:** The WIREEXTENSION statement only extends wires and not vias. For 65nm and below, WIREEXTENSION is no longer recommended because it may generate some advance rule violations if wires and vias have different widths. See [Illustration of WIREEXTENSION](#) on page 308.

### Example 1-35 Nondefault Rule Statement

Assume two default via rules were defined:

```
VIARULE via12rule GENERATE DEFAULT
    LAYER metall ;
    ...
END via12rule
VIARULE via23rule GENERATE DEFAULT
    LAYER metal2 ;
    ...
END via23rule
```

- Assuming the minimum width is 1.0  $\mu\text{m}$ , the following nondefault rule creates a 1.5x minimum width wire using default spacing:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
NONDEFAULTRULE wide1_5x
  LAYER metall1
    WIDTH 1.5 ; #metall1 has a 1.5 um width
  END metall1
  LAYER metal2
    WIDTH 1.5 ;
  END metal2
  LAYER metal3
    WIDTH 1.5 ;
  END metal3
END wide1_5x
```

**Note:** If there were no default via rules, then a VIA, USEVIA, or USEVIARULE keyword would be required. Because there are none defined, the default via rules are implicitly inherited for this nondefault rule; therefore, via12rule and via23rule would be used for this routing rule.

- The following nondefault rule creates a 3x minimum width wire using default spacing with at least two-cut vias:

```
NONDEFAULTRULE wide3x
  LAYER metall1
    WIDTH 3.0 ; #metall1 has 3.0 um width
  END metall1
  LAYER metal2
    WIDTH 3.0 ;
  END metal2
  LAYER metal3
    WIDTH 3.0 ;
  END metal3
  #viarule12 and viarule23 are used implicitly
  MINCUTS cut12 2 ; #at least two-cut vias are required for cut12
  MINCUTS cut23 2 ;
END wide3x
```

- The following nondefault rule creates an “analog” rule with its own special vias, and with hard extra spacing:

```
NONDEFAULTRULE analog_rule
  HARDSPACING ;      #do not let any other signal close to this one
  LAYER metall1
    WIDTH 1.5 ;      #metall1 has 1.5 um width
    SPACING 3.0 ;    #extra spacing of 3.0 um
  END metall1
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
LAYER metal2
    WIDTH 1.5
    SPACING 3.0
END metal2
LAYER metal3
    WIDTH 1.5
    SPACING 3.0
END metal3
#Use predefined "analog vias"
#The DEFAULT VIARULES will not be inherited.
USEVIA via12_fixed_analog_via ;
USEVIA via_23_fixed_analog_via ;
END analog_rule
```

## Property Definitions

```
[PROPERTYDEFINITIONS
    [objectType propName propType [RANGE min max]
    [value | "stringValue"]
    ;] ...
END PROPERTYDEFINITIONS]
```

Lists all properties used in the LEF file. You must define properties in the PROPERTYDEFINITIONS statement before you can refer to them in other sections of the LEF file.

### *objectType*

Specifies the object type being defined. You can define properties for the following object types:

LAYER  
LIBRARY  
MACRO  
NONDEFAULTRULE  
PIN  
VIA  
VIARULE



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

*propName*

Specifies a unique property name for the object type.

*propType*

Specifies the property type for the object type. You can specify one of the following property types:

INTEGER

REAL

STRING

RANGE *min max*

Limits real number and integer property values to a specified range. That is, the value must be greater than or equal to *min* and less than or equal to *max*.

*value* | "*stringValue*"

Assigns a numeric value or a name to a `LIBRARY` object type.

**Note:** Assign values to other properties in the section of the LEF file that describes the object to which the property applies.

### Example 1-36 Property Definitions Statement

The following example shows library, via, and macro property definitions.

```
PROPERTYDEFINITIONS
  LIBRARY versionNum INTEGER 12;
  LIBRARY title STRING "Cadence96";
  VIA count INTEGER RANGE 1 100;
  MACRO weight REAL RANGE 1.0 100.0;
  MACRO type STRING;
END PROPERTYDEFINITIONS
```

### Site

```
SITE siteName
  CLASS {PAD | CORE} ;
  [SYMMETRY {X | Y | R90} ... ;]
  [ROWPATTERN {previousSiteName siteOrient} ... ;]
  SIZE width BY height ;
END siteName
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines a placement site in the design. A placement site gives the placement grid for a family of macros, such as I/O, core, block, analog, digital, short, tall, and so forth. `SITE` definitions can be used in `DEF ROW` statements.

`CLASS {PAD | CORE}` Specifies whether the site is an I/O pad site or a core site.

`ROWPATTERN {previousSiteName siteOrient}`

Specifies a set of previously defined sites and their orientations that together form *siteName*.

*previousSiteName*

Specifies the name of a previously defined site. The height of each previously defined site must be the same as the height specified for *siteName*, and the sum of the widths of the previously defined sites must equal the width specified for *siteName*.

*siteOrient*

Specifies the orientation for the previously defined site. This value must be one of N, S, E, W, FN, FS, FE, and FW. For more information on orientations, see [“Specifying Orientation”](#) in the `DEF COMPONENT` section.

### Example 1-37 Site Row Pattern Statement

The following example defines three sites: `Fsite`; `Lsite`; and `mySite`, which consists of a pattern of `Fsite` and `Lsite` sites:

```
SITE Fsite
    CLASS CORE ;
    SIZE 4.0 BY 7.0 ; #4.0 um width, 7.0 um height
END Fsite
SITE Lsite
    CLASS CORE ;
    SIZE 6.0 BY 7.0 ; #6.0 um width, 7.0 um height
END Lsite
SITE mySite
    ROWPATTERN Fsite N Lsite N Lsite FS ; #Pattern of F + L + flipped L
    SIZE 16.0 BY 7.0 ;                #Width = width(F + L + L)
END mySite
```

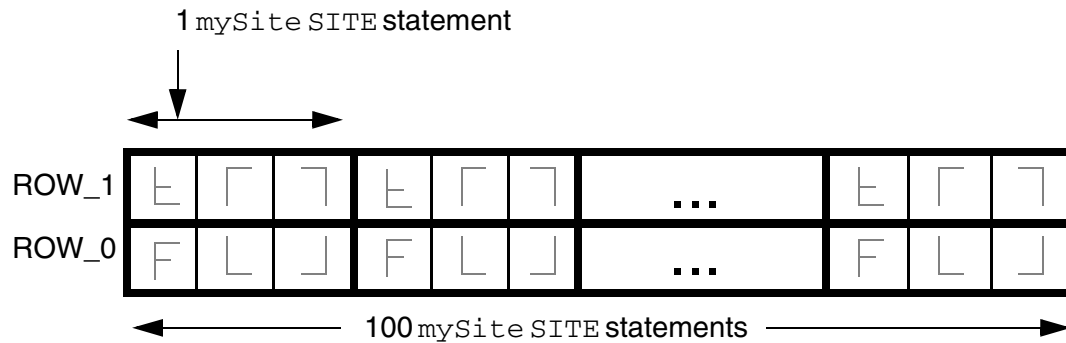
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Figure 1-318 on page 611 illustrates some DEF rows made up of `mySite` sites.

**Figure 1-318**



```
ROW ROW_0 mySite 1000 1000 N DO 100 BY 1 STEP 1600 0 ;
ROW ROW_1 mySite 1000 1700 FS DO 100 BY 1 STEP 1600 0 ;
```

`SITE siteName` Specifies the name for the placement site.

`SIZE width BY height`

Specifies the dimensions of the site in normal (or north) orientation, in microns.

`SYMMETRY {X | Y | R90}`

Indicates which site orientations are equivalent. The sites in a given row all have the same orientation as the row. Generally, site symmetry should be used to control the flipping allowed inside the rows. For more information on defining symmetry, see [“Defining Symmetry”](#) on page 578.

Possible orientations include:

- X Site is symmetric about the x axis. This means that N and FS sites are equivalent, and FN and S sites are equivalent. A macro with an orientation of N matches N or FS rows.
- Y Site is symmetric about the y axis. This means that N and FN sites are equivalent, and FS and S sites are equivalent. A macro with an orientation of N matches N or FN rows.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

- X Y      Site is symmetric about the x and y axis. This means that N, FN, FS, and S sites are equivalent. A macro with orientation N matches N, FN, FS, or S rows.
- R90      Site is symmetric when rotated 90 degrees. Typically, this value is not used.

**Note:** Typically, a site for single-height standard cells uses symmetry Y, and a site for double-height standard cells uses symmetry X Y.

## Units

```
[UNITS
    [TIME NANOSECONDS convertFactor ;]
    [CAPACITANCE PICO FARADS convertFactor ;]
    [RESISTANCE OHMS convertFactor ;]
    [POWER MILLIWATTS convertFactor ;]
    [CURRENT MILLIAMPS convertFactor ;]
    [VOLTAGE VOLTS convertFactor ;]
    [DATABASE MICRONS LEFconvertFactor ;]
    [FREQUENCY MEGAHERTZ convertFactor ;]
END UNITS]
```

Defines the units of measure in LEF. The values tell you how to interpret the numbers found in the LEF file. Units are fixed with a *convertFactor* for all unit types, except database units and capacitance. For more information, see [“Convert Factors”](#) on page 614. Currently, other values for *convertFactor* appearing in the UNITS statement are ignored.

The UNITS statement is optional and, when used, must precede the LAYER statements.

CAPACITANCE PICO FARADS *convertFactor*

Interprets one LEF capacitance unit as 1 picofarad.

CURRENT MILLIAMPS *convertFactor*

Interprets one LEF current unit as 1 milliamp.

DATABASE MICRONS *LEFconvertFactor*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Interprets one LEF distance unit as multiplied when converted into database units.

If you omit the `DATABASE MICRONS` statement, a default value of 100 is recorded as the *LEFconvertFactor* in the database. In this case, one micron would equal 100 database units.

`FREQUENCY MEGAHERTZ convertFactor`

Interprets one LEF frequency unit as 1 megahertz.

`POWER MILLIWATTS convertFactor`

Interprets one LEF power unit as 1 milliwatt.

`RESISTANCE OHMS convertFactor`

Interprets one LEF resistance unit as 1 ohm.

`TIME NANOSECONDS convertFactor`

Interprets one LEF time unit as 1 nanosecond.

`VOLTAGE VOLTS convertFactor`

Interprets one LEF voltage unit as 1 volt.

### **Database Units Information**

Database precision is relative to Standard International (SI) units. LEF values are converted to integer values in the library database as follows.

SI unit	Database precision
1 nanosecond	= 1,000 DBUs
1 picofarad	= 1,000,000 DBUs
1 ohm	= 10,000 DBUs
1 milliwatt	= 10,000 DBUs
1 milliamp	= 10,000 DBUs
1 volt	= 1,000 DBUs

## LEF/DEF 5.8 Language Reference

### LEF Syntax

#### **Convert Factors**

LEF supports values of 100, 200, 400, 800, 1000, 2000, 4000, 8000, 10,000, and 20,000 for *LEFconvertFactor*. The following table illustrates the conversion of LEF distance units into database units.

LEFconvertFactor	LEF	Database Units
100	1 micron	100
200	1 micron	200
400	1 micron	400
800	1 micron	800
1000	1 micron	1000
2000	1 micron	2000
4000	1 micron	4000
8000	1 micron	8000
10,000	1 micron	10,000
20,000	1 micron	20,000

The DEF database precision cannot be more precise than the LEF database precision. This means the DEF convert factor must always be less than or equal to the LEF convert factor. The LEF convert factor must also be an integer multiple of the DEF convert factor so no round-off of DEF database unit values is required (e.g., a LEF convert factor of 1000 allows DEF convert factors of 100, 200, 1000, but not 400, 800). The following table shows the valid pairings of the LEF convert factor and the corresponding DEF convert factor.

LEFconvertFactor	Legal DEFconvertFactors
100	100
200	100, 200
400	100, 200, 400
800	100, 200, 400, 800
1000	100, 200, 1000
2000	100, 200, 400, 1000, 2000
4000	100, 200, 400, 800, 1000, 2000, 4000

## LEF/DEF 5.8 Language Reference

### LEF Syntax

LEFconvertFactor	Legal DEFconvertFactors
8000	100, 200, 400, 800, 1000, 2000, 4000, 8000
10,000	100, 200, 400, 1000, 2000, 10,000
20,000	100, 200, 400, 800, 1000, 2000, 4000, 10,000, 20,000

An incremental LEF should have the same value as a previous LEF. An error message warns you if an incremental LEF has a different value than what is recorded in the database.

## Use Min Spacing

```
[USEMINSPACING OBS { ON | OFF } ;]
```

Defines how minimum spacing is calculated for obstruction (blockage) geometries.

OBS {ON | OFF}

Specifies how to calculate minimum spacing for obstruction geometries (MACRO OBS shapes).

*Default:* ON

OFF

Spacing is computed to MACRO OBS shapes as if they were actual routing shapes. A wide OBS shape would use wide wire spacing rules, and a thin OBS shapes would use thin wire spacing rules.

ON

Spacing is computed as if the MACRO OBS shapes were min-width wires. Some LEF models abstract many min-width wires as a single large OBS shape; therefore using wide wire spacing would be too conservative.

**Note:** OFF is the recommended value to specify because it is a better abstract model for the various wide wire spacing rules that are more common at process nodes of 130nm and smaller. Certain older style LEF abstracts use ON, but it can have unexpected side effects (such as hidden DRC errors) if the abstracts are not created very carefully. You cannot mix both types of LEF abstracts at the same time.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## Version

VERSION *number* ;

Specifies which version of the LEF syntax is being used. *number* is a string of the form *major.minor[.subMinor]*, such as 5.8.

**Note:** Many applications default to the latest version of LEF/DEF supported by the application (which depends on how old the application is). The latest version as described by this document is 5.8. However, a default value of 5.8 is not formally part of the language definition; therefore, you cannot be sure that all applications use this default value. Also, because the default value varies with the latest version, you should not depend on this.

## Via

```
VIA viaName [DEFAULT]
{ VIARULE viaRuleName ;
  CUTSIZE xSize ySize ;
  LAYERS botMetalLayer cutLayer topMetalLayer ;
  CUTSPACING xCutSpacing yCutSpacing ;
  ENCLOSURE xBotEnc yBotEnc xTopEnc yTopEnc ;
  [ROWCOL numCutRows numCutCols ;]
  [ORIGIN xOffset yOffset ;]
  [OFFSET xBotOffset yBotOffset xTopOffset yTopOffset ;]
  [PATTERN cutPattern ;]
}
| {[RESISTANCE resistValue ;]
  {LAYER layerName ;
    { RECT [MASK maskNum] pt pt ;
      | POLYGON [MASK maskNum] pt pt pt ... ;} ...
    } ...
  }
  [PROPERTY propName propVal ;] ...
END viaName
```

Defines two types of vias: fixed vias and generated vias. All vias consist of shapes on three layers: a cut layer and two routing (or masterslice) layers that connect through that cut layer.

A fixed via is defined using rectangles or polygons, and does not use a `VIARULE`. The fixed via name must mean the same via in all associated LEF and DEF files.

A generated via is defined using `VIARULE` parameters to indicate that it was derived from a `VIARULE GENERATE` statement. For a generated via, the via name is only used locally inside this LEF file. The geometry and parameters are maintained, but the name can be freely changed by applications that use this via when writing out LEF and DEF files. For example, large blocks that include generated vias as part of the LEF `MACRO PIN` statement can define



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

generated vias inside the same LEF file without concern about via name collisions in other LEF files.

**Note:** Use the `VIARULE GENERATE` statement to define special wiring.

`CUTSIZE xSize ySize`

Specifies the required width (*xSize*) and height (*ySize*) of the cut layer rectangles.

*Type:* Float, specified in microns

`CUTSPACING xCutSpacing yCutSpacing`

Specifies the required x and y spacing between cuts. The spacing is measured from one cut edge to the next cut edge.

*Type:* Float, specified in microns

`DEFAULT`

Identifies the via as the default via between the defined layers. Default vias are used for default routing by the signal routers.

If you define more than one default via for a layer pair, the router chooses which via to use. You must define default vias between *metal1* and masterslice layers if there are pins on the masterslice layers.

All vias consist of shapes on three layers: a cut layer and two routing (or masterslice) layers that connect through that cut layer. There should be at least one `RECT` or `POLYGON` on each of the three layers.

`ENCLOSURE xBotEnc yBotEnc xTopEnc yTopEnc`

Specifies the required x and y enclosure values for the bottom and top metal layers. The enclosure measures the distance from the cut array edge to the metal edge that encloses the cut array.

*Type:* Float, specified in microns

**Note:** It is legal to specify a negative number, as long as the resulting metal size is positive.

`LAYER layerName`

Specifies the layer on which to create the rectangles that make up the via. All vias consist of shapes on three layers: a cut layer and two routing (or masterslice) layers that connect through that cut layer. There should be at least one `RECT` or `POLYGON` on each of the three layers.

`LAYERS botMetalLayer cutLayer topMetalLayer`

Specifies the required names of the bottom routing (or masterslice) layer, cut layer, and top routing (or masterslice) layer. These layer names must be previously defined in layer definitions, and must match the layer names defined in the specified LEF

*viaRuleName*.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**MASK** *maskNum*

Specifies which mask for double- or triple-patterning lithography is to be applied to the shapes defined in **RECT** or **POLYGON** of the via master. The *maskNum* variable must be a positive integer. Most applications only support values of 1, 2, or 3. For a fixed via made up of **RECT** or **POLYGON** statements, the cut-shapes should either be all colored or not colored at all. It is an error to have partially colored cuts for one via. Uncolored cut shapes should be automatically colored if the layer is a multi-mask layer.

The metal shapes with a shape per layer of the via-master do not need colors because the via instance has the mask color, but some readers will color them as `mask 1` for internal consistency (see [Figure 1-324](#) on page 627 ). So a writer may write out **MASK 1** for the metal shapes even if they are read in with no **MASK** value.

**OFFSET** *xBotOffset yBotOffset xTopOffset yTopOffset*

Specifies the x and y offset for the bottom and top metal layers. By default, the 0, 0 origin of the via is the center of the cut array, and the enclosing metal rectangles. These values allow each metal layer to be offset independently. After the non-shifted via is computed, the metal layer rectangles are offset by adding the appropriate values—the x/y *BotOffset* values to the metal layer below the cut layer, and the x/ y *TopOffset* values to the metal layer above the cut layer. These offsets are in addition to any offset caused by the **ORIGIN** values.

**Default:** 0, for all values

**Type:** Float, specified in microns

**ORIGIN** *xOffset yOffset*

Specifies the x and y offset for all of the via shapes. By default, the 0, 0 origin of the via is the center of the cut array, and the enclosing metal rectangles. After the non-shifted via is computed, all cut and metal rectangles are offset by adding these values.

**Default:** 0, for both values

**Type:** Float, specified in microns

**PATTERN** *cutPattern*

Specifies the cut pattern encoded as an ASCII string. This parameter is only required when some of the cuts are missing from the array of cuts, and defaults to “all cuts are present,” if not specified.

For information on and examples of via cut patterns, see [Creating Via Cut Patterns](#).

The *cutPattern* syntax uses “\_” as a separator, and is defined as follows:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

*numRows\_rowDefinition*

*[\_numRows\_rowDefinition] ...*

*numRows* Specifies a hexadecimal number that indicates how many times to repeat the following row definition. This number can be more than one digit.

*rowDefinition* Defines one row of cuts, from left to right.

The *rowDefinition* syntax is defined as follows:

*{[RrepeatNumber]hexDigitCutPattern} ...*

*hexDigitCutPattern*

Specifies a single hexadecimal digit that encodes a 4-bit binary value, in which 1 indicates a cut is present, and 0 indicates a cut is not present.

*repeatNumber* Specifies a single hexadecimal digit that indicates how many times to repeat *hexDigitCutPattern*.

For parameterized vias (with + VIARULE ...), the *cutPattern* has an optional suffix added to allow three types of mask color patterns. The default mask color pattern (no suffix) is a checker-board defined as an alternating pattern starting with MASK 1 at the bottom left. Then the mask cycles left-to-right, and from bottom-to-top, as shown in [Figure 1-322](#) on page 624. The other two patterns supported are alternating rows, and alternating columns, see [Figure 1-323](#) on page 625.

The optional suffixes are:

<cut\_pattern>\_MR alternating rows

<cut\_pattern>\_MC alternating columns

POLYGON *pt pt pt*

Specifies a sequence of at least three points to generate a polygon geometry. The polygon edges must be parallel to the x axis, to the y axis, or at a 45-degree angle. Each POLYGON keyword defines a polygon generated by connecting each successive point, and then connecting the first and last points. The *pt* syntax corresponds to an x y coordinate pair, such as -0.2 1.0.

*Type:* Float, specified in microns

### Example 1-38 Via Rules

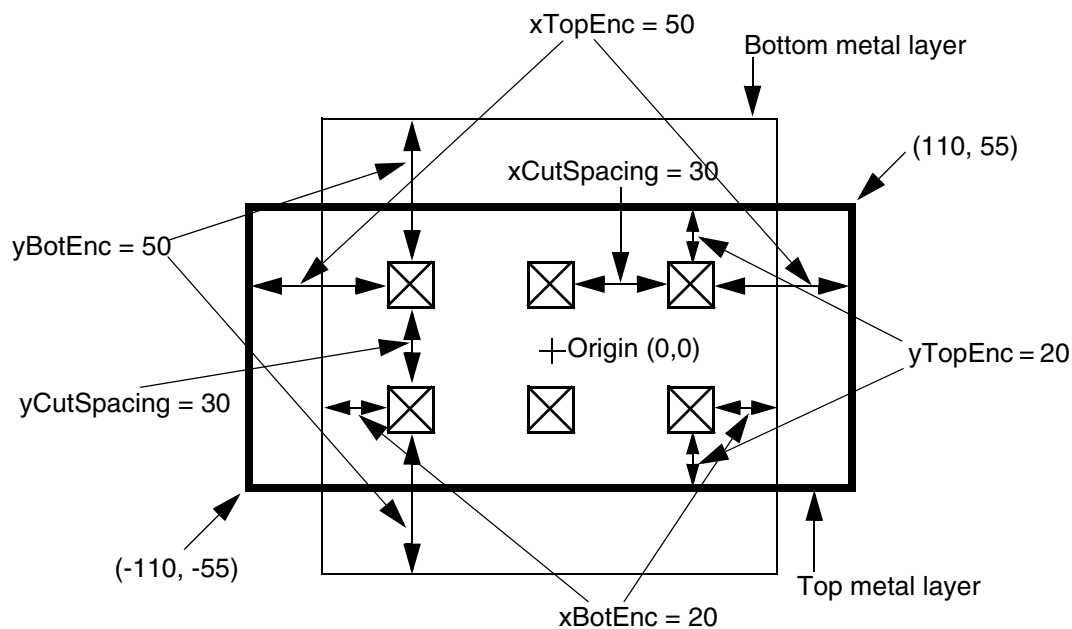
## LEF/DEF 5.8 Language Reference

### LEF Syntax

The following via rule describes a non-shifted via (that is, a via with no `OFFSET` or `ORIGIN` parameters). There are two rows and three columns of via cuts. [Figure 1-319](#) on page 620 illustrates this via rule.

```
VIA myVia
  VIARULE myViaRule ;
  CUTSIZE 20 20 ;                #xCutSize yCutSize
  LAYERS metall cut12 metal2 ;
  CUTSPACING 30 30 ;             #xCutSpacing yCutSpacing
  ENCLOSURE 20 50 50 20 ;        #xBotEnc yBotEnc xTopEnc yTopEnc
  ROWCOL 2 3 ;
END myVia
```

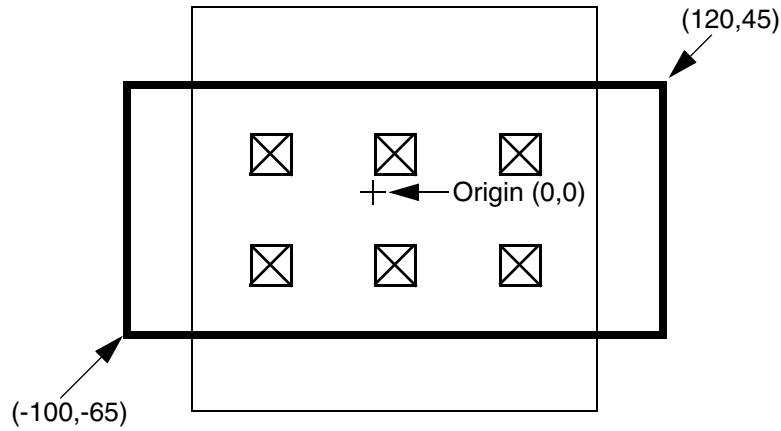
**Figure 1-319 Via Rule**



The same via rule with the following `ORIGIN` parameter shifts all of the metal and cut rectangles by 10 in the x direction, and by -10 in the y direction (see [Figure 1-320](#) on page 621):

```
ORIGIN 10 -10 ;
```

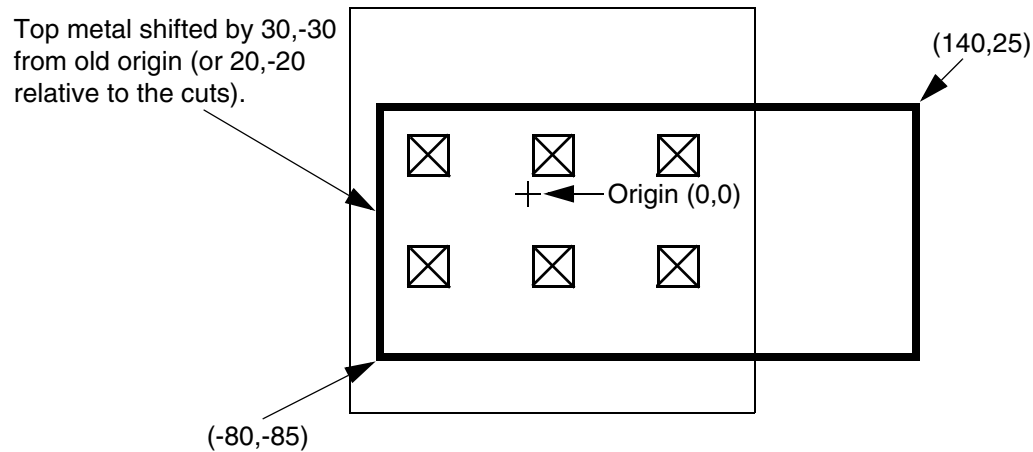
**Figure 1-320 Via Rule With Origin**



If the same via rule contains the following `ORIGIN` and `OFFSET` parameters, all of the rectangles shift by 10, -10. In addition, the top layer metal rectangle shifts by 20, -20, which means that the top metal shifts by a total of 30, -30.

```
ORIGIN 10 -10 ;
OFFSET 0 0 20 -20 ;
```

**Figure 1-321 Via Rule With Origin and Offset**



### Example 1-39 Via Polygon

The following via definition creates a polygon geometry used by X-routing applications:

```
VIA myVia23
    LAYER metal2 ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
POLYGON -2.1 -1.0 -0.2 1.0 2.1 1.0 0.2 -1.0 ;
LAYER cut23 ;
RECT -0.4 -0.4 0.4 0.4 ;
LAYER metal3 ;
POLYGON -0.2 -1.0 -2.1 1.0 0.2 1.0 2.1 -1.0 ;
END myVia23
```

PROPERTY *propName propVal*

Specifies a numerical or string value for a via property defined in the PROPERTYDEFINITIONS statement. The *propName* you specify must match the *propName* listed in the PROPERTYDEFINITIONS statement.

RECT *pt pt*

Specify the corners of a rectangular shape in the via. The *pt* syntax corresponds to an x y coordinate pair, such as `-0.4 -4.0`. For vias used only in macros or pins, reference locations and rectangle coordinates must be consistent.

*Type:* Float, specified in microns

RESISTANCE *resistValue*

Specifies the lumped resistance for the via. This is not a resistance per via-cut value; it is the total resistance of the via. By default, via resistance is computed from the via LAYER RESISTANCE value; however, you can override that value with this value. *resistValue* is ignored if a via rule is specified, because only the VIARULE definition or a cut layer RESISTANCE value gives the resistance for generated vias.

*Type:* Float, specified in ohms

**Note:** A RESISTANCE value attached to an individual via is no longer recommended.

ROWCOL *numCutRows numCutCols*

Specifies the number of cut rows and columns that make up the via array.

*Default:* 1, for both values

*Type:* Positive integer, for both values

*viaName*

Specifies the name for the via.

VIARULE *viaRuleName*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies the name of the LEF `VIARULE` that produced this via. This indicates that the via is the result of automatic via generation, and that the via name is only used locally inside this LEF file. The geometry and parameters are maintained, but the name can be freely changed by applications that use this via when writing out LEF and DEF files.

*viaRuleName* must be specified before you define any of the other parameters, and must refer to a previously defined `VIARULE GENERATE` rule name. It cannot refer to a `VIARULE` without a `GENERATE` keyword.

Specifying the reserved via rule name of `DEFAULT` indicates that the via should use a previously defined `VIARULE GENERATE` rule with the `DEFAULT` keyword that exists for this routing-cut-routing (or masterslice-cut-masterslice) layer combination. This makes it possible for an IP block user to use existing via rules from the normal LEF technology section instead of requiring it to locally create its own via rules for just one LEF file.

#### Example 1-40 Generated Via Rule

The following via definition defines a generated via that is used only in this LEF file.

```
VIA myBlockVia
  VIARULE DEFAULT ;                                #Use existing VIARULE GENERATE rule with
                                                    #the DEFAULT keyword
  CUTSIZE 0.1 0.1 ;                                #Cut is 0.1 x 0.1 um
  LAYERS metall vial2 metal2 ;                     #Bottom metal, cut, and top metal layers
  CUTSPACING 0.1 0.1 ;                             #Space between cut edges is 0.1 um
  ENCLOSURE 0.05 0.01 0.01 0.05 ;                 #metall enclosure is 0.05 in x, 0.01 in y
                                                    #metal2 enclosure is 0.01 in x, 0.05 in y
  ROWCOL 1 2 ;                                     #1 row, 2 columns = 2 cuts
END myBlockVia
```

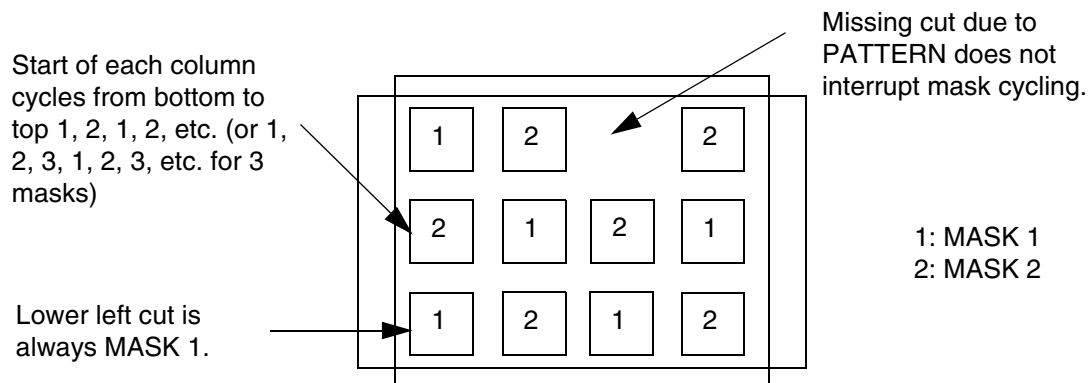
#### Example 1-41 Parameterized via cut-mask pattern

The following example shows a `VIARULE` parameterized via:

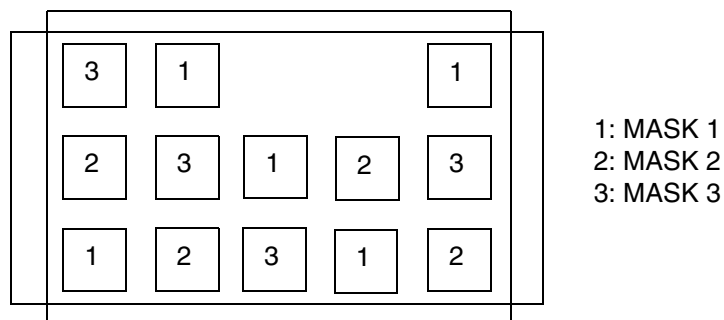
```
VIA myParamVia1
  VIARULE myGenVia1                                CUTSIZE 0.4 0.4
  LAYERS M1 VIA1 M2                                CUTSPACING 0.4 0.4
  ENCLOSURE 0.4 0 0 0.4                            ROWCOL 3 4                #3 rows, 4 columns
  PATTERN 2_F_1_D;                                  #1 cut in top row is missing
```

Example of a parameterized via checker-board cut-mask pattern for a 3-mask layer with 2 missing cuts. For parameterized vias (with `VIARULE . . .`), the mask of the cuts are pre-defined as an alternating pattern starting with `MASK 1` at the bottom left. The mask cycles from left-to-right and bottom-to-top are shown.

**Figure 1-322 Parameterized via cut-mask pattern using `PATTERN`**



Assuming that the layer `VIA1` is a two-mask cut-layer, then the mask color for each cut-shape is a checker-board as shown above. The default checker-board pattern is: the bottom left cut starts at 1, and then goes from left to right, 1, 2, 1, 2. The next row starts at 2, and goes 2, 1, 2, 1, etc. The missing cuts due to `PATTERN` statement do not change the mask assignments.



Example of a parameterized via checker-board cut-mask pattern for a 3-mask layer with 2 missing cuts. For parameterized vias (with `VIARULE . . .`), the mask of the cuts are pre-defined as an alternating pattern starting with `MASK 1` at the bottom left. The mask cycles from left-to-right and bottom-to-top are shown.

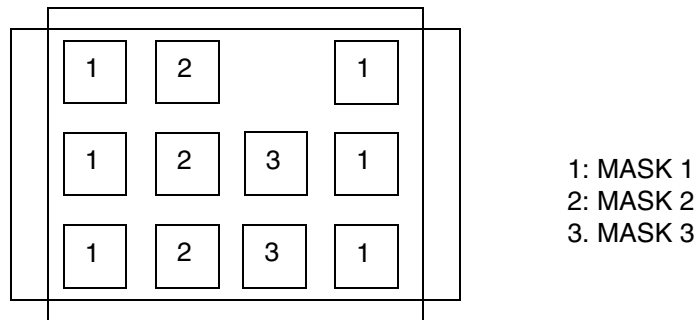
- The following examples show a parameterized via with cut-mask patterns for a 3-mask layer and 2-mask layer using `_MC` and `_MR` suffixes:



**Figure 1-323 Parameterized via cut-mask pattern using Suffixes**

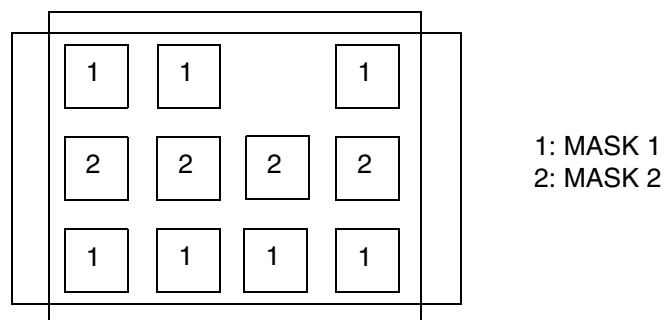
Mask color pattern for a parameterized via with 3-mask cut-layer:

```
VIA myParamVia1
  VIARULE myGenVia1      CUTSIZE 0.4 0.4
  LAYERS M1 VIA1 M2      CUTSPACING 0.4 0.4
  ENCLOSURE 0.4 0 0 0.4  ROWCOL 3 4
  PATTERN 2_F_1_D_MC;    #alternating mask color columns due to _MC suffix
```



For a VIARULE parameterized via like:

```
VIA myParamVia1
  VIARULE myGenVia1      CUTSIZE 0.4 0.4
  LAYERS M1 VIA1 M2      CUTSPACING 0.4 0.4
  ENCLOSURE 0.4 0 0 0.4  ROWCOL 3 4
  PATTERN 2_F_1_D_MR;    #alternating mask color rows due to _MR suffix
```



### Example 1-42 Fixed-via with pre-colored cut shapes

The following example shows a fixed-via with pre-colored cut shapes:

```
VIA myVia1
  LAYER m1 ;
  RECT -0.4 -0.2 1.2 0.2 ;           #no mask, some readers will set to mask 1
  LAYER vial ;
  RECT MASK 1 -0.2 -0.2 0.2 0.2 ;    #first cut on mask 1
```

## LEF/DEF 5.8 Language Reference

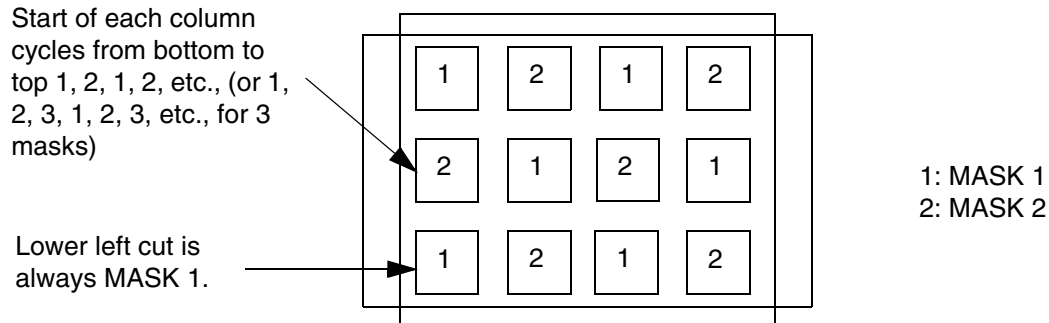
### LEF Syntax

---

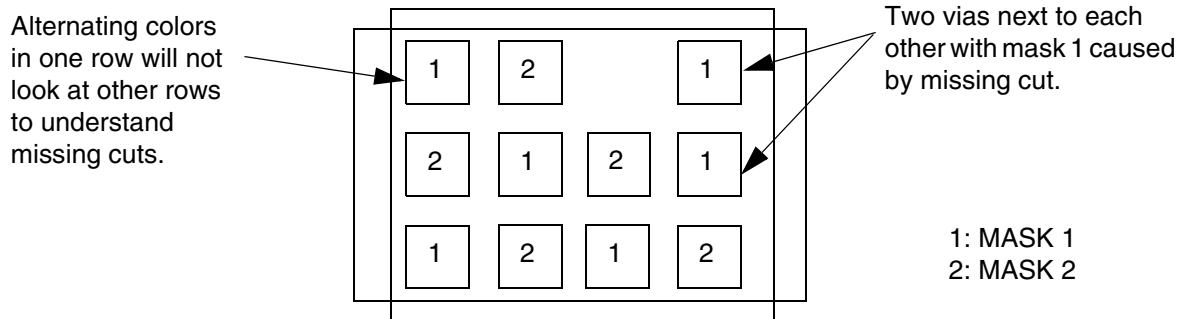
```
RECT MASK 2 0.6 -0.2 1.0 0.2 ;    #second cut on mask 2
LAYER m2 ;
RECT -0.2 -0.4 1.0 0.4 ;          #no mask, some readers will set to mask 1
END myVial
```

For a fixed via made up of `RECT` or `POLYGON` statements, the cut shapes must all be colored or not colored at all. If the cuts are not colored, they will be automatically colored in a checkerboard pattern as described above for parameterized vias. Each via-cut with the same lower-left Y value is considered one row, and each via in one row is a new column. For common "array" style vias with no missing cuts, this coloring is a good one. For vias that do not have a row and column structure, or are missing cuts this coloring may not be good (see [Figure 1-324](#) on page 627). If the metal layers having only one shape per layer are not colored, some applications will color them to `MASK 1` for internal consistency, even though the via-master metal shape colors are not really used by LEF/DEF via instances. For multiple disjoint metal shapes, it is highly recommended to provide proper color.

**Figure 1-324 Fixed-via with pre-colored cut shapes**



Uncolored fixed vias (from RECT/POLYGON statements) will get similar coloring as a parameterized via. Each cut with the same Y value is one row. The cuts inside one row alternate. This works well for cut-arrays without missing cuts as shown above.



Unlike parameterized vias, there is no real row/column structure required in a fixed via, so the automatic coloring will not work well for a fixed via with missing cuts (or cuts that are not aligned). This type of via must be pre-colored to be useful.

See the MACRO Layer Geometries statement to see how a via-instance uses these via-master mask values.

## Via Rule

```
VIARULE viaRuleName
    LAYER layerName ;
        DIRECTION {HORIZONTAL | VERTICAL} ;
        [WIDTH minWidth TO maxWidth ;]
    LAYER layerName ;
        DIRECTION {HORIZONTAL | VERTICAL} ;
        [WIDTH minWidth TO maxWidth ;]
    {VIA viaName ;} ...
    [PROPERTY propName propVal ;] ...
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

END *viaRuleName*

Defines which vias to use at the intersection of special wires of the same net.

**Note:** You should only use VIARULE GENERATE statements to create a via for the intersection of two special wires. In earlier versions of LEF, VIARULE GENERATE was not complete enough to cover all situations. In those cases, a fixed VIARULE (without a GENERATE keyword) was sometimes used. This is no longer required.

DIRECTION {HORIZONTAL | VERTICAL}

Specifies the wire direction. If you specify a WIDTH range, the rule applies to wires of the specified DIRECTION that fall within the range. Otherwise, the rule applies to all wires of the specified DIRECTION on the layer.

LAYER *layerName*

Specifies the routing or masterslice layers for the top or bottom of the via.

PROPERTY *propName propVal*

Specifies a numerical or string value for a via rules property defined in the PROPERTYDEFINITIONS statement. The *propName* you specify must match the *propName* listed in the PROPERTYDEFINITIONS statement.

VIA *viaName*

Specifies a previously defined via to test for the current via rule. The first via in the list that can be placed at the location without design rule violations is selected. The vias must all have exactly three layers in them. The three layers must include the same routing or masterslice layers as listed in the LAYER statements of the VIARULE, and a cut layer that is between the two routing or masterslice layers.

VIARULE *viaRuleName*

Specifies the name to identify the via rule.

WIDTH *minWidth* TO *maxWidth*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Specifies a wire width range. If the widths of two intersecting special wires fall within the wire width range, the `VIARULE` is used. To fall within the range, the widths must be greater than or equal to *minWidth* and less than or equal to *maxWidth*.

**Note:** `WIDTH` is defined by wire direction, not by layer. If you specify a `WIDTH` range, the rule applies to wires of the specified `DIRECTION` that fall within the range.

#### Example 1-43 Via Rule Statement

In the following example, whenever a *metal1* wire with a width between 0.5 and 1.0 intersects a *metal2* wire with a width between 1.0 and 2.0, the via generation code attempts to put a *via12\_1* at the intersection first. If the *via12\_1* causes a DRC violation, a *via12\_2* is then tried. If both fail, the default behavior from a `VIARULE GENERATE` statement for *metal1* and *metal2* is used.

```
VIARULE viaRule1
  LAYER metal1 ;
    DIRECTION HORIZONTAL ;
    WIDTH 0.5 TO 1.0 ;
  LAYER metal2 ;
    DIRECTION VERTICAL ;
    WIDTH 1.0 TO 2.0 ;
  VIA via12_1 ;
  VIA via12_2 ;
END viaRule1
```

#### Via Rule Generate

```
VIARULE viaRuleName GENERATE [DEFAULT]
  LAYER routingLayerName ;
    ENCLOSURE overhang1 overhang2 ;
    [WIDTH minWidth TO maxWidth ;]
  LAYER routingLayerName ;
    ENCLOSURE overhang1 overhang2 ;
    [WIDTH minWidth TO maxWidth ;]
  LAYER cutLayerName ;
    RECT pt pt ;
    SPACING xSpacing BY ySpacing ;
    [RESISTANCE resistancePerCut ;]
END viaRuleName
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

Defines formulas for generating via arrays. You can use the `VIARULE GENERATE` statement to cover special wiring that is not explicitly defined in the `VIARULE` statement.

Rather than specifying a list of vias for the situation, you can create a formula to specify how to generate the cut layer geometries.

**Note:** Any vias created automatically from a `VIARULE GENERATE` rule that appear in the `DEF NETS` or `SPECIALNETS` sections must also appear in the `DEF VIA` section.

<code>DEFAULT</code>	Specifies that the via rule can be used to generate vias for the default routing rule. There can only be one <code>VIARULE GENERATE DEFAULT</code> for a given routing-cut-routing (or masterslice-cut-masterslice) layer combination.
----------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### Example 1-44 Via Rule Generate Default

The following example defines a rule for generating vias for the default routing or masterslice rule:

```
VIARULE via12 GENERATE DEFAULT
    LAYER m1 ;
    ENCLOSURE 0.03 0.01 ; #2 sides need >= 0.03, 2 other sides need >= 0.01
    LAYER m2 ;
    ENCLOSURE 0.05 0.01 ; #2 sides need >= 0.05, 2 other sides need >= 0.01
    LAYER cut12 ;
    RECT -0.1 -0.1 0.1 0.1 ; # cut is .20 by .20
    SPACING 0.40 BY 0.40 ; #center-to-center spacing
    RESISTANCE 20 ; #ohms per cut
END via12
```

```
ENCLOSURE overhang1 overhang2
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

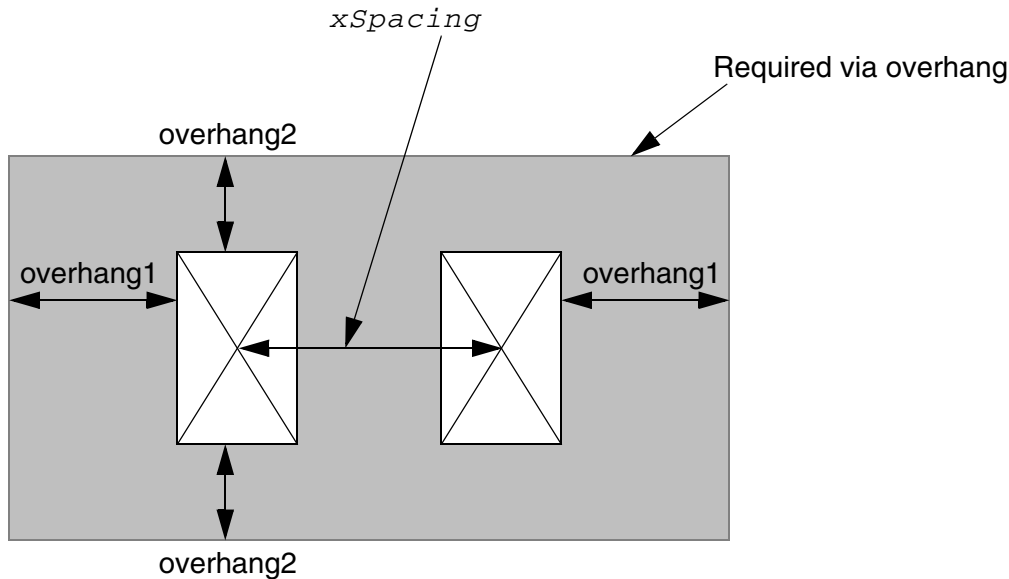
Specifies that the via must be covered by metal on two opposite sides by at least *overhang1*, and on the other two sides by at least *overhang2* (see [Figure 1-325](#) on page 632). The via generation code then chooses the direction of overhang that best maximizes the number of cuts that can fit in the via.

**Note:** If there are also `ENCLOSURE` rules for the cut layer that apply to a given via, the via generation code will choose the `ENCLOSURE` rule with values that match the `ENCLOSURE` in `VIARULE GENERATE`. If there is no such match, the via generation code will ignore the `ENCLOSURE` in `VIARULE GENERATE` and choose which `ENCLOSURE` rule is best in `LAYER ENCLOSURE` values that apply to the same width via being generated. This means that only `ENCLOSURE` statements in `LAYER CUT` are honored, and one of them will be used.

For example, `VIARULE GENERATE ENCLOSURE 0.2 0.0` combined with a `LAYER CUT` rule of `ENCLOSURE 0.2 0.0`, `ENCLOSURE 0.1 0.1` and `ENCLOSURE 0.15 0.15 WIDTH 0.5`, would mean that any via inside a wire with width that is greater than or equal to 0.5 wide, `0.15 0.15` enclosure values are used. Otherwise, `0.2 0.0` enclosure values are used. See the `LAYER CUT ENCLOSURE` statement for more information on handling multiple enclosure rule.

*Type:* Float, specified in microns

**Figure 1-325 Overhang**



### Example 1-45 Via Rule Generate Enclosure

The following example describes a formula for generating via cuts:

```
VIARULE via12 GENERATE
    LAYER m1 ;
        ENCLOSURE 0.05 0.01 ; #2 sides must be >=0.05, 2 other sides must be >=0.01
        WIDTH 0.2 TO 100.0 ; #for m1, between 0.2 to 100 microns wide
    LAYER m2 ;
        ENCLOSURE 0.05 0.01 ; #2 sides must be >=0.05, 2 other sides must be >=0.01
        WIDTH 0.2 TO 100.0 ; #for m2, between 0.2 to 100 microns wide
    LAYER cut12
    RECT -0.07 -0.07 0.07 0.07 ; #cut is .14 by .14
    SPACING 0.30 BY 0.30 ; #center-to-center spacing
END via12
```

The cut layer SPACING ADJACENTCUTS statement can override the VIARULE cut layer SPACING statements. For example, assume the following cut layer information is also defined in the LEF file:

```
LAYER cut12
...
    SPACING 0.20 ADJACENTCUTS 3 WITHIN 0.22 ;
...
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

The 0.20  $\mu\text{m}$  edge-to-edge spacing in the `ADJACENTCUTS` statement is larger than the `VIARULE GENERATE` example spacing of 0.16 (0.30 – 0.14). Whenever the `VIARULE GENERATE` rule creates a via that is larger than 2x2 cuts (that is, 2x3, 3x2, 3x3 and so on), the 0.20 spacing from the `ADJACENTCUTS` statement is used instead.

**Note:** The spacing in `VIARULE GENERATE` is center-to-center spacing, whereas the spacing in `ADJACENTCUTS` is edge-to-edge.

<code>GENERATE</code>	Defines a formula for generating the appropriate via.
<code>LAYER <i>cutLayerName</i></code>	Specifies the cut layer for the generated via.
<code>LAYER <i>routingLayerName</i></code>	Specifies the routing (or masterslice) layers for the top and bottom of the via.
<code>RECT <i>pt pt</i></code>	Specifies the location of the lower left contact cut rectangle.
<code>RESISTANCE <i>resistancePerCut</i></code>	<p>Specifies the resistance of the cut layer, given as the resistance per contact cut.</p> <p><i>Default:</i> The resistance value in the <code>LAYER (Cut)</code> statement</p> <p><i>Type:</i> Float</p>
<code>SPACING <i>xSpacing</i> BY <i>ySpacing</i></code>	<p>Defines center-to-center spacing in the x and y dimensions to create an array of contact cuts. The number of cuts of an array in each direction is the most that can fit within the bounds of the intersection formed by the two special wires. Cuts are only generated where they do not violate stacked or adjacent via design rules.</p> <p><b>Note:</b> This value can be overridden by the <code>SPACING ADJACENTCUTS</code> value in the cut layer statement.</p>
<code>VIARULE <i>viaRuleName</i></code>	<p>Specifies the name for the rule.</p> <p>The name <code>DEFAULT</code> is reserved and should not be used for any via rule name. In the LEF and DEF <code>VIA</code> definitions that use generated via parameters, the reserved <code>DEFAULT</code> name indicates the via rule with the <code>DEFAULT</code> keyword.</p>

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`WIDTH minWidth TO maxWidth`

Specifies a wire width range to use for this VIARULE. This VIARULE can be used for wires with a width greater than or equal to ( $\geq$ ) *minWidth*, and less than or equal to ( $\leq$ ) *maxWidth* for the given routing (or masterslice) layer. If no WIDTH statement is specified, the VIARULE can be used for all wire widths on the given routing (or masterslice) layer.

---

## ALIAS Statements

---

This chapter contains information about the following topics.

- [ALIAS Statements](#) on page 635
  - ❑ [ALIAS Definition](#) on page 636
  - ❑ [ALIAS Examples](#) on page 636
  - ❑ [ALIAS Expansion](#) on page 637

### ALIAS Statements

You can use alias statements in LEF and DEF files to define commands or parameters associated with the library or design. An alias statement can appear anywhere in a LEF or DEF file as follows:

```
&ALIAS  &&aliasName  =  aliasDefinition  &ENDALIAS
```

`&ALIAS` and `&ENDALIAS` are both reserved keywords and are not case sensitive. An alias statement has the following requirements:

- `&ALIAS` must be the first token in the line in which it appears.
- *aliasName* is string name and must appear on the same line as `&ALIAS`. It is case sensitive based on the value of `NAMESCASESENSITIVE` in the LEF input, or the value of `Input.Lef.Names.Case.Sensitive`.
- *aliasName* cannot contain any of the following special characters: #, space, tab, or control characters.
- `&ENDALIAS` must be the last token in the line in which it appears.
- Multiple commands can appear in the alias definition, separated by semicolons. However, the last command must not be terminated by a semicolon.

## ALIAS Definition

The alias name (*aliasName*) is an identifier for the associated alias definition (*aliasDefinition*). The data reader stores the alias definition in the database. If the associated alias name already exists in the database, a warning is issued and the existing definition is replaced.

Alias definitions are text strings with the following properties:

- *aliasDefinition* is any text excluding “&ENDALIAS”.
- All EOL, space, and tab characters are preserved.
- *aliasDefinition* text can expand to multiple lines.

## ALIAS Examples

The following examples include legal and illegal alias statements:

- The following statement is legal.

```
&ALIAS &&MAC = SROUTE ADDCELL AREA &&CORE &ENDALIAS
```

- The following statement is illegal because MAC does not start with “&&”.

```
&ALIAS MAC = SROUTE AREA &&CORE &ENDALIAS
```

- The following statement is illegal because &ALIAS is not the first token in this line.

```
( 100 200 ) &ALIAS &&MAC = SROUTE AREA &&CORE &ENDALIAS
```

- The following statement is legal. It contains multiple commands; the last command is not terminated by a semicolon.

```
$ALIAS $$ = INPUT LEF myfile.txt;  
VERIFY LIBRARY  
ENDALIAS
```

The following examples show legal and illegal alias names:

- “Engineer\_change” is a legal alias name.

```
&&Engineer_change
```

- “&Version&History&&” is a legal alias name.

```
&&&Version&History&&
```

- “design history” is an illegal alias name. It contains a space character and is considered as two tokens: an *aliasName* token “&&design,” and a non-*aliasName* token “history”.

```
&&design history
```

## LEF/DEF 5.8 Language Reference

### ALIAS Statements

---

- “someName#IO-pin-Num” is an illegal alias name. It contains a “#” character and is translated as one *aliasName* token “&&someName”. The “#” is considered a comment character.

&&someName#IO-pin-Num

## ALIAS Expansion

Alias expansion is the reverse operation of alias definition. The following is the syntax for alias expansion.

&&*aliasName*

where *aliasName* is any name previously defined by an alias statement. If an *aliasName* does not exist in the database, no substitution occurs.

You use aliases as string expansion parameters for LEF or DEF files. An alias can substitute for any token of a LEF or DEF file.

## LEF/DEF 5.8 Language Reference

### ALIAS Statements

---

---

## Working with LEF

---

This chapter contains information about the following topics.

- [Incremental LEF](#) on page 639
- [Error Checking](#) on page 640

### Incremental LEF

`INPUT LEF` can add new data to the current database, providing an incremental LEF capability. Although it is possible to put an entire LEF library in one file, some systems require that you put certain data in separate files.

This feature also is useful, for example, when combined with the `INPUT GDSII` command, to extract geometric data from a GDSII-format file and add the data to the database.

When using `INPUT LEF` on a database that has been modified previously, save the previous version before invoking `INPUT LEF`. This provides a backup in case the library information has problems and the database gets corrupted or lost.



The original LEF file, created with `FINPUT LEF` (or with `INPUT LEF` when no database is loaded), must contain all the layers.

### Adding Objects to the Library

`INPUT LEF` can add the following objects to the database:

- New via
- New via rule
- Samenet spacings (if none have been specified previously)
- New macro

## LEF/DEF 5.8 Language Reference

### Working with LEF

---

If geometries have not been specified for an existing via, `INPUT LEF` can add layers and associated rectangle geometries. If not specified previously for a macro, `INPUT LEF` can add the following:

- `FOREIGN` statement
- `EEQ`
- `LEQ`
- `Size`
- `Overlap` geometries
- `Obstruction` geometries

If not previously specified for an existing macro pin, `INPUT LEF` can add the following:

- `Mustjoins`
- `Ports` and geometries

The database created by `INPUT LEF` can contain a partial library. Run `VERIFY LIBRARY` before proceeding.

If new geometries are added to a routed database, run `VERIFY GEOMETRY` and `VERIFY CONNECTIVITY` to identify new violations.

#### *Important*

When defining a pin with no port geometries with the intent of incrementally adding them, *do not* include an empty `PORT` statement as shown below.

```
MACRO abc
...
  PIN a
    ...
    PORT # dummy pin-port, do not
    END  # include these two lines
  END a
  ...
```

## Error Checking

To help develop, test, and debug generic libraries and parametric macros, LEF and DEF have a user-defined error checking facility. This facility consists of seven utilities that you can use



## LEF/DEF 5.8 Language Reference

### Working with LEF

---

from within a LEF or DEF file during the scanning phase of LEF/DEF readers. These utilities have the following features:

- A message facility that writes to one or more text files during LEF or DEF input
- An error handling facility that logs user detected warnings, errors, and fatal errors

The error checking utilities have the following syntax:

```
&CREATEFILE &fileAlias =  
    { stringExpression  
      | stringIF-ELSEexpression } ;  
  
&OPENFILE &fileAlias ;  
  
&CLOSEFILE &fileAlias ;  
  
&MESSAGE  
    {&fileAlias | &MSGWINDOW} = message;  
  
&WARNING  
    {&fileAlias | &MSGWINDOW} = message ;  
  
&ERROR  
    {&fileAlias | &MSGWINDOW} = message ;  
  
&FATALERROR  
    {&fileAlias | &MSGWINDOW} = message;  
  
message =  
    { &fileAlias | stringExpression  
      | stringIF-ELSEexpression  
      | stringIFexpression }
```

## Message Facility

The message facility outputs user-defined messages during the scanning phase of LEF and DEF input. These messages can be directed to the message window.

### **&CREATEFILE**

The &CREATEFILE utility first assigns a token (*&fileAlias*) to represent a named file. The file name is derived from a previously defined string, a quoted string, or an IF-ELSE expression that evaluates to a string. The following example illustrates these three cases.

```
&DEFINES &messagefile = "demo1.messages" ;  
&CREATEFILE &outfile = &messagefile ;  
&CREATEFILE &msgs =  
    "/usr/asics/cmos/fif4/errors.txt" ;  
&CREATEFILE &messages =  
    IF &errortrap  
    THEN "errs.txt"  
    ELSE "/dev/null" ;
```

The derived file name must be a legal file name in the host environment. The default directory is the current working directory. The file names are case sensitive.

## LEF/DEF 5.8 Language Reference

### Working with LEF

---

`&CREATEFILE` creates an empty file with the given name and opens the file. If the token is already bound to another open file, a warning is issued, the file is closed, and the new file is opened. If the file already exists, the version number is incremented.

#### ***&CLOSEFILE and &OPENFILE***

The `&CLOSEFILE` utility closes the file bound to a given token; `&OPENFILE` opens the file bound to a given token. `&CLOSEFILE` and `&OPENFILE` control the number of open files. Each operating system has a limit for the number of open files. Therefore, `&CLOSEFILE` might be needed to free up extra file descriptors.

Files are closed in the following ways.

- All user files are closed at the end of the scanning phase of the LEF and DEF readers.
- All user files are closed if the scanning phase aborts.
- If `&CREATEFILE` is invoked with a token that is already bound to an open file, that file is closed before opening the new file.

#### ***&MESSAGE***

The `&MESSAGE` utility appends text to the file represented by the *&fileAlias* token, or to the message window if `&MSGWINDOW` is specified.

`&MSGWINDOW` is a special file alias that is not created, opened, or closed. The assigned expression (right side of the statement) can be one of the following:

*&fileAlias*

Must correspond to a valid file that has been successfully opened. The contents of the file are appended to the target file (or message window).

*stringExpression*

Either a string or a string token.

For example:

```
&DEFINES &romword16 =  
    "ROM word size = 16 bits" ;  
&MESSAGE &mesgs = "ROM size = 256" ;  
&MESSAGE &mesgs = &romword16 ;
```

*stringIF-ELSEexpression*

String IF-ELSE expressions evaluate a Boolean expression and then branch to string values, for example:

## LEF/DEF 5.8 Language Reference

### Working with LEF

---

```
&&MESSAGE &msgs =  
  IF (&&c_flag = 0)  
    THEN "FLAG C set to 0"  
  ELSE IF ( &&c_flag = 1 )  
    THEN "FLAG C set to 1"  
  ELSE "FLAG C set to 2" ;
```

As shown in this example, IF-ELSE expressions can be nested.

#### *stringIFexpression*

A string IF expression is an IF-ELSE expression without the ELSE phrase. The Boolean expression is evaluated, and if true, the THEN string is sent to the target file; if false, no string is sent, for example,

```
&MESSAGE &msgs =  
  IF ( &&buf = "INV_BIG" )  
    THEN "INV_BIG buffers" ;
```

Neither the file alias token nor &MSGWINDOW can be part of the assigned expression.

## Error-Checking Facility

In addition to the message facility, you have partial control of the error checking facility of the LEF and DEF readers. When scanning LEF or DEF input, the readers record warnings, errors, and fatal errors. At the end of the scan, the total number of each is sent to the message window before proceeding with the reader phase.

If a fatal error is detected, input is aborted after the scanning phase.

With the user interface to the error checking facility, the LEF and DEF files can include custom error checking. User detected warnings, errors, and fatal errors, can be logged, thereby incrementing the DEF/LEF reader's warning, error, and fatal error counts.

A user-detected fatal error terminates input just as with the resident error checking facility. In addition, the user defined error checking facility utilities can send message strings to the message window.

### ***&WARNING, &ERROR, and &FATALERROR***

The &WARNING, &ERROR, and &FATALERROR utilities use the same syntax as the &MESSAGE utility. These utilities can send message strings to files and to the message window in the same manner as &MESSAGE. In addition, when the assigned expression is a string IF expression, or a string IF-ELSE expression, then the associated counter (warnings, errors, or fatal errors) is incremented by 1 if any IF condition evaluates to true.

## **LEF/DEF 5.8 Language Reference**

### Working with LEF

---

---

## DEF Syntax

---

This chapter contains information about the following topics:

- About Design Exchange Format Files on page 646
  - ❑ General Rules on page 647
  - ❑ Character Information on page 647
    - Name Escaping Semantics for Identifiers on page 648
    - Escaping Semantics for Quoted Property Strings on page 649
  - ❑ Order of DEF Statements on page 651
- DEF Statement Definitions on page 652
  - ❑ Blockages on page 652
  - ❑ Bus Bit Characters on page 656
  - ❑ Components on page 657
  - ❑ Design on page 664
  - ❑ Die Area on page 665
  - ❑ Divider Character on page 665
  - ❑ Extensions on page 665
  - ❑ Fills on page 666
  - ❑ GCell Grid on page 670
  - ❑ Groups on page 672
  - ❑ History on page 673
  - ❑ Nets on page 673
    - Regular Wiring Statement on page 680

- ❑ [Nondefault Rules](#) on page 696
- ❑ [Pins](#) on page 700
- ❑ [Pin Properties](#) on page 717
- ❑ [Property Definitions](#) on page 718
- ❑ [Regions](#) on page 719
- ❑ [Rows](#) on page 720
- ❑ [Scan Chains](#) on page 722
- ❑ [Slots](#) on page 728
- ❑ [Special Nets](#) on page 729
  - [Special Wiring Statement](#) on page 733
- ❑ [Styles](#) on page 746
- ❑ [Technology](#) on page 758
- ❑ [Tracks](#) on page 758
- ❑ [Units](#) on page 760
- ❑ [Version](#) on page 761
- ❑ [Vias](#) on page 761

## About Design Exchange Format Files

A Design Exchange Format (DEF) file contains the design-specific information of a circuit and is a representation of the design at any point during the layout process. The DEF file is an ASCII representation using the syntax conventions described in “[Typographic and Syntax Conventions](#)” on page 7.

DEF conveys logical design data to, and physical design data from, place-and-route tools. Logical design data can include internal connectivity (represented by a netlist), grouping information, and physical constraints. Physical data includes placement locations and orientations, routing geometry data, and logical design changes for backannotation. Place-and-route tools also can read physical design data, for example, to perform ECO changes.

For standard-cell-based/ASIC flow tools, floorplanning is part of the design flow. You typically use the various floorplanning commands to interactively create a floorplan. This data then becomes part of the physical data output for the design using the `ROWS`, `TRACKS`,

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

GCELLGRID, and DIEAREA statements. You also can manually enter this data into DEF to create the floorplan.

It is legal for a DEF file to contain only floorplanning information, such as ROWS. In many cases, the DEF netlist information is in a separate format, such as Verilog, or in a separate DEF file. It is also common to have a DEF file that only contains a COMPONENTS section to pass placement information.

## General Rules

Note the following information about creating DEF files:

- Identifiers like net names and cell names are limited to 2,048 characters.
- DEF statements end with a semicolon ( ; ). You *must* leave a space before the semicolon.
- Each section can be specified only once. Sections end with `END SECTION`.
- You must define all objects before you reference them except for the + ORIGINAL argument in the NETS section.

## Character Information

LEF and DEF identifiers can contain any printable ASCII character, except space, tab, or new-line characters. This means the following characters are allowed along with all alpha-numeric characters:

! " # \$ % & ' ( ) \* + , / : ; < = > ? @ [ \ ] ^ \_ ` { | } ~

A LEF or DEF property string value is contained inside a pair of quotes, like this "`<string>`". The `<string>` value can contain any printable ASCII character as shown above, including space, tab, or new-line characters.

Some characters have reserved meanings unless escaped with \.

- [ ] Default special characters for bus bits inside a net or pin name unless overridden by `BUSBITCHARS`
- / Default special character for hierarchy inside a net or component name unless overridden by `DIVIDERCHAR`
- # The comment character. If preceded by a space, tab, or new-line, everything after # until the next new-line is treated as a comment.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

- \* Matches any sequence of characters for SPECIALNETS or GROUPS component identifiers.
- % Matches any single character for SPECIALNETS or GROUPS component identifiers.
- " The start and end character of a property string value. It has no special meaning for an identifier.
- \ The escape character

You can use the backslash (\) as an escape character before each of the special characters shown above. When the backslash precedes a character that has a special meaning in LEF or DEF, the special meaning of the character is ignored.

### Name Escaping Semantics for Identifiers

Here are some examples depicting the use of the escape character (\) in identifiers:

- A DEF file with `BUSBITCHARS " [ ] "` and net or pin name:

`A[0]` is the 0<sup>th</sup> member of bus A  
`A<0>` is a scalar named A<0>  
`A\[0\]` is a scalar named A[0]

- A DEF file with `DIVIDERCHAR " / "` and net or component names like:

`A/B/C` is a 3-level hierarchical name, where C is inside B and B is inside A  
`A/B\C` is a 2-level hierarchical name where B/C is inside A  
`A\B` is a flat name A/B

- The " character has no special meaning for an identifier. So an identifier would not need a \ before a ". An identifier like:

`name_with_"_in_it`

or even

`"_name_starts_with_goute`

is legal without a \.



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

- An identifier that starts with # would be treated as a comment unless the \ is present like this:

```
\#_name_with_hash_for_first_char
```

Note, the # needs to be preceded by white-space to be treated as a comment char, so it has no special meaning inside an identifier. So an identifier like

```
name_with_#_in_it
```

is legal without a \.

- Pattern matching characters \* or % inside SPECIALNETS or GROUPS component identifiers can be disabled like this:

```
GROUPS 1 ;
```

```
- myGroup i1/i2/\* ...
```

or

```
SPECIALNETS 1 ;
```

```
- VDD ( i1/i2/\* VDD ) ...
```

These will match the exact name i1/i2/\* and not match i1/i2/i3 or other components starting with i1/i2/.

Note, the \* and % have no special meaning in other identifiers, so no \ is needed for them.

- A real \ char in an identifier needs to be escaped like this:

```
name_with_\_in_it ....
```

The first \ escapes the second \, so the real name is just name\_with\_\\_in\_it.

### Escaping Semantics for Quoted Property Strings

Properties may have string type values, placed within double quotes ("). However, if you need to use a double quote as a part of the string value itself, you would need to precede it with the escape character (\) to avoid breaking the property syntax. The escape sequence \" is converted to " during parsing.

The example below depicts the use of the escape character in a quoted property string:

```
PROPERTY stringQuotedProp "string with \" quote and single  
backslash \and double backslash \\" ;
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

The actual value of the property in the database will be:

string with " quote and single backslash and double backslash\

Here:

- The first \ escapes the " so it does not end the property string.
- The next \ has no effect on the subsequent a character.
- The first \ in \\ escapes the second \ character. This means that the second \ in \\ is treated as a real \ character, and it does not escape the final " character, which ends the property string.

Note that the other special characters like [ ] / # \* % have no special meaning inside a property string and do not need to be escaped.

#### ***LEF/DEF to LEF/DEF Equivalence***

In DEF syntax, \ is only used to escape characters that have a special meaning if they are not escaped.

Consider the following LEF/DEF header specification:

- LEFDEF/[ ] is equivalent to LEF or DEF with DIVIDERCHAR "/" and BUSBITCHARS "[ ]"
- LEFDEF|<> is equivalent to LEF or DEF with DIVIDERCHAR "|" and BUSBITCHARS "<>"

In the following examples, <> are not special characters for LEFDEF/[ ] files and [ ] are not special characters for LEFDEF|<> files. Observe how the header settings (listed above) affect the semantic meaning of the names:

- A<0> with LEFDEF/[ ] is not equivalent to A<0> with LEFDEF|<>
- A<0> with LEFDEF/[ ] is equivalent to A\<0\> with LEFDEF|<>
- A[0] with LEFDEF/[ ] is equivalent to A<0> with LEFDEF|<>

#### ***Verilog and DEF Equivalence***

For Verilog and DEF equivalence, consider the following DEF header specification:

- DEF/[ ] is equivalent to DEF with DIVIDERCHAR "/" and BUSBITCHARS "[ ]"

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

- DEF | <> is equivalent to DEF with DIVIDERCHAR " | " and BUSBITCHARS "<>"

In the following examples (showing net names), <> are not special characters for DEF/ [ ] files and [ ] are not special characters for DEF | <> files:

- A<0> in DEF/ [ ] is equivalent to \A<0> in Verilog  
A<0> in DEF | <> is equivalent to A[0] in Verilog (bit 0 of bus A)
- A[0] in DEF/ [ ] is equivalent to A[0] in Verilog (bit 0 of bus A)  
A[0] in DEF | <> is equivalent to \A[0] in Verilog
- A\<0> in DEF/ [ ] is equivalent to \A<0> in Verilog  
A\<0> in DEF | <> is equivalent to \A<0> in Verilog
- A\[0\] in DEF/ [ ] is equivalent to \A[0] in Verilog  
A\[0\] in DEF | <> is equivalent to \A[0] in Verilog \*

The following example shows instance path names for Verilog and DEF equivalence:

- A/B in DEF/ [ ] represents instance path A.B (instance A in the top module, with instance B inside the module referenced by instance A) in Verilog.
- A\B in DEF/ [ ] represents instance \A/B in Verilog.
- A\B/C in DEF/ [ ] represents \A/B .C in Verilog (escaped instance \A/B in the top module, with instance C inside the module referenced by instance \A/B).
- The net and instance path A\B/C/D[0] in DEF/ [ ] will represent \A/B .C.D[0] in Verilog (escaped instance \A/B in the top module, with instance C inside the module referenced by instance \A/B, and bus D in that module with bit 0 being specified).

### ***Comparison of DEF and Verilog Escaping Semantics***

The DEF escape \ applies only to the next character and prevents the character from having a special meaning.

The Verilog escape \ affects the complete "token" and is terminated by a trailing white space (" ", Tab, Enter, etc.).

## **Order of DEF Statements**

Standard DEF files can contain the following statements and sections. You can define the statements and sections in any order; however, data must be defined before it is used. For

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

example, you must specify the `UNITS` statement before any statements that use values dependent on `UNITS` values, and `VIAS` statements must be defined before statements that use via names. If you specify statements and sections in the following order, all data is defined before being used.

```
[ VERSION statement ]
[ DIVIDERCHAR statement ]
[ BUSBITCHARS statement ]
DESIGN statement
[ TECHNOLOGY statement ]
[ UNITS statement ]
[ HISTORY statement ] ...
[ PROPERTYDEFINITIONS section ]
[ DIEAREA statement ]
[ ROWS statement ] ...
[ TRACKS statement ] ...
[ GCELLGRID statement ] ...
[ VIAS statement ]
[ STYLES statement ]
[ NONDEFAULTRULES statement ]
[ REGIONS statement ]
[ COMPONENTMASKSHIFT statement ]
[ COMPONENTS section ]
[ PINS section ]
[ PINPROPERTIES section ]
[ BLOCKAGES section ]
[ SLOTS section ]
[ FILLS section ]
[ SPECIALNETS section ]
[ NETS section ]
[ SCANCHAINS section ]
[ GROUPS section ]
[ BEGINEXT section ] ...
END DESIGN statement
```

## DEF Statement Definitions

The following definitions describe the syntax arguments for the statements and sections that make up a DEF file. The statements and sections are listed in alphabetical order, *not* in the order they must appear in a DEF file. For the correct order, see [Order of DEF Statements](#) on page 651.

### Blockages

```
[BLOCKAGES numBlockages ;
  [- LAYER layerName
    [ + SLOTS | + FILLS]
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
[ + PUSHDOWN]
[ + EXCEPTPGNET]
[ + COMPONENT compName]
[ + SPACING minSpacing | + DESIGNRULEWIDTH effectiveWidth]
[ + MASK maskNum]
    {RECT pt pt | POLYGON pt pt pt ...} ...
;] ...
[- PLACEMENT
    [ + SOFT | + PARTIAL maxDensity]
    [ + PUSHDOWN]
    [ + COMPONENT compName
        {RECT pt pt} ...
    ]
;] ...
END BLOCKAGES]
```

Defines placement and routing blockages in the design. You can define simple blockages (blockages specified for an area), or blockages that are associated with specific instances (components). Only placed instances can have instance-specific blockages. If you move the instance, its blockage moves with it.

**COMPONENT *compName*** Specifies a component with which to associate a blockage. Specify with **LAYER *layerName*** to create a blockage on *layerName* associated with a component. Specify with **PLACEMENT** to create a placement blockage associated with a component.

**DESIGNRULEWIDTH *effectiveWidth*** Specifies that the blockage has a width of *effectiveWidth* for the purposes of spacing calculations. If you specify **DESIGNRULEWIDTH**, you cannot specify **SPACING**. As a lot of spacing rules in advanced nodes no longer just rely on wire width, **DESIGNRULEWIDTH** is not allowed for 20nm and below nodes.  
*Type:* DEF database units

**EXCEPTPGNET** Indicates that the blockage only blocks signal net routing, and does not block power or ground net routing.

This can be used above noise sensitive blocks, to prevent signal routing on specific layers above the block, but allow power routing connections.

**FILLS** Creates a blockage on the specified layer where metal fill shapes cannot be placed.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

LAYER <i>layerName</i>	<p>Normally only cut or routing layers have blockages, but it is legal to create a blockage on any layer.</p> <p><b>Note:</b> Cut-layer blockages will prevent vias from being placed in that area.</p>
MASK <i>maskNum</i>	<p>Specifies which mask for double or triple patterning lithography to use for the specified shapes. The <i>maskNum</i> variable must be a positive integer. Most applications support values of only 1, 2, or 3. Shapes without any defined mask have no mask set (are uncolored).</p> <p>For example,</p> <pre>- LAYER metall + PUSHDOWN + MASK 1     RECT ( -300 -310 ) ( 320 330 )    #rectangle on mask 1     RECT ( -150 -160 ) ( 170 180 );  #rectangle on mask 1</pre>
<i>numBlockages</i>	<p>Specifies the number of blockages in the design specified in the BLOCKAGES section.</p>
PARTIAL <i>maxDensity</i>	<p>Indicates that the initial placement should not use more than <i>maxDensity</i> percentage of the blockage area for standard cells. Later placement of clock tree buffers, or buffers added during timing optimization ignore this blockage. The <i>maxDensity</i> value is calculated as:</p> $\text{standard cell area in blockage area} / \text{blockage area} \leq \text{maxDensity}$ <p>This can be used to reduce the density in a locally congested area, and preserve it for buffer insertion.</p> <p><b>Type:</b> Float <b>Value:</b> Between 0.0 and 100.0</p>
PLACEMENT	<p>Creates a placement blockage. You can create a simple placement blockage, or a placement blockage attached to a specific component.</p>
POLYGON <i>pt pt pt</i>	<p>Specifies a sequence of at least three points to generate a polygon geometry. The polygon edges must be parallel to the x axis, the y axis, or at a 45-degree angle. Each POLYGON statement defines a polygon generated by connecting each successive point, and then the first and last points. The <i>pt</i> syntax corresponds to a coordinate pair, such as <i>x y</i>. Specify an asterisk (*) to repeat the same value as the previous <i>x</i> or <i>y</i> value from the last point.</p>

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

PUSHDOWN	Specifies that the blockage was pushed down into the block from the top level of the design.
RECT <i>pt pt</i>	Specifies the coordinates of the blockage geometry. The coordinates you specify are absolute. If you associate a blockage with a component, the coordinates are not relative to the component's origin.
SOFT	Indicates that the initial placement should not use the area, but later phases, such as timing optimization or clock tree synthesis, can use the blockage area. This can be used to preserve certain areas (such as small channels between blocks) for buffer insertion after the initial placement.
SLOTS	Creates a blockage on the specified layer where slots cannot be placed.
SPACING <i>minSpacing</i>	<p>Specifies the minimum spacing allowed between this particular blockage and any other shape. The <i>minSpacing</i> value overrides all other normal LAYER-based spacing rules, including wide-wire spacing rules, end-of-line rules, parallel run-length rules, and so on. A blockage with SPACING is not "seen" by any other DRC check, except the simple check for <i>minSpacing</i> to any other routing shape on the same layer.</p> <p>The <i>minSpacing</i> value cannot be larger than the maximum spacing defined in the SPACING or SPACINGTABLE for that layer. Tools may change larger values to the maximum spacing value with a warning.</p> <p><i>Type:</i> Integer, specified in DEF database units</p>

#### Example 4-1 Blockages Statements

- The following BLOCKAGES section defines eight blockages in the following order: two *metal2* routing blockages, a pushed down routing blockage, a routing blockage attached to component |i4, a floating placement blockage, a pushed down placement blockage, a placement blockage attached to component |i3, and a fill blockage.

```
BLOCKAGES 7 ;
- LAYER metall
  RECT ( -300 -310 ) ( 320 330 )
  RECT ( -150 -160 ) ( 170 180 ) ;
- LAYER metall + PUSHDOWN
  RECT ( -150 -160 ) ( 170 180 ) ;
- LAYER metall + COMPONENT |i4
  RECT ( -150 -160 ) ( 170 180 ) ;
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
- PLACEMENT
    RECT ( -150 -160 ) ( 170 180 ) ;
- PLACEMENT + PUSHDOWN
    RECT ( -150 -160 ) ( 170 180 ) ;
- PLACEMENT + COMPONENT |i3
    RECT ( -150 -160 ) ( 170 180 ) ;
- LAYER metall + FILLS
    RECT ( -160 -170 ) ( 180 190 ) ;
```

END BLOCKAGES

- The following BLOCKAGES section defines two blockages. One requires minimum spacing of 1000 database units for its rectangle and polygon. The other requires that its rectangle's width be treated as 1000 database units for DRC checking.

BLOCKAGES 2 ;

```
- LAYER metall
    + SPACING 1000      #RECT and POLYGON require at least 1000 dbu spacing
    RECT ( -300 -310 ) ( 320 300 )
    POLYGON ( 0 0 ) ( * 100 ) ( 100 * ) ( 200 200 ) ( 200 0 ) ; #Has 45-degree
                                                                #edge
- LAYER metall
    + DESIGNRULEWIDTH 1000 #Treat the RECT as 1000 dbu wide for DRC checking
    RECT ( -150 -160 ) ( 170 180 ) ;
```

END BLOCKAGES

## Bus Bit Characters

BUSBITCHARS "delimiterPair" ;

Specifies the pair of characters used to specify bus bits when DEF names are mapped to or from other databases. The characters must be enclosed in double quotation marks. For example:

```
BUSBITCHARS "()" ;
```

If one of the bus bit characters appears in a DEF name as a regular character, you must use a backslash (\) before the character to prevent the DEF reader from interpreting the character as a bus bit delimiter.

If you do not specify the BUSBITCHARS statement in your DEF file, the default value is "[ ]".

## Component Mask Shift

```
[COMPONENTMASKSHIFT layer1 [layer2 ...] ;]
```



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Defines which layers of a component are allowed to be shifted from the original mask colors in the LEF. This can be useful to shift all the layers of a specific component in order to align the masks with other component or router mask settings to increase routing density. This definition allows a specific component to compactly describe the mask shifting for that component.

All the listed layers must have a LEF MASK statement to indicate that the specified layer is either a two or three mask layer. The order of the layers must be increasing from the highest layer down to the lowest layer in the LEF layer order.

#### Example 4-2 Component Mask Shift

The following example indicates that any given component can shift the mask on layers M3, M2, VIA1, or M1:

```
COMPONENTMASKSHIFT M3 M2 VIA1 M1 ;
```

This layer list is used to interpret the + MASKSHIFT *shiftLayerMasks* value for a specific component as shown in the example given below:

```
- i1/i2 AND2
  + MASKSHIFT 1102 #M3 shifted by 1, M2 by 1, VIA1 by 0, M1 by 2
  ...
```

For details on components, see the [Components](#) section.

## Components

```
COMPONENTS numComps ;
  [- compName modelName
    [+ EEQMASTER macroName]
    [+ SOURCE {NETLIST | DIST | USER | TIMING}]
    [+ {FIXED pt orient | COVER pt orient | PLACED pt orient
        | UNPLACED} ]
    [+ MASKSHIFT shiftLayerMasks]
    [+ HALO [SOFT] left bottom right top]
    [+ ROUTEHALO haloDist minLayer maxLayer]
    [+ WEIGHT weight]
    [+ REGION regionName]
    [+ PROPERTY {propName propVal} ...]...
  ;] ...
END COMPONENTS
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Defines design components, their location, and associated attributes.

*compName modelName* Specifies the component name in the design, which is an instance of *modelName*, the name of a model defined in the library. A *modelName* must be specified with each *compName*.

COVER *pt orient* Specifies that the component has a location and is a part of a cover macro. A COVER component cannot be moved by automatic tools or interactive commands. You must specify the component's location and its orientation.

EEQMASTER *macroName* Specifies that the component being defined should be electrically equivalent to the previously defined *macroName*.

FIXED *pt orient* Specifies that the component has a location and cannot be moved by automatic tools, but can be moved using interactive commands. You must specify the component's location and orientation.

HALO [SOFT] *left bottom right top* Specifies a placement blockage around the component. The halo extends from the LEF macro's left edge(s) by *left*, from the bottom edge(s) by *bottom*, from the right edge(s) by *right*, and from the top edge(s) by *top*. The LEF macro edges are either defined by the rectangle formed by the MACRO SIZE statement, or, if OVERLAP obstructions exist (OBS shapes on a layer with TYPE OVERLAP), the polygon formed by merging the OVERLAP shapes.

If SOFT is specified, the placement halo is honored only during initial placement; later phases, such as timing optimization or clock tree synthesis, can use the halo area. This can be used to preserve certain areas (such as small channels between blocks) for buffer insertion.

*Type:* Integer, specified in DEF database units

MASKSHIFT *shiftLayerMasks*

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Specifies shifting the cell-master masks used in double or triple patterning for specific layers of an instance of the cell-master. This is mostly used for standard cells where the placer or router may shift one or more layer mask assignments for better density.

The *shiftLayerMasks* variable is a hex-encoded digit, with one digit per multi-mask layer:

```
...<thirdLayerShift><secondLayerShift><bottomLayerShift>
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

The *bottomLayerShift* value is the mask-shift for the bottom-most multi-mask layer defined in the COMPONENTMASKSHIFT statement. The *secondMaskShift*, *thirdMaskShift*, and so on, are the shift values for each layer in order above the bottom-most multi-mask layer. The missing digits indicate that no shift is needed so 002 and 2 have the same meaning.

For 2-mask layers, the *LayerShift* value must be 0 or 1 and indicates:

0 - No mask-shift

1 - Shift the mask colors by 1 (mask 1->2, and 2->1)

For 3-mask layers, the *LayerShift* value can be 0, 1, 2, 3, 4, or 5 that indicates:

0 - No mask shift

1 - Shift by 1 (1->2, 2->3, 3->1)

2 - Shift by 2 (1->3, 2->1, 3->2)

3 - Mask 1 is fixed, swap 2 and 3

4 - Mask 2 is fixed, swap 1 and 3

5 - Mask 3 is fixed, swap 1 and 2

The purpose of 3, 4, 5 is for standard cells that have a fixed power-rail mask color, but the pins between the power-rails can still be shifted. Suppose you had a standard cell with mask 1 power rails, and three signal pins on mask 1, 2, and 3. A *LayerShift* of 3, will keep the mask 1 power rails and signal pin fixed on mask 1, while mask 2 and mask 3 signal pin shapes will swap mask colors.

See [Example 4-3](#) on page 660.

### Example 4-3 Mask Shift Layers for Components

The following example shows a LEF section that has a three-mask layer defined for M1, and two-mask layer defined for layers VIA1 and M2:

```
COMPONENTMASKSHIFT M2 VIA1 M1 ;
COMPONENTS 100 ;
- i1/i2 AND2
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

```
+ MASKSHIFT 2          #M1 layer masks are shifted by 2, no shift for others
...
- i1/i3 OR2
+ MASKSHIFT 103        ##M1 layer has shift 3, VIA1 0, M2 1
...
```

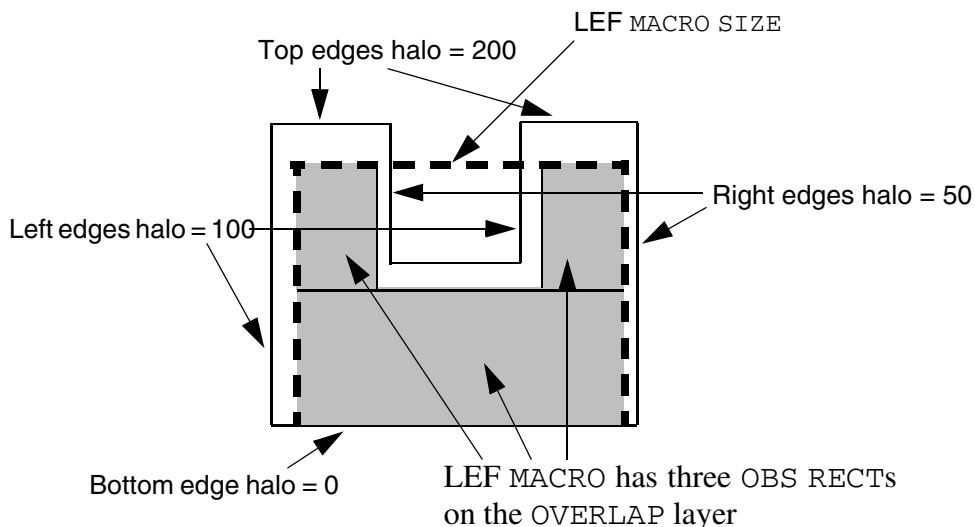
If an application shifts the layers M1, VIA1, and M2, then the above example indicates that the instance of AND2 cell shifts the M1 layer masks by 2. Since M1 is a three-mask layer, this shows that the cell-master M1 layer mask 1 shifts to 3, mask 2 shifts to 1, and mask 3 shifts to 2. The other layer masks are not shifted. The instance of OR2 cell shifts the M1 layer masks by 3. For a 3-mask layer this means keep mask 1 fixed, mask 2 shifts to 3, and mask 3 shifts to 2). The VIA1 layer does not shift and the M2 layer masks are shifted by 1 (for a two-mask layer this means that mask 1 shifts to 2, and 2 shifts to 1).

#### Example 4-4 Component Halo

The following statement creates a placement blockage for a “U-shaped” LEF macro, as illustrated in [Figure 4-1](#) on page 661:

```
- i1/i2
+ PLACED ( 0 0 ) N
+ HALO 100 0 50 200 ;
```

**Figure 4-1 Component Halo**



*numComps*

Specifies the number of components defined in the COMPONENTS section.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

`PLACED pt orient`

Specifies that the component has a location, but can be moved using automatic layout tools. You must specify the component's location and orientation.

`PROPERTY propName propVal`

Specifies a numerical or string value for a component property defined in the `PROPERTYDEFINITIONS` statement. The *propName* you specify must match the *propName* listed in the `PROPERTYDEFINITIONS` statement.

`REGION regionName`

Specifies a region in which the component must lie. *regionName* specifies a region already defined in the `REGIONS` section. If the region is smaller than the bounding rectangle of the component itself, the DEF reader issues an error message and ignores the argument. If the region does not contain a legal location for the component, the component remains unplaced after the placement step.

`ROUTEHALO haloDist minLayer maxLayer`

Specifies that signal routing in the “halo area” around the block boundary should be perpendicular to the block edge in order to reach the block pins. The halo area is the area within *haloDist* of the block boundary (see the Figure below). A routing-halo is intended to be used to minimize cross coupling between routing at the current level of the design, and routing inside the block. It has no effect on power routing. Note that this also means it is allowed to route in the “halo corners” because routing in the “halo corner” is not adjacent to the block boundary, and will not cause any significant cross-coupling with routing inside the block.

The routing halo exists for the routing layers between *minLayer* and *maxLayer*. The layer you specify for *minLayer* must be a lower routing layer than *maxLayer*.

*Type:* Integer, specified in DEF database units (*haloDist*); string that matches a LEF routing layer name (*minLayer* and *maxLayer*)

### Example 4-5 Route Halo Example

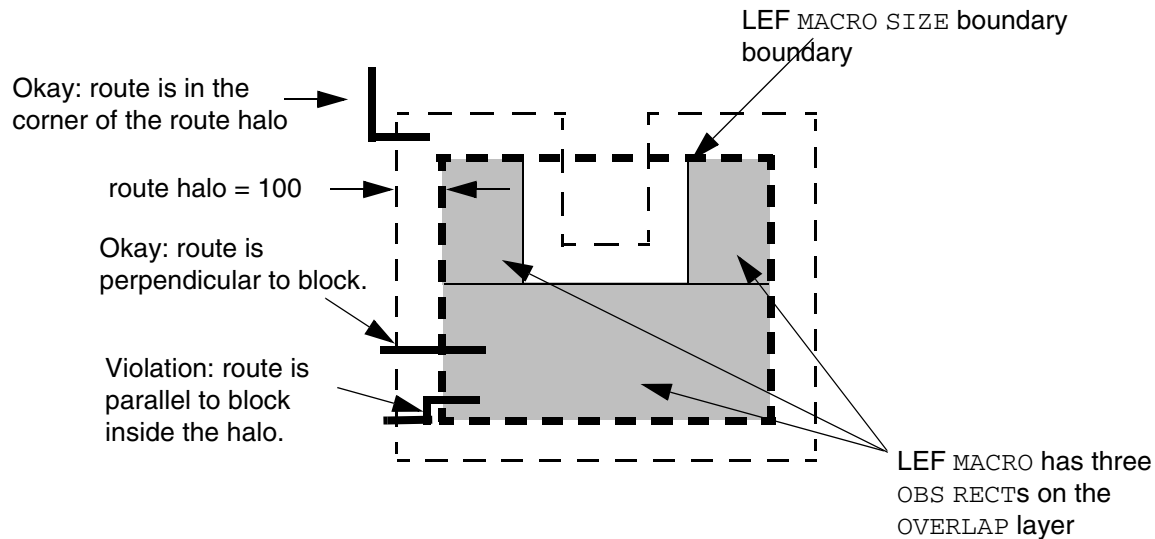
For a U-shaped macro, the following component description results in the halo shown in [Figure 4-2](#) on page 663.

```
- il/i2
  + PLACED ( 0 0 ) N
  + ROUTEHALO 100 metal1 metal3 ;
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

**Figure 4-2 Route Halo**



SOURCE {NETLIST | DIST | USER | TIMING}

Specifies the source of the component.

*Value:* Specify one of the following:

DIST	Component is a physical component (that is, it only connects to power or ground nets), such as filler cells, well-taps, and decoupling caps.
NETLIST	Component is specified in the original netlist. This is the default value, and is normally not written out in the DEF file.
TIMING	Component is a logical rather than physical change to the netlist, and is typically used as a buffer for a clock-tree, or to improve timing on long nets.
USER	Component is generated by the user for some user-defined reason.

UNPLACED

Specifies that the component does not have a location.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

**WEIGHT** *weight*









Specifies the weight of the component, which determines whether or not automatic placement attempts to keep the component near the specified location. *weight* is only meaningful when the component is placed. All non-zero weights have the same effect during automatic placement.

*Default:* 0

### Specifying Orientation

If a component has a location, you must specify its location and orientation. A component can have any of the following orientations: N, S, W, E, FN, FS, FW, or FE.

Orientation terminology can differ between tools. The following table maps the orientation terminology used in LEF and DEF files to the OpenAccess database format.

LEF/DEF	OpenAccess	Definition
N (North)	R0	
S (South)	R180	
W (West)	R90	
E (East)	R270	
FN (Flipped North)	MY	
FS (Flipped South)	MX	
FW (Flipped West)	MX90	
FE (Flipped East)	MY90	

Components are always placed such that the lower left corner of the cell is the origin (0,0) after any orientation. When a component flips about the y axis, it flips about the component center. When a component rotates, the lower left corner of the bounding box of the component's sites remains at the same placement location.

### Design

**DESIGN** *designName* ;

Specifies a name for the design. The DEF reader reports a warning if this name is different from that in the database. In case of a conflict, the just specified name overrides the old name.



## Die Area

```
[DIEAREA pt pt [pt] ... ;]
```

If two points are defined, specifies two corners of the bounding rectangle for the design. If more than two points are defined, specifies the points of a polygon that forms the die area. The edges of the polygon must be parallel to the x or y axis (45-degree shapes are not allowed), and the last point is connected to the first point. All points are integers, specified as DEF database units.

Geometric shapes (such as blockages, pins, and special net routing) can be outside of the die area, to allow proper modeling of pushed down routing from top-level designs into sub blocks. However, routing tracks should still be inside the die area.

### Example 4-6 Die Area Statements

The following statements show various ways to define the die area.

```
DIEAREA ( 0 0 ) ( 100 100 ) ;                               #Rectangle from 0,0 to 100,100
DIEAREA ( 0 0 ) ( 0 100 ) ( 100 100 ) ( 100 0 ) ;          #Same rectangle as a polygon
DIEAREA ( 0 0 ) ( 0 100 ) ( 50 100 ) ( 50 50 ) ( 100 50 ) ( 100 0 ) ; #L-shaped polygon
```

## Divider Character

```
DIVIDERCHAR "character" ;
```

Specifies the character used to express hierarchy when DEF names are mapped to or from other databases. The character must be enclosed in double quotation marks. For example:

```
DIVIDERCHAR "/" ;
```

If the divider character appears in a DEF name as a regular character, you must use a backslash (\) before the character to prevent the DEF reader from interpreting the character as a hierarchy delimiter.

If you do not specify the DIVIDERCHAR statement in your LEF file, the default value is "/".

## Extensions

```
[BEGINEXT "tag"
      extensionText
ENDEXT]
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Adds customized syntax to the DEF file that can be ignored by tools that do not use that syntax. You can also use extensions to add new syntax not yet supported by your version of LEF/DEF, if you are using version 5.1 or later. Add extensions as separate sections.

*extensionText*

Defines the contents of the extension.

*"tag"*

Identifies the extension block. You must enclose *tag* in quotes.

#### Example 4-7 Extension Statement

```
BEGINEXT "1VSI Signature 1.0"
    CREATOR "company name"
    DATE "timestamp"
    REVISION "revision number"
ENDEXT
```

## Fills

```
[FILLS numFills ;
    [- LAYER layerName [+ MASK maskNum] [+ OPC]
        {RECT pt pt | POLYGON pt pt pt ...} ... ;] ...
    [- VIA viaName [+ MASK viaMaskNum] [+ OPC] pt ... ;] ...
END FILLS]
```

Defines the rectangular shapes that represent metal fills in the design. Each fill is defined as an individual rectangle.

LAYER <i>layerName</i>	Specifies the layer on which to create the fill.
MASK <i>maskNum</i>	Specifies which mask for double or triple patterning lithography to use for the given rectangles or polygons. The <i>maskNum</i> variable must be a positive integer. Most applications support values of 1, 2, or 3 only. Shapes without any defined mask have no mask setting (are uncolored).

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

`MASK viaMaskNum`

Specifies which mask for double or triple patterning lithography to be applied to via shapes on each layer.

The *viaMaskNum* variable is a hex-encoded three digit value of the form:

*<topMaskNum><cutMaskNum><bottomMaskNum>*

For example, MASK 113 means that the top metal and cut layer *maskNum* is 1, and the bottom metal layer *maskNum* is 3. A value of 0 means the shape on that layer has no mask assignment (is uncolored), so 013 means the top layer is uncolored. If either the first or second digit is missing, they are assumed to be 0, so 013 and 13 mean the same thing. Most applications support *maskNums* of 0, 1, 2, or 3 for double or triple patterning.

The *topMaskNum* and *bottomMaskNum* variables specify which mask the corresponding metal shape belongs to. The via-master metal mask values have no effect.

For the cut layer, the *cutMaskNum* defines the mask for the bottom-most, and then the left-most cut. For multi-cut vias, the via-instance cut masks are derived from the via-master cut mask values. The via-master must have a mask defined for all the cut shapes and every via-master cut mask is "shifted" (from 1 to 2, 2 to 1 for two mask layers, and from 1 to 2, 2 to 3, and 3 to 1 for three mask layers) so the lower-left cut matches the *cutMaskNum* value.

Similarly, for the metal layer, the *topMaskNum/bottomMaskNum* would define the mask for the bottom-most, then leftmost metal shape. For multiple disjoint metal shapes, the via-instance metal masks are derived from the via-master metal mask values. The via-master must have a mask defined for all of the metal shapes, and every via-master metal mask is "shifted" (1->2, 2->1 for two mask layers, 1->2, 2->3, 3->1 for three mask layers) so the lower-left cut matches the *topMaskNum/bottomMaskNum* value.

See [Example 4-9](#) on page 669.

`numFills`

Specifies the number of LAYER statements in the FILLS statement, *not* the number of rectangles.

`OPC`

Indicates that the FILL shapes require OPC correction during mask generation.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

<code>POLYGON <i>pt pt pt</i></code>	Specifies a sequence of at least three points to generate a polygon geometry. The polygon edges must be parallel to the x axis, the y axis, or at a 45-degree angle. Each <code>POLYGON</code> statement defines a polygon generated by connecting each successive point, and then the first and last points. The <i>pt</i> syntax corresponds to a coordinate pair, such as <i>x y</i> . Specify an asterisk (*) to repeat the same value as the previous <i>x</i> or <i>y</i> value from the last point.
<code>RECT <i>pt pt</i></code>	Specifies the lower left and upper right corner coordinates of the fill geometry.
<code>VIA <i>viaName pt</i></code>	Places the via named <i>viaName</i> at the specified (x y) location ( <i>pt</i> ). <i>viaName</i> must be a previously defined via in the DEF VIAS or LEF VIA section. <i>Type: (pt)</i> Integers, specified in DEF database units

#### Example 4-8 Fills Statements

- The following `FILLS` statement defines fill geometries for layers *metal1* and *metal2*:

```
FILLS 2 ;
- LAYER metal1
    RECT ( 1000 2000 ) ( 1500 4000 )
    RECT ( 2000 2000 ) ( 2500 4000 )
    RECT ( 3000 2000 ) ( 3500 4000 ) ;
- LAYER metal2
    RECT ( 1000 2000 ) ( 1500 4000 )
    RECT ( 1000 4500 ) ( 1500 6500 )
    RECT ( 1000 7000 ) ( 1500 9000 )
    RECT ( 1000 9500 ) ( 1500 11500 ) ;
END FILLS
```

- The following `FILLS` statement defines two rectangles and one polygon fill geometries:

```
FILLS 1 ;
-LAYER metal1
    RECT ( 100 200 ) ( 150 400 )
    POLYGON ( 100 100 ) ( 200 200 ) ( 300 200 ) ( 300 100 )
    RECT ( 300 200 ) ( 350 400 ) ;
END FILLS
```

- Shapes on the `TRIMMETAL` layers are written out in the `FILLS` section in DEF in the following format:

```
- LAYER TM1 + MASK x RECT (x x) (x x) ;
```

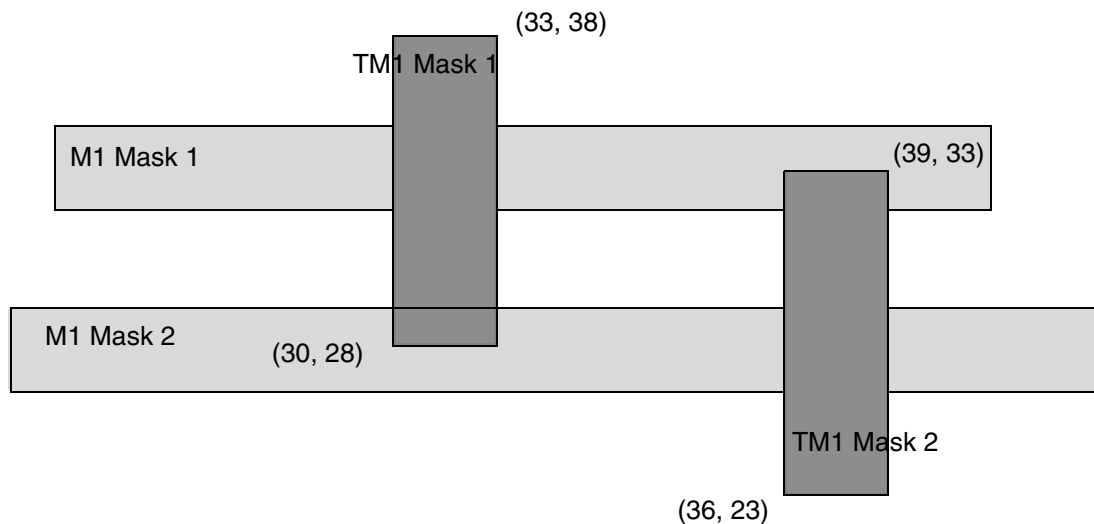
## LEF/DEF 5.8 Language Reference

### DEF Syntax

The following `FILLS` statement defines two rectangular shapes on the `TRIMMETAL` layer `TM1`, which is defined in a property statement in the masterslice layer section in LEF:

```
FILLS 2 ;
- LAYER TM1 + MASK 1 RECT (30 28) (33 38)
- LAYER TM1 + MASK 2 RECT (36 23) (39 33)
...
END FILLS
```

**Figure 4-3 Trim Metal Layer Shapes in the FILLS Section**



- The following `FILLS` statement defines two rectangles and two via fill geometries for layer *metal1*. The rectangles and one of the via fill shapes require OPC correction.

```
FILLS 3 ;
-LAYER metal1 + OPC
  RECT ( 0 0 ) ( 100 100 )
  RECT ( 200 200 ) ( 300 300 ) ;
-VIA via26
  ( 500 500 )
  ( 800 800 ) ;
-VIA via28 + OPC
  ( 900 900 ) ;
END FILLS
```

### Example 4-9 Multi-Mask Patterns for Fills

The following example shows multi-mask patterning for fills:

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
- LAYER M1 + MASK 1 RECT ( 10 10 ) ( 11 11 ) ; #RECT on MASK 1
- LAYER M2 RECT ( 10 10 ) ( 11 11 ) ; #RECT is uncolored
- VIA VIA1_1 + MASK 031 ( 10 10 ) ; #VIA with top-cut-bot mask 031
```

This indicates that the:

- M1 rectangle shape is on MASK 1
- M2 rectangle shape has no mask set and is uncolored
- VIA1\_1 via will have:
  - no mask set for the top metal shape - it is uncolored (*topMaskNum* is 0 in the 031 value). Note that the via-master color of the top metal shape does not matter.
  - mask 1 for the bottom metal shape (*bottomMaskNum* is 1 in the 031 value)
  - for the cut layer, the bottom-most, and then the left-most cut of the via-instance is mask 3. The mask for the other cuts of the via-instance are derived from the via-master by "shifting" the via-master cut masks to match. So if the via-master's bottom- left cut is MASK 1, then the via-master cuts on mask 1 become mask 3 for the via-instance, and similarly cuts on 2 become 1, and cuts on 3 become 2. See [Figure 4-11](#) on page 740.

## GCell Grid

```
[GCELLGRID
  {X start DO numColumns+1 STEP space} ...
  {Y start DO numRows+1 STEP space ;} ...]
```

Defines the gcell grid for a standard cell-based design. Each GCELLGRID statement specifies a set of vertical (x) and horizontal (y) lines, or tracks, that define the gcell grid.

Typically, the GCELLGRID is automatically generated by a particular router, and is not manually created by the designer.

DO *numColumns+1*

Specifies the number of columns in the grid.

DO *numRows+1*

Specifies the number of rows in the grid.

STEP *space*

Specifies the spacing between tracks.

*X start, Y start*

Specify the location of the first vertical (x) and first horizontal (y) track.

### ***GCell Grid Boundary Information***

The boundary of the gcell grid is the rectangle formed by the extreme vertical and horizontal lines. The gcell grid partitions the routing portion of the design into rectangles, called gcells. The lower left corner of a gcell is the origin. The x size of a gcell is the distance between the upper and lower bounding vertical lines, and the y size is the distance between the upper and lower bounding horizontal lines.

For example, the grid formed by the following two GCELLGRID statements creates gcells that are all the same size (100 x 200 in the following):

```
GCELLGRID X 1000 DO 101 STEP 100 ;  
GCELLGRID Y 1000 DO 101 STEP 200 ;
```

A gcell grid in which all gcells are the same size is called a uniform gcell grid. Adding GCELLGRID statements can increase the granularity of the grid, and can also result in a nonuniform grid, in which gcells have different sizes.

For example, adding the following two statements to the above grid generates a nonuniform grid:

```
GCELLGRID X 3050 DO 61 STEP 100 ;  
GCELLGRID Y 5100 DO 61 STEP 200 ;
```

When a track segment is contained inside a gcell, the track segment belongs to that gcell. If a track segment is aligned on the boundary of a gcell, that segment belongs to the gcell only if it is aligned on the left or bottom edges of the gcell. Track segments aligned on the top or right edges of a gcell belong to the next gcell.

### ***GCell Grid Restrictions***

Every track segment must belong to a gcell, so gcell grids have the following restrictions:

- The x coordinate of the last vertical track must be less than, and not equal to, the x coordinate of the last vertical gcell line.
- The y coordinate of the last horizontal track must be less than, and not equal to, the y coordinate of the last horizontal gcell line.

Gcells grids also have the following restrictions:

- Each GCELLGRID statement must define two lines.

- Every gcell need not contain the vertex of a track grid. But, those that do must be at least as large in both directions as the default wire widths on all layers.

## Groups

```
[GROUPS numGroups ;  
  [- groupName [compNamePattern ... ]  
    [+ REGION regionNam]  
    [+ PROPERTY {propName propVal} ...] ...  
  ;] ...  
END GROUPS]
```

Defines groups in a design.

*compNamePattern*

Specifies the components that make up the group. Do not assign any component to more than one group. You can specify any of the following:

- ❑ A component name, for example C3205
- ❑ A list of component names separated by spaces, for example, I01 I02 C3204 C3205
- ❑ A pattern for a set of components, for example, IO\* and C320\*

**Note:** An empty group with no component names is allowed.

*groupName*

Specifies the name for a group of components.

*numGroups*

Specifies the number of groups defined in the GROUPS section.

PROPERTY *propName propVal*

Specifies a numerical or string value for a group property defined in the PROPERTYDEFINITIONS statement. The *propName* you specify must match the *propName* listed in the PROPERTYDEFINITIONS statement.

REGION *regionName*

Specifies a rectangular region in which the group must lie. *regionName* specifies a region previously defined in the REGIONS section. If region restrictions are specified in both COMPONENT and GROUP statements for the same component, the component restriction overrides the group restriction.



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

## History

```
[HISTORY anyText ;] ...
```

Lists a historical record about the design. Each line indicates one record. Any text excluding a semicolon (;) can be included in *anyText*. The semicolon terminates the HISTORY statement. Linefeed and Return do not terminate the HISTORY statement. Multiple HISTORY lines can appear in a file.

## Nets

```
NETS numNets ;
  [- { netName
    [ ( {compName pinName | PIN pinName} [+ SYNTHESIZED] ) ] ...
    | MUSTJOIN ( compName pinName ) }
  [+ SHIELDNET shieldNetName ] ...
  [+ VPIN vpinName [LAYER layerName] pt pt
    [PLACED pt orient | FIXED pt orient | COVER pt orient] ] ...
  [+ SUBNET subnetName
    [ ( {compName pinName | PIN pinName | VPIN vpinName} ) ] ...
    [NONDEFAULTRULE rulename]
    [regularWiring] ...] ...
  [+ XTALK class]
  [+ NONDEFAULTRULE ruleName]
  [regularWiring] ...
  [+ SOURCE {DIST | NETLIST | TEST | TIMING | USER}]
  [+ FIXEDBUMP]
  [+ FREQUENCY frequency]
  [+ ORIGINAL netName]
  [+ USE {ANALOG | CLOCK | GROUND | POWER | RESET | SCAN | SIGNAL
    | TIEOFF}]
  [+ PATTERN {BALANCED | STEINER | TRUNK | WIREDLOGIC}]
  [+ ESTCAP wireCapacitance]
  [+ WEIGHT weight]
  [+ PROPERTY {propName propVal} ...] ...
;] ...

END NETS
```

Defines netlist connectivity and regular-routes for nets containing regular pins. These routes are normally created by a signal router that can rip-up and repair the routing. The SPECIALNETS statement defines netlist connectivity and routing for special-routes that are created by "special routers" or "manually" and should not be modified by a signal router. Special routes are normally used for power-routing, fixed clock-mesh routing, high-speed buses, critical analog routes, or flip-chip routing on the top-metal layer to bumps.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Input arguments for a net can appear in the `NETS` section or the `SPECIALNETS` section. In case of conflicting values, the DEF reader uses the last value encountered. `NETS` and `SPECIALNETS` statements can appear more than once in a DEF file. If a particular net has mixed wiring or pins, specify the special wiring and pins first.

### Arguments

*compName pinName*

Specifies the name of a regular component pin on a net or a subnet. LEF `MUSTJOIN` pins, if any, are not included; only the master pin (that is, the one without the `MUSTJOIN` statement) is included. If a subnet includes regular pins, the regular pins must be included in the parent net.

`COVER pt orient`

Specifies that the pin has a location and is a part of the cover macro. A `COVER` pin cannot be moved by automatic tools or by interactive commands. You must specify the pin's location and orientation.

`ESTCAP wireCapacitance`

Specifies the estimated wire capacitance for the net. `ESTCAP` can be loaded with simulation data to generate net constraints for timing-driven layout.

`FIXED pt orient`

Specifies that the pin has a location and cannot be moved by automatic tools, but can be moved by interactive commands. You must specify the pin's location and orientation.

`FIXEDBUMP`

Indicates that the bump net cannot be reassigned to a different pin.

It is legal to have a pin without geometry to indicate a logical connection, and to have a net that connects that pin to two other instance pins that have geometry. Area I/Os have a logical pin that is connected to a bump and an input driver cell. The bump and driver cell have pin geometries (and, therefore, should be routed and extracted), but the logical pin is the external pin name without geometry (typically the Verilog pin name for the chip). Because bump nets are usually routed with special routing, they also can be specified in the `SPECIALNETS` statement. If a net name appears in both the `NETS` and `SPECIALNETS` statements, the `FIXEDBUMP` keyword also should appear in both statements. However, the value only exists once within a given application's database for the net name.

Because DEF is often used incrementally, the last value read in is used. Therefore, in a typical DEF file, if the same net appears in both statements, the `FIXEDBUMP` keyword (or lack of it) in the `NETS` statement is the value that is used, because the `NETS` statement

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

is defined after the `SPECIALNETS` statement.

For an example specifying the `FIXEDBUMP` keyword, see [“Fixed Bump”](#) on page 730.

`FREQUENCY` *frequency*

Specifies the frequency of the net, in hertz. The frequency value is used by the router to choose the correct number of via cuts required for a given net, and by validation tools to verify that the AC current density rules are met. For example, a net described with `+FREQUENCY 100` indicates the net has 100 rising and 100 falling transitions in 1 second.

*Type:* Float

`LAYER` *layerName*

Specifies the layer on which the virtual pin lies.

`MUSTJOIN` (*compName pinName*)

Specifies that the net is a mustjoin. If a net is designated `MUSTJOIN`, its name is generated by the system. Only one net should connect to any set of mustjoin pins.

Mustjoin pins for macros are defined in LEF. The only reason to specify a `MUSTJOIN` net in DEF (identified arbitrarily by one of its pins) is to specify prewiring for the `MUSTJOIN` connection.

Otherwise, nets are generated automatically where needed for mustjoin connections specified in the library. If the input file specifies that a mustjoin pin is connected to a net, the DEF reader connects the set of mustjoin pins to the same net. If the input file does not specify connections to any of the mustjoin pins, the DEF reader creates a local `MUSTJOIN` net.

*netName*

Specifies the name for the net. Each statement in the `NETS` section describes a single net. There are two ways of identifying the net: *netName* or `MUSTJOIN`. If the *netName* is given, a list of pins to connect to the net also can be specified. Each pin is identified by a component name and pin name pair (*compName pinName*) or as an I/O pin (`PIN pinName`). Parentheses ensure readability of output. The keyword `MUSTJOIN` cannot be used as a *netName*.

`NONDEFAULTRULE` *ruleName*

By default the width of any route segment in the `NETS regularWiring` section is defined by the default width (`LEF WIDTH` statement value for the routing layer).

This keyword specifies a nondefault rule to use instead of the default rule when creating the net and wiring. When specified with `SUBNET`, identifies the nondefault rule to use when creating the subnet and its wiring.

The width of any route segment is defined by the corresponding `NONDEFAULTRULE WIDTH` for that layer.

### Wrong-way Width Rules

Some technologies required larger widths for wrong-way routing than in the preferred direction. If the wrong-way width is larger than the default or NDR width, then the wrong-way width is used for wrong-way routes on that layer. The implicit routing extension is still half of the default or NDR width, even for wrong-way routes.

The following LEF DRC rules allow a `WRONGDIRECTION` keyword that defines wrong-way widths that will affect the width of any wrong-way routes in the `DEF NETS` section:

`LEF58_WIDTH`

`LEF58_WIDHTABLE`

`LEF58_SPANLENGHTABLE`

See the "[Impact of Wrong-way Width Rules on page 689](#)" section for examples and more details.

*numNets*

Specifies the number of nets defined in the `NETS` section.

`ORIGINAL netName`

Specifies the original net partitioned to create multiple nets, including the net being defined.

`PATTERN {BALANCED | STEINER | TRUNK | WIREDLOGIC}`

Specifies the routing pattern used for the net.

*Default:* STEINER

*Value:* Specify one of the following:

BALANCED	Used to minimize skews in timing delays for clock nets.
STEINER	Used to minimize net length.
TRUNK	Used to minimize delay for global nets.
WIREDLOGIC	Used in ECL designs to connect output and mustjoin pins before routing to the remaining pins.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

PIN *pinName*

Specifies the name of an I/O pin on a net or a subnet.

PLACED *pt orient*

Specifies that the pin has a location, but can be moved during automatic layout. You must specify the pin's location and orientation.

```
PROPERTY propName propVal
```

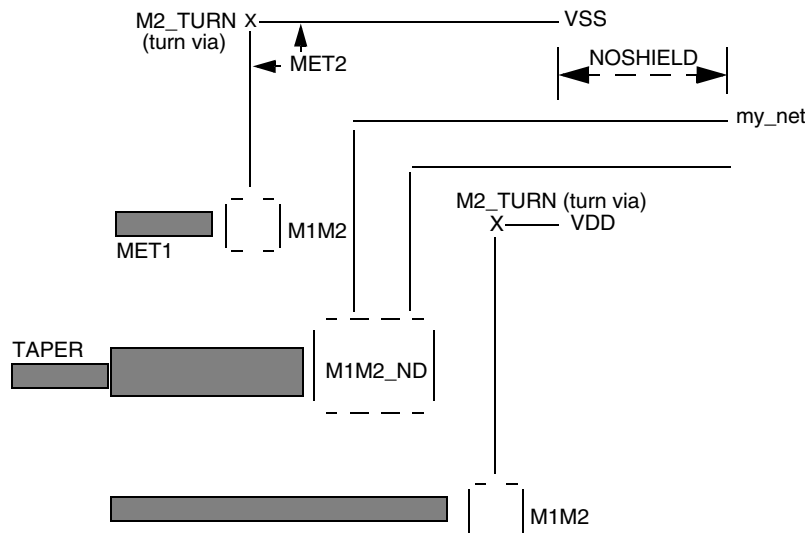
Specifies a numerical or string value for a net property defined in the `PROPERTYDEFINITIONS` statement. The *propName* you specify must match the *propName* listed in the `PROPERTYDEFINITIONS` statement.

*regularWiring*

Specifies the regular physical wiring for the net or subnet. For regular wiring syntax, see [“Regular Wiring Statement”](#) on page 680.

SHIELDNET *shieldNetName*

Specifies the name of a special net that shields the regular net being defined. A shield net for a regular net is defined earlier in the DEF file in the `SPECIALNETS` section.



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

SOURCE {DIST | NETLIST | TEST | TIMING | USER}

Specifies the source of the net. The value of this field is preserved when input to the DEF reader.

*Value:* Specify one of the following:

DIST	Net is the result of adding physical components (that is, components that only connect to power or ground nets), such as filler cells, well-taps, tie-high and tie-low cells, and decoupling caps.
NETLIST	Net is defined in the original netlist. This is the default value, and is not normally written out in the DEF file.
TEST	Net is part of a scanchain.
TIMING	Net represents a logical rather than physical change to netlist, and is used typically as a buffer for a clock-tree, or to improve timing on long nets.
USER	Net is user defined.

SUBNET *subnetName*

Names and defines a subnet of the regular net *netName*. A subnet must have at least two pins. The subnet pins can be virtual pins, regular pins, or a combination of virtual and regular pins. A subnet pin cannot be a mustjoin pin.

SYNTHESIZED

Used by some tools to indicate that the pin is part of a synthesized scan chain.

USE {ANALOG | CLOCK | GROUND | POWER | RESET | SCAN | SIGNAL | TIEOFF}

Specifies how the net is used.

*Value:* Specify one of the following:

ANALOG	Used as an analog signal net.
CLOCK	Used as a clock net.
GROUND	Used as a ground net.
POWER	Used as a power net.
RESET	Used as a reset net.
SCAN	Used as a scan net.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

SIGNAL	Used as a digital signal net.
TIEOFF	Used as a tie-high or tie-low net.

`VPIN vpinName pt pt`

Specifies the name of a virtual pin, and its physical geometry. Virtual pins can be used only in subnets. A SUBNET statement refers to virtual pins by the *vpinName* specified here. You must define each virtual pin in a + VPIN statement before you can list it in a SUBNET statement.

#### Example 4-10 Virtual Pin

The following example defines a virtual pin:

```
+ VPIN M7K.v2 LAYER MET2 ( -10 -10 ) ( 10 10 ) FIXED ( 10 10 )
+ SUBNET M7K.2 ( VPIN M7K.v2 ) ( /PREG_CTRL/I$73/A I )
  NONDEFAULTRULE rule1
  ROUTED MET2 ( 27060 341440 ) ( 26880 * ) ( * 213280 )
  M1M2 ( 95040 * ) ( * 217600 ) ( 95280 * )
  NEW MET1 ( 1920 124960 ) ( 87840 * )
  COVER MET2 ( 27060 341440 ) ( 26880 * )
```

`WEIGHT weight`

Specifies the weight of the net. Automatic layout tools attempt to shorten the lengths of nets with high weights. A value of 0 indicates that the net length for that net can be ignored. The default value of 1 specifies that the net should be treated normally. A larger weight specifies that the tool should try harder to minimize the net length of that net. For normal use, timing constraints are generally a better method to use for controlling net length than net weights. For the best results, you should typically limit the maximum weight to 10, and not add weights to more than 3 percent of the nets.

**Default:** 1

**Type:** Integer

`XTALK class`

Specifies the crosstalk class number for the net. If you specify the default value (0), XTALK will not be written to the DEF file.

**Default:** 0

**Type:** Integer

**Value:** 0 to 200

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

#### Regular Wiring Statement

```
{+ COVER | + FIXED | + ROUTED | + NOSHIELD}  
  layerName [TAPER | TAPERRULE ruleName] [STYLE styleNum]  
    routingPoints  
  [NEW layerName [TAPER | TAPERRULE ruleName] [STYLE styleNum]  
    routingPoints  
  ] ...
```

Specifies regular wiring for the net.

#### COVER

Specifies that the wiring cannot be moved by either automatic layout or interactive commands. If no wiring is specified for a particular net, the net is unrouted. If you specify COVER, you must also specify *layerName*.

#### FIXED

Specifies that the wiring cannot be moved by automatic layout, but can be changed by interactive commands. If no wiring is specified for a particular net, the net is unrouted. If you specify FIXED, you must also specify *layerName*.

#### *layerName*

Specifies the layer on which the wire lies. You must specify *layerName* if you specify COVER, FIXED, ROUTED, or NEW. Specified layers must be routable; reference to a cut layer generates an error.

#### NEW *layerName*

Indicates a new wire segment (that is, there is no wire segment between the last specified coordinate and the next coordinate), and specifies the name of the layer on which the new wire lies. Noncontinuous paths can be defined in this manner.

#### NOSHIELD

Specifies that the last wide segment of the net is not shielded. If the last segment is not shielded, and is tapered, specify TAPER under the LAYER argument, instead of NOSHIELD.

#### ROUTED

Specifies that the wiring can be moved by the automatic layout tools. If no wiring is specified for a particular net, the net is unrouted. If you specify ROUTED, you must also specify *layerName*. An example of DEF NETS routing is shown in [DEF NETS Examples](#) on page 691.



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

#### *routingPoints*

Defines the center line coordinates of the route on *layerName*. For information about using routing points, see [“Defining Routing Points”](#) on page 688.

As described above, the width of the routes is defined by the default width (e.g., LEF WIDTH statement on the routing layer) or a NONDEFAULTRULE width for the routing layer. In addition, some technologies require larger widths for wrong-way routes that may increase the width. See the [“Impact of Wrong-way Width Rules on page 689”](#) section for more details.

The *routingPoints* syntax is defined as follows:

```
{ ( x y [extValue] )
  { [MASK maskNum] ( x y [extValue] )
    | [MASK viaMaskNum] viaName [orient]
    | [MASK maskNum] RECT ( deltax1 deltax1 deltax2 deltax2 )
    | VIRTUAL ( x y ) } } ...
```

*extValue*

Specifies the amount by which the wire is extended past the endpoint of the segment. The extension value must be greater than or equal to 0 (zero).

*Default:* Half the wire width

*Type:* Integer, specified in database units

Some tools only allow 0 or the WIREEXTENSION value from the LAYER or NONDEFAULTRULE statement.

*MASK maskNum*

Specifies which mask for double or triple patterning lithography to use for the next wire or RECT. The *maskNum* variable must be a positive integer - most applications support values of 1, 2, or 3 only. Shapes without any defined mask have no mask set (that is, they are uncolored).

*MASK viaMaskNum*

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Specifies which mask for double or triple patterning lithography is to be applied to the next via's shapes on each layer.

The *viaMaskNum* variable is a hex-encoded three-digit value of the form:

*<topMaskNum><cutMaskNum><bottomMaskNum>*

For example, MASK 113 means that the top metal and cut layer *maskNum* is 1, and the bottom metal layer *maskNum* is 3. A value of 0 means that the shape on the layer has no mask assignment (is uncolored), so 013 means the top layer is uncolored. If either the first or second digit is missing, they are assumed to be 0, so 013 and 13 mean the same thing. Most applications support *maskNums* of 0, 1, 2, or 3 only for double or triple patterning.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

The *topMaskNum* and *bottomMaskNum* variables specify which mask the corresponding metal shape belongs to. The via-master metal mask values have no effect.

For the cut layer, the *cutMaskNum* variable defines the mask for the bottom-most, and then the left-most cut. For multi-cut vias, the via-instance cut masks are derived from the via-master cut mask values. The via-master must have a mask defined for all the cut shapes and every via-master cut mask is "shifted" (from 1 to 2, and 2 to 1 for two mask layers, and from 1 to 2, 2 to 3, and 3 to 1 for three mask layers), so the lower-left cut matches the *cutMaskNum* value.

Similarly, for the metal layer, the *topMaskNum/bottomMaskNum* would define the mask for the bottom-most, then leftmost metal shape. For multiple disjoint metal shapes, the via-instance metal masks are derived from the via-master metal mask values. The via-master must have a mask defined for all of the metal shapes, and every via-master metal mask is "shifted" (1->2, 2->1 for two mask layers, 1->2, 2->3, 3->1 for three mask layers) so the lower-left cut matches the *topMaskNum/bottomMaskNum* value.

See [Example 4-11](#) on page 685.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

*orient*

Specifies the orientation of the *viaName* that precedes it, using the standard DEF orientation values of N, S, E, W, FN, FS, FE, and FW (See [“Specifying Orientation”](#) on page 664).

If you do not specify *orient*, N (North) is the default non-rotated value used. All other orientation values refer to the flipping or rotation around the via origin (the 0, 0 point in the via shapes). The via origin is still placed at the ( *x y* ) value given in the routing statement just before the *viaName*.

**Note:** Some tools do not support orientation of vias inside their internal data structures; therefore, they are likely to translate vias with an orientation into a different but equivalent via that does not require an orientation.

RECT ( *deltax1 deltax1 deltax2 deltax2* )

Indicates that a rectangle is created from the previous ( *x y* ) routing point using the delta values. The RECT values leave the current point and layer unchanged.

See [Example 4-12](#) on page 687.

*viaName*

Specifies a via to place at the last point. If you specify a via, *layerName* for the next routing coordinates (if any) is implicitly changed to the other routing layer for the via. For example, if the current layer is *metal1*, a *via12* changes the layer to *metal2* for the next routing coordinates.

VIRTUAL ( *x y* )

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Indicates that there is a virtual (non-physical zero-width) connection between the previous point and the new ( *x* *y* ) point. An '\*' indicates that the *x* or *y* value is to be used from the previous point. The layer remains unchanged.

You can use this keyword to retain the symbolic routing graph.

See [Example 4-12](#) on page 687.

( *x* *y* )

Specifies the route coordinates. You cannot specify a route with zero length.

For more information, see [“Specifying Coordinates”](#) on page 689.

*Type:* Integer, specified in database units

STYLE *styleNum*

Specifies a previously defined style from the `STYLES` section in this DEF file. If a style is specified, the wire's shape is defined by the center line coordinates and the style.

TAPER

Specifies that the next contiguous wire segment on *layerName* is created using the default rule.

TAPERRULE *ruleName*

Specifies that the next contiguous wire segment on *layerName* is created using the specified nondefault rule.

### Example 4-11 Multi-mask Patterns for Routing Points

The following example shows a routing statement that specifies three-mask layers M1 and VIA1, and a two-mask layer M2:

```
+ ROUTED M1 (10 0 ) MASK 3 (10 20 ) VIA1_1  
  NEW M2 ( 10 10 ) (20 10) MASK 1 ( 20 20 ) MASK 031 VIA1_2 ;
```

This indicates that the:

- M1 wire shape (10 0) to (10 20) belongs to mask 3
- VIA1\_1 via has no preceding MASK statement so all the metal and cut shapes have no mask and are uncolored

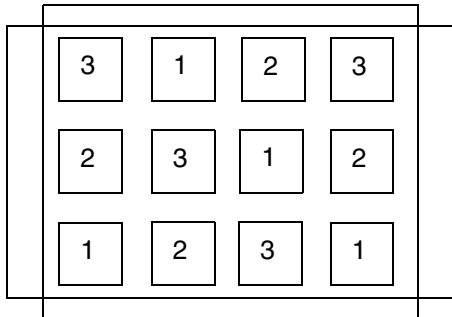
## LEF/DEF 5.8 Language Reference

### DEF Syntax

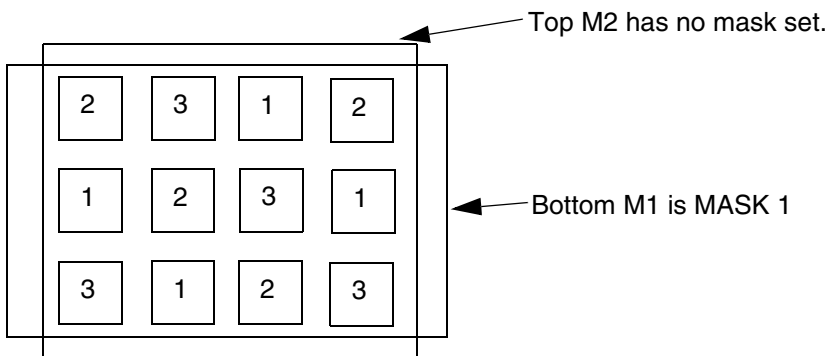
---

- first `NEW M2` wire shape (10 10) to (20 10) has no mask set and is uncolored
- second `M2` wire shape (20 10) to (20 20) is on mask 1
- `VIA1_2` via has a `MASK 031` (it can be `MASK 31` also) so:
  - *topMaskNum* is 0 in the 031 value, so no mask is set for the top metal (`M2`) shape
  - *bottomMaskNum* is 1 in the 031 value, so mask 1 is used for the bottom metal (`M1`) shape
  - *cutMaskNum* is 3 in the 031 value, so the bottom-most, and then the left-most cut of the via-instance is mask 3. The mask for the other cuts of the via-instance are derived from the via-master by "shifting" the via-master's cut masks to match. So if the via-master's bottom-left cut is mask 1, then the via-master cuts on mask 1 become mask 3 for the via-instance, and similarly cuts on 2 shift to 1, and cuts on 3 shift to 2, as shown in [Figure 4-4](#) on page 687.

**Figure 4-4 Via-master multi-mask patterns**



Via-master cut masks for VIA1\_2.



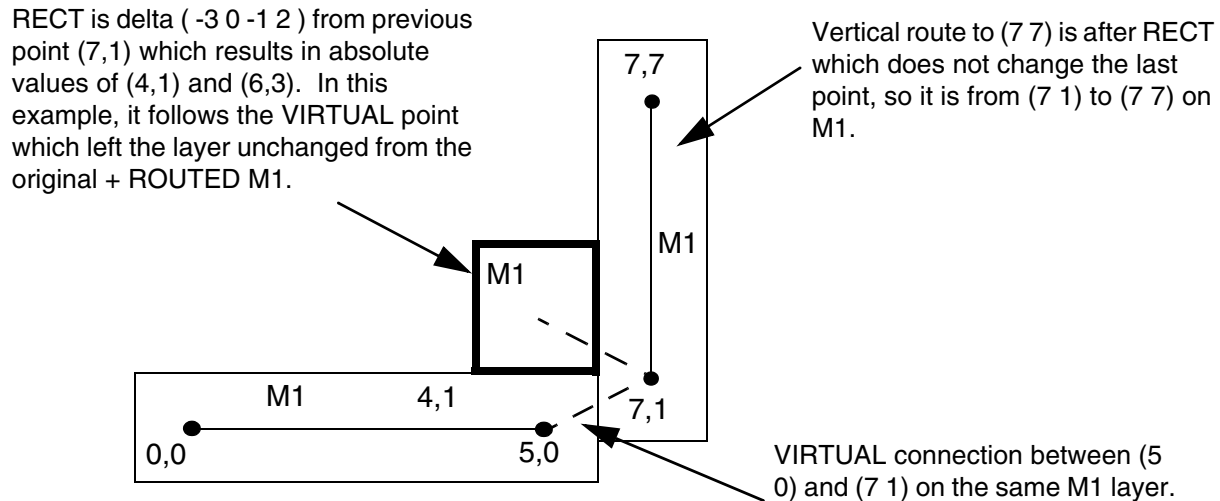
Masks for via-instance: ... ( 20 20 ) MASK 031 VIA1\_2;  
Bottom M1 is MASK 1. Top M2 has no mask set. Lower-left cut is MASK 3, and other via-instance cut shape masks are derived from via-master cut-masks as shown above by "shifting" the via-master masks to match: 1->3, 2->1, 3->2.

### **Example 4-12 Routing Points - Usage of Virtual and Rect**

Figure 4-5 on page 688 shows the results of the following routing statement:

```
+ ROUTED M1 ( 0 0 ) ( 5 0 ) VIRTUAL ( 7 1 ) RECT ( -3 0 -1 2 ) ( 7 7 ) ;
```

**Figure 4-5 Routing Points - Usage of Virtual and Rect**



### Defining Routing Points

Routing points define the center line coordinates of the route for a specified layer. Routes that are 90 degrees, have a width defined by the routing rule for this wire, and extend from one coordinate ( $x\ y$ ) to the next coordinate.

If either endpoint has an extension value (*extValue*), the wire is extended by that amount past the endpoint. Some applications require the extension value to be 0, half of the wire width, or the same as the routing rule wire extension value. If you do not specify an extension value, the default value of half of the wire width is used.

If a coordinate with an extension value is specified after a via, the wire extension is added to the beginning of the next wire segment after the via (zero-length wires are not allowed).

If the wire segment is a 45-degree edge, and no *STYLE* is specified, the default octagon style is used for the endpoints. The routing rule width must be an even multiple of the manufacturing grid in order to keep all of the coordinates of the resulting outer wire boundary on the manufacturing grid.

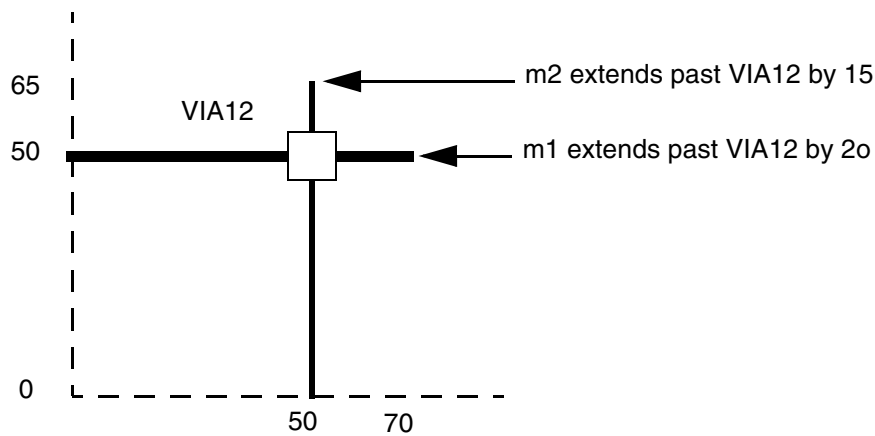
If a *STYLE* is defined for 90-degree or 45-degree routes, the routing shape is defined by the center line coordinates and the style. No corrections, such as snapping to manufacturing grid, can be applied, and any extension values are ignored. The DEF file should contain values that are already snapped, if appropriate. The routing rule width indicates the desired user width, and represents the minimum allowed width of the wire that results from the style when the 45-degree edges are properly snapped to the manufacturing grid.



### Specifying Coordinates

To maximize compactness of the design files, the coordinates allow for the asterisk ( `*` ) convention. Here, ( `x *` ) indicates that the y coordinate last specified in the wiring specification is used; ( `* y` ) indicates that the x coordinate last specified is used. Use ( `* * extValue` ) to specify a wire extension at a via.

```
ROUTED M1 ( 0 50 ) ( 50 * 20 ) VIA12 ( * * 15 ) ( * 0 )
```



Each coordinate sequence defines a connected orthogonal path through the points. The first coordinate in a sequence must not have an `*` element.

Because nonorthogonal segments are not allowed, subsequent points in a connected sequence must create orthogonal paths. For example, the following sequence is a valid path:

```
( 100 200 ) ( 200 200 ) ( 200 500 )
```

The following sequence is an equivalent path:

```
( 100 200 ) ( 200 * ) ( * 500 )
```

The following sequence is not valid because it represents a nonorthogonal segment.

```
( 100 200 ) ( 300 500 )
```

### Impact of Wrong-way Width Rules

Some technologies require larger widths for wrong-way routing than in the preferred direction. If the wrong-way width is larger than the default or NDR width, then the wrong-way width is used for wrong-way routes on that layer. The implicit routing extension is still half of the default or NDR width, even for wrong-way routes.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Some older tools may not understand this behavior. Normally, they will still read/write and round-trip the DEF routing properly, but they may not understand that the width is slightly larger for wrong-way routes. If these tools check wrong-way width, then the DRC rules may flag false violations. RC extraction that does not understand the wrong-way width will also be incorrect, although wrong-way routes are generally short and the width difference is small, so the RC error is normally negligible.

The following LEF DRC rules allow a `WRONGDIRECTION` keyword that defines wrong-way widths that will affect the width of any wrong-way routes in the `DEF NETS` section:

<code>LEF58_WIDTH</code>	Defines the default routing width to use for all regular wiring on the layer.
<code>LEF58_WIDHTHABLE</code>	Defines all the allowable legal widths on the routing layer.
<code>LEF58_SPANLENGHTHABLE</code>	Defines all the allowable legal span lengths on the routing layer.

These width rules are mutually exclusive, so only one of the 3 rules is allowed on one routing layer.

The full syntax for `WIDHTHABLE` and `SPANLENGHTHABLE` have an optional `ORTHOGONAL` keyword or `ORTHOGONAL` keyword with a value. The `ORTHOGONAL` keyword and any value after it can be ignored, and has no effect on `DEF NETS` routing interpretation.

The full syntax for these rules is:

```
WIDTH defaultWidth ;
```

Along with:

```
PROPERTY LEF58_WIDTH  
    "WIDTH minWidth [WRONGDIRECTION] ;" ;
```

Or

```
PROPERTY LEF58_SPANLENGHTHABLE  
"SPANLENGHTHABLE {spanLength} ... [WRONGDIRECTION]  
    [ORTHOGONAL length]  
; " ;
```

Or

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
PROPERTY LEF58_WIDHTHABLE
    "WIDHTHABLE {width}...[WRONGDIRECTION] [ORTHOGONAL]
    ]; " ;
```

For more information on LEF width rules, see [Layer\(Routing\)](#) section in the "LEF Syntax" chapter in the LEF/DEF Language Reference.

### DEF NETS Examples

#### Example 4-13 Impact of default and nondefault rules on wrong-way segment

- This following example shows how LEF width rules will affect the width of any wrong-way routes in the DEF NETS section. An example of each type of rule (WIDTH, WIDHTHABLE, and SPANLENGHTHABLE) is shown below for METAL2:

```
LAYER METAL2
...
DIRECTION VERTICAL ;
#0.6 is the default routing rule width in the vertical direction
WIDTH 0.06 ;
#wrong direction (horizontal) metal width must be >= 0.12
PROPERTY LEF58_WIDTH
    "WIDTH 0.12 WRONGDIRECTION ; " ;
...
END METAL2
```

or

```
LAYER METAL2
...
DIRECTION VERTICAL ;
#0.06 is the default routing rule width in the vertical direction
WIDTH 0.06 ;
#wrong direction (horizontal) metal width must be 0.12, 0.16 or >= 0.20
PROPERTY LEF58_WIDHTHABLE
    "WIDHTHABLE 0.06 0.08 0.12 0.16 0.20 ;
    WIDHTHABLE 0.12 0.16 0.20 WRONGDIRECTION ; " ;
...
END METAL2
```

or

```
LAYER METAL2
...
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```

DIRECTION VERTICAL ;
#0.06 is the default routing rule width in the vertical direction
WIDTH 0.06 ;
#wrong direction (horizontal) metal width must be 0.12, 0.16 or >= 0.20
PROPERTY LEF58_SPANLENGHTHABLE
    "SPANLENGHTHABLE 0.06 0.08 0.12 0.16 0.20 ;
    SPANLENGHTHABLE 0.12 0.16 0.20 WRONGDIRECTION ; " ;
...
END METAL2

```

For the above rules, any `METAL2` vertical routes are in the preferred direction so they will have the normal widths and extensions as given by the default rule width, or the `NONDEFAULTRULE` width definition. The horizontal routes are in the wrong-direction, so they will use the first `WRONGDIRECTION` value in the rules above, that is greater than or equal to the preferred-direction width.

If the rule width is larger than the largest wrong-direction value, then the wrong-direction width is the same as the rule width as shown for `NDR7` below.

The table below shows examples of different routing rule widths and the corresponding vertical and horizontal route widths and extensions for the `WIDTHTABLE` and `SPANLENGHTHABLE` rules shown above. They both have 0.12, 0.16, and 0.20 as the legal `WRONGDIRECTION` width values.

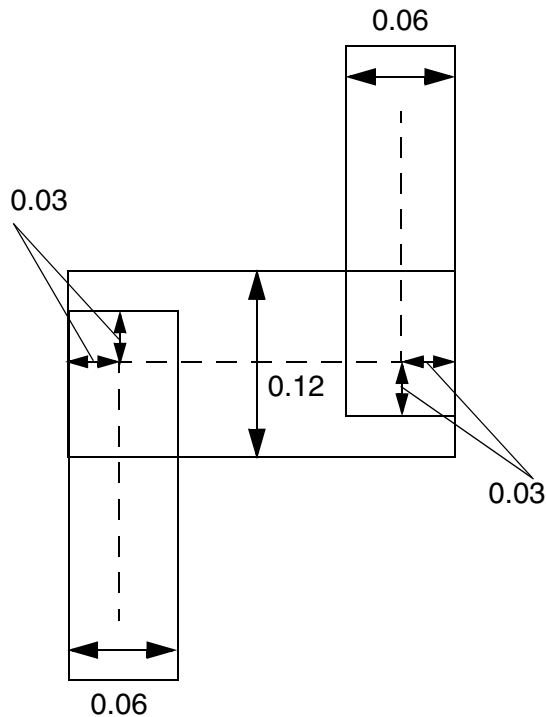
Rule	Rule width	Vertical Route Width	Vertical Route Extension	Horizontal Route Width	Horizontal Route Extension
default	0.06	0.06	0.03	0.12	0.03
NDR1	0.08	0.08	0.04	0.12	0.04
NDR2	0.12	0.12	0.06	0.12	0.06
NDR3	0.14	0.14	0.07	0.16	0.07
NDR4	0.16	0.16	0.08	0.16	0.08
NDR5	0.18	0.18	0.09	0.20	0.09
NDR6	0.20	0.20	0.10	0.20	0.10
NDR7	0.30	0.30	0.15	0.30	0.15

For example, the default rule in the table shows that a vertical route in the `NETS` section will have a width of 0.06  $\mu\text{m}$  with extension of 0.03  $\mu\text{m}$ , while a horizontal route will have

a wrong-way width of  $0.12\text{ }\mu\text{m}$  with extension  $0.03\text{ }\mu\text{m}$ , as shown in the DEF NETS routing example below:

```
- NET1 (...)  
  + ROUTED METAL2 ( 1 0 ) ( 1 2 ) ( 3 2 ) ( 3 4 ) ;
```

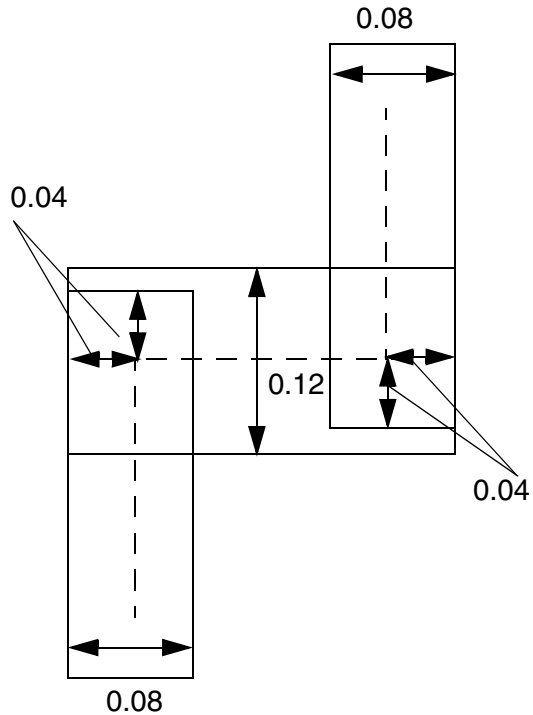
**Figure 4-6 Default rule on wrong way segment**



- NDR1 has a width of  $0.08\text{ }\mu\text{m}$ , so it will use the wrong-way width of  $0.12$ , because  $0.8$  is less than or equal to  $0.12$  (the first wrong-direction width value). So a vertical route will have a width of  $0.08\text{ }\mu\text{m}$  with extension of  $0.04\text{ }\mu\text{m}$  on a vertical route, while a horizontal route will have a width of  $0.12\text{ }\mu\text{m}$  with extension  $0.04\text{ }\mu\text{m}$ :

```
- NET2 (...)  
  + NONDEFAULTRULE NDR1  
  + ROUTED METAL2 ( 1 0 ) ( 1 3 ) ( 4 3 ) ( 4 5 ) ;
```

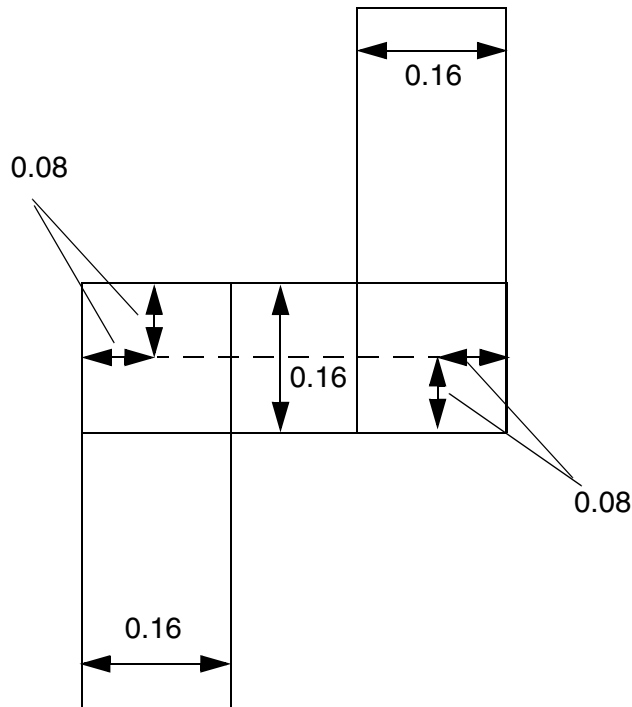
Figure 4-7 Non-Default rule on wrong way segment



- NDR4 in the table has a width of 0.16  $\mu\text{m}$ . Width of 0.16 is larger than the first wrongdirection value of 0.12, but is less than or equal to the second wrong-direction value of 0.16, so it will use the wrong-way width of 0.16. In this case both the vertical and horizontal routes will have a width of 0.16  $\mu\text{m}$  with extension of 0.08  $\mu\text{m}$ .

```
- NET3 (...)  
  + NONDEFAULTRULE NDR4  
  + ROUTED METAL2 (1 0) (1 3) (4 3) (4 4) ;
```

**Figure 4-8 Non-Default rule on wrong way segment**



### ***Specifying Orientation***


If you specify the pin's placement status, you must specify its location and orientation. A pin can have any of the following orientations: N, S, W, E, FN, FS, FW, or FE.

Orientation terminology can differ between tools. The following table maps the orientation terminology used in LEF and DEF files to the OpenAccess database format.

<b>LEF/DEF</b>	<b>OpenAccess</b>	<b>Definition</b>
N (North)	R0	
S (South)	R180	
W (West)	R90	
E (East)	R270	
FN (Flipped North)	MY	
FS (Flipped South)	MX	
FW (Flipped West)	MX90	

## LEF/DEF 5.8 Language Reference

### DEF Syntax

LEF/DEF	OpenAccess	Definition
FE (Flipped East)	MY90	

#### Example 4-14 Shielded Net

The following example defines a shielded net:

```
NETS 1 ;
- my_net ( I1 CLK ) ( BUF OUT )
+ SHIELDNET VSS
+ SHIELDNET VDD
ROUTED
MET2 ( 14000 341440 ) ( 9600 * ) ( * 282400 )
M1M2 ( 2400 * )
+ NOSHIELD MET2 ( 14100 341440 ) ( 14000 * )
+ TAPER MET1 ( 2400 282400 ) ( 240 * )
END NETS
```

#### Nondefault Rules

```
NONDEFAULTRULES numRules ;
{- ruleName
    [+ HARDSPACING]
    [+ LAYER layerName
        WIDTH minWidth
        [DIAGWIDTH diagWidth]
        [SPACING minSpacing]
        [WIREEXT wireExt]
    } ...
    [+ VIA viaName] ...
    [+ VIARULE viaRuleName] ...
    [+ MINCUTS cutLayerName numCuts] ...
    [+ PROPERTY {propName propVal} ...] ...
    [PROPERTY LEF58 USEVIACUTCLASS
    "USEVIACUTCLASS cutLayerName className
        [ROWCOL numCutRows numCutCols]
        ;... " ;]
    ;} ...
END NONDEFAULTRULES
```

Defines any nondefault rules used in this design that are not specified in the LEF file. This section can also contain the default rule and LEF nondefault rule definitions for reference. These nondefault rule names can be used anywhere in the DEF `NETS` section that requires a nondefault rule name.



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

If a nondefault rule name collides with an existing LEF or DEF nondefault rule name that has different parameters, the application should use the DEF definition when reading this DEF file, though it can change the DEF nondefault rule name to make it unique. This is typically done by adding a unique extension, such as `_1` or `_2` to the rule name.

All vias must be previously defined in the LEF `VIA` or DEF `VIAS` sections. Every nondefault rule must specify a width for every layer. If a nondefault rule does not specify a via or via rule for a particular routing-cut-routing layer combination, then there must be a `VIARULE GENERATE DEFAULT` rule that it inherited for that combination.

`DIAGWIDTH` *diagWidth*

Specifies the diagonal width for *layerName*, when 45-degree routing is used.

*Default:* 0 (no diagonal routing allowed)

*Type:* Integer, specified in DEF database units

`HARDSPACING`

Specifies that any spacing values that exceed the LEF `LAYER ROUTING` spacing requirements are “hard” rules instead of “soft” rules. By default, routers treat extra spacing requirements as soft rules that are high cost to violate, but not real spacing violations. However, in certain situations, the extra spacing should be treated as a hard, or real, spacing violation, such as when the route will be modified with a post-process that replaces some of the extra space with metal.

`LAYER` *layerName*

Specifies the layer for the various width and spacing values. *layerName* must be a routing layer. Each routing layer must have at least a minimum width specified.

`MINCUTS` *cutLayerName numCuts*

Specifies the minimum number of cuts allowed for any via using the specified cut layer.

All vias (generated or fixed vias) used for this nondefault rule must have at least *numCuts* cuts in the via.

*Type:* (*numCuts*) Positive integer

*numRules*

Specifies the number of nondefault rules defined in the `NONDEFAULTRULES` section.

`PROPERTY` *propName propValue*

Specifies a property for this nondefault rule. The *propName* must be defined as a `NONDEFAULTRULE` property in the `PROPERTYDEFINITIONS` section, and the *propValue* must match the type for *propName* (that is, integer, real, or string).

### Use Via Cut Class Rule

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

You can use the use via cut class only rule to specify the cut class to be used with the routing rule.

You can create a use via cut class rule by using the following property definition:

```
PROPERTY LEF58_USEVIACUTCLASS
    USEVIACUTCLASS cutLayerName className
        [ROWCOL numCutRows numCutCols]
        ;... " ;
```

Where:

```
USEVIACUTCLASS cutLayerName className
    [ROWCOL numCutRows numCutCols]
```

Specifies *className* cuts on *cutLayerName*, which must be a cut layer, to be used with this routing rule. ROWCOL specifies the number of cut rows and columns of a multiple-cut via of the cut class *className* to be used with this routing rule. By default, a single cut via or the given cut number defined in MINCUTS is used.

*rulename*

Specifies the name for this nondefault rule. This name can be used in the NETS section wherever a nondefault rule name is allowed. The reserved name DEFAULT can be used to indicate the default routing rule used in the NETS section.

SPACING *minSpacing*

Specifies the minimum spacing for *layerName*. The LEF LAYER SPACING or SPACINGTABLE definitions always apply; therefore it is only necessary to add a SPACING value if the desired spacing is larger than the LAYER rules already require.  
*Type:* Integer, specified in DEF database units.

VIA *viaName*

Specifies a previously defined LEF or DEF via to use with this rule.

VIARULE *viaRuleName*

Specifies a previously defined LEF VIARULE GENERATE to use with this routing rule. If no via or via rule is specified for a given routing-cut-routing layer combination, then a VIARULE GENERATE DEFAULT via rule must exist for that combination, and it is implicitly inherited.

WIDTH *minWidth*

Specifies the required minimum width allowed for *layerName*.  
*Type:* Integer, specified in DEF database units

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

WIREEXT *wireExt*

Specifies the distance by which wires are extended at vias. Enter 0 (zero) to specify no extension. Values other than 0 must be greater than or equal to half of the routing width for the layer, as defined in the nondefault rule.

*Default:* Wires are extended half of the routing width

*Type:* Float, specified in microns

#### Example 4-15 Nondefault Rules

The following NONDEFAULTRULES statement is based on the assumption that there are VIARULE GENERATE DEFAULT rules for each routing-cut-routing combination, and that the default width is 0.3  $\mu\text{m}$ .

```
NONDEFAULTRULES 5 ;
- doubleSpaceRule #Needs extra space, inherits default via rules
  + LAYER metal1 WIDTH 200 SPACING 1000
  + LAYER metal2 WIDTH 200 SPACING 1000
  + LAYER metal3 WIDTH 200 SPACING 1000 ;
- lowerResistance #Wider wires and double cut vias for lower resistance
  #and higher current capacity. No special spacing rules,
  #therefore the normal LEF LAYER specified spacing rules
  #apply. Inherits the default via rules.
  + LAYER metal1 WIDTH 600 #Metal1 is thinner, therefore a little wider
  + LAYER metal2 WIDTH 500
  + LAYER metal3 WIDTH 500
  + MINCUTS cut12 2 #Requires at least two cuts
  + MINCUTS cut23 2 ;
- myRule #Use default width and spacing, change via rules. The
  #default via rules are not inherited.
  + LAYER metal1 WIDTH 200
  + LAYER metal2 WIDTH 200
  + LAYER metal3 WIDTH 200
  + VIARULE myvia12rule
  + VIARULE myvia23rule ;
- myCustomRule #Use new widths, spacing and fixed vias. The default
  #via rules are not inherited because vias are defined.
  + LAYER metal1 WIDTH 500 SPACING 1000
  + LAYER metal2 WIDTH 500 SPACING 1000
  + LAYER metal3 WIDTH 500 SPACING 1000
  + VIA myvia12_custom1
  + VIA myvia12_custom2
  + VIA myvia23_custom1
  + VIA myvia23_custom2 ;
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
END NONDEFAULTRULES
```

## Pins

```
[PINS numPins ;
  [ - pinName + NET netName
    [+ SPECIAL]
    [+ DIRECTION {INPUT | OUTPUT | INOUT | FEEDTHRU}]
    [+ NETEXPR "netExprPropName defaultNetName"]
    [+ SUPPLYSENSITIVITY powerPinName]
    [+ GROUNDSENSITIVITY groundPinName]
    [+ USE {SIGNAL | POWER | GROUND | CLOCK | TIEOFF | ANALOG
            | SCAN | RESET}]
    [+ ANTENNAPINPARTIALMETALAREA value [LAYER layerName]] ...
    [+ ANTENNAPINPARTIALMETALSIDEAREA value [LAYER layerName]] ...
    [+ ANTENNAPINPARTIALCUTAREA value [LAYER layerName]] ...
    [+ ANTENNAPINDIFFFAREA value [LAYER layerName]] ...
    [+ ANTENNAMODEL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}] ...
    [+ ANTENNAPINGATEAREA value [LAYER layerName]] ...
    [+ ANTENNAPINMAXAREACAR value LAYER layerName] ...
    [+ ANTENNAPINMAXSIDEAREACAR value LAYER layerName] ...
    [+ ANTENNAPINMAXCUTCAR value LAYER layerName] ...
    [[+ PORT]
      [+ LAYER layerName
        [MASK maskNum]
        [SPACING minSpacing | DESIGNRULEWIDTH effectiveWidth]
        pt pt
      ]
      [+ POLYGON layerName
        [MASK maskNum]
        [SPACING minSpacing | DESIGNRULEWIDTH effectiveWidth]
        pt pt pt ...
      ]
      [+ VIA viaName
        [MASK viaMaskNum]
        pt
      ]
      ] ...
    [+ COVER pt orient | FIXED pt orient | PLACED pt orient]
  ]...
; ] ...
END PINS]
```

Defines external pins. Each pin definition assigns a pin name for the external pin and associates the pin name with a corresponding internal net name. The pin name and the net name can be the same.

When the design is a chip rather than a block, the PINS statement describes logical pins, without placement or physical information.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

`ANTENNAMODEL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}`

Specifies the oxide model for the pin. If you specify an `ANTENNAMODEL` statement, that value affects all `ANTENNAGATEAREA` and `ANTENNA*CAR` statements for the pin that follow it until you specify another `ANTENNAMODEL` statement. The `ANTENNAMODEL` statement does not affect `ANTENNAPARTIAL*AREA` and `ANTENNADIFFAREA` statements because they refer to the total metal, cut, or diffusion area connected to the pin, and do not vary with each oxide model.

*Default:* `OXIDE1`, for a new `PIN` statement

Because DEF is often used incrementally, if an `ANTENNA` statement occurs twice for the same oxide model, the last value specified is used.

Usually, you only need to specify a few `ANTENNA` values; however, for a block with six routing layers, it is possible to have six different `ANTENNAPARTIAL*AREA` values and six different `ANTENNAPINDIFFAREA` values per pin. It is also possible to have six different `ANTENNAPINGATEAREA` and `ANTENNAPINMAX*CAR` values for each oxide model on each pin.

See [Example 4-16](#) on page 710.

`ANTENNAPINDIFFAREA value [LAYER layerName]`

Specifies the diffusion (diode) area to which the pin is connected on a layer. If you do not specify *layerName*, the value applies to all layers. This is not necessary for output pins.

*Type:* Integer

*Value:* Area specified in (DEF database units)<sup>2</sup>

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

`ANTENNAPINGATEAREA value [LAYER layerName]`

Specifies the gate area to which the pin is connected on a layer. If you do not specify *layerName*, the value applies to all layers. This is not necessary for input pins.

*Type:* Integer

*Value:* Area specified in (DEF database units)<sup>2</sup>

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

`ANTENNAPINMAXAREACAR value LAYER layerName`

For hierarchical process antenna effect calculation, specifies the maximum cumulative antenna ratio value, using the metal area at or below the current pin layer, excluding the pin area itself. Use this to calculate the actual cumulative antenna ratio on the pin layer, or the layer above it.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAPINMAXCUTCAR *value* LAYER *layerName*

For hierarchical process antenna effect calculation, specifies the maximum cumulative antenna ratio value, using the cut area at or below the current pin layer, excluding the pin area itself. Use this to calculate the actual cumulative antenna ratio for the cuts above the pin layer.

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAPINMAXSIDEAREACAR *value* LAYER *layerName*

For hierarchical process antenna effect calculation, specifies the maximum cumulative antenna ratio value, using the metal side wall area at or below the current pin layer, excluding the pin area itself. Use this to calculate the actual cumulative antenna ratio on the pin layer, or the layer above it.

*Type:* Integer

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAPINPARTIALCUTAREA *value* [LAYER *cutLayerName*]

Specifies the partial cut area above the current pin layer and inside the macro cell on a layer. If you do not specify *layerName*, the value applies to all layers. For hierarchical designs, only the cut layer above the I/O pin layer is needed for partial antenna ratio calculation.

*Type:* Integer

*Value:* Area specified in (DEF database units)<sup>2</sup>

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAPINPARTIALMETALAREA *value* [LAYER *layerName*]

Specifies the partial metal area connected directly to the I/O pin and the inside of the macro cell on a layer. If you do not specify *layerName*, the value applies to all layers. For hierarchical designs, only the same metal layer as the I/O pin, or the layer above it, is needed for partial antenna ratio calculation.

*Type:* Integer

*Value:* Area specified in (DEF database units)<sup>2</sup>

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

ANTENNAPINPARTIALMETALSIDEAREA *value* [LAYER *layerName*]

Specifies the partial metal side wall area connected directly to the I/O pin and the inside of the macro cell on a layer. If you do not specify *layerName*, the value applies to all layers. For hierarchical designs, only the same metal layer as the I/O pin, or the layer above it, is needed for partial antenna ratio calculation.

*Type:* Integer

*Value:* Area specified in (DEF database units)<sup>2</sup>

For more information on process antenna calculation, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

DIRECTION {INPUT | OUTPUT | INOUT | FEEDTHRU}

Specifies the pin type. Most current tools do not usually use this keyword. Typically, pin directions are defined by timing library data, and not from DEF.

*Value:* Specify one of the following:

INPUT	Pin that accepts signals coming into the cell.
OUTPUT	Pin that drives signals out of the cell.
INOUT	Pin that can accept signals going either in or out of the cell.
FEEDTHRU	Pin that goes completely across the cell.

GROUNDSENSITIVITY *groundPinName*

Specifies that if this pin is connected to a tie-low connection (such as 1'b0 in Verilog), it should connect to the same net to which *groundPinName* is connected.

*groundPinName* must match another pin in this PINS section that has a + USE GROUND attribute. The ground pin definition can follow later in this PINS section; it does not have to be defined before this pin definition. It is a semantic error to put this attribute on an existing ground pin.

**Note:** GROUNDSENSITIVITY is useful only when there is more than one ground net connected to pins in the PINS section. By default, if there is only one net connected to all + USE GROUND pins, the tie-low connections are already implicitly defined (that is, tie-low connections are connected to the same net as any ground pin).

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

NETEXPR "*netExprPropName defaultNetName*"

Specifies a net expression property name (such as `power1` or `power2`) and a default net name. If *netExprPropName* matches a net expression property higher up in the netlist (for example, in Verilog, VHDL, or OpenAccess), then the property is evaluated, and the software identifies a net to which to connect this pin. If the property does not exist, *defaultNetName* is used for the net name.

*netExprPropName* must be a simple identifier in order to be compatible with other languages, such as Verilog and CDL. Therefore, it can only contain alphanumeric characters, and the first character cannot be a number. For example, `power2` is a legal name, but `2power` is not. You cannot use characters such as `$` and `!`. The *defaultName* can be any legal DEF net name.

If more than one pin connects to the same net, only one pin should have a NETEXPR added to it. It is redundant and unnecessary to add NETEXPR to every ground pin connected to one ground net, and it is illegal to have different NETEXPR values for pins connected to the same net.

See [Example 4-17](#) on page 710.

*numPins*

Specifies the number of pins defined in the PINS section.

*pinName* + NET *netName*

Specifies the name for the external pin, and the corresponding internal net (defined in NETS or SPECIALNETS statements).

PORT

Indicates that the following LAYER, POLYGON, and VIA statements are all part of one PORT connection, until another PORT statement occurs. In a PIN definition without any PORT statement, all of the LAYER, POLYGON, and VIA statements are assumed to be part of a single implicit PORT for the PIN.

This commonly occurs for power and ground pins. All of the shapes of one port (rectangles, polygons, and vias) should already be connected with just the port shapes; therefore, the router only needs to connect to one of the shapes for the port. Separate ports should each be connected by routing inside the block (and each DEF PORT should map to a single LEF PORT in the equivalent LEF abstract for this block).

The syntax for describing PORT statements is defined as follows:

```
[ [+ PORT]
  [+ LAYER layerName
```



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
[MASK maskNum]  
[SPACING minSpacing | DESIGNRULEWIDTH effectiveWidth]  
    pt pt  
|+ POLYGON layerName  
    [MASK maskNum]  
    [SPACING minSpacing | DESIGNRULEWIDTH effectiveWidth]  
    pt pt pt ...  
|+ VIA viaName  
    [MASK viaMaskNum]  
    pt  
] ...  
[+ COVER pt orient | FIXED pt orient | PLACED pt orient]  
] ...
```

COVER *pt orient*

Specifies the pin's location, orientation, and that it is a part of the cover macro. A COVER pin cannot be moved by automatic tools or by interactive commands. If you specify a placement status for a pin, you must also include a LAYER statement.

DESIGNRULEWIDTH *effectiveWidth*

Specifies that this pin has a width of *effectiveWidth* for the purpose of spacing calculations. If you specify DESIGNRULEWIDTH, you cannot specify SPACING. As a lot of spacing rules in advanced nodes no longer just rely on wire width, DESIGNRULEWIDTH is not allowed for 20nm and below nodes. (See [Example 4-18](#) on page 711.)

Type: Integer, specified in DEF database units

FIXED *pt orient*

Specifies the pin's location, orientation, and that its location cannot be moved by automatic tools, but can be moved by interactive commands. If you specify a placement status for a pin, you must also include a LAYER statement

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

LAYER *layerName* *pt* *pt*

Specifies the routing layer used for the pin, and the pin geometry on that layer. If you specify a placement status for a pin, you must include a LAYER statement.

MASK *maskNum*

Specifies which mask from double or triple patterning to use for the specified shape. The *maskNum* variable must be a positive integer - most applications support values of 1, 2, or 3 only. Shapes without any defined mask do not have a mask set (are uncolored).

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

`MASK viaMaskNum`

Specifies which mask for double or triple patterning lithography is to be applied to via shapes on each layer.

The *viaMaskNum* variable is a hex-encoded three digit value of the form:

`<topMaskNum><cutMaskNum><bottomMaskNum>`

For example, MASK 113 means the top metal and cut layer *maskNum* is 1, and the bottom metal layer *maskNum* is 3. A value of 0 indicates that the shape on that layer does not have a mask assignment (is uncolored), so 013 means the top layer is uncolored. If either the first or second digit is missing, they are assumed to be 0, so 013 and 13 mean the same thing. Most applications support *maskNums* of 0, 1, 2, or 3 for double or triple patterning.

The *topMaskNum* and *bottomMaskNum* variables specify the mask to which the corresponding metal shape belongs. The via-master metal mask values have no effect.

For the cut-layer, the *cutMaskNum* variable defines the mask for the bottom-most, then the left-most cut in the North (R0) orientation. For multi-cut vias, the via-instance cut masks are derived from the via-master cut mask values. The via-master must have a mask defined for all of the cut shapes and every via-master cut mask is "shifted" (from 1 to 2 and 2 to 1 for two mask layers, and from 1 to 2, 2 to 3, and 3 to 1 for three mask layers), so the lower-left cut matches the *cutMaskNum* value. See [Example 4-16](#) on page 710.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Similarly, for the metal layer, the *topMaskNum/bottomMaskNum* would define the mask for the bottom-most, then leftmost metal shape. For multiple disjoint metal shapes, the via-instance metal masks are derived from the via-master metal mask values. The via-master must have a mask defined for all of the metal shapes, and every via-master metal mask is “shifted” (1->2, 2->1 for two mask layers, 1->2, 2->3, 3->1 for three mask layers) so the lower-left cut matches the *topMaskNum/bottomMaskNum* value.

Shapes without any defined mask, that need to be assigned, can be assigned to an arbitrary choice of mask by applications.

`PLACED pt orient`

Specifies the pin’s location, orientation, and that it’s location is fixed, but can be moved during automatic layout. If you specify a placement status for a pin, you must also include a `LAYER` statement.

`POLYGON layerName pt pt pt`

Specifies the layer and a sequence of at least three points to generate a polygon for this pin. The polygon edges must be parallel to the x axis, the y axis, or at a 45-degree angle.

Each `POLYGON` statement defines a polygon generated by connecting each successive point, and then the first and last points. The *pt* syntax corresponds to a coordinate pair, such as *x y*. Specify an asterisk (\*) to repeat the same value as the previous *x* or *y* value from the last point. (See [Example 4-21](#) on page 714.)

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

SPACING *minSpacing*

Specifies the minimum spacing allowed between this pin and any other routing shape. This distance must be greater than or equal to *minSpacing*. If you specify SPACING, you cannot specify DESIGNRULEWIDTH. (See [Example 4-18](#) on page 711.)

*Type:* Integer, specified in DEF database units

VIA *viaName pt*

Places the via named *viaName* at the specified (x y) location (*pt*). *viaName* must be a previously defined via in the DEF VIAS or LEF VIA section.

*Type:* (*pt*) Integers, specified in DEF database units

SPECIAL

Identifies the pin as a special pin. Regular routers do not route to special pins. The special router routes special wiring to special pins.

SUPPLYSENSITIVITY *powerPinName*

Specifies that if this pin is connected to a tie-high connection (such as 1'b1 in Verilog), it should connect to the same net to which *powerPinName* is connected.

*powerPinName* must match another pin in this PINS section that has a + USE POWER attribute. The power pin definition can follow later in this PINS section; it does not have to be defined before this pin definition. It is a semantic error to put this attribute on an existing power pin. For an example, see [Example 4-17](#) on page 710.

**Note:** POWERSENSITIVITY is useful only when there is more than one power net connected to pins in the PINS section. By default, if there is only one net connected to all + USE POWER pins, the tie-high connections are already implicitly defined (that is, tie-high connections are connected to the same net as the single power pin).

USE {ANALOG | CLOCK | GROUND | POWER | RESET | SCAN | SIGNAL | TIEOFF}

Specifies how the pin is used.

*Default:* SIGNAL

*Value:* Specify one of the following:

ANALOG

Pin is used for analog connectivity.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

CLOCK	Pin is used for clock net connectivity.
GROUND	Pin is used for connectivity to the chip-level ground distribution network.
POWER	Pin is used for connectivity to the chip-level power distribution network.
RESET	Pin is used as reset pin.
SCAN	Pin is used as scan pin.
SIGNAL	Pin is used for regular net connectivity.
TIEOFF	Pin is used as tie-high or tie-low pin.

#### Example 4-16 Antenna Model Statement

The following example describes the OXIDE1 and OXIDE2 models for pin `clock1`. Note that the `ANTENNAPINPARTIALMETALAREA` and `ANTENNAPINDIFFAREA` values are not affected by the oxide values.

```
PINS 100 ;
- clock1 + NET clock1
...
+ ANTENNAPINPARTIALMETALAREA 1000 LAYER m1
+ ANTENNAPINDIFFAREA 500 LAYER m1
...
+ ANTENNAMODEL OXIDE1                                #not required, but good practice
+ ANTENNAPINGATEAREA 1000
+ ANTENNAMAXAREACAR 300 LAYER m1
...
+ ANTENNAMODEL OXIDE2                                #start of OXIDE2 values
+ ANTENNAPINGATEAREA 2000
+ ANTENNAMAXAREACAR 100 LAYER m1
...
```

#### Example 4-17 Net Expression and Supply Sensitivity

The following `PINS` statement defines sensitivity and net expression values for five pins in the design `myDesign`:

```
DESIGN myDesign
...
PINS 4 ;
- in1 + NET myNet
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
...
+ SUPPLYSENSITIVITY vddpin1 ; #If in1 is connected to 1'b1, use
                                #net that is connected to vddpin1.
                                #No GROUNDSENSITIVITY is needed because
                                #only one ground net is used by PINS.
                                #Therefore, 1'b0 implicitly means net
                                #from any +USE GROUND pin.
- vddpin1 + NET VDD1 + USE POWER
...
+ NETEXPR "power1 VDD1" ; #If an expression named power1 is defined in
                            #the netlist, use it to find the net.
                            #Otherwise, use net VDD1.
- vddpin2 + NET VDD2 + USE POWER
...
+ NETEXPR "power2 VDD2" ; #If an expression named power2 is defined in
                            #the netlist, use it to find the net.
                            # Otherwise, use net VDD2.
- gndpin1 + NET GND + USE GROUND
...
+ NETEXPR "gnd1 GND" ; #If an expression named gnd1 is defined in
                        #the netlist, use it to find net
                        #connection. Otherwise, use net GND.
```

END PINS

### Example 4-18 Design Rule Width and Spacing Rules

The following statements create spacing rules using the DESIGNRULEWIDTH and SPACING statements:

```
PINS 3 ;
- myPin1 + NET myNet1 + DIRECTION INPUT
  + LAYER metall
    DESIGNRULEWIDTH 1000    #Pin is effectively 1000 dbu wide
    ( -100 0 ) ( 100 200 ) #Pin is 200 x 200 dbu
  + FIXED ( 10000 5000 ) S ;
- myPin2 + NET myNet2 + DIRECTION INPUT
  + LAYER metall
    SPACING 500              #Requires >= 500 dbu spacing
    ( -100 0 ) ( 100 200 ) #Pin is 200 x 200 dbu
  + COVER ( 10000 5000 ) S ;
- myPin3 + NET myNet1        #Pin with two shapes
  + DIRECTION INPUT
  + LAYER metal2 ( 200 200 ) ( 300 300 ) #100 x 100 dbu shape
  + POLYGON metall ( 0 0 ) ( 100 100 ) ( 200 100 ) ( 200 0 ) #Has 45-degree edge
  + FIXED ( 10000 5000 ) N ;
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

END PINS

#### Example 4-19 Multi-Mask Patterns for Pins

The following example shows via master masks:

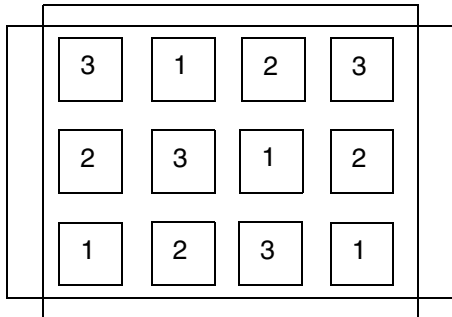
```
clock + NET clock
+ LAYER M1 MASK 2 ( -25 0 ) ( 25 50 )      #m1 rectangle is on mask 2
+ LAYER M2 ( -10 0 ) ( 10 75 )             #m2 rectangle, no mask
+ VIA VIA1 MASK 031 ( 0 25 )               #via1 with mask 031
...
```

The VIA1 via will have:

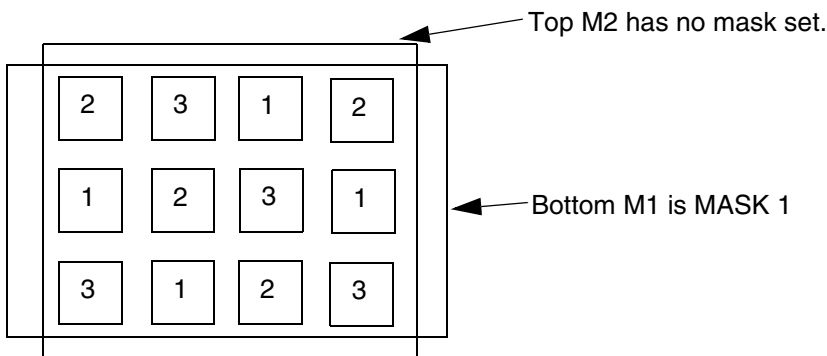
- no mask set for the top metal shape (*topMaskNum* is 0 in the 031 value)
- MASK 1 for the bottom metal shape (*botMaskNum* is 1 in the 031 value)
- the bottom-most, and then the left-most cut of the via-instance is MASK 3. The mask for the other cuts of the via-instance are derived from the via-master by "shifting" the via-master cut masks to match. So if the via-master's bottom-left cut is MASK 1, then the via-master cuts on MASK 1 become MASK 3 for the via-instance. Similarly cuts on 2 shift to 1, and cuts on 3 shift to 2. See [Figure 4-9](#) on page 713.



**Figure 4-9 Multi-Mask Patterns for Pins**



Via-master cut masks for VIA1.



Masks for via-instance: + VIA VIA1 MASK 031 ( 0 25 )  
Bottom M1 is MASK 1. Top M2 has no mask set. Lower-left cut is MASK 3, and other via-instance cut shape masks are derived from via-master cut-masks as shown above by "shifting" the via-master masks to match: 1->3, 2->1, 3->2.

### Example 4-20 Port Example

Assume a block that is 5000 x 5000 database units with a 0,0 origin in the middle of the block. If you have the following pins defined, [Figure 4-10](#) on page 714 illustrates how pin BUSA[0] is created for two different placement locations and orientations:

```
PINS 2 ;
- BUSA[0] + NEY BUSA[0] + DIRECTION IN{UT + USE SIGNAL
  + LAYER M1 ( -25 0 ) ( 25 50 )      #m1, m2, and via12
  + LAYER M2 ( -10 0 ) ( 10 75 )
  + VIA via12 ( 0 25 )
  + PLACED ( 0 -2500 ) N ;           #middle of bottom side
- VDD + NET VDD + DIRECTION INOUT + USE POWER + SPECIAL
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

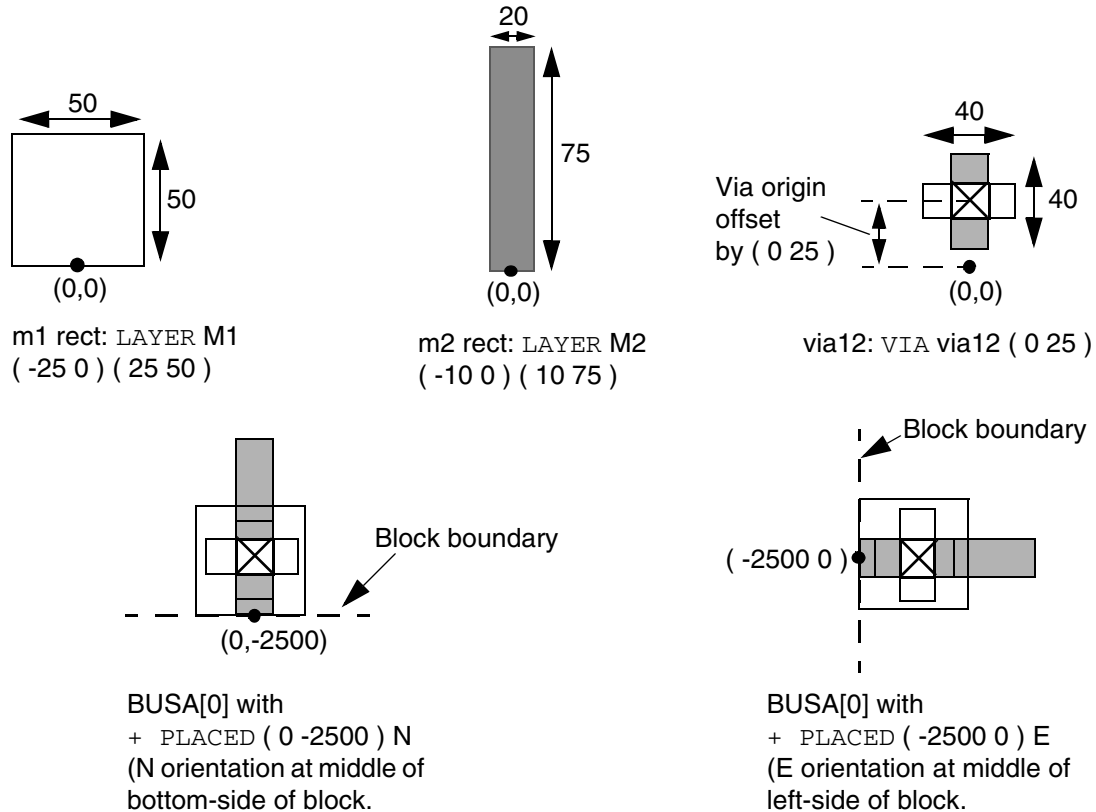
---

```

+ PORT
  + LAYER M2 ( -25 0 ) ( 25 50 )
  + PLACED ( 0 2500 ) S          #middle of top side
+ PORT
  + LAYER M1 ( -25 0 ) ( 25 50 )
  + PLACED ( -2500 0 ) E        #middle of left side
+ PORT
  + LAYER M1 ( -25 0 ) ( 25 50 )
  + PLACED ( 2500 0 ) W ;       #middle of right side
END PINS

```

**Figure 4-10 Port Illustration**



### Example 4-21 Port Statement With Polygon

The following PINS statement creates a polygon with a 45-degree angle:

```

PINS 2 ;
- myPin3 + NET myNet1 + DIRECTION INPUT
+ PORT

```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
+ POLYGON metall ( 0 0 ) ( 100 100 ) ( 200 100 ) ( 200 0 ) #45-degree angle
+ FIXED ( 10000 5000 ) N ;

...
END PINS
```

#### ***Extra Physical PIN(S) for One Logical PIN***

In the design of place and route blocks, you sometimes want to add extra physical connection points to existing signal ports (usually to enable the signal to be accessed from two sides of the block). One pin has the same name as the net it is connected to. Any other pins added to the net must use the following naming conventions.

For extra non-bus bit pin names, use the following syntax:

*pinname.extraN*

*N* is a positive integer, incremented as the physical pins are added

For example:

```
PINS n ;
- a + NET a .... ;
- a.extra1 + NET a ... ;
```

For extra bus bit pin names, use the following syntax:

*basename.extraN[index]*

*basename* is simple part of bus bit pin/net name . *N* is a positive integer, incremented as the physical pins are added. [*index*] identifies the specific bit of the bus, if it is a bus bit.

For example:

```
PINS n ;
- a[0] + net a[0] ... ;
- a.extra1[0] + net a[0] ... ;
```

**Note:** The brackets [ ] are the `BUSBITCHARS` as defined in the `DEF BUSBITCHARS` statement.









#### ***Specifying Orientation***

If you specify the pin's placement status, you must specify its location and orientation. A pin can have any of the following orientations: N, S, W, E, FN, FS, FW, or FE.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

Orientation terminology can differ between tools. The following table maps the orientation terminology used in LEF and DEF files to the OpenAccess database format.

LEF/DEF	OpenAccess	Definition
N (North)	R0	
S (South)	R180	
W (West)	R90	
E (East)	R270	
FN (Flipped North)	MY	
FS (Flipped South)	MX	
FW (Flipped West)	MX90	
FE (Flipped East)	MY90	

#### Example 4-22 Pin Statements

The following example describes a physical I/O pin.

```
# M1 width = 50, track spacing = 120
# M2 width = 60, track spacing = 140

DIEAREA ( -5000 -5000 ) ( 5000 5000 ) ;
TRACKS Y -4900 DO 72 STEP 140 LAYER M2 M1 ;
TRACKS X -4900 DO 84 STEP 120 LAYER M1 M2 ;
PINS 4 ;
    # Pin on the left side of the block
    - BUSA[0] + NET BUSA[0] + DIRECTION INPUT
      + LAYER M1 ( -25 0 ) ( 25 165 ) # .5 M1 W + 1 M2 TRACK
      + PLACED ( -5000 2500 ) E ;
    # Pin on the right side of the block
    - BUSA[1] + NET BUSA[1] + DIRECTION INPUT
      + LAYER M1 ( -25 0 ) ( 25 165 ) # .5 M1 W + 1 M2 TRACK
      + PLACED ( 5000 -2500 ) W ;
    # Pin on the bottom side of the block
    - BUSB[0] + NET BUSB[0] + DIRECTION INPUT
      + LAYER M2 ( -30 0 ) ( 30 150 ) # .5 M2 W + 1 M1 TRACK
      + PLACED ( -2100 -5000 ) N ;
    # Pin on the top side of the block
    - BUSB[1] + NET BUSB[1] + DIRECTION INPUT
```

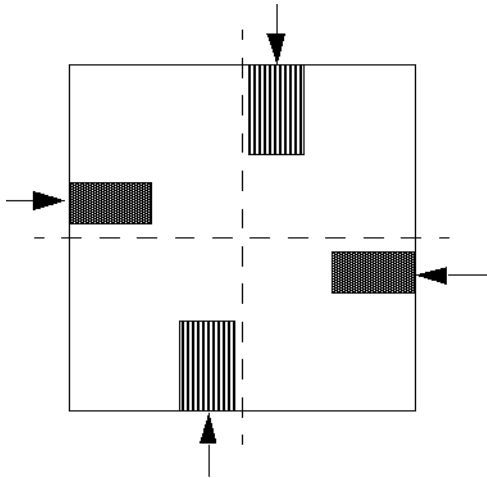
## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
+ LAYER M2 ( -30 0 ) ( 30 150 ) # .5 M2 W + 1 M1 TRACK
+ PLACED ( 2100 5000 ) S ;

END PINS
```



The following example shows how a logical I/O pin would appear in the DEF file. The pin is first defined in Verilog for a chip-level design.

```
module chip (OUT, BUSA, BUSB) ;
    input [0:1] BUSA, BUSB;
    output OUT;
    ....
endmodule
```

The following description for this pin is in the PINS section in the DEF file:

```
PINS 5 ;
- BUSA[0] + NET BUSA[0] + DIRECTION INPUT ;
- BUSA[1] + NET BUSA[1] + DIRECTION INPUT ;
- BUSB[0] + NET BUSB[0] + DIRECTION INPUT ;
- BUSB[1] + NET BUSB[1] + DIRECTION INPUT ;
- OUT + NET OUT + DIRECTION OUTPUT ;

END PINS
```

## Pin Properties

```
[PINPROPERTIES num;
    [- {compName pinName | PIN pinName}
        [+ PROPERTY {propName propVal} ...] ...
    ;] ...

END PINPROPERTIES]
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Defines pin properties in the design.

*compName pinName*

Specifies a component pin. Component pins are identified by the component name and pin name.

*num*

Specifies the number of pins defined in the PINPROPERTIES section.

PIN *pinName*

Specifies an I/O pin.

PROPERTY *propName propVal*

Specifies a numerical or string value for a pin property defined in the PROPERTYDEFINITIONS statement. The *propName* you specify must match the *propName* listed in the PROPERTYDEFINITIONS statement.

#### Example 4-23 Pin Properties Statement

```
PINPROPERTIES 3 ;
- CORE/g76 CKA + PROPERTY CLOCK "FALLING" ;
- compl A + PROPERTY CLOCK "EXCLUDED" ;
- rp/regB clk + PROPERTY CLOCK "INSERTION" ;
END PINPROPERTIES
```

### Property Definitions

```
[PROPERTYDEFINITIONS
  [objectType propName propType [RANGE min max]
   [value | stringValue]
  ;] ...
END PROPERTYDEFINITIONS]
```

Lists all properties used in the design. You must define properties in the PROPERTYDEFINITIONS statement before you can refer to them in other sections of the DEF file.

*objectType*

Specifies the object type being defined. You can define properties for the following object types:

COMPONENT

COMPONENTPIN

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

DESIGN  
GROUP  
NET  
NONDEFAULTRULE  
REGION  
ROW  
SPECIALNET

*propName*

Specifies a unique property name for the object type.

*propType*

Specifies the property type for the object type. You can specify one of the following property types:

INTEGER  
REAL  
STRING

RANGE *min max*

Limits real number and integer property values to a specified range.

*value* | *stringValue*

Assigns a numeric value or a name to a DESIGN object.

**Note:** Assign values to properties for component pins in the PINPROPERTIES section. Assign values to other properties in the section of the LEF file that describes the object to which the property applies.

## Regions

```
[REGIONS numRegions ;  
  [- regionName {pt pt} ...  
    [+ TYPE {FENCE | GUIDE}]  
    [+ PROPERTY {propName propVal} ...] ...  
  ;] ...  
END REGIONS]
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Defines regions in the design. A region is a physical area to which you can assign a component or group.

*numRegions*

Specifies the number of regions defined in the design.

PROPERTY *propName propVal*

Specifies a numerical or string value for a region property defined in the PROPERTYDEFINITIONS statement. The *propName* you specify must match the *propName* listed in the PROPERTYDEFINITIONS statement.

*regionName pt pt*

Names and defines a region. You define a region as one or more rectangular areas specified by pairs of coordinate points.

TYPE {FENCE | GUIDE}

Specifies the type of region.

*Default:* All instances assigned to the region are placed inside the region boundaries, and other cells are also placed inside the region.

*Value:* Specify one of the following:

FENCE	All instances assigned to this type of region must be exclusively placed inside the region boundaries. No other instances are allowed inside this region.
GUIDE	All instances assigned to this type of region should be placed inside this region; however, it is a preference, not a hard constraint. Other constraints, such as wire length and timing, can override this preference.

### Example 4-24 Regions Statement

```
REGIONS 1 ;  
- REGION1 ( 0 0 ) ( 1200 1200 )  
+ PROPERTY REGIONORDER 1 ;
```

## Rows

```
[ROW rowName siteName origX origY siteOrient  
[DO numX BY numY [STEP stepX stepY]]  
[+ PROPERTY {propName propVal} ...] ... ;] ...
```



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Defines rows in the design.

DO *numX* BY *numY*

Specifies a repeating set of sites that create the row. You must specify one of the values as 1. If you specify 1 for *numY*, then the row is horizontal. If you specify 1 for *numX*, the row is vertical.

*Default:* Both *numX* and *numY* equal 1, creating a single site at this location (that is, a horizontal row with one site).

*origX origY*

Specifies the location of the first site in the row.

Type: Integer, specified in DEF database units

PROPERTY *propName propVal*

Specifies a numerical or string value for a row property defined in the PROPERTYDEFINITIONS statement. The *propName* you specify must match the *propName* listed in the PROPERTYDEFINITIONS statement.

*rowName*

Specifies the row name for this row.

*siteName*

Specifies the LEF SITE to use for the row. A site is a placement location that can be used by LEF macros that match the same site. *siteName* can also refer to a site with a row pattern in its definition, in which case, the row pattern indicates a repeating set of sites that are abutted. For more information, see [“Site”](#) and [“Macro”](#) in “LEF Syntax.”

*siteOrient*

Specifies the orientation of all sites in the row. *siteOrient* must be one of N, S, E, W, FN, FS, FE, or FW. For more information on orientations, see [“Specifying Orientation”](#) on page 664.

STEP *stepX stepY*

Specifies the spacing between sites in horizontal and vertical rows.

### Example 4-25 Row Statements

Assume *siteA* is 200 by 900 database units.

```
ROW row_0 siteA 1000 1000 N ; #Horizontal row is one-site wide at 1000, 1000
ROW row_1 siteA 1000 1000 N DO 1 BY 1 ; #Same as row_0
ROW row_2 siteA 1000 1000 N DO 1 BY 1 STEP 200 0 ; #Same as row_0
ROW row_3 siteA 1000 1000 N DO 10 BY 1 ; #Horizontal row is 10 sites wide,
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
                                #so row width is 200*10=2000 dbu
ROW row_4 siteA 1000 1000 N DO 10 BY 1 STEP 200 0 ; #Same as row_3
ROW row_5 siteA 1000 1000 N DO 1 BY 10 ; #Vertical row is 10 sites high, so
                                #total row height is 900*10=9000 dbu
ROW row_6 siteA 1000 1000 N DO 1 BY 10 STEP 0 900 ; #Same as row_5
```

## Scan Chains

```
[SCANCHAINS numScanChains ;
  [- chainName
    [+ PARTITION partitionName [MAXBITS maxbits]]
    [+ COMMONSCANPINS [ ( IN pin ) ] [( OUT pin ) ] ]
    + START {fixedInComp | PIN} [outPin]
    [+ FLOATING
      {floatingComp [ ( IN pin ) ] [( OUT pin ) ] [( BITS numBits ) ]} ...]
    [+ ORDERED
      {fixedComp [ ( IN pin ) ] [( OUT pin ) ] [( BITS numBits ) ]} ...
    ] ...
    + STOP {fixedOutComp | PIN} [inPin] ]
  ;] ...
END SCANCHAINS]
```

Defines scan chains in the design. Scan chains are a collection of cells that contain both scan-in and scan-out pins. These pins must be defined in the PINS section of the DEF file with + USE SCAN.

*chainName*

Specifies the name of the scan chain. Each statement in the SCANCHAINS section describes a single scan chain.

COMMONSCANPINS [ ( IN *pin* ) ] [( OUT *pin* ) ]

Specifies the scan-in and scan-out pins for each component that does not have a scan-in and scan-out pin specified. You must specify either common scan-in and scan-out pins, or individual scan-in and scan-out pins for each component.

FLOATING {*floatingComp* [ ( IN *pin* ) ] [( OUT *pin* ) ] [( BITS *numBits* ) ]}

Specifies the floating list. You can have one or zero floating lists. If you specify a floating list, it must contain at least one component.

*floatingComp* Specifies the component name.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

<code>( IN <i>pin</i> )</code>	Specifies the scan-in pin. If you do not specify a scan-in pin, the router uses the pin you specified for the common scan pins.
<code>( OUT <i>pin</i> )</code>	Specifies the scan-out pin. If you do not specify a scan-out pin, the router uses the pin you specified for the common scan pins.
<code>BITS <i>numBits</i></code>	<p>Specifies the sequential bit length of any chain element. This allows application tools that do not have library access to determine the sequential bit length contribution of any chain element to ensure the <code>MAXBITS</code> constraints are not violated for chains in a given partition. You can specify 0 to indicate when elements are nonsequential.</p> <p><i>Default:</i> 1</p> <p><i>Type:</i> Integer</p>

**Note:** Scan chain reordering commands can use floating components in any order to synthesize a scan chain. Floating components cannot be shared with other scan chains unless they are in the same `PARTITION`. Each component should only be used once in synthesizing a scan chain.

`MAXBITS maxBits`

When specified with chains that include the `PARTITION` keyword, sets the maximum bit length (flip-flop bit count) that the chain can grow to in the partition.

*Default:* 0 (tool-specific defaults apply, which is probably the number of bits in each chain)

*Type:* Integer

*Value:* Specify a value that is at least as large as the size of the current chain.

`numScanChains`

Specifies the number of scan chains to synthesize.

`ORDERED { fixedComp [( IN pin )] [( OUT pin )] [( BITS numBits )] }`

Specifies an ordered list. You can specify none or several ordered lists. If you specify an ordered list, you must specify at least two fixed components for each ordered list.

<code><i>fixedComp</i></code>	Specifies the component name.
-------------------------------	-------------------------------

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

<code>( IN <i>pin</i> )</code>	Specifies the scan-in pin. If you do not specify a scan-in pin, the router uses the pin you specified for the common scan pins.
<code>( OUT <i>pin</i> )</code>	Specifies the scan-out pin. If you do not specify a scan-out pin, the router uses the pin you specified for the common scan pins.
<code>BITS <i>numBits</i></code>	<p>Specifies the sequential bit length of any chain element. This allows application tools that do not have library access to determine the sequential bit length contribution of any chain element to ensure the <code>MAXBITS</code> constraints are not violated for chains in a given partition. You can specify 0 to indicate when elements are nonsequential.</p> <p><i>Default:</i> 1</p> <p><i>Type:</i> Integer</p>

**Note:** Scan chain reordering commands should synthesize these components in the same order that you specify them in the list. Ordered components cannot be shared with other scan chains unless they are in the same `PARTITION`. Each component should only be used once in synthesizing a scan chain.

`PARTITION partitionName`

Specifies a partition name. This statement allows reordering tools to determine inter-chain compatibility for element swapping (both `FLOATING` elements and `ORDERED` elements). It associates each chain with a partition group, which determines their compatibility for repartitioning by swapping elements between them.

Chains with matching `PARTITION` names constitute a swap-compatible group. You can change the length of chains included in the same partition (up to the `MAXBITS` constraint on the chain), but you cannot eliminate chains or add new ones; the number of chains in the partition is always preserved.

If you do not specify the `PARTITION` keyword, chains are assumed to be in their own single partition, and reordering can be performed only within that chain.

#### Example 4-26 Partition Scanchain

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

In the following definition, chain `chain1_clock1` is specified without a `MAXBITS` keyword. The maximum allowed bit length of the chain is assumed to be the sequential length of the longest chain in any `clock1` partition.

```
SCANCHAINS 77 ;
- chain1_clock1
  + PARTITION clock1
  + START block1/bsr_reg_0 Q
  + FLOATING
    block1/pgm_cgm_en_reg_reg ( IN SD ) ( OUT QZ )
    ...
    block1/start_reset_dd_reg ( IN SD ) ( OUT QZ )
  + STOP block1/start_reset_d_reg SD ;
```

In the following definition, chain `chain2_clock2` is specified with a `PARTITION` statement that associates it with `clock2`, and a maximum bit length of 1000. The third element statement in the `FLOATING` list is a scannable register bank that has a sequential bit length of 4. The `ORDERED` list element statements have total bit lengths of 1 each because the muxes are specified with a maximum bit length of 0.

```
- chain2_clock2
  + PARTITION clock2
    MAXBITS 1000
  + START block1/current_state_reg_0_QZ
  + FLOATING
    block1/port2_phy_addr_reg_0_ ( IN SD ) ( OUT QZ )
    block1/port2_phy_addr_reg_4_ ( IN SD ) ( OUT QZ )
    block1/port3_intf ( IN SD ) ( OUT MSB ) ( BITS 4 )
    ...
  + ORDERED
    block1/mux1 ( IN A ) ( OUT X ) ( BITS 0 )
    block1/ff1 ( IN SD ) ( OUT Q )
  + ORDERED
    block1/mux2 ( IN A ) ( OUT X ) ( BITS 0 )
    block1/ff2 ( IN SD ) ( OUT Q ) ;
```

In the following definition, chain `chain3_clock2` is also specified with a `PARTITION` statement that associates it with `clock2`. This means it is swap-compatible with `chain2_clock2`. The specified maximum bit length for this chain is 1200.

```
- chain3_clock2
  + PARTITION clock2
    MAXBITS 1200
  + START block1/LV_testpoint_0_Q_reg Q
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
+ FLOATING
  block1/LV_testpoint_0_Q_reg ( IN SE ) ( OUT Q )
  block1/tm_state_reg_1_ (IN SD ) ( OUT QZ )
  ...
```

In the following definition, chain `chain4_clock3` is specified with a `PARTITION` statement that associates it with `clock3`. The second element statement in the `FLOATING` list is a scannable register bank that has a sequential bit length of 8, and default pins. The `ORDERED` list element statements have total bit lengths of 2 each because the mux is specified with a maximum bit length of 0.

```
- chain4_clock3
  + PARTITION clock3
  + START block1/prescaler_IO/lfsr_reg1
  + FLOATING
    block1/dp1_timers
    block1/bus8 ( BITS 8 )
    ...
  + ORDERED
    block1/ds1/ff1 ( IN SD ) ( OUT Q )
    block1/ds1/mux1 ( IN B ) ( OUT Y ) ( BITS 0 )
    block1/ds1/ff2 ( IN SD ) ( OUT Q )
    ...
```

`START {fixedInComp | PIN} [outPin]`

Specifies the start point of the scan chain. You must specify this point. The starting point can be either a component, *fixedInComp*, or an I/O pin, `PIN`. If you do not specify *outPin*, the router uses the pin specified for common scan pins.

`STOP {fixedOutComp | PIN} [inPin]`

Specifies the endpoint of the scan chain. You must specify this point. The stop point can be either a component, *fixedOutComp*, or an I/O pin, `PIN`. If you do not specify *inPin*, the router uses the pin specified for common scan pins.

### Scan Chain Rules

Note the following when defining scan chains.

- Each scan-in/scan-out pin pair of adjacent components in the ordered list cannot have different owning nets.
- No net can connect a scan-out pin of one component to the scan-in pin of a component in a different scan chain.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

- For incremental DEF, if you have a COMPONENTS section and a SCANCHAINS section in the same DEF file, the COMPONENTS section must appear before the SCANCHAINS section. If the COMPONENTS section and SCANCHAINS section are in different DEF files, you must read the COMPONENTS section or load the database before reading the SCANCHAINS section.

#### Example 4-27 Scan Chain Statements

```
Nets 100; #Number of nets resulting after scan chain synthesis
- SCAN-1 ( C1 SO + SYNTHESIZED )
          ( C4 SI + SYNTHESIZED ) + SOURCE TEST ;
- ...
- N1 ( C3 SO + SYNTHESIZED )
     ( C11 SI + SYNTHESIZED ) ( AND1 A ) ;
- ...
END NETS
SCANCHAINS 2; #Specified before scan chain ordering
- S1
  + COMMONSCANPINS ( IN SI ) ( OUT SO )
  + START SIPAD OUT
  + FLOATING C1 C2 ( IN D ) ( OUT Q ) C3 C4 C5...CN
  + ORDERED A1 ( OUT Q ) A2 ( IN D ) ( OUT Q ) ...
    AM ( N D )
  + ORDERED B1 B2 ... BL
  + STOP SOPAD IN ;
- S2 ... ;
END SCANCHAINS
SCANCHAINS 2 ; #Specified after scan chain ordering
- S1
  + START SIPAD OUT
  + FLOATING C1 ( IN SI ) ( OUT SO )
    C2 ( IN D ) ( OUT Q )
    C3 ( IN SI ( OUT SO ) ... CN ( IN SI ) ( OUT SO )
  + ORDERED A1 ( IN SI ) ( OUT Q )
    A2 ( IN D ) ( OUT Q ) ... AM ( IN D ) ( OUT SO )
  + ORDERED B1 ( IN SI ) ( OUT SO )
    B2 ( IN SI ) ( OUT SO ) ...
  + STOP SOPAD IN ;
- S2 ... ;
END SCANCHAINS
```

## Slots

```
[SLOTS numSlots ;  
    [- LAYER layerName  
        {RECT pt pt | POLYGON pt pt pt ... } ...  
    ;] ...  
END SLOTS]
```

Defines the rectangular shapes that form the slotting of the wires in the design. Each slot is defined as an individual rectangle.

*LAYER layerName*

Specifies the layer on which to create slots.

*numSlots*

Specifies the number of **LAYER** statements in the **SLOTS** statement, *not* the number of rectangles.

*POLYGON pt pt pt*

Specifies a sequence of at least three points to generate a polygon geometry. The polygon edges must be parallel to the x axis, the y axis, or at a 45-degree angle. Each **POLYGON** statement defines a polygon generated by connecting each successive point, and then the first and last points. The *pt* syntax corresponds to a coordinate pair, such as *x y*. Specify an asterisk (\*) to repeat the same value as the previous *x* or *y* value from the last point.

*Type:* DEF database units

*RECT pt pt*

Specifies the lower left and upper right corner coordinates of the slot geometry.

### Example 4-28 Slots Statements

The following statement defines slots for layers **MET1** and **MET2**.

```
SLOTS 2 ;  
- LAYER MET1  
    RECT ( 1000 2000 ) ( 1500 4000 )  
    RECT ( 2000 2000 ) ( 2500 4000 )  
    RECT ( 3000 2000 ) ( 3500 4000 ) ;  
- LAYER MET2  
    RECT ( 1000 2000 ) ( 1500 4000 )  
    RECT ( 1000 4500 ) ( 1500 6500 )  
    RECT ( 1000 7000 ) ( 1500 9000 )  
    RECT ( 1000 9500 ) ( 1500 11500 ) ;
```



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
END SLOTS
```

The following `SLOTS` statement defines two rectangles and one polygon slot geometries:

```
SLOTS 1 ;
    - LAYER metall
        RECT ( 100 200 ) ( 150 400 )
        POLYGON ( 100 100 ) ( 200 200 ) ( 300 200 ) ( 300 100 )
        RECT ( 300 200 ) ( 350 400 ) ;
END SLOTS
```

## Special Nets

```
[SPECIALNETS numNets ;
    [- netName
        [ ( {compName pinName | PIN pinName} [+ SYNTHESIZED] ) ] ...
        [+ VOLTAGE volts]
        [specialWiring] ...
        [+ SOURCE {DIST | NETLIST | TIMING | USER}]
        [+ FIXEDBUMP]
        [+ ORIGINAL netName]
        [+ USE {ANALOG | CLOCK | GROUND | POWER | RESET | SCAN | SIGNAL | TIEOFF}]
        [+ PATTERN {BALANCED | STEINER | TRUNK | WIREDLOGIC}]
        [+ ESTCAP wireCapacitance]
        [+ WEIGHT weight]
        [+ PROPERTY {propName propVal} ...] ...
    ;] ...
END SPECIALNETS]
```

Defines netlist connectivity and special-routes for nets containing special pins. Special-routes are created by "special routers" or "manually", and should not be modified by a signal router. Special routes are normally used for power-routing, fixed clock-mesh routing, high-speed buses, critical analog routes, or flip-chip routing on the top-metal layer to bumps.

Input parameters for a net can appear in the `NETS` section or the `SPECIALNETS` section. In case of conflicting values for an argument, the DEF reader uses the last value encountered for the argument. `NETS` and `SPECIALNETS` statements can appear more than once in a DEF file. If a particular net has mixed wiring or pins, specify the special wiring and pins first.

You can also specify the netlist in the `COMPONENTS` statement. If the netlist is specified in both `NETS` and `COMPONENTS` statements, and if the specifications are not consistent, an error message appears. On output, the writer outputs the netlist in either format, depending on the command arguments of the output command.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

*compNamePattern pinName*

Specifies the name of a special pin on the net and its corresponding component. You can use a *compNamePattern* to specify a set of component names. During evaluation of the pattern match, components that match the pattern but do not have a pin named *pinName* are ignored. The pattern match character is \* (asterisk). For example, a component name of *abc/def* would be matched by *a\**, *abc/d\**, or *abc/def*.

*ESTCAP wireCapacitance*

Specifies the estimated wire capacitance for the net. *ESTCAP* can be loaded with simulation data to generate net constraints for timing-driven layout.

*FIXEDBUMP*

Indicates that the bump net cannot be reassigned to a different pin.

It is legal to have a pin without geometry to indicate a logical connection and to have a net that connects that pin to two other instance pins that have geometry. Area I/Os have a logical pin that is connected to a bump and an input driver cell. The bump and driver cell have pin geometries (and, therefore, should be routed and extracted), but the logical pin is the external pin name without geometry (typically the Verilog pin name for the chip).

Bump nets also can be specified in the *NETS* statement. If a net name appears in both the *NETS* and *SPECIALNETS* statements, the *FIXEDBUMP* keyword also should appear in both statements. However, the value only exists once within a given application's database for the net name.

Because DEF is often used incrementally, the last value read in is used. Therefore, in a typical DEF file, if the same net appears in both statements, the *FIXEDBUMP* keyword (or lack of it) in the *NETS* statement is the value that is used because the *NETS* statement is defined after the *SPECIALNETS* statement.

### Example 4-29 Fixed Bump

The following example describes a logical pin that is connected to a bump and an input driver cell. The I/O driver cell and bump cells are specified in the *COMPONENTS* statement. Bump cells are usually placed with + *COVER* placement status so they cannot be moved manually by mistake.

```
COMPONENTS 200
- driver1 drivercell + PLACED ( 100 100 ) N ;
...
- bumpa1 bumpcell + COVER ( 100 100 ) N ;
- bumpa2 bumpcell + COVER ( 200 100 ) N ;
```

The pin is assigned in the *PIN* statement.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

PINS 100

- n1 + NET n1 + SPECIAL + DIRECTION INPUT ;
- n2 + NET n2 + SPECIAL + DIRECTION INPUT ;

In the **SPECIALNETS** statement, the net *n1* is assigned to *bumpa1* and cannot be reassigned. Note that another net *n2* is assigned to *bumpa2*; however, I/O optimization commands are allowed to reassign *bumpa2* to a different net.

SPECIALNETS 100

- n1 ( driver1 in ) ( bumpa1 bumpin ) + FIXEDBUMP ;
- n2 ( driver2 in ) ( bumpa2 bumpin ) ;

*netName*

Specifies the name of the net.

ORIGINAL *netName*

Specifies the original net partitioned to create multiple nets, including the current net.

PATTERN {BALANCED | STEINER | TRUNK | WIREDLOGIC}

Specifies the routing pattern used for the net.

**Default:** STEINER

**Value:** Specify one of the following:

BALANCED	Used to minimize skews in timing delays for clock nets.
STEINER	Used to minimize net length.
TRUNK	Used to minimize delay for global nets.
WIREDLOGIC	Used in ECL designs to connect output and mustjoin pins before routing to the remaining pins.

PIN *pinName*

Specifies the name of an I/O pin on a net or a subnet.

PROPERTY *propName propVal*

Specifies a numerical or string value for a net property defined in the **PROPERTYDEFINITIONS** statement. The *propName* you specify must match the *propName* listed in the **PROPERTYDEFINITIONS** statement.

*specialWiring*

Specifies the special wiring for the net. For syntax information, see [“Special Wiring Statement”](#) on page 733.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

SOURCE {DIST | NETLIST | TIMING | USER}

Specifies how the net is created. The value of this field is preserved when input to the DEF reader.

DIST	Net is the result of adding physical components (that is, components that only connect to power or ground nets), such as filler cells, well-taps, tie-high and tie-low cells, and decoupling caps.
NETLIST	Net is defined in the original netlist. This is the default value, and is not normally written out in the DEF file.
TEST	Net is part of a scanchain.
TIMING	Net represents a logical rather than physical change to netlist, and is used typically as a buffer for a clock-tree, or to improve timing on long nets.
USER	Net is user defined.

SYNTHESIZED

Used by some tools to indicate that the pin is part of a synthesized scan chain.

USE {ANALOG | CLOCK | GROUND | POWER | RESET | SCAN | SIGNAL | TIEOFF}

Specifies how the net is used.

Value: Specify one of the following:

ANALOG	Used as an analog signal net.
CLOCK	Used as a clock net.
GROUND	Used as a ground net.
POWER	Used as a power net.
RESET	Used as a reset net.
SCAN	Used as a scan net.
SIGNAL	Used as a digital signal net.
TIEOFF	Used as a tie-high or tie-low net.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

VOLTAGE *volts*

Specifies the voltage for the net, as an integer in units of .001 volts. For example, VOLTAGE 1500 in DEF is equal to 1.5 V.

WEIGHT *weight*

Specifies the weight of the net. Automatic layout tools attempt to shorten the lengths of nets with high weights. Do not specify a net weight larger than 10, or assign weights to more than 3 percent of the nets in a design.

**Note:** The net constraints method of controlling net length is preferred over using net weights.

### Special Wiring Statement

```
[[+ COVER | + FIXED | + ROUTED | + SHIELD shieldNetName]
  [+ SHAPE shapeType] [+ MASK maskNum]
  + POLYGON layerName pt pt pt ...
  | + RECT layerName pt pt
  | + VIA viaName [orient] pt ...
|{+ COVER | + FIXED | + ROUTED | + SHIELD shieldNetName}
  layerName routeWidth
  [+ SHAPE
    {RING | PADRING | BLOCKRING | STRIPE | FOLLOWPIN
    | IOWIRE | COREWIRE | BLOCKWIRE | BLOCKAGEWIRE | FILLWIRE
    | FILLWIREOPC | DRCFILL}]
  [+ STYLE styleNum]
  routingPoints
[NEW layerName routeWidth
  [+ SHAPE
    {RING | PADRING | BLOCKRING | STRIPE | FOLLOWPIN
    | IOWIRE | COREWIRE | BLOCKWIRE | BLOCKAGEWIRE | FILLWIRE
    | FILLWIREOPC | DRCFILL}]
  [+ STYLE styleNum]
  routingPoints
] ...

] ...
```

Defines the wiring for both routed and shielded nets.

COVER

Specifies that the wiring cannot be moved by either automatic layout or interactive commands. If no wiring is specified for a particular net, the net is unrouted. If you specify COVER, you must also specify *layerName width*.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

#### FIXED

Specifies that the wiring cannot be moved by automatic layout, but can be changed by interactive commands. If no wiring is specified for a particular net, the net is unrouted. If you specify `FIXED`, you must also specify *layerName width*.

#### *layerName routeWidth*

Specifies the width for wires on *layerName*. Normally, only routing layers use this syntax, but it is legal for any layer (cut layer shapes or other layers like masterslice layers normally use the `RECT` or `POLYGON` statements). For more information, see [“Defining Routing Points”](#) on page 743.

Vias do not change the route width. When a via is used in special wiring, the previously established *routeWidth* is used for the next wire in the new layer. To change the *routeWidth*, a new path must be specified using `NEW layerName routeWidth`.

Many applications require *routeWidth* to be an even multiple of the manufacturing grid in order to be fabricated, and to keep the center line on the manufacturing grid.

*Type:* Integer, specified in database units

#### `NEW layerName routewidth`

Indicates a new wire segment (that is, that there is no wire segment between the last specified coordinate and the next coordinate) on *layerName*, and specifies the width for the wire. Noncontinuous paths can be defined in this manner. For more information, see [“Defining Routing Points”](#) on page 743.

*Type:* Integer, specified in database units

#### `POLYGON layerName pt pt pt`

Specifies a sequence of at least three points to generate a polygon geometry on *layerName*. The polygon edges must be parallel to the x axis, the y axis, or at a 45-degree angle. Each polygon statement defines a polygon generated by connecting each successive point, then connecting the first and last points. The *pt* syntax corresponds to a coordinate pair, such as *x y*. Specify an asterisk (\*) to repeat the same value as the previous *x* or *y* value from the last point.

*Type:* (*x y*) Integer, specified in database units

#### `RECT layerName pt pt`

Specifies a rectangle on layer *layerName*. The two points define opposite corners of the rectangle. The *pt* syntax corresponds to a coordinate pair, such as *x y*. You cannot define the same *x* and *y* values for both points (that is, a zero-area rectangle is not legal).

*Type:* (*x y*) Integer, specified in database units

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

#### ROUTED

Specifies that the wiring can be moved by automatic layout tools. If no wiring is specified for a particular net, the net is unrouted. If you specify `ROUTED`, you must also specify *layerName width*.

#### *routingPoints*

Defines the center line coordinates of the route on *layerName*. For information on using routing points, see [“Defining Routing Points”](#) on page 743. For an example of special wiring with routing points, see [Example 4-31](#) on page 739.

The *routingPoints* syntax is defined as follows:

```
( x y [extValue])
  { [MASK maskNum] ( x y [extValue])
    | [MASK viaMaskNum] viaName [orient]
    [DO numX BY numY STEP stepX stepY]
  } ...
```

`DO numX BY numY STEP stepX stepY`

Creates an array of power vias of the via specified with *viaName*.

*numX* and *numY* specify the number of vias to create, in the x and y directions. Do not specify 0 as a value.

*Type:* Integer

*stepX* and *stepY* specify the step distance between vias, in the x and y directions, in DEF distance database units.

*Type:* Integer

For an example of a via array, see [Example 4-30](#) on page 738.

*extValue*

Specifies the amount by which the wire is extended past the endpoint of the segment.

*Type:* Integer, specified in database units

*Default:* 0

`MASK maskNum`

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

Specifies which mask for double or triple patterning lithography to use for the next wire. The *maskNum* variable must be a positive integer. Most applications support values of 1, 2, or 3 only. Shapes without any defined mask have no mask set (that is, they are uncolored).

`MASK viaMaskNum`

Specifies which mask for double or triple patterning lithography is to be applied to the next via's shapes on each layer.

The *viaMaskNum* variable is a hex-encoded three digit value of the form:

`<topMaskNum><cutMaskNum><bottomMaskNum>`

For example, MASK 113 means the top metal and cut layer *maskNum* is 1, and the bottom metal layer *maskNum* is 3. A value of 0 means the shape on that layer has no mask assignment (is uncolored), so 013 means the top layer is uncolored. If either the first or second digit is missing, they are assumed to be 0, so 013 and 13 mean the same thing. Most applications support *maskNum* values of 0, 1, 2, or 3 only for double or triple patterning.



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

The *topMaskNum* and *bottomMaskNum* variables specify which mask the corresponding metal shape belongs to. The via-master metal mask values have no effect.

For the cut-layer, the *cutMaskNum* variable will define the mask for the bottom-most, and then the left-most cut. For multi-cut vias, the via-instance cut masks are derived from the via-master cut mask values. The via-master must have a mask defined for all the cut shapes and every via-master cut mask is "shifted" (from 1 to 2, and 2 to 1 for two mask layers, and from 1 to 2, 2 to 3, and 3 to 1 for three mask layers), so the lower-left cut matches the *cutMaskNum* value.

Similarly, for the metal layer, the *topMaskNum/bottomMaskNum* would define the mask for the bottom-most, then leftmost metal shape. For multiple disjoint metal shapes, the via-instance metal masks are derived from the via-master metal mask values. The via-master must have a mask defined for all of the metal shapes, and every via-master metal mask is "shifted" (1->2, 2->1 for two mask layers, 1->2, 2->3, 3->1 for three mask layers) so the lower-left cut matches the *topMaskNum/bottomMaskNum* value.

See [Example 4-32](#) on page 739.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

<i>orient</i>	<p>Specifies the orientation of the <i>viaName</i> that precedes it, using the standard DEF orientation values of N, S, E, W, FN, FS, FE, and FW (See <a href="#">“Specifying Orientation”</a> on page 664).</p> <p>If you do not specify <i>orient</i>, N (North) is the default non-rotated value used. All other orientation values refer to the flipping or rotation around the via origin (the 0, 0 point in the via shapes). The via origin is still placed at the (<i>x y</i>) value given in the routing statement just before the <i>viaName</i>.</p> <p><b>Note:</b> Some tools do not support orientation of vias inside their internal data structures; therefore, they are likely to translate vias with an orientation into a different but equivalent via that does not require an orientation.</p>
<i>viaName</i>	<p>Specifies a via to place at the last point. If you specify a via, <i>layerName</i> for the next routing coordinates (if any) is implicitly changed to the other routing layer for the via. For example, if the current layer is <i>metal1</i>, a <i>via12</i> changes the layer to <i>metal2</i> for the next routing coordinates.</p>
( <i>x y</i> )	<p>Specifies the route coordinates. You cannot specify a route with zero length.</p> <p>For more information, see <a href="#">“Specifying Coordinates”</a> on page 744.</p> <p><i>Type:</i> Integer, specified in database units</p>

### Example 4-30 Via Arrays

The following example specifies arrays of via VIAGEN21\_2 on *metal1* and *metal2*.

```
SPECIALNETS 2 ;
-vdd ( * vdd )
+ ROUTED metal1 150 ( 100 100 ) ( 200 * )
NEW metal1 0 ( 200 100 ) VIAGEN21_2 DO 10 BY 20 STEP 10000 20000
NEW metal2 0 (-900 -30 ) VIAGEN21_2 DO 1000 BY 1 STEP 5000 0
...
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

As with any other VIA statement, the DO statement does not change the previous coordinate. Therefore, the following statement creates a *metal1* wire of width 50 from ( 200 100 ) to ( 200 200 ) along with the via array that starts at ( 200 100 ).

```
NEW metal1 50 ( 200 100 ) VIAGEN21_2 DO 10 BY 20 STEP 1000 2000 ( 200 200 )
```

#### Example 4-31 Special Wiring With Routing Points

```
SPECIALNETS 1 ;
- vdd (*vdd)
+ USE POWER
+ POLYGON metal1 ( 0 0 ) ( 0 100 ) ( 100 100 ) ( 200 200 ) ( 200 0 )
+ POLYGON metal2 ( 100 100 ) ( * 200 ) ( 200 * ) ( 300 300 ) ( 300 100 )
+ RECT metal1 ( 0 0 ) ( 100 200 )
+ ROUTED metal1 100 ( 0 0 50 ) ( 100 0 50 ) via12 ( 100 100 50 )
+ ROUTED metal2 100 + SHAPE RING + STYLE 1 ( 0 0 ) ( 100 100 ) ( 200 100 )
;
END SPECIALNETS
```

#### Example 4-32 Multi-Mask Layers with Special Wiring

The following example shows a routing statement that specifies three-mask layers M1 and VIA1, and a two-mask layer M2:

```
+ FIXED + SHAPE RING + MASK 2 + RECT M3 ( 0 0 ) ( 10 10 )
+ ROUTED M1 2000 (10 0 ) MASK 3 (10 20 ) VIA1_1
  NEW M2 1000 ( 10 10 ) (20 10) MASK 1 ( 20 20 ) MASK 031 VIA1_2
+ SHAPE STRIPE + VIA VIA3_3 ( 30 30 ) ( 40 40 )
;
```

This indicates that the:

- M3 rectangle shape is on mask 2, has FIXED route status, and shape RING
- M1 wire shape from (10 0) to (10 20) is on mask 3.
- VIA1\_1 via has no preceding MASK statement so all the metal and cut shapes have no mask and are uncolored
- first NEW M2 wire shape (10 10) to (20 10) has no mask set and is uncolored
- second M2 wire shape (20 10) to (20 20) is on mask 1
- VIA1\_2 via has a MASK 031 (it can be MASK 31 also) so:
  - *topMaskNum* is 0 in the 031 value, so no mask is set for the top metal (M2) shape

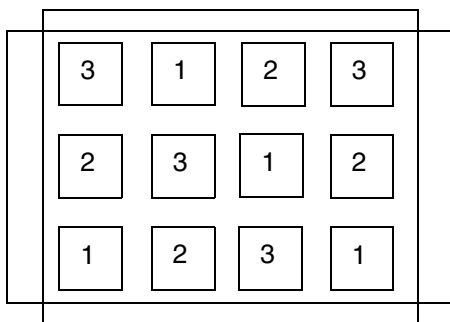
## LEF/DEF 5.8 Language Reference

### DEF Syntax

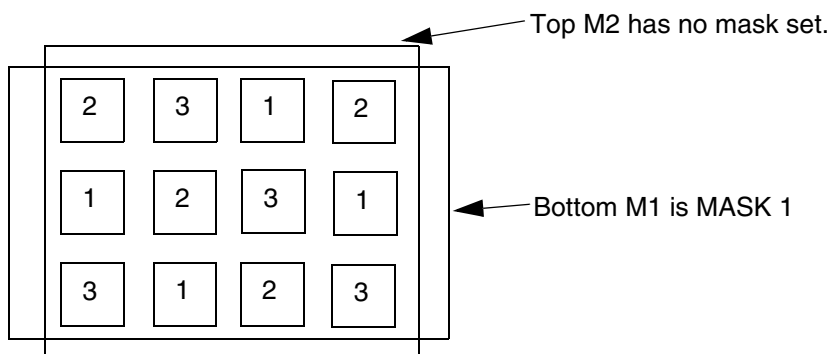
- ❑ *bottomMaskNum* is 1 in the 031 value, so mask 1 is used for the bottom metal (M1) shape
- ❑ *cutMaskNum* is 3 in the 031 value, so the bottom-most, then left-most cut of the via-instance is mask 3. The mask for the other cuts of the via-instance are derived from the via-master by "shifting" the via-master's cut masks to match. So, if the via-master's bottom-left cut is mask 1, then the via-master cuts on mask 1 become mask 3 for the via-instance, and similarly cuts on 2 shift to 1, and cuts on 3 shift to 2. See [Figure 4-11](#) on page 740.

The VIA3\_3 has shape STRIPE, and the via is at both (30 30) and (40 40). There is no wire segment between (30 30) and (40 40). If the route status is not specified, it is considered as + ROUTED.

**Figure 4-11 Multi-Mask Patterns with Special Wiring**



Via-master cut masks for VIA1.



Masks for via-instance: ... ( 20 20 ) MASK 031 VIA1\_2

Bottom M1 is MASK 1. Top M2 has no mask set. Lower-left cut is MASK 3, and other via-instance cut shape masks are derived from via-master cut-masks as shown above by "shifting" the via-master masks to match: 1->3, 2->1, 3->2.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

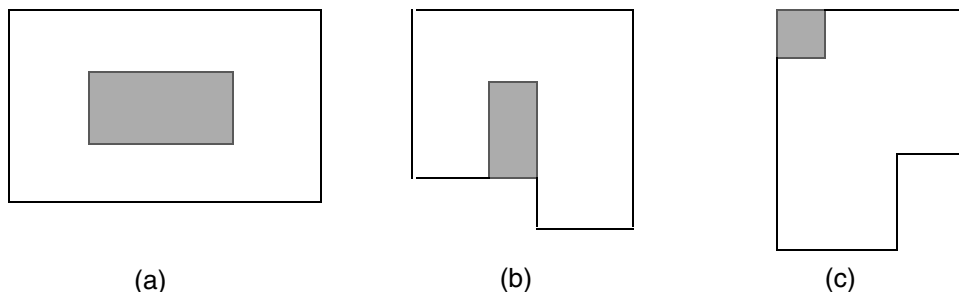
#### SHAPE

Specifies a wire with special connection requirements because of its shape. This applies to vias as well as wires.

*Value:* Specify one of the following:

RING	Used as ring, target for connection
PADRING	Connects padings
BLOCKRING	Connects rings around the blocks
STRIPE	Used as stripe
FOLLOWPIN	Connects standard cells to power structures.
IOWIRE	Connects I/O to target
COREWIRE	Connects endpoints of followpin to target
BLOCKWIRE	Connects block pin to target
BLOCKAGEWIRE	Connects blockages
FILLWIRE	Represents a fill shape that does not require OPC. It is normally connected to a power or ground net. Floating fill shapes should be in the <code>FILL</code> section.
FILLWIREOPC	Represents a fill shape that requires OPC. It is normally connected to a power or ground net. Floating fill shapes should be in the <code>FILL</code> section.
DRCFILL	Used as a fill shape to correct DRC errors, such as <code>SPACING</code> , <code>MINENCLOSEDAREA</code> , or <code>MINSTEP</code> violations on wires and pins of the same net (see <a href="#">Figure 4-12</a> on page 742.)

**Figure 4-12 Fill Shapes**



Examples of fill inside (a) a `MINENCLOSEDAREA` violation, (b) a `SPACING` violation, and (c) a `MINSTEP` violation.

`SHIELD` *shieldNetName*

Specifies the name of a regular net to be shielded by the special net being defined.

After describing shielded routing for a net, use `+ ROUTED` to return to the routing of the special net being defined.

`STYLE` *styleNum*

Specifies a previously defined style from the `STYLES` section in this DEF file. The style is used with the endpoints of each routing segment to define the routing shape, and applies to all routing segments defined in one *routingPoints* statement.

`VIA` *viaName* [*orient*] *pt* ...

Specifies the name of the via placed at every point in the list with an optional orientation. For example, the statement

`VIA myVia ( 0 0 ) ( 1 1 )` indicates an instance of `myVia` at 0,0 and at 1,1.

...

### Example 4-33 Special Nets Statements

Signoff DRC tools may require metal shapes under the trim metal shapes to fill up the gaps between the line-end of wires sandwiched by the trim metal shape to be output in DEF. Those metal shapes would be written out in `_TRIMMETAL_FILLS_RESERVED` with the `DRCFILL` tag in the `SPECIALNETS` section in the following format:

```
- _TRIMMETAL_FILLS_RESERVED
  + ROUTED + SHAPE DRCFILL + MASK x + RECT M1 (x x) (x x)
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

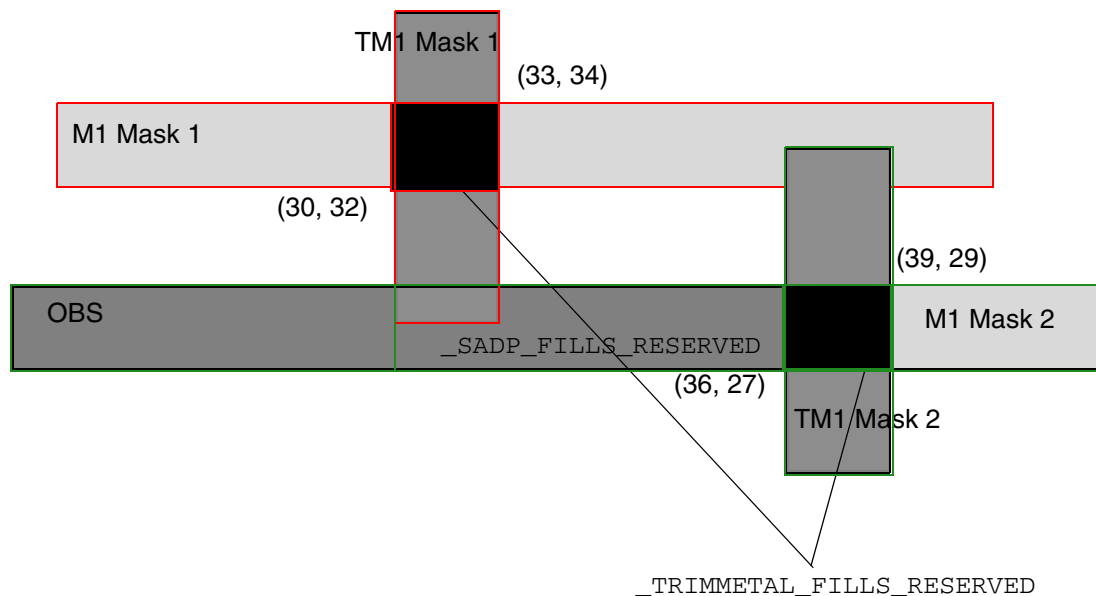
As trim metal shapes need to be aligned and merged, dummy patches are often added even on OBS and unconnected pins. Those patches would be written out in `_SADP_FILLS_RESERVED` with the `DRCFILL` tag in the `SPECIALNETS` section in the following format:

```
- _SADP_FILLS_RESERVED
    + ROUTED + SHAPE DRCFILL + MASK x + RECT M1 (x x) (x x)
```

The following `SPECIALNETS` statement defines two metal shapes under the trim metal shapes and a dummy patch written out with the `DRCFILL` tag:

```
SPECIALNETS 2 ;
- _TRIMMETAL_FILLS_RESERVED
    + ROUTED + SHAPE DRCFILL + MASK 1 + RECT M1 (30 32) (33 34)
    + ROUTED + SHAPE DRCFILL + MASK 2 + RECT M1 (36 27) (39 29)
- _SADP_FILLS_RESERVED
    + ROUTED + SHAPE DRCFILL + MASK 2 + RECT M1 (30 27) (36 29)
END SPECIALNETS
```

**Figure 4-13 Trim Metal and SADP Fills in the SPECIALNETS Section**



### Defining Routing Points

Routing points define the center line coordinates of a route. If a route has a 90-degree edge, it has a width of `routeWidth`, and extends from one coordinate (`x y`) to the next coordinate.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

If either endpoint has an optional extension value (*extValue*), the wire is extended by that amount past the endpoint. If a coordinate with an extension value is specified after a via, the wire extension is added to the beginning of the next wire segment after the via (zero-length wires are not allowed). Some applications convert the extension value to an equivalent route that has the *x* and *y* points already extended, with no extension value. If no extension value is defined, the wire extension is 0, and the wire is truncated at the endpoint.

The *routeWidth* must be an even value to ensure that the corners of the route fall on a legal database coordinate without round off. Because most vendors specify a manufacturing grid, *routeWidth* must be an even multiple of the manufacturing grid in order to be fabricated.

If the wire segment is a 45-degree edge, and no *STYLE* is specified, the default octagon style is used for the endpoints. The *routeWidth* must be an even multiple of the manufacturing grid in order to keep all of the coordinates of the resulting outer wire boundary on the manufacturing grid.

If a *STYLE* is defined for 90-degree or 45-degree routes, the routing shape is defined by the center line coordinates and the style. No corrections, such as snapping to manufacturing grid, should be applied, and any extension values are ignored. The DEF file should contain values that are already snapped, if appropriate. The *routeWidth* indicates the desired user width, and represents the minimum allowed width of the wire that results from the style when the 45-degree edges are snapped to the manufacturing grid. See [Figure 4-15](#) on page 749 through [Figure 4-24](#) on page 758 for examples.

### Specifying Coordinates

To maximize compactness of the design files, the coordinates allow for the asterisk ( *\** ) convention. For example, ( *x* *\** ) indicates that the *y* coordinate last specified in the wiring specification is used; ( *\** *y* ) indicates that the *x* coordinate last specified is used.

Each coordinate sequence defines a connected orthogonal or 45-degree path through the points. The first coordinate in a sequence must not have an *\** element.

All subsequent points in a connected sequence must create orthogonal or 45-degree paths. For example, the following sequence is a valid path:

```
( 100 200 ) ( 200 200 ) ( 200 500 )
```

The following sequence is an equivalent path:

```
( 100 200 ) ( 200 * ) ( * 500 )
```

The following sequence is not valid because it is not an orthogonal or 45-degree segment.

```
( 100 200 ) ( 300 500 )
```



### ***Special Pins and Wiring***

Pins that appear in the `SPECIALNETS` statement are special pins. Regular routers do not route to these pins. The special router routes special wiring to special pins. If you use a component-based format to input the connectivity for the design, special pins to be routed by the special router also must be specified in the `SPECIALNETS` statement, because pins included in the `COMPONENTS` statement are considered regular.

The following example inputs connectivity in a component-based format, specifies `VDD` and `VSS` pins as special pins, and marks `VDD` and `VSS` nets for special routing:

```
COMPONENTS 3 ;
    C1 AND N1 N2 N3 ;
    C2 AND N4 N5 N6 ;
END COMPONENTS

SPECIALNETS 2 ;
    VDD ( * VDD ) + WIDTH M1 5 ;
    VSS ( * VSS ) ;
END SPECIALNETS
```

### ***Shielded Routing***

If, in a non-routed design, a net has + `SHIELDNET` attributes, the router adds shielded routing to this net. + `NOSHIELD` indicates the last wide segment of the net is not shielded. If the last segment is not shielded and is tapered, use the + `TAPER` keyword instead of + `NOSHIELD`. For example:

```
+ SHIELDNET VSS      # both sides will be shielded with VSS
+ SHIELDNET VDD      # one side will be shielded with VDD and
+ SHIELDNET VSS      # one side will be shielded with VSS
```

After you add shielded routing to a special net, it has the following syntax:

```
+ SHIELD regularNetName
    MET2 regularWidth ( x y )
```

A shield net specified for a regular net should be defined earlier in the DEF file in the `SPECIALNETS` section. After describing shielded routing for a net, use + `ROUTED` to return to the routing of the current special net.

For example:

```
SPECIALNETS 2 ;
    - VSS
        + ROUTED MET2 200
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
...
+ SHIELD my_net MET2 100 ( 14100 342440 ) ( 13920 * )
    M2_TURN ( * 263200 ) M1M2 ( 2400 * ) ;
- VDD
    + ROUTED MET2 200
...
+ SHIELD my_net MET2 100 ( 14100 340440 ) ( 8160 * )
    M2_TURN ( * 301600 ) M1M2 ( 2400 * ) ;
END SPECIALNETS
```

## Styles

```
[STYLES numStyles ;
    {- STYLE styleNum pt pt ... ;} ...
END STYLES]
```

Defines a convex polygon that is used at each of the endpoints of a wire to precisely define the wire's outer boundary. A style polygon consists of two to eight points. Informally, half of the style polygon defines the first endpoint wire boundary, and the other half of the style polygon defines the second endpoint wire boundary. Octagons and squares are the most common styles.

*numStyles*

Specifies the number of styles specified in the `STYLES` section.

`STYLE styleNum pt pt`

Defines a new style. *styleNum* is an integer that is greater than or equal to 0 (zero), and is used to reference the style later in the DEF file. When defining multiple styles, the first *styleNum* must be 0 (zero), and any following *styleNum* should be numbered consecutively so that a table lookup can be used to find them easily.

Style numbers are keys used locally in the DEF file to reference a particular style, but not actual numbers preserved in the application. Each style number must be unique. Style numbers can only be used inside the same DEF file, and are not preserved for use in other DEF files. Because applications are not required to preserve the style number itself, an application that writes out an equivalent DEF file might use different style numbers.

*Type:* Integer

The *pt* syntax specifies a sequence of at least two points to generate a polygon geometry. The syntax corresponds to a coordinate pair, such as *x y*. Specify an asterisk (\*) to repeat the same value as the previous *x* (or *y*) value from the last point. The polygon must be convex. The polygon edges must be parallel to the x axis, the y axis, or

at a 45-degree angle, and must enclose the point (0 0).

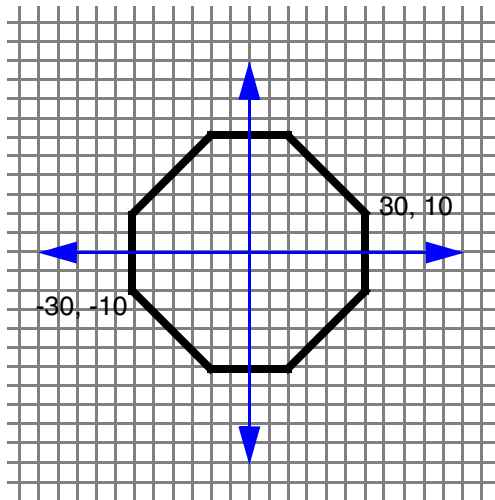
*Type:* Integer, specified in DEF database units

#### **Example 4-34 Styles Statement**

The following `STYLES` statement defines the basic octagon shown in [Figure 4-14](#) on page 747.

```
STYLES 1 ;  
    - STYLE 1 ( 30 10 ) ( 10 30 ) ( -10 30 ) ( -30 10 ) ( -30 -10 )  
              ( -10 -30 ) ( 10 -30 ) ( 30 -10 ) ;  
END STYLES
```

**Figure 4-14**



#### **Defining Styles**

A style is defined as a polygon with points  $P_1$  through  $P_n$ . The center line is given as  $(X_0, Y_0)$  to  $(X_1, Y_1)$ . Two sets of points are built ( $P_{0,1}$  through  $P_{0,n}$  and  $P_{1,1}$  through  $P_{1,n}$ ) as follows:

$$P_{0,i} = P_i + (X_0, Y_0) \text{ for } 1 \leq i \leq n$$

$$P_{1,i} = P_i + (X_1, Y_1) \text{ for } 1 \leq i \leq n$$

The resulting wire segment shape is a counterclockwise, eight-sided polygon ( $S_1$  through  $S_8$ ) that can be computed in the following way:

$S_1$  = lowest point in (left-most points in ( $P_{0,1}$  through  $P_{0,n}$   $P_{1,1}$  through  $P_{1,n}$ ))

$S_2$  = left-most point in (lowest points in ( $P_{0,1}$  through  $P_{0,n}$   $P_{1,1}$  through  $P_{1,n}$ ))

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

$S3$  = right-most point in (lowest points in ( $P0,1$  through  $P0,n$   $P1,1$  through  $P1,n$ ))

$S4$  = lowest point in (right-most points in ( $P0,1$  through  $P0,n$   $P1,1$  through  $P1,n$ ))

$S5$  = highest point in (right-most points in ( $P0,1$  through  $P0,n$   $P1,1$  through  $P1,n$ ))

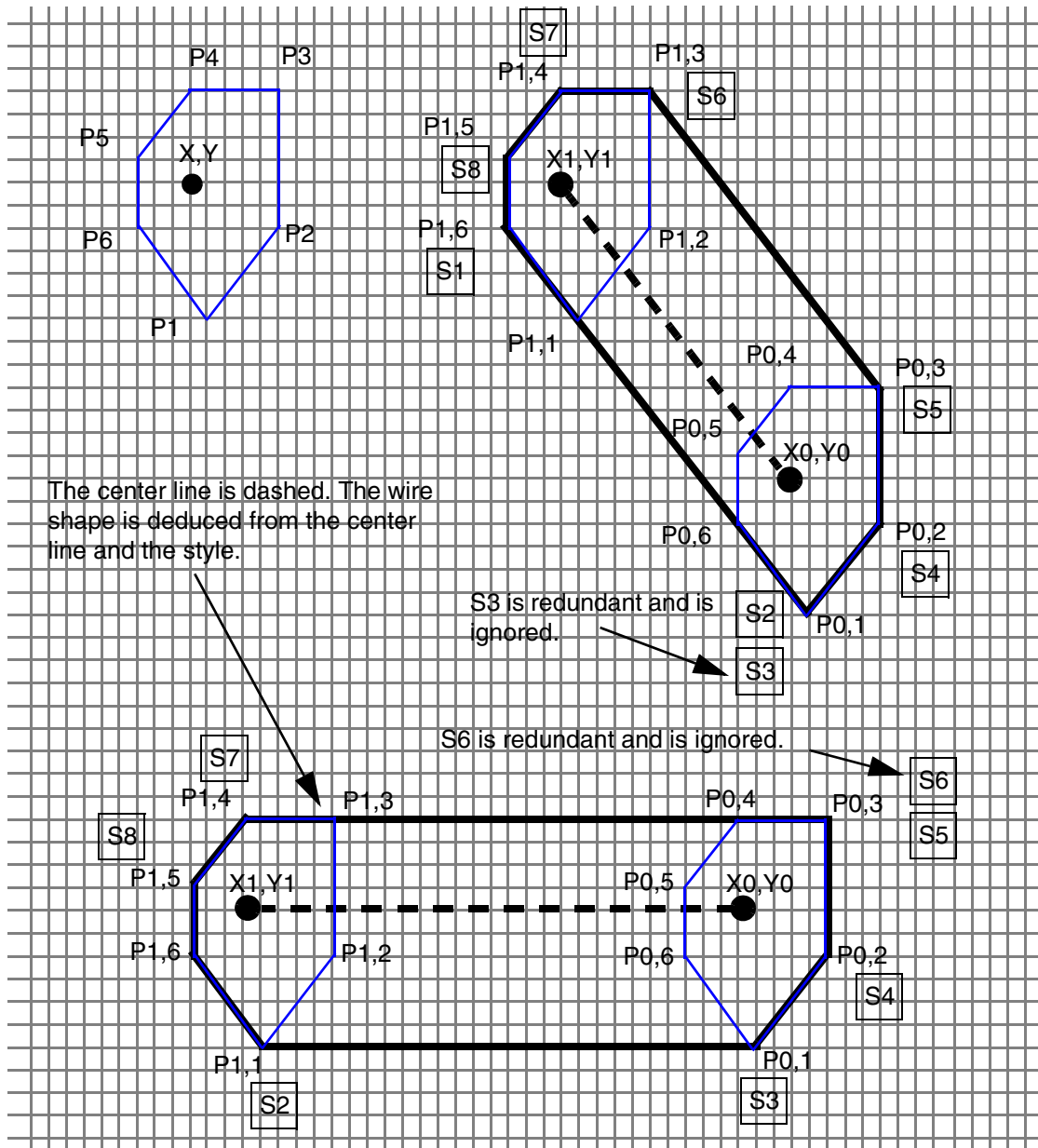
$S6$  = right-most point in (highest points in ( $P0,1$  through  $P0,n$   $P1,1$  through  $P1,n$ ))

$S7$  = left-most point in (highest points in ( $P0,1$  through  $P0,n$   $P1,1$  through  $P1,n$ ))

$S8$  = highest point in (left-most points in ( $P0,1$  through  $P0,n$   $P1,1$  through  $P1,n$ ))

When consecutive points are collinear, only one of them is relevant, and the resulting shape has less than eight sides, as shown in [Figure 4-15](#) on page 749. A more advanced algorithm can order the points and only have to check a subset of the points, depending on which endpoint was used, and whether the wire was horizontal, vertical, a 45-degree route, or a 135-degree route.

Figure 4-15



### Examples of X Routing with Styles

The following examples illustrate the use of styles for X routing. In two cases, there are examples of SPECIALNETS syntax and NETS syntax that result in the same geometry.

#### Example 1

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

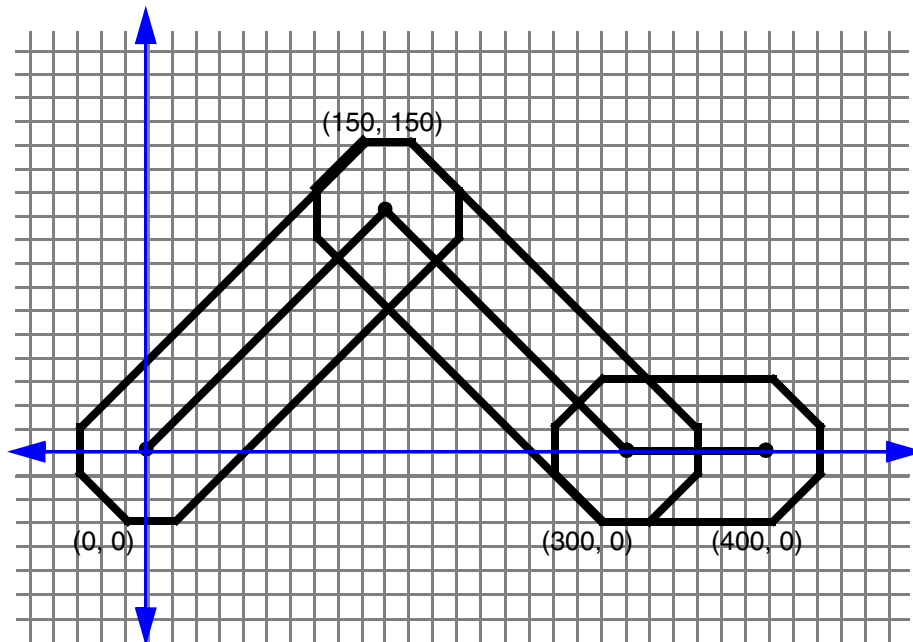
The following statements define an X wire with octagonal ends, as shown in [Figure 4-16](#) on page 750.

```
STYLES 1 ;
- STYLE 0 ( 30 10 ) ( 10 30 ) ( -10 30 ) ( -30 10 ) ( -30 -10 ) ( -10 -30 )
  ( 10 -30 ) ( 30 -10 ) ;          #An octagon.
END STYLES

SPECIALNETS 1 ;
- VSS ...
+ ROUTED metal3 50 + STYLE 0 ( 0 0 ) ( 150 150 ) ( 300 0 ) ( 400 0 ) ;
  #The style applies to all the segments until a NEW statement or ";"
  #at the end of the net.
END SPECIALNETS

NETS 1 ;
- mySignal ...
+ ROUTED metal3 STYLE 0 ( 0 0 ) ( 150 150 ) ( 300 0 ) ( 400 0 ) ;
  #The style applies to all the segments in the ROUTED statement
END NETS
```

**Figure 4-16**



### Example 2

## LEF/DEF 5.8 Language Reference

### DEF Syntax

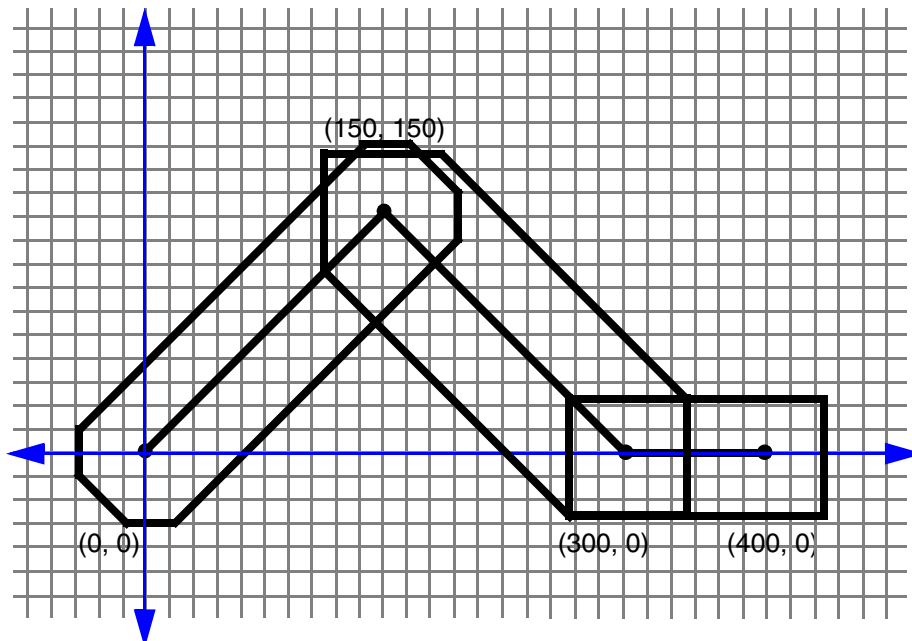
The following statements define the same X wire with mixed octagonal and manhattan styles, as shown in [Figure 4-17](#) on page 751.

```
STYLES 2 ;
- STYLE 0 ( 30 10 ) ( 10 30 ) ( -10 30 ) ( -30 10 ) ( -30 -10 ) ( -10 -30 )
  ( 10 -30 ) ( 30 -10 ) ; #An octagon
- STYLE 1 ( 25 25 ) ( -25 25 ) ( -25 -25 ) ( 25 -25 ) ; #A square
END STYLES

SPECIALNETS 1 ;
- POWER (* power)
  + ROUTED metal3 50 + STYLE 0 ( 0 0 ) ( 150 150 )
  NEW metal3 50 + STYLE 1 ( 150 150 ) ( 300 0 ) ( 400 0 ) ;
END SPECIALNETS

NETS 1 ;
- mySignal ...
  + ROUTED metal3 STYLE 0 ( 0 0 ) ( 150 150 )
  NEW metal3 STYLE 1 ( 150 150 ) ( 300 0 ) ( 400 0 ) ;
END NETS
```

**Figure 4-17**



**Note:** The square ends might be necessary for connecting to manhattan wires or pins, or in cases where vias have a manhattan shape even on X routing layers. In practice, the middle

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

wire probably would not use a simple square, such as `style2`; it would use a combination of an octagon and a square for the middle segment style, in order to smooth out the resulting outline at the (150,150) point.

#### Example 3

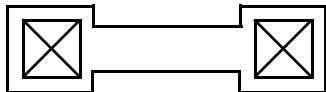
The following statements define a manhattan wire with a width of 70, as shown in [Figure 4-18](#) on page 752.

This example emphasizes that the style overrides the width of 100 units. In this case, the style polygon is a square 70 x 70 units wide, and the vias (`via12`) are 100 x 100 units wide. The application that creates the styles is responsible for meeting any particular width requirements. Normally, the resulting style-computed width is equal to or larger than the wire width given in the routing statement.

```
STYLES 1 ;
- STYLE 0 ( 35 35 ) ( -35 35 ) ( -35 -35 ) ( 35 -35 ) ;
END STYLES

SPECIALNETS 1 ;
- POWER ...
+ ROUTED metall 100 + STYLE 0 ( 0 0 ) via12 ( 600 * ) via12 ;
END SPECIALNETS
```

**Figure 4-18**



#### Example 4

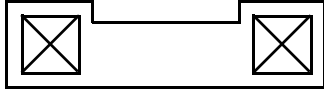
The following statements define a similar wire that is offset from the center, as shown in [Figure 4-19](#) on page 753. Similar to Example 3, the center line in both runs through the middle of the X in the vias.

```
STYLES 1 ;
- STYLE 0 ( 35 20 ) ( -35 20 ) ( -35 -50 ) ( 35 -50 ) ; #70 x 70 offset square
END STYLES

SPECIALNETS 1 ;
- POWER ...
+ ROUTED metall 100 + STYLE 0 ( 0 0 ) via12 ( 600 * ) via12 ;
END SPECIALNETS
```



**Figure 4-19**



### Example 5

The following statements define a wire that uses a “2-point line” style, as shown in [Figure 4-20](#) on page 753.

**Note:** This example shows the simplest style possible, which is a 2-point line. Generally, it would be easier to use a normal route without a style.

```
STYLES 1 ;  
    - STYLE 0 ( 0 -10 ) ( 0 10 ) ; #a vertical line  
END STYLES  
  
SPECIALNETS 1 ;  
    - POWER ...  
        + ROUTED metall 20 + STYLE 0 ( 0 0 ) ( 100 0 ) ;  
END SPECIALNETS
```

**Figure 4-20**

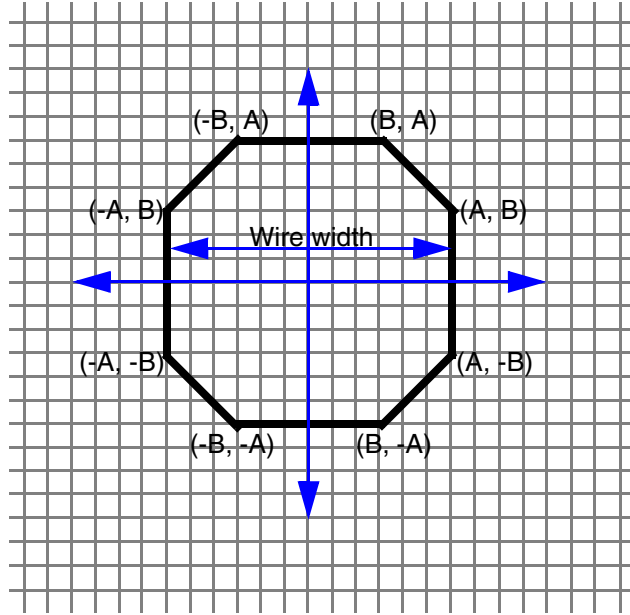


### ***45-Degree Routing Without Styles***

Because many applications only store the wire endpoints and the width of the wire, DEF includes a specific style default definition. If a style is not explicitly defined, the default style is implicitly included with any 45-degree routing segment. It is computed directly from the wire width and endpoints, at the expense of some loss in flexibility.

The default style is an octagon (shown in [Figure 4-21](#) on page 754) whose coordinates are computed from the wire width and the manufacturing grid.

**Figure 4-21**



The octagon is always symmetric about the x and y axis. The coordinates are computed to be exactly the same wire width as equivalent horizontal or vertical wire widths, and as close as possible for the diagonal widths (they are always slightly bigger because of rounding of irrational values), while forcing the coordinates to remain on the manufacturing grid. The wire width must be an even multiple of the manufacturing grid in order to keep A and B on the manufacturing grid.

Assume the following rules:

- $W$  = wire width
- $M$  = manufacturing grid (mgrid). This is derived from the LEF `MANUFACTURINGGRID` statement.
- $D$  = diagonal width
- ceiling = round up the result to the nearest integer

The octagon coordinates are computed as:

$$A = W/2$$

$$B = [\text{ceiling}(W/(\text{sqrt}(2) * M) * M) - A$$

The derivation of B can be understood as:

$$D = \text{sqrt}((A + B)^2 + (A + B)^2) \text{ or } D = \text{sqrt}(2) * (A + B)$$

## LEF/DEF 5.8 Language Reference

### DEF Syntax

The diagonal width (D) must be greater than or equal to the wire width (W), and B must be on the manufacturing grid, so D must be equal to W, which results in:

$$D/\sqrt{2} = A + B$$

$$B = D/\sqrt{2} - A \text{ or } W/\sqrt{2} - A$$

To force B to be on the manufacturing grid, and keep the diagonal width greater than or equal to the wire width:

$$B \text{ on mgrid} = \text{ceiling}(B / M) * M$$

Which results in the computation:

$$B = [\text{ceiling}(W/(\sqrt{2} * M) * M) - A$$

The following table lists examples coordinate computations:

**Table 4-1**

W = Width (μm)	M = mgrid (μm)	D = W/(sqrt(2)*M)	ceiling (D)	A (μm)	B (μm)	Diagonal width (μm)
1.0	0.005	141.42	142	0.5	0.21	1.0041
0.5	0.005	70.71	71	0.25	0.105	0.5020
0.15	0.005	21.21	22	0.075	0.035	0.1556
0.155*	0.005	21.92	22	0.0775*	0.0325*	0.1556
* A width of 0.155 is an odd multiple of the manufacturing grid and is not allowed because it would create coordinates for A and B that are off the manufacturing grid. It is shown for completeness to illustrate how the result is off grid.						

The default style only applies to 45-degree route segments; it does not apply to 90-degree route segments.

### Example 1

The following two routes produce identical routing shapes, as shown in [Figure 4-22](#) on page 756.

```
SPECIALNETS 1 ;  
    - POWER (* power)
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

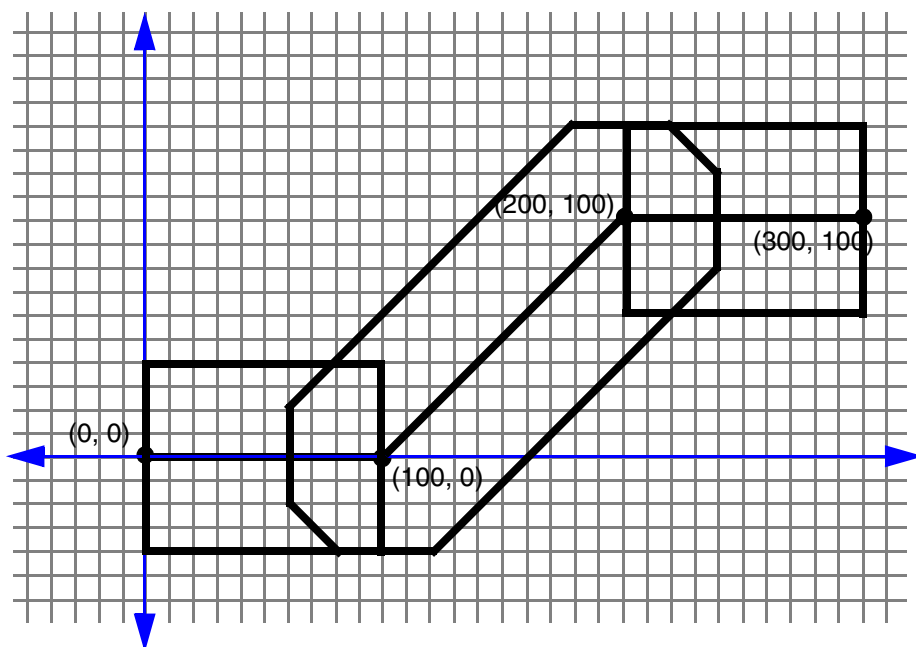
```

+ ROUTED metal3 80 ( 0 0 ) ( 100 0 ) ( 200 100 ) ( 300 100 ) ;
END SPECIALNETS

NETS 1 ;
- mySignal ... #mySignal uses the default routing rule width of 80
+ ROUTED metal3 ( 0 0 0 ) ( 100 0 0 ) ( 200 100 0 ) ( 300 100 0 ) ;
#The wire extension was set to 0 for every point. The wire extension
#is ignored for 45-degree route segments; the default octagon
#overrides it.
END NETS

```

**Figure 4-22**



### Example 2

The following regular route definition, using the traditional default wire extension of  $1/2 * width$  for the first and last 90-degree endpoints, produces the route shown in [Figure 4-23](#) on page 757.

```

SPECIALNETS 1;
- POWER (* power) #The half-width extensions are given for the first and last
+ ROUTED metal3 80 ( 0 0 40 ) ( 100 0 ) ( 200 100 ) ( 300 100 40 ) ;
#The default extension is 0 for SPECIALNETS, so it is not given for
#two middle points.
END SPECIALNETS

```

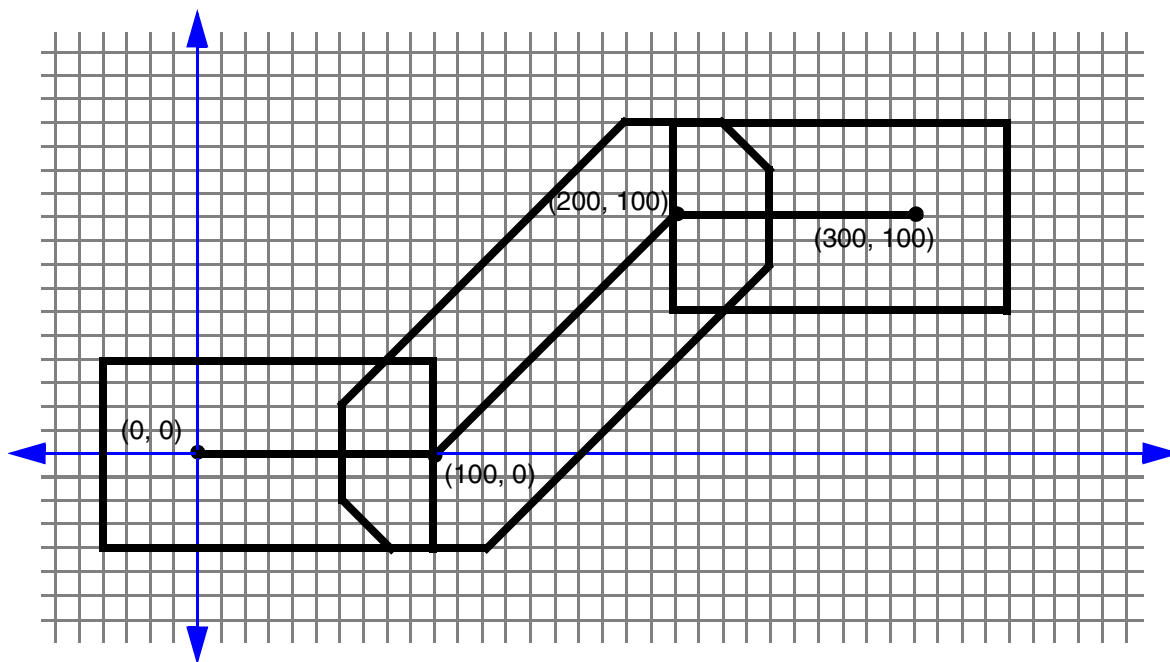
## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
NETS 1 ;
- mySignal ... #mySignal uses the default routing rule with width of 80
+ ROUTED metal3 ( 0 0 ) ( 100 0 0 ) ( 200 100 0 ) ( 300 100 ) ;
  #The default extension is half the width for NETS, so it is not
  #included for the first and last end-points.
END NETS
```

**Figure 4-23**



### Example 3

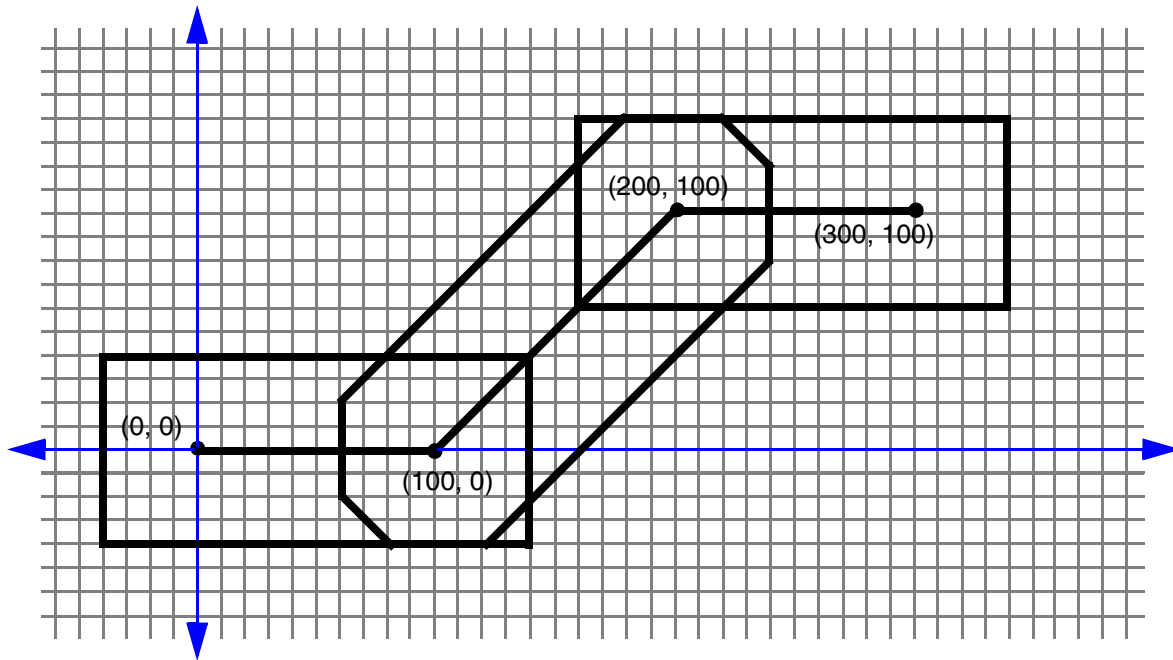
The following definition, using the traditional default wire extension of  $1/2 * width$  for all of the points, produces the route in [Figure 4-24](#) on page 758.

```
SPECIALNETS 1 ;
- POWER (* power)    #The half-width extensions are given explicitly
+ ROUTED metal3 80 ( 0 0 40 ) ( 100 40 ) ( 200 100 40 ) ( 300 100 40 ) ;
END SPECIALNETS

NETS 1 ;
- mySignal ... #mySignal uses the default routing rule width of 80
+ ROUTED metal3 ( 0 0 ) ( 100 0 ) ( 200 100 ) ( 300 100 ) ;
  #All points use the implicit default  $1/2 * width$  wire extensions.
```

END NETS

**Figure 4-24**



## Technology

```
[TECHNOLOGY technologyName ;]
```

Specifies a technology name for the design in the database. In case of a conflict, the previous name remains in effect.

## Tracks

```
[TRACKS  
  [{X | Y} start DO numtracks STEP space  
    [MASK maskNum [SAMEMASK]]  
    [LAYER layerName ...]  
  ;] ...]
```

Defines the routing grid for a standard cell-based design. Typically, the routing grid is generated when the floorplan is initialized. The first track is located at an offset from the placement grid set by the `OFFSET` value for the layer in the LEF file. The track spacing is the `PITCH` value for the layer defined in LEF.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

`DO numTracks`

Specifies the number of tracks to create for the grid. You cannot specify 0 *numtracks*.

`LAYER layerName`

Specifies the routing layer used for the tracks. You can specify more than one layer.

`MASK maskNum [SAMEMASK]`

Specifies which mask for double or triple patterning lithography to use for the first routing track. The *maskNum* variable must be a positive integer - most applications support values of 1, 2, or 3 only. The layer(s) must be declared as two or three mask layers in the LAYER section.

By default, the tracks cycle through all the masks. So you will see alternating masks, such as, 1, 2, 1, 2, etc. for a two-mask layer and 1, 2, 3, 1, 2, 3, etc., for a three-mask layer.

If the `SAMEMASK` keyword is specified, then all the routing tracks are the same mask as the first track mask. Tracks without any defined mask do not have a mask set (that is, they are uncolored).

See [Example 4-35](#) on page 759.

`STEP space`

Specifies the spacing between the tracks.

`{X | Y} start`

Specifies the location and direction of the first track defined. *X* indicates vertical lines; *Y* indicates horizontal lines. *start* is the *X* or *Y* coordinate of the first line. For example, `X 3000` creates a set of vertical lines, with the first line going through (3000 0).

### Example 4-35 Mask Assignments for Routing Tracks

- The following example shows a three-mask layer `M1` that has a first track of mask 2 with cycling mask numbers after that:

```
TRACKS X 0 DO 20 STEP 5 MASK 2 LAYER M1 ;
```

This statement will result in `M1` vertical tracks at *X* coordinates with mask assignments of 0 (mask 2), 5 (mask 3), 10 (mask 1), 15 (mask 2), etc., for 20 tracks.

- The following statement will result in `M1` vertical tracks at *X* coordinates with mask assignments of 0 (mask 1), 10 (mask 1), 20 (mask 1), 30 (mask 1), etc., for 20 tracks.

```
TRACKS X 0 DO 20 STEP 10 MASK 1 SAMEMASK LAYER M1 ;
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

### Units

```
[UNITS DISTANCE MICRONS dbuPerMicron ;]
```

Specifies the database units per micron (*dbuPerMicron*) to convert DEF distance units into microns.

LEF supports values of 100, 200, 400, 800, 1000, 2000, 4000, 8000, 10,000, and 20,000 for the LEF *dbuPerMicron*. The LEF *dbuPerMicron* must be greater than or equal to the DEF *dbuPerMicron*, otherwise you can get round-off errors. The LEF convert factor must also be an integer multiple of the DEF convert factor so no round-off of DEF database unit values is required (e.g., a LEF convert factor of 1000 allows DEF convert factors of 100, 200, 1000, but not 400, 800).

The following table shows the valid pairings of the LEF *dbuPerMicron* and the corresponding legal DEF *dbuPerMicron* values.

LEF <i>dbuPerMicron</i>	Legal DEF <i>dbuPerMicron</i>
100	100
200	100, 200
400	100, 200, 400
800	100, 200, 400, 800
1000	100, 200, 1000
2000	100, 200, 400, 1000, 2000
4000	100, 200, 400, 800, 1000, 2000, 4000
8000	100, 200, 400, 800, 1000, 2000, 4000, 8000
10,000	100, 200, 400, 1000, 2000, 10,000
20,000	100, 200, 400, 800, 1000, 2000, 4000, 10,000, 20,000

### Using DEF Units

The following table shows examples of how DEF units are used:

Units	DEF Units	DEF Value Example	Real Value
Time	.001 nanosecond	1500	1.5 nanoseconds



## LEF/DEF 5.8 Language Reference

### DEF Syntax

Units	DEF Units	DEF Value Example	Real Value
Capacitance	.000001 picofarad	1,500,000	1.5 picofarads
Resistance	.0001 ohm	15,000	1.5 ohms
Power	.0001 milliwatt	15,000	1.5 milliwatts
Current	.0001 milliamp	15,000	1.5 milliamps
Voltage	.001 volt	1500	1.5 volts

The DEF reader assumes divisor factors such that DEF data is given in the database units shown below.

Unit	Database Precision
1 nanosecond	= 1000 DBUs
1 picofarad	= 1,000,000 DBUs
1 ohm	= 10,000 DBUs
1 milliwatt	= 10,000 DBUs
1 milliamperere	= 10,000 DBUs
1 volt	= 1000 DBUs

## Version

```
[VERSION versionNumber ;]
```

Specifies which version of the DEF syntax is being used.

**Note:** The `VERSION` statement is not required in a DEF file; however, you should specify it, because it prevents syntax errors caused by the inadvertent use of new versions of DEF with older tools that do not support the new version syntax.

## Vias

```
[VIAS numVias ;  
  [- viaName  
    [+ VIARULE viaRuleName  
      + CUTSIZE xSize ySize  
      + LAYERS botmetalLayer cutLayer topMetalLayer  
      + CUTSPACING xCutSpacing yCutSpacing  
      + ENCLOSURE xBotEnc yBotEnc xTopEnc yTopEnc
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
[+ ROWCOL numCutRows NumCutCols]  
[+ ORIGIN xOffset yOffset]  
[+ OFFSET xBotOffset yBotOffset xTopOffset yTopOffset]  
[+ PATTERN cutPattern ]  
| [ + RECT layerName [+ MASK maskNum] pt pt  
    | + POLYGON layerName [+ MASK maskNum] pt pt pt] ...]  
;] ...  
END VIAS]
```

Lists the names and geometry definitions of all vias in the design. Two types of vias can be listed: fixed vias and generated vias. All vias consist of shapes on three layers: a cut layer and two routing (or masterslice) layers that connect through that cut layer.

A fixed via is defined using rectangles or polygons, and does not use a `VIARULE`. The fixed via name must mean the same via in all associated LEF and DEF files.

A generated via is defined using `VIARULE` parameters to indicate that it was derived from a `VIARULE GENERATE` statement. For a generated via, the via name is only used locally inside this DEF file. The geometry and parameters are maintained, but the name can be freely changed by applications that use this via when writing out LEF and DEF files to avoid possible via name collisions with other DEF files.

`CUTSIZE` *xSize* *ySize*

Specifies the required width (*xSize*) and height (*ySize*) of the cut layer rectangles.  
*Type*: Integer, specified in DEF database units

`CUTSPACING` *xCutSpacing* *yCutSpacing*

Specifies the required x and y spacing between cuts. The spacing is measured from one cut edge to the next cut edge.  
*Type*: Integer, specified in DEF database units

`ENCLOSURE` *xBotEnc* *yBotEnc* *xTopEnc* *yTopEnc*

Specifies the required x and y enclosure values for the bottom and top metal layers. The enclosure measures the distance from the cut array edge to the metal edge that encloses the cut array (see [Figure 4-25](#) on page 766).  
*Type*: Integer, specified in DEF database units

`LAYERS` *botMetalLayer* *cutLayer* *TopMetalLayer*

Specifies the required names of the bottom routing/masterslice layer, cut layer, and top routing/masterslice layer. These layer names must be previously defined in layer definitions, and must match the layer names defined in the specified LEF *viaRuleName*.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

#### `MASK maskNum`

Specifies which mask for double or triple patterning lithography is to be applied to the shapes defined in `RECT` or `POLYGON` statements of the via master. The *maskNum* variable must be a positive integer - most applications support values of 1, 2, or 3 only. For a fixed via made up of `RECT`/`POLYGON` statements, the cut-shapes must be either colored or uncolored. It is an error to have partially colored cuts for one via. Uncolored cut shapes should be automatically colored by the reader if the layer is a multi-mask layer.

The metal shapes of the via-master do not need colors because the via-instance has the mask color. Some readers may, however, color them for internal consistency (see [Example 4-38](#) on page 770). So a writer may write out `MASK 1` for metal shapes even if they were read in with no mask value.

For uncolored fixed vias, or parameterized vias (with `+ VIARULE ...`), the mask of the cuts are pre-defined as an alternating pattern starting with `MASK 1` at the bottom-left. The mask cycles, from left-to-right and bottom-to-top, for the cuts are as shown in [Figure 4-30](#) on page 771.

#### `numVias`

Specifies the number of vias listed in the `VIA` statement.

#### `OFFSET xBotOffset yBotOffset xTopOffset yTopOffset`

Specifies the x and y offset for the bottom and top metal layers. These values allow each metal layer to be offset independently.

By default, the 0, 0 origin of the via is the center of the cut array, and the enclosing metal rectangles. After the non-shifted via is computed, the metal layer rectangles are shifted by adding the appropriate values—the *x/yBotOffset* values to the metal layer below the cut layer, and the *x/yTopOffset* values to the metal layer above the cut layer.

These offset values are in addition to any offset caused by the `ORIGIN` values. For an example and illustration of this syntax, see [Example 4-36](#) on page 766.

*Default:* 0, for all values

*Type:* Integer, in DEF database units

#### `ORIGIN xOffset yOffset`

Specifies the x and y offset for all of the via shapes. By default, the 0, 0 origin of the via is the center of the cut array, and the enclosing metal rectangles. After the non-shifted via is computed, all cut and metal rectangles are shifted by adding these values. For an example and illustration of this syntax, see [Example 4-36](#) on page 766.

*Default:* 0, for both values

*Type:* Integer, in DEF database units

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

PATTERN *cutPattern*

Specifies the cut pattern encoded as an ASCII string. This parameter is only required when some of the cuts are missing from the array of cuts, and defaults to “all cuts are present,” if not specified.

For information on and examples of via cut patterns, see [“Creating Via Cut Patterns”](#) on page 771.

The *cutPattern* syntax is defined as follows:

*numRows\_rowDefinition*

*[\_numRows\_rowDefinition]* ...

*numRows*

Specifies a hexadecimal number that indicates how many times to repeat the following row definition. This number can be more than one digit.

*rowDefinition*

Defines one row of cuts, from left to right.

The *rowDefinition* syntax is defined as follows:

{*[RrepeatNumber]hexDigitCutPattern*} ...

*hexDigitCutPattern*

Specifies a single hexadecimal digit that encodes a 4-bit binary value in which 1 indicates a cut is present, and 0 indicates a cut is not present.

*repeatNumber*

Specifies a single hexadecimal digit that indicates how many times to repeat *hexDigitCutPattern*.

For parameterized vias (with + VIARULE ...), the *cutPattern* has an optional suffix added to allow three types of mask color patterns. The default mask color pattern (no suffix) is a checker-board (see [Figure 4-28](#) on page 768). The other two patterns supported are alternating rows, and alternating columns (see [Figure 4-29](#) on page 769).

The optional suffixes are:

<*cut\_pattern*>\_MR alternating rows

<*cut\_pattern*>\_MC alternating columns

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

POLYGON *layerName pt pt pt*

Defines the via geometry for the specified layer. You must specify at least three points to generate the polygon, and the edges must be parallel to the x axis, the y axis, or at a 45-degree angle.

*Type:* (*x y*) Integer, specified in database units

Each POLYGON statement defines a polygon generated by connecting each successive point, and then the first and last points. The *pt* syntax corresponds to a coordinate pair, such as (*x y*). Specify an asterisk (\*) to repeat the same value as the previous *x* or *y* value from the last point.

For example, + POLYGON ( 0 0 ) ( 10 10 ) ( 10 0 ) creates a triangle shape.

All vias consist of shapes on three layers: a cut layer and two routing (or masterslice) layers that connect through that cut layer. There should be at least one RECT or POLYGON on each of the three layers.

RECT *layerName pt pt*

Defines the via geometry for the specified layer. The points are specified with respect to the via origin. In most cases, the via origin is the center of the via bounding box. All geometries for the via, including the cut layers, are output by the DEF writer.

*Type:* (*x y*) Integer, specified in database units

All vias consist of shapes on three layers: a cut layer and two routing (or masterslice) layers that connect through that cut layer. There should be at least one RECT or POLYGON on each of the three layers.

ROWCOL *numCutRows numCutCols*

Specifies the number of cut rows and columns that make up the cut array.

*Default:* 1, for both values

*Type:* Positive integer, for both values

*viaName*

Specifies the via name. Via names are generated by appending a number after the rule name. Vias are numbered in the order in which they are created.

VIARULE *viaRuleName*

Specifies the name of the LEF VIARULE that produced this via. This name must be specified before you define any of the other parameters, and must refer to a VIARULE GENERATE via rule. It cannot refer to a VIARULE without a GENERATE keyword.

Specifying the reserved via rule name of DEFAULT indicates that the via should use the previously defined VIARULE GENERATE rule with the DEFAULT keyword that exists for

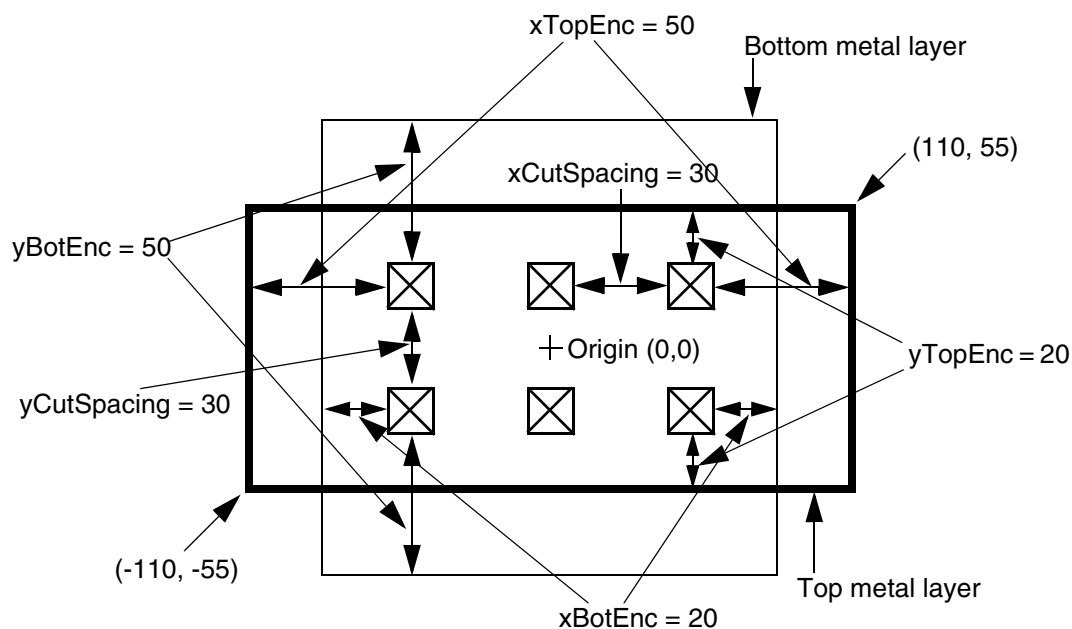
this routing-cut-routing (or masterslice-cut-masterslice) layer combination. This makes it possible for a tool that does not use the LEF `VIARULE` technology section to still generate DEF generated-via parameters by using the default rule.

### Example 4-36 Via Rules

The following via rule describes a non-shifted via (that is, a via with no `OFFSET` or `ORIGIN` parameters). There are two rows and three columns of via cuts. [Figure 4-25](#) on page 766 illustrates this via rule.

```
- myUnshiftedVia
+ VIARULE myViaRule
+ CUTSIZE 20 20          #xCutSize yCutSize
+ LAYERS metal1 cut12 metal2
+ CUTSPACING 30 30       #xCutSpacing yCutSpacing
+ ENCLOSURE 20 50 50 20  #xBotEnc yBotEnc xTopEnc yTopEnc
+ ROWCOL 2 3 ;
```

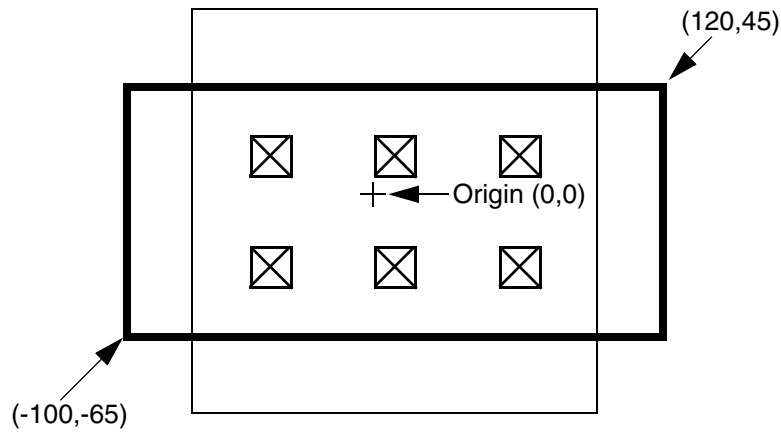
**Figure 4-25 Via Rule**



The same via rule with the following `ORIGIN` parameter shifts all of the metal and cut rectangles by 10 in the x direction, and by -10 in the y direction (see [Figure 4-26](#) on page 767):

```
+ ORIGIN 10 -10
```

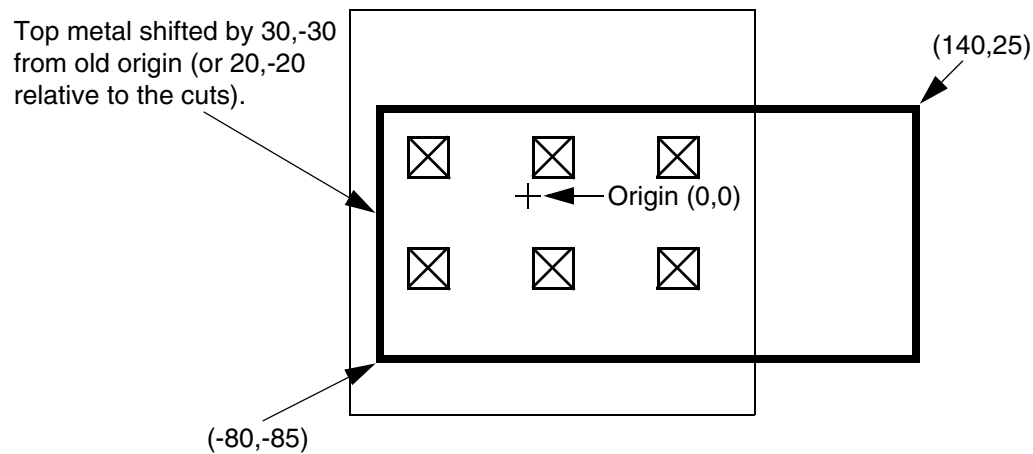
**Figure 4-26 Via Rule With Origin**



If the same via rule contains the following `ORIGIN` and `OFFSET` parameters, all of the rectangles shift by 10, -10. In addition, the top layer metal rectangle shifts by 20, -20, which means that the top metal shifts by a total of 30, -30.

```
+ ORIGIN 10 -10
+ OFFSET 0 0 20 -20
```

**Figure 4-27 Via Rule With Origin and Offset**



### Example 4-37 Multi-Mask Patterns for Parameterized Vias with Via Rule

The following via rule describes a via cut mask pattern for a parameterized via:

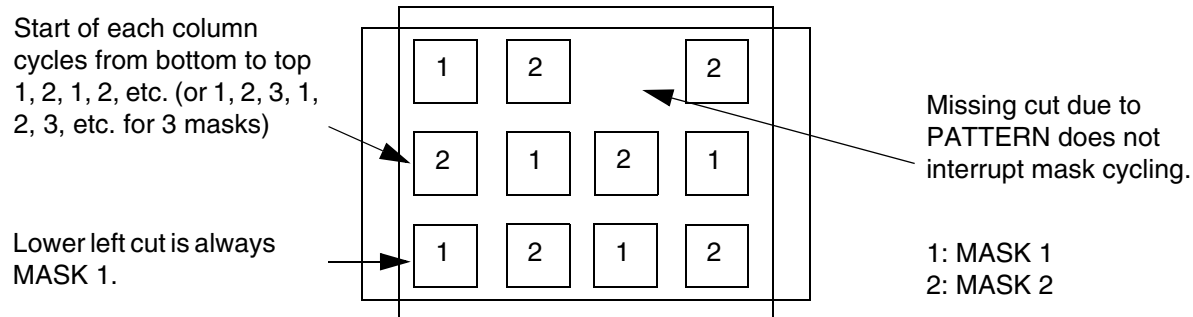
```
- myParamVia1
  + VIARULE myGenVia1      + CUTSIZE 40 40
  + LAYERS M1 VIA1 M2      + CUTSPACING 40 40
```

## LEF/DEF 5.8 Language Reference

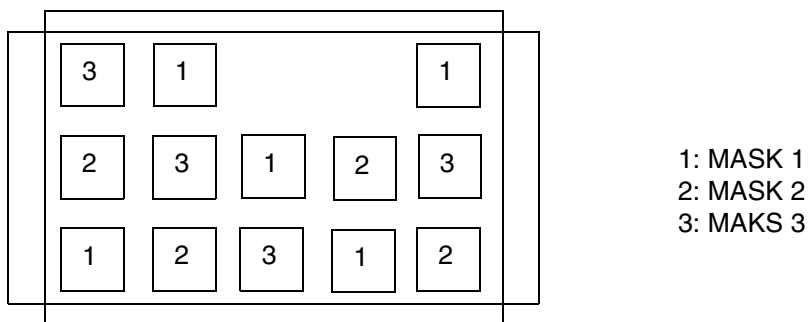
### DEF Syntax

```
+ ENCLOSURE 40 0 0 40      + ROWCOL 3 4
+ PATTERN 2_F_1_D ;          #1 cut in top row is missing
```

**Figure 4-28 Multi-Mask Patterns for Parameterized Vias**



Assuming VIA1 is a 2-mask cut-layer, then the mask for each cut-shape is shown above. The default checker-board pattern is: the bottom left cut starts at 1, and goes left to right, 1, 2, 1, 2. The next row up starts at 2, and goes 2, 1, 2, 1, etc. Missing cuts due to PATTERN do not change the mask assignments.



Example of a check-board parameterized via cut-mask pattern for a 3-mask layer with 2 missing cuts due to PATTERN statement.

- The following examples show a parameterized via with cut-mask patterns for a 3-mask layer and 2-mask layer using `_MC` and `_MR` suffixes:



## LEF/DEF 5.8 Language Reference

### DEF Syntax

**Figure 4-29 Multi-Mask Patterns for Parameterized Vias using Suffixes**

On a 3 mask cut layer

Mask colors for a parameterized via like:

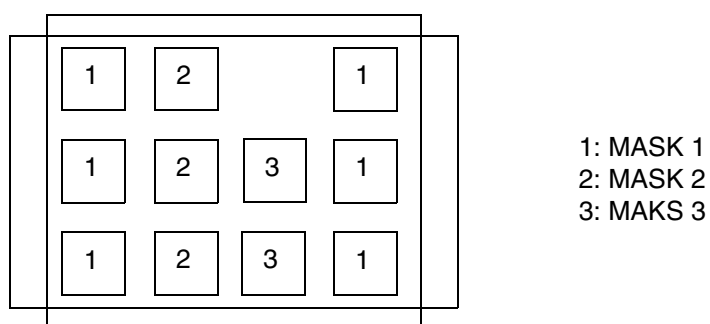
- myParamVia1

+ VIARULE myGenVia1      + CUTSIZE 40 40

+ LAYERS M1 VIA1 M2      + CUTSPACING 40 40

+ ENCLOSURE 40 0 0 40    + ROWCOL 3 4

+ PATTERN 2\_F\_1\_D\_MC;    #alternating mask color columns due to \_MC suffix



On a 2 mask cut layer

Mask colors for a parameterized via like:

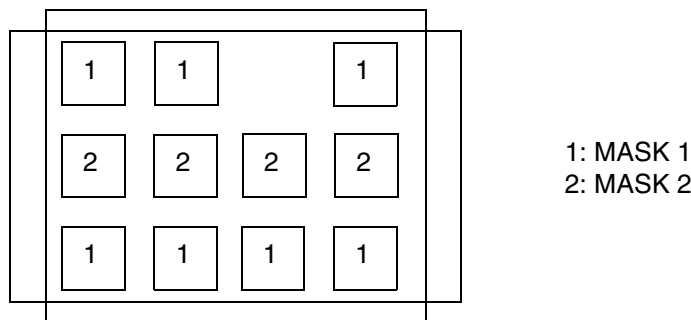
- myParamVia1

+ VIARULE myGenVia1      + CUTSIZE 40 40

+ LAYERS M1 VIA1 M2      + CUTSPACING 40 40

+ ENCLOSURE 40 0 0 40    + ROWCOL 3 4

+ PATTERN 2\_F\_1\_D\_MR;    #alternating mask color rows due to \_MR suffix



For a fixed via specified using `RECT` or `POLYGON` statements, the cut shapes must either be all colored or uncolored. If the cuts are not colored, they will be automatically colored in a checkerboard pattern as shown in [Figure 4-28](#) on page 768. Each via cut with the same lower-left Y value is considered as one row, and each via in one row is a new column. For common "array" style vias with no missing cuts, this coloring is a good one. For vias that do

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

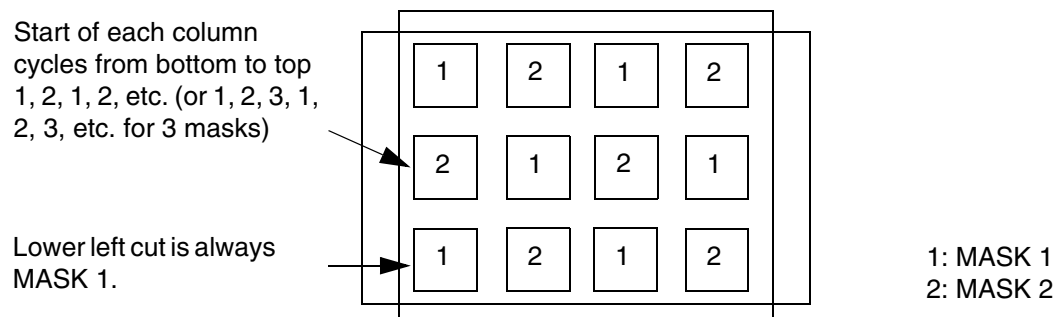
not have a row and column structure or are missing cuts, then this coloring may not be good (see [Figure 4-30](#) on page 771). If the metal layers are not colored, some applications will color them to mask 1 for internal consistency, even though the via master metal shape colors are not really used by LEF or DEF via instances.

#### Example 4-38 Multi-Mask Patterns for Fixed Via

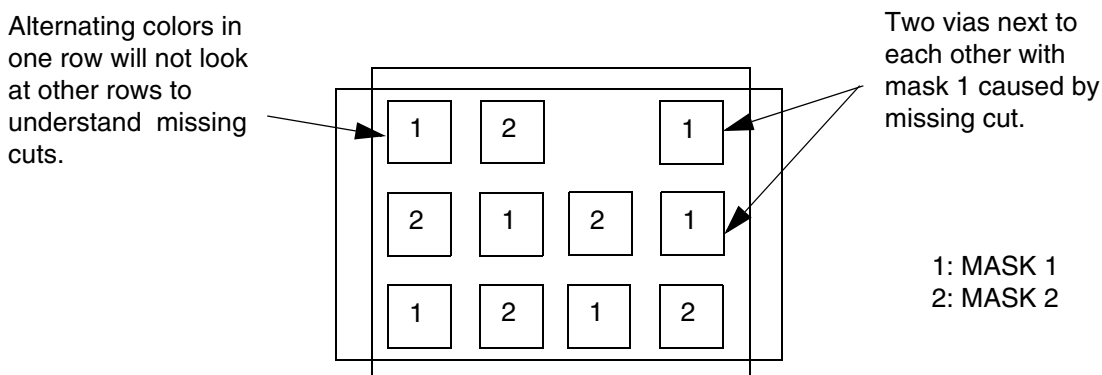
The following example shows a fixed-via with pre-colored cut shapes:

```
- myVia1
  + RECT M1 ( -40 -20 ) ( 120 20 )           #no mask, some readers set to 1
  + RECT VIA1 + MASK 1 ( -20 -20 ) ( 20 20 )   #first cut on mask 1
  + RECT VIA1 + MASK 2 ( 60 -20 ) ( 100 20 )   #second cut on mask 2
  + RECT ( -20 -40 ) ( 100 40 )               #no mask, some readers set to 1
```

**Figure 4-30 Multi-Mask Patterns for Fixed Via**



Uncolored fixed vias (from RECT/POLYGON statements) will get similar coloring as a parameterized via. Each cut with the same Y value is one row. The cuts inside one row alternate. This works well for cut-arrays without missing cuts as shown here.



Unlike parameterized vias, there is no real row/column structure required in a fixed via, so the automatic coloring will not work well for a fixed via with missing cuts (or cuts that are not aligned). This type of via must be pre-colored to be useful.

See the [Fills](#), [Nets](#), and [Special Nets](#) routing statements to see how a via instance uses these via-master mask values.

### ***Creating Via Cut Patterns***

Via cuts are defined as a series of rows, starting at the bottom, left corner. Each row definition defines one row of cuts, from left to right, and rows are numbered from bottom to top.

The `PATTERN` syntax that defines rows uses the `ROWCOL` parameters to specify the cut array. If the row has more bits than the `numCutCols` value in the `ROWCOL` parameter for this via,

the last bits are ignored. The number of rows defined must equal the *numCutRows* value in the ROWCOL parameter.

Figure 4-31 on page 772 illustrates the following via cut pattern syntax:

```
- myVia
  + VIARULE myViaRule
  ...
  + ROWCOL 5 5
  + PATTERN 2_F0_2_F8_1_78 ;]
```

The last three bits of F0, F8, and 78 are ignored because only five bits are allowed in a row. Therefore, the following PATTERN syntax gives the identical pattern:

```
+ PATTERN 2_F7_2_FF_1_7F
```

**Figure 4-31**

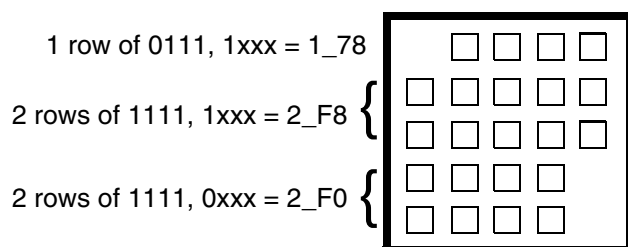


Figure 4-32 on page 773 illustrates the following via cut pattern syntax:

```
- myVia
  + VIARULE myViaRule
  ...
  + ROWCOL 5 14
  + PATTERN 2_FFE0_3_R4F ;
```

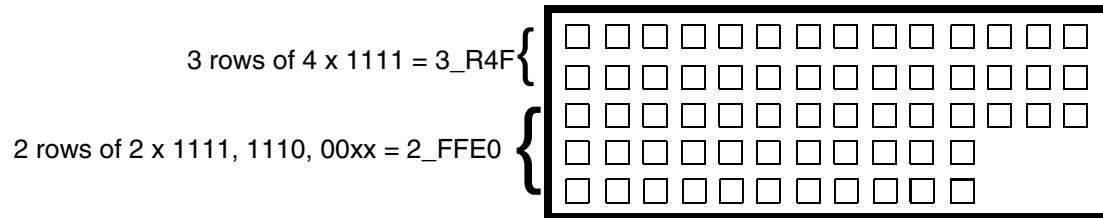
The R4F value indicates a repeat of four Fs. The last two bits of each row definition are ignored because only 14 bits allowed in each row.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

**Figure 4-32**



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

---

## Examples

---

This appendix contains information about the following topics.

- [LEF](#) on page 775
- [DEF](#) on page 786
- [Scan Chain Synthesis Example](#) on page 791

## LEF

```
VERSION 5.7 ;
```

```
# DEMO4 CHIP - 1280 ARRAY
```

```

&alias &&area = (73600,74400) (238240,236400) &endalias
&alias &&core = (85080,85500) (226760,224700) &endalias
&alias &&m2stripes = srout stripe net vss net vdd layer m2
width
320 count 2 pattern 87900 4200 218100
area &&area core &&core &endalias
&alias &&m3stripes = srout stripe net vss net vdd layer m3
width
600 count 2 pattern 89840 6720 217520
area &&area core &&core &endalias
&alias &&powerfollowpins = srout follow net vss net vdd layer
m1 width 560
area &&area core &&core &endalias
&alias &&powerrepair = srout repair net vss net vdd area
&&area core &&core &endalias
# PLACEMENT SITE SECTION
SITE CORE1 SIZE 67.2 BY 6 ; # GCD of all Y sizes of Macros END
CORE1
SITE IOX SIZE 37.8 BY 444 ; # 151.2 / 4 = 37.8 , 4 sites per pad END IOX
SITE IOY SIZE 436.8 BY 30 ; # 150 / 5 = 30 , 5 sites per pad END IOY
SITE SQUAREBLOCK SIZE 268.8 BY 252 ; END SQUAREBLOCK
SITE I2BLOCK SIZE 672 BY 504 ; END I2BLOCK
SITE LBLOCK SIZE 201.6 BY 168 ; END LBLOCK
SITE CORNER SIZE 436.8 BY 444 ; END CORNER
LAYER POLYS TYPE MASTERSLICE ; END POLYS

```

## LEF/DEF 5.8 Language Reference Examples

---

```
LAYER PW TYPE MASTERSLICE ; END PW
LAYER NW TYPE MASTERSLICE ; END NW
LAYER PD TYPE MASTERSLICE ; END PD
LAYER ND TYPE MASTERSLICE ; END ND
LAYER CUT01 TYPE CUT ; END CUT01
LAYER M1 TYPE ROUTING ; DIRECTION VERTICAL ; PITCH 5.6 ; WIDTH2.6 ;
    SPACING 1.5 ;
END M1
LAYER CUT12 TYPE CUT ; END CUT12
LAYER M2 TYPE ROUTING ; DIRECTION HORIZONTAL ; PITCH 6.0 ;
    WIDTH 3.2 ;SPACING 1.6 ;
END M2
LAYER CUT23 TYPE CUT ; END CUT23
LAYER M3 TYPE ROUTING ; DIRECTION VERTICAL ; PITCH 5.6 ; WIDTH 3.6;
    SPACING 1.6 ;
END M3
LAYER OVERLAP TYPE OVERLAP ; END OVERLAP
VIA C2PW DEFAULT LAYER PW ; RECT -2.0 -2.0 2.0 2.0 ;
    LAYER CUT01 ; RECT -0.6 -0.6 0.6 0.6 ;
    LAYER M1 ; RECT -2.0 -2.0 2.0 2.0 ;
END C2PW
VIA C2NW DEFAULT LAYER NW ; RECT -2.0 -2.0 2.0 2.0 ;
    LAYER CUT01 ; RECT -0.6 -0.6 0.6 0.6 ;
    LAYER M1 ; RECT -2.0 -2.0 2.0 2.0 ;
END C2NW
VIA C2PD DEFAULT LAYER PD ; RECT -2.0 -2.0 2.0 2.0 ;
    LAYER CUT01 ; RECT -0.6 -0.6 0.6 0.6 ;
    LAYER M1 ; RECT -2.0 -2.0 2.0 2.0 ;
END C2PD
VIA C2ND DEFAULT LAYER ND ; RECT -2.0 -2.0 2.0 2.0 ;
    LAYER CUT01 ; RECT -0.6 -0.6 0.6 0.6 ;
    LAYER M1 ; RECT -2.0 -2.0 2.0 2.0 ;
END C2ND
VIA C2POLY DEFAULT LAYER POLYS ; RECT -2.0 -2.0 2.0 2.0 ;
    LAYER CUT01 ; RECT -0.6 -0.6 0.6 0.6 ;
    LAYER M1 ; RECT -2.0 -2.0 2.0 2.0 ;
END C2POLY
VIA VIA12 DEFAULT LAYER M1 ; RECT -2.0 -2.0 2.0 2.0 ;
    LAYER CUT12 ; RECT -0.7 -0.7 0.7 0.7 ;
    LAYER M2 ; RECT -2.0 -2.0 2.0 2.0 ;
END VIA12
VIA VIA23 DEFAULT LAYER M3 ; RECT -2.0 -2.0 2.0 2.0 ;
    LAYER CUT23 ; RECT -0.8 -0.8 0.8 0.8 ;
    LAYER M2 ; RECT -2.0 -2.0 2.0 2.0 ;
END VIA23
    SPACING SAMENET CUT01 CUT12 4.0 ;
    SAMENET CUT12 CUT23 4.0 ;
END SPACING
VIA VIACENTER12 LAYER M1 ; RECT -4.6 -2.2 4.6 2.2 ;
    LAYER CUT12 ; RECT -3.1 -0.8 -1.9 0.8 ; RECT 1.9 -0.8 3.1 0.8 ;
    LAYER M2 ; RECT -4.4 -2.0 4.4 2.0 ;
```



## LEF/DEF 5.8 Language Reference Examples

---

```
END VIACENTER12
VIA VIATOP12 LAYER M1 ; RECT -2.2 -2.2 2.2 8.2 ;
    LAYER CUT12 ; RECT -0.8 5.2 0.8 6.8 ;
    LAYER M2 ; RECT -2.2 -2.2 2.2 8.2 ;
END VIATOP12
VIA VIABOTTOM12 LAYER M1 ; RECT -2.2 -8.2 2.2 2.2 ;
    LAYER CUT12 ; RECT -0.8 -6.8 0.8 -5.2 ;
    LAYER M2 ; RECT -2.2 -8.2 2.2 2.2 ;
END VIABOTTOM12
VIA VIALEFT12 LAYER M1 ; RECT -7.8 -2.2 2.2 2.2 ;
    LAYER CUT12 ; RECT -6.4 -0.8 -4.8 0.8 ;
    LAYER M2 ; RECT -7.8 -2.2 2.2 2.2 ;
END VIALEFT12
VIA VIARIGHT12 LAYER M1 ; RECT -2.2 -2.2 7.8 2.2 ;
    LAYER CUT12 ; RECT 4.8 -0.8 6.4 0.8 ;
    LAYER M2 ; RECT -2.2 -2.2 7.8 2.2 ;
END VIARIGHT12
VIA VIABIGPOWER12 LAYER M1 ; RECT -21.0 -21.0 21.0 21.0 ;
    LAYER CUT12 ; RECT -2.4 -0.8 2.4 0.8 ;
        RECT -19.0 -19.0 -14.2 -17.4 ; RECT -19.0 17.4 -14.2
        19.0 ;
        RECT 14.2 -19.0 19.0 -17.4 ; RECT 14.2 17.4 19.0 19.0 ;
        RECT -19.0 -0.8 -14.2 0.8 ; RECT -2.4 -19.0 2.4 -17.4 ;
        RECT 14.2 -0.8 19.0 0.8 ; RECT -2.4 17.4 2.4 19.0 ;
    LAYER M2 ; RECT -21.0 -21.0 21.0 21.0 ;
END VIABIGPOWER12
VIARULE VIALIST12 LAYER M1 ; DIRECTION VERTICAL ; WIDTH 9.0 TO
    9.6 ;
    LAYER M2 ; DIRECTION HORIZONTAL ; WIDTH 3.0 TO 3.0 ;
        VIA VIACENTER12 ; VIA VIATOP12 ; VIA VIABOTTOM12 ;
        VIA VIALEFT12 ; VIA VIARIGHT12 ;
END VIALIST12
VIARULE VIAGEN12 GENERATE
    LAYER M1 ;
        ENCLOSURE 0.01 0.05 ;
    LAYER M2 ;
        ENCLOSURE 0.01 0.05 ;
    LAYER CUT12 ;
        RECT -0.06 -0.06 0.06 0.06 ;
        SPACING 0.14 BY 0.14 ;
END VIAGEN12
VIA VIACENTER23 LAYER M3 ; RECT -2.2 -2.2 2.2 2.2 ;
    LAYER CUT23 ; RECT -0.8 -0.8 0.8 0.8 ;
    LAYER M2 ; RECT -2.0 -2.0 2.0 2.0 ;
END VIACENTER23
VIA VIATOP23 LAYER M3 ; RECT -2.2 -2.2 2.2 8.2 ;
    LAYER CUT23 ; RECT -0.8 5.2 0.8 6.8 ;
    LAYER M2 ; RECT -2.2 -2.2 2.2 8.2 ;
END VIATOP23
VIA VIABOTTOM23 LAYER M3 ; RECT -2.2 -8.2 2.2 2.2 ;
    LAYER CUT23 ; RECT -0.8 -6.8 0.8 -5.2 ;
```

## LEF/DEF 5.8 Language Reference Examples

---

```
        LAYER M2 ; RECT -2.2 -8.2 2.2 2.2 ;
END VIABOTTOM23
VIA VIALEFT23 LAYER M3 ; RECT -7.8 -2.2 2.2 2.2 ;
        LAYER CUT23 ; RECT -6.4 -0.8 -4.8 0.8 ;
        LAYER M2 ; RECT -7.8 -2.2 2.2 2.2 ;
END VIALEFT23
VIA VIARIGHT23 LAYER M3 ; RECT -2.2 -2.2 7.8 2.2 ;
        LAYER CUT23 ; RECT 4.8 -0.8 6.4 0.8 ;
        LAYER M2 ; RECT -2.2 -2.2 7.8 2.2 ;
END VIARIGHT23
VIARULE VIALIST23 LAYER M3 ; DIRECTION VERTICAL ; WIDTH 3.6 TO
    3.6 ;
        LAYER M2 ; DIRECTION HORIZONTAL ; WIDTH 3.0 TO 3.0 ;
        VIA VIACENTER23 ; VIA VIATOP23 ; VIA VIABOTTOM23 ;
        VIA VIALEFT23 ; VIA VIARIGHT23 ;
END VIALIST23
VIARULE VIAGEN23 GENERATE
    LAYER M2 ;
        ENCLOSURE 0.01 0.05 ;
    LAYER M3 ;
        ENCLOSURE 0.01 0.05 ;
    LAYER CUT23 ;
        RECT -0.06 -0.06 0.06 0.06 ;
        SPACING 0.14 BY 0.14 ;
END VIAGEN23
MACRO CORNER CLASS ENDCAP BOTTOMLEFT ; SIZE 436.8 BY 444 ; SYMMETRY X Y ; SITE
CORNER ;
    PIN VDD SHAPE RING ; DIRECTION INOUT ;
        PORT LAYER M2 ; WIDTH 20 ; PATH 426.8 200 200 200 200 434 ;END
    END VDD
    PIN VSS SHAPE RING ; DIRECTION INOUT ;
        PORT LAYER M2 ; WIDTH 20 ; PATH 100 434 100 100 ; LAYER M1;
        WIDTH 20 ; PATH 100 100 426.8 100 ;END
    END VSS
END CORNER

MACRO IN1X class pad ; FOREIGN IN1X ; SIZE 151.2 BY 444 ;
    SYMMETRY X ; SITE IOX ;
    PIN Z DIRECTION OUTPUT ;
        PORT LAYER M1 ; PATH 61.6 444 72.8 444 ; END
    END Z
    PIN PO DIRECTION OUTPUT ;
        PORT LAYER M1 ; PATH 78.4 444 84.0 444 ; END
    END PO
    PIN A DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 95.2 444 100.8 444 ; END
    END A
    PIN PI DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 106.4 444 112 444 ; END
    END PI
    PIN VDD DIRECTION INOUT ; SHAPE ABUTMENT ;
```

## LEF/DEF 5.8 Language Reference Examples

---

```
        PORT LAYER M2 ; WIDTH 20 ; PATH 10 200 141.2 200 ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M1 ; WIDTH 20 ; PATH 10 100 141.2 100 ; END
    END VSS
END IN1X

MACRO IN1Y EEQ IN1X ; FOREIGN IN1Y ; class pad ; SIZE 436.8 BY 150 ;
    SYMMETRY Y ; SITE IOY ;
    PIN Z DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 0 69 0 75 ; END
    END Z
    PIN PO DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 0 81 0 87 ; END
    END PO
    PIN A DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 0 51 0 57 ; END
    END A
    PIN PI DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 0 39 0 45 ; END
    END PI
    PIN VDD DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M2 ; WIDTH 20 ; PATH 236.8 10 236.8 140 ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M2 ; WIDTH 20 ; PATH 336.8 10 336.8 140 ; END
    END VSS
END IN1Y

MACRO FILLER FOREIGN FILLER ; SIZE 67.2 BY 6 ; SYMMETRY X Y ;
    SITE CORE1 ;
    PIN VDD DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M1 ; RECT 45.8 0 55 6 ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M1 ; RECT 12.2 0 21.4 6 ; END
    END VSS
    OBS LAYER M1 ; RECT 24.1 1.5 43.5 4.5 ; END
END FILLER

MACRO INV FOREIGN INVS ; SIZE 67.2 BY 24 ; SYMMETRY X Y ; SITE CORE1 ;
    PIN Z DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 30.8 9 42 9 ; END
    END Z
    PIN A DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 25.2 15 ; END
    END A
    PIN VDD DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M1 ; WIDTH 5.6 ; PATH 50.4 4.6 50.4 13.4 ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M1 ; WIDTH 5.6 ; PATH 16.8 4.6 16.8 13.4 ; END
```

## LEF/DEF 5.8 Language Reference Examples

---

```
        END VSS
        OBS LAYER M1 ; RECT 24.1 1.5 43.5 16.5 ; END
    END INV

MACRO BUF FOREIGN BUFS ; SIZE 67.2 BY 126 ; SYMMETRY X Y ; SITE
    CORE1 ;
    PIN Z DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 25.2 39 42 39 ; END
    END Z
    PIN A DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 30.8 33 ; END
    END A
    PIN VDD DIRECTION INOUT ; SHAPE FEEDTHRU ;
        PORT LAYER M1 ; WIDTH 5.6 ;
        PATH 50.4 4.6 50.4 10.0 56.0 10.0 56.0 115.8 50.4 115.8
            50.4 121.4 ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE FEEDTHRU ;
        PORT LAYER M1 ; WIDTH 5.6 ;
        PATH 16.8 4.6 16.8 10.0 11.2 10.0 11.2 115.8 16.8 115.8
            16.8 121.4 ; END
    END VSS
    OBS LAYER M1 ; RECT 24.1 1.5 43.5 124.5 ; END
END BUF

MACRO BIDIR1X FOREIGN BIDIR1X ; class pad ; SIZE 151.2 BY 444 ;
    SYMMETRY X ; SITE IOX ;
    PIN IO DIRECTION INOUT ;
        PORT LAYER M1 ; PATH 61.6 444 67.2 444 ; END
    END IO
    PIN ZI DIRECTION OUTPUT ;
        PORT LAYER M1 ; PATH 78.4 444 84.0 444 ; END
    END ZI
    PIN PO DIRECTION OUTPUT ;
        PORT LAYER M1 ; PATH 95.2 444 100.8 444 ; END
    END PO
    PIN A DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 106.4 444 112.0 444 ; END
    END A
    PIN EN DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 134.4 444 140.0 444 ; END
    END EN
    PIN TN DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 28.0 444 33.6 444 ; END
    END TN
    PIN PI DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 44.8 444 50.4 444 ; END
    END PI
    PIN VDD DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M2 ; WIDTH 20 ; PATH 10 200 141.2 200 ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE ABUTMENT ;
```

## LEF/DEF 5.8 Language Reference Examples

---

```
        PORT LAYER M1 ; WIDTH 20 ; PATH 10 100 141.2 100 ; END
    END VSS
END BIDIR1X

MACRO BIDIR1Y EEQ BIDIR1X ; class pad ; FOREIGN BIDIR1Y ; SIZE 436.8
    BY 150 ; SYMMETRY Y ; SITE IOY ;
    PIN IO DIRECTION INOUT ;
        PORT LAYER M2 ; PATH 0 69 0 75 ; END END IO
    PIN ZI DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 0 93 0 99 ; END END ZI
    PIN PO DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 0 81 0 87 ; END END PO
    PIN A DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 0 15 0 21 ; END END A
    PIN EN DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 0 27 0 33 ; END END EN
    PIN TN DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 0 39 0 45 ; END END TN
    PIN PI DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 0 51 0 57 ; END END PI
    PIN VDD DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M2 ; WIDTH 20 ; PATH 236.8 10 236.8 140 ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE ABUTMENT ;
        PORT LAYER M2 ; WIDTH 20 ; PATH 336.8 10 336.8 140 ; END
    END VSS
END BIDIR1Y

MACRO OR2 FOREIGN OR2S ; SIZE 67.2 BY 42 ; SYMMETRY X Y ; SITE
    CORE1 ;
    PIN Z DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 25.2 39 42 39 ; END
    END Z
    PIN A DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 25.2 15 ; END
    END A
    PIN B DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 25.2 3 ; END
    END B
    PIN VDD DIRECTION INOUT ; SHAPE FEEDTHRU ;
        PORT LAYER M1 ; WIDTH 5.6 ;
        PATH 50.4 4.6 50.4 10.0 ; PATH 50.4 27.4 50.4 37.4 ;
        VIA 50.4 3 C2PW ; VIA 50.4 21 C2PW ; VIA 50.4 33 C2PW ;
        VIA 50.4 39 C2PW ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE FEEDTHRU ;
        PORT LAYER M1 ; WIDTH 5.6 ; PATH 16.8 4.6 16.8 10.0 ;
        PATH 16.8 27.4 16.8 37.4 ;
        VIA 16.8 3 C2NW ; VIA 16.8 15 C2NW ; VIA 16.8 21 C2NW ;
        VIA 16.8 33 C2NW ; VIA 16.8 39 C2NW ; END
```

## LEF/DEF 5.8 Language Reference Examples

---

```
END VSS
OBS LAYER M1 ; RECT 24.1 1.5 43.5 40.5 ; END
END OR2

MACRO AND2 FOREIGN AND2S ; SIZE 67.2 BY 84 ; SYMMETRY X Y ; SITE
CORE1 ;
PIN Z DIRECTION OUTPUT ;
    PORT LAYER M2 ; PATH 25.2 39 42 39 ; END
END Z
PIN A DIRECTION INPUT ;
    PORT LAYER M1 ; PATH 42 15 ; END
END A
PIN B DIRECTION INPUT ;
    PORT LAYER M1 ; PATH 42 3 ; END
END B
PIN VDD DIRECTION INOUT ; SHAPE ABUTMENT ;
    PORT LAYER M1 ; WIDTH 5.6 ; PATH 50.4 4.6 50.4 79.4 ; END
END VDD
PIN VSS DIRECTION INOUT ; SHAPE ABUTMENT ;
    PORT LAYER M1 ; WIDTH 5.6 ; PATH 16.8 4.6 16.8 79.4 ; END
END VSS
OBS LAYER M1 ; RECT 24.1 1.5 43.5 82.5 ; END
END AND2

MACRO DFF3 FOREIGN DFF3S ; SIZE 67.2 BY 210 ; SYMMETRY X Y ; SITE
CORE1 ;
PIN Q DIRECTION OUTPUT ;
    PORT LAYER M2 ; PATH 19.6 99 47.6 99 ; END
END Q
PIN QN DIRECTION OUTPUT ;
    PORT LAYER M2 ; PATH 25.2 123 42 123 ; END
END QN
PIN D DIRECTION INPUT ;
    PORT LAYER M1 ; PATH 30.8 51 ; END
END D
PIN G DIRECTION INPUT ;
    PORT LAYER M1 ; PATH 25.2 3 ; END
END G
PIN CD DIRECTION INPUT ;
    PORT LAYER M1 ; PATH 36.4 75 ; END
END CD
PIN VDD DIRECTION INOUT ; SHAPE FEEDTHRU ;
    PORT LAYER M1 ; WIDTH 5.6 ; PATH 50.4 4.6 50.4 205.4 ;
END
END VDD
PIN VSS DIRECTION INOUT ; SHAPE FEEDTHRU ;
    PORT LAYER M1 ; WIDTH 5.6 ; PATH 16.8 4.6 16.8 205.4 ;
END
END VSS
OBS LAYER M1 ; RECT 24.1 1.5 43.5 208.5 ; PATH 8.4 3 8.4 123 ;
    PATH 58.8 3 58.8 123 ; PATH 64.4 3 64.4 123 ; END
END DFF3
```

## LEF/DEF 5.8 Language Reference Examples

---

```
MACRO NOR2 FOREIGN NOR2S ; SIZE 67.2 BY 42 ; SYMMETRY X Y ; SITE
    CORE1 ;
    PIN Z DIRECTION OUTPUT ;
        PORT LAYER M1 ; PATH 42 33 ; END
    END Z
    PIN A DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 25.2 15 ; END
    END A
    PIN B DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 36.4 9 ; END
    END B
    PIN VDD DIRECTION INOUT ; SHAPE FEEDTHRU ;
        PORT LAYER M1 ; WIDTH 5.6 ; PATH 50.4 4.6 50.4 37.4 ; END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE FEEDTHRU ;
        PORT LAYER M1 ; WIDTH 5.6 ; PATH 16.8 4.6 16.8 37.4 ; END
    END VSS
    OBS LAYER M1 ; RECT 24.1 1.5 43.5 40.5 ; END
END NOR2
```

```
MACRO AND2J EEQ AND2 ; FOREIGN AND2SJ ; SIZE 67.2 BY 48 ;
    SYMMETRY X Y ; ORIGIN 0 6 ; SITE CORE1 ;
    PIN Z DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 25.2 33 42 33 ; END
    END Z
    PIN A DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 42 15 ; END
    END A
    PIN B DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 42 3 ; END
    END B
    PIN VDD DIRECTION INOUT ; SHAPE FEEDTHRU ;
        PORT LAYER M1 ; WIDTH 5.6 ; PATH 50.4 -1.4 50.4 37.4 ;
    END
    END VDD
    PIN VSS DIRECTION INOUT ; SHAPE FEEDTHRU ;
        PORT LAYER M1 ; WIDTH 5.6 ; PATH 16.8 -1.4 16.8 37.4 ;
    END
    END VSS
    OBS LAYER M1 ; RECT 24.1 1.5 43.5 34.5 ; END
END AND2J
```

```
MACRO SQUAREBLOCK FOREIGN SQUAREBLOCKS ; CLASS RING ; SIZE 268.8
    BY 252 ; SITE SQUAREBLOCK ;
    PIN Z DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 22.8 21 246.0 21 ; END
    END Z
    PIN A DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 64.4 33 137.2 33 ;
        PATH 137.2 33 137.2 69 ; PATH 137.2 69 204.4 69 ; END
    END A
    PIN B DIRECTION INPUT ;
```

## LEF/DEF 5.8 Language Reference Examples

---

```
        PORT LAYER M2 ; PATH 22.8 129 246.0 129 ; END
END B
PIN C DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 70 165 70 153 ; PATH 70 153 126 153 ;
        END
END C
PIN D DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 22.8 75 64.4 75 ; END
END D
PIN E DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 22.8 87 64.4 87 ; END
END E
PIN F DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 22.8 99 64.4 99 ; END
END F
PIN G DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 22.8 111 64.4 111 ; END
END G
PIN VDD DIRECTION INOUT ; SHAPE RING ;
        PORT LAYER M1 ; WIDTH 3.6 ; PATH 4.0 3.5 4.0 248 ;
        PATH 264.8 100 264.8 248 ; PATH 150 3.5 150 100 ;
        LAYER M2 ; WIDTH 3.6 ; PATH 4.0 3.5 150 3.5 ;
        PATH 150 100 264.8 100 ; PATH 4.0 248 264.8 248 ; END
END VDD
PIN VSS DIRECTION INOUT ; SHAPE RING ;
        PORT LAYER M1 ; WIDTH 3.6 ; PATH 10 10 10 150 ;
        PATH 100 150 100 200 ; PATH 50 200 50 242 ;
        PATH 258.8 10 258.8 242 ; LAYER M2 ; WIDTH 3.6 ;
        PATH 10 150 100 150 ; PATH 100 200 50 200 ;
        PATH 10 10 258.8 10 ; PATH 50 242 258.8 242 ; END
END VSS
OBS LAYER M1 ; RECT 13.8 14.0 255.0 237.2 ; END
END SQUAREBLOCK

MACRO I2BLOCK FOREIGN I2BLOCKS ; CLASS RING ; SIZE 672 BY 504 ;
SITE I2BLOCK ;
PIN Z DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 22.8 21 649.2 21 ; END
END Z
PIN A DIRECTION OUTPUT ;
        PORT LAYER M2 ; PATH 22.8 63 154.0 63 ; PATH 154.0 63 154.0
        129;
        PATH 154.0 129 447.6 129 ; END
END A
PIN B DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 137.2 423 447.6 423 ; END
END B
PIN C DIRECTION INPUT ;
        PORT LAYER M2 ; PATH 204.4 165 271.6 165 ; END
END C
PIN D DIRECTION INPUT ;
```



## LEF/DEF 5.8 Language Reference Examples

---

```
        PORT LAYER M2 ; PATH 204.4 171 271.6 171 ; END
END D
PIN E DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 204.4 213 204.4 213 ; END
END E
PIN F DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 406 249 406 273 ; END
END F
PIN G DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 338.8 249 338.8 273 ; END
END G
PIN H DIRECTION INPUT ;
        PORT LAYER M1 ; PATH 372.4 357 372.4 381 ; END
END H
PIN VDD DIRECTION INOUT ; SHAPE RING ;
        PORT LAYER M1 ; WIDTH 3.6 ; PATH 668 3.5 668 80.5 ;
        PATH 467 80.5 467 465.5 ; PATH 668 465.5 668 500.5 ;
        PATH 4 500.5 4 465.5 ; PATH 138 465.5 138 80.5 ;
        PATH 4 80.5 4 3.5 ; LAYER M2 ; WIDTH 3.6 ; PATH 4 3.5 668 3.5 ;
        PATH 668 80.5 467 80.5 ; PATH 467 465.5 668 465.5 ;
        PATH 668 500.5 4 500.5 ; PATH 4 465.5 138 465.5 ;
        PATH 138 80.5 4 80.5 ; END
END VDD
PIN VSS DIRECTION INOUT ; SHAPE RING ;
        PORT LAYER M1 ; WIDTH 3.6 ; PATH 662 10 662 74 ;
        PATH 461 74 461 472 ; PATH 662 472 662 494 ; PATH 10 494 10
        472 ;
        PATH 144 472 144 74 ; PATH 10 74 10 10 ; LAYER M2 ; WIDTH
        3.6 ;
        PATH 10 10 662 10 ; PATH 662 74 461 74 ; PATH 461 472 662
        472 ;
        PATH 662 494 10 494 ; PATH 10 472 144 472 ; PATH 144 74 10
        74 ;
        END
END VSS
OBS LAYER M1 ; RECT 14 14 658 70 ; RECT 14 476 658 490 ;
        RECT 148 14 457 490 ; # rectilinear shape description
        LAYER OVERLAP ; RECT 0 0 672 84 ; RECT 134.4 84 470.4 462 ;
        RECT 0 462 672 504 ; END
END I2BLOCK

MACRO LBLOCK FOREIGN LBLOCKS ; CLASS RING ; SIZE 201.6 BY 168 ; SITE
        LBLOCK ;
        PIN Z DIRECTION OUTPUT ;
                PORT LAYER M2 ; PATH 2.8 15 198.8 15 ; END
        END Z
        PIN A DIRECTION OUTPUT ;
                PORT LAYER M2 ; PATH 2.8 81 137.2 81 ; PATH 137.2 81 137.2
                69 ;
                PATH 137.2 69 198.8 69 ; END
        END A
```

## LEF/DEF 5.8 Language Reference Examples

---

```
PIN B DIRECTION INPUT ;
      PORT LAYER M2 ; PATH 2.8 165 64.4 165 ; END
END B
PIN C DIRECTION INPUT ;
      PORT LAYER M1 ; PATH 2.8 93 2.8 105 ; END
END C
PIN D DIRECTION INPUT ;
      PORT LAYER M1 ; PATH 64.4 93 64.4 105 ; END
END D
PIN E DIRECTION INPUT ;
      PORT LAYER M1 ; PATH 198.8 39 198.8 39 ; END
END E
PIN F DIRECTION INPUT ;
      PORT LAYER M1 ; PATH 198.8 45 198.8 45 ; END
END F
PIN G DIRECTION INPUT ;
      PORT LAYER M1 ; PATH 2.8 111 2.8 111 ; END END G
      PORT LAYER M2 ; WIDTH 3.6 ; PATH 1.8 27 199.8 27 ; END
END VDD
PIN VSS DIRECTION INOUT ;
      PORT LAYER M2 ; WIDTH 3.6 ; PATH 1.8 57 199.8 57 ; END
END VSS
OBS LAYER M2 ; RECT 1.0 80 66.2 166.5 ; RECT 1.0 1.5 200.6 23 ;
      RECT 1.0 31 200.6 53 ; RECT 1.0 61 200.6 82.5 ;
      # rectilinear shape description
      LAYER OVERLAP ; RECT 0 0 201.6 84 ; RECT 0 84 67.2 168 ;
      END
END LBLOCK

END LIBRARY
```

## DEF

The following example shows a design netlist.

```
DESIGN DEMO4CHIP ;
      TECHNOLOGY DEMO4CHIP ;
      UNITS DISTANCE MICRONS 100 ;
      COMPONENTS 243 ;

- CORNER1 CORNER ; - CORNER2 CORNER ; - CORNER3 CORNER ;
- CORNER4 CORNER ; - C01 IN1X ; - C02 IN1Y ; - C04 IN1X ;
- C05 IN1X ; - C06 IN1Y ;
- C07 IN1Y ; - C08 IN1Y ; - C09 IN1Y ; - C10 IN1X ; - C11 IN1X ;
- C13 BIDIR1Y ; - C14 INV ; - C15 BUF ; - C16 BUF ; - C17 BUF ;
- C19 BIDIR1Y ; - C20 INV ; - C21 BUF ; - C22 BUF ; - C23 BUF ;
- C25 BIDIR1Y ; - C26 INV ; - C27 BUF ; - C28 BUF ; - C29 BUF ;
- C31 BIDIR1Y ; - C32 INV ; - C33 BUF ; - C34 BUF ; - C35 BUF ;
- C37 BIDIR1X ; - C39 INV ; - C40 BUF ; - C41 BUF ; - C42 BUF ;
- C44 BIDIR1X ; - C45 INV ; - C46 BUF ; - C47 BUF ; - C48 BUF ;
```

## LEF/DEF 5.8 Language Reference Examples

---

```
- C50 BIDIR1Y ; - C51 INV ; - C52 BUF ; - C53 BUF ; - C54 BUF ;
- C56 BIDIR1X ; - C57 INV ; - C58 BUF ; - C59 BUF ; - C60 BUF ;
- D02 BIDIR1X ; - D03 INV ; - D04 BUF ; - D05 BUF ; - D06 BUF ;
- D08 BIDIR1X ; - D09 INV ; - D10 BUF ; - D11 BUF ; - D12 BUF ;
- D14 BIDIR1X ; - D15 INV ; - D16 BUF ; - D17 BUF ; - D19 BUF ;
- D33 BIDIR1Y ; - D34 INV ; - D35 BUF ; - D36 BUF ; - D37 BUF ;
- D39 BIDIR1Y ; - D40 INV ; - D41 BUF ; - D42 BUF ; - D43 BUF ;
- D45 BIDIR1Y ; - D46 INV ; - D47 BUF ; - D48 BUF ; - D49 BUF ;
- D82 OR2 ; - D83 OR2 ; - D84 OR2 ; - D85 OR2 ; - D86 OR2 ;
- D87 OR2 ; - D88 OR2 ; - D89 OR2 ; - D90 OR2 ; - D91 OR2 ;
- D92 OR2 ; - D93 OR2 ;
- E01 AND3 ; - E02 AND3 ; - E03 AND3 ; - E04 AND3 ; - E05 AND3 ;
- E06 AND3 ; - E07 AND3 ; - E08 AND3 ; - E09 AND3 ; - E10 AND3 ;
- E11 AND3 ; - E12 AND3 ; - E13 AND3 ; - E14 AND3 ; - E15 AND3 ;
- E16 AND3 ;
- EE16 IN1X ; - E17 IN1X ; - E18 IN1X ; - E19 IN1X ; - E20 IN1X ;
- E21 IN1X ; - E22 IN1X ; - E23 IN1Y ; - E24 IN1Y ; - E25 IN1Y ;
- E26 INV ; - E27 AND2 ; - E28 AND2 ; - E29 AND2 ; - E30 AND2 ;
- E31 AND2 ; - E32 AND2 ; - E33 OR2 ; - E34 OR2 ; - E35 OR2 ;
- E36 OR2 ; - E37 IN1Y ; - E38A01 DFF3 ; - E38A02 DFF3 ;
- E38A03 DFF3 ; - E38A04 DFF3 ; - E38A05 DFF3 ; - F01 I2BLOCK ;
- F04 OR2 ; - F06 OR2 ; - F07 OR2 ; - F08 OR2 ; - F09 SQUAREBLOCK ;
- F12 LBLOCK ;
- Z14 INV ; - Z15 BUF ; - Z16 BUF ; - Z17 BUF ; - Z20 INV ;
- Z21 BUF ; - Z22 BUF ; - Z23 BUF ; - Z26 INV ; - Z27 BUF ;
- Z28 BUF ; - Z29 BUF ; - Z32 INV ; - Z33 BUF ; - Z34 BUF ;
- Z35 BUF ; - Z39 INV ; - Z40 BUF ; - Z41 BUF ; - Z42 BUF ;
- Z45 INV ; - Z46 BUF ; - Z47 BUF ; - Z48 BUF ; - Z51 INV ;
- Z52 BUF ; - Z53 BUF ; - Z54 BUF ; - Z57 INV ; - Z58 BUF ; - Z59 BUF ;
- Z60 BUF ; - Z103 INV ; - Z104 BUF ; - Z105 BUF ; - Z106 BUF ;
- Z109 INV ; - Z110 BUF ; - Z111 BUF ; - Z112 BUF ; - Z115 INV ;
- Z116 BUF ; - Z117 BUF ; - Z119 BUF ; - Z134 INV ; - Z135 BUF ;
- Z136 BUF ; - Z137 BUF ; - Z140 INV ; - Z141 BUF ; - Z142 BUF ;
- Z143 BUF ; - Z146 INV ; - Z147 BUF ; - Z148 BUF ; - Z149 BUF ;
- Z182 OR2 ; - Z183 OR2 ; - Z184 OR2 ; - Z185 OR2 ; - Z186 OR2 ;
- Z187 OR2 ; - Z188 OR2 ; - Z189 OR2 ; - Z190 OR2 ; - Z191 OR2 ;
- Z192 OR2 ; - Z193 OR2 ; - Z201 AND3 ; - Z202 AND3 ; - Z203 AND3 ;
- Z204 AND3 ; - Z205 AND3 ; - Z206 AND3 ; - Z207 AND3 ; - Z208 AND3 ;
- Z209 AND3 ; - Z210 AND3 ; - Z211 AND3 ; - Z212 AND3 ; - Z213 AND3 ;
- Z214 AND3 ; - Z215 AND3 ; - Z216 AND3 ; - Z226 INV ; - Z227 AND2 ;
- Z228 AND2 ; - Z229 AND2 ; - Z230 AND2 ; - Z231 AND2 ; - Z232 AND2 ;
- Z233 OR2 ; - Z234 OR2 ; - Z235 OR2 ; - Z236 OR2 ; - Z38A01 DFF3 ;
- Z38A02 DFF3 ; - Z38A03 DFF3 ; - Z38A04 DFF3 ; - Z38A05 DFF3 ;
END COMPONENTS
```

NETS 222 ;

```
- VDD ( Z216 B ) ( Z215 B ) ( Z214 C ) ( Z214 B )
( Z213 C ) ( Z213 B ) ( Z212 C ) ( Z212 B ) ( Z211 C ) ( Z211 B )
( Z210 C ) ( E23 Z ) ( Z143 Z ) ( Z142 Z ) ( Z141 Z ) ( Z119 Z )
( Z117 Z ) ( Z116 Z ) ( Z106 Z ) ( Z105 Z ) ( Z104 Z ) ( Z34 Z )
( Z33 Z ) ( Z28 Z ) ( Z27 Z ) ( Z22 Z ) ( Z21 Z ) ( Z16 Z )
```

## LEF/DEF 5.8 Language Reference Examples

---

```
( Z15 Z ) ( D45 PO ) ( D14 PO ) ( C01 PI ) ( D45 TN ) ( D39 TN )
( D33 TN ) ( D14 TN ) ( D08 TN ) ( D02 TN ) ( C56 TN ) ( C50 TN )
( C44 TN ) ( C37 TN ) ( C31 TN ) ( C25 TN ) ( C19 TN ) ( C13 TN ) ;
- VSS ( Z209 C ) ( Z208 C ) ( Z207 C ) ( Z206 C ) ( Z205 C )
( Z204 C ) ( Z203 C ) ( Z202 C ) ( Z201 C ) ( Z149 Z ) ( Z148 Z )
( Z147 Z ) ( Z137 Z ) ( Z136 Z ) ( Z135 Z ) ( Z112 Z ) ( Z111 Z )
( Z110 Z ) ( Z60 Z ) ( Z59 Z ) ( Z58 Z ) ( Z54 Z ) ( Z53 Z )
( Z52 Z ) ( Z47 Z ) ( Z46 Z ) ( Z41 Z ) ( Z40 Z ) ( E18 Z )
( D49 Z ) ( D43 Z ) ( D45 A ) ( D39 A ) ( D33 A ) ( D14 A )
( D08 A ) ( D02 A ) ( C56 A ) ( C50 A ) ( C44 A ) ( C37 A )
( C31 A ) ( C25 A ) ( C19 A ) ( C13 A ) ; - XX1001 ( Z38A04 G )
( Z38A02 G ) ; - XX100 ( Z38A05 G ) ( Z38A03 G ) ( Z38A01 G ) ;
- XX907 ( Z236 B ) ( Z235 B ) ; - XX906 ( Z234 B ) ( Z233 B ) ;
- XX904 ( Z232 B ) ( Z231 B ) ; - XX903 ( Z230 B ) ( Z229 B ) ;
- XX902 ( Z228 B ) ( Z227 B ) ;
- XX900 ( Z235 A ) ( Z233 A ) ( Z232 A ) ( Z230 A ) ( Z228 A ) ( Z226 A ) ;
- Z38QN4 ( Z38A04 QN ) ( Z210 B ) ; - COZ131 ( Z38A04 Q ) ( Z210 A ) ;
- Z38QN3 ( Z38A03 QN ) ( Z209 B ) ; - COZ121 ( Z38A03 Q ) ( Z209 A ) ;
- Z38QN2 ( Z38A02 QN ) ( Z208 B ) ; - COZ111 ( Z38A02 Q ) ( Z208 A ) ;
- Z38QN1 ( Z38A01 QN ) ( Z207 B ) ; - COZ101 ( Z38A01 Q ) ( Z207 A ) ;
- XX901 ( Z236 A ) ( Z234 A ) ( Z231 A ) ( Z229 A ) ( Z227 A ) ( Z226 Z )
( Z193 A ) ;
- X415 ( Z149 A ) ( Z148 A ) ( Z147 A ) ( Z146 Z ) ; - X413 ( Z143 A )
( Z142 A ) ( Z141 A ) ( Z140 Z ) ;
- X411 ( Z137 A ) ( Z136 A ) ( Z135 A ) ( Z134 Z ) ;
- X405 ( Z119 A ) ( Z117 A ) ( Z116 A ) ( Z115 Z ) ;
- X403 ( Z112 A ) ( Z111 A ) ( Z110 A ) ( Z109 Z ) ;
- X401 ( Z106 A ) ( Z105 A ) ( Z104 A ) ( Z103 Z ) ;
- X315 ( Z60 A ) ( Z59 A ) ( Z58 A ) ( Z57 Z ) ;
- X313 ( Z54 A ) ( Z53 A ) ( Z52 A ) ( Z51 Z ) ;
- DIS051 ( Z216 A ) ( Z48 Z ) ;
- X311 ( Z48 A ) ( Z47 A ) ( Z46 A ) ( Z45 Z ) ;
- DIS041 ( Z215 A ) ( Z42 Z ) ; - X309 ( Z42 A ) ( Z41 A ) ( Z40 A )
( Z39 Z ) ;
- X307 ( Z35 A ) ( Z34 A ) ( Z33 A ) ( Z32 Z ) ;
- DIS031 ( Z214 A ) ( Z35 Z ) ; - DIS021 ( Z213 A ) ( Z29 Z ) ;
- X305 ( Z29 A ) ( Z28 A ) ( Z27 A ) ( Z26 Z ) ;
- DIS011 ( Z212 A ) ( Z23 Z ) ;
- X303 ( Z23 A ) ( Z22 A ) ( Z21 A ) ( Z20 Z ) ;
- DIS001 ( Z211 A ) ( Z17 Z ) ;
- X301 ( Z17 A ) ( Z16 A ) ( Z15 A ) ( Z14 Z ) ;
- X1000 ( E38A05 G ) ( E38A03 G ) ( E38A01 G ) ( E37 Z ) ;
- CNTEN ( Z38A05 Q ) ( E38A05 Q ) ( E25 A ) ;
- VIH20 ( E37 PI ) ( E25 PO ) ; - X0907 ( E36 B ) ( E35 B ) ( E25 Z ) ;
- CCLK0 ( F09 A ) ( E24 A ) ; - VIH19 ( E25 PI ) ( E24 PO ) ;
- X0906 ( E34 B ) ( E33 B ) ( E24 Z ) ; - CATH1 ( F09 Z ) ( E23 A ) ;
- VIH18 ( E24 PI ) ( E23 PO ) ; - CRLIN ( F08 Z ) ( E22 A ) ;
- VIH17 ( E23 PI ) ( E22 PO ) ; - X0904 ( E32 B ) ( E31 B ) ( E22 Z ) ;
- NXLIN ( F07 Z ) ( E21 A ) ; - VIH16 ( E22 PI ) ( E21 PO ) ;
- X0903 ( E30 B ) ( E29 B ) ( E21 Z ) ; - RPT1 ( F06 Z ) ( E20 A ) ;
- VIH15 ( E21 PI ) ( E20 PO ) ; - X0902 ( E28 B ) ( E27 B ) ( E20 Z ) ;
```

## LEF/DEF 5.8 Language Reference Examples

---

```
- AGISL ( F04 Z ) ( E19 A ) ; - VIH14 ( E20 PI ) ( E19 PO ) ;
- X0900 ( E35 A ) ( E33 A ) ( E32 A ) ( E30 A ) ( E28 A ) ( E26 A )
  ( E19 Z ) ;
- TSTCN ( Z38A05 QN ) ( E38A05 QN ) ( E18 A ) ;
- VIH13 ( E19 PI ) ( E18 PO ) ; - BCLK1 ( F01 A ) ( E17 A ) ;
- VIH12 ( E18 PI ) ( E17 PO ) ; - CLR0 ( F01 Z ) ( EE16 A ) ;
- VIH11 ( E17 PI ) ( EE16 PO ) ; - BCLKX1 ( Z216 C ) ( E17 Z )
  ( E16 C ) ; - CLRX0 ( Z38A05 CD ) ( Z38A03 CD ) ( Z38A01 CD )
  ( Z215 C ) ( E38A05 CD ) ( E38A03 CD ) ( E38A01 CD ) ( EE16 Z )
  ( E15 C ) ; - E38QN4 ( E38A04 QN ) ( E10 B ) ;
- CAX131 ( E38A04 Q ) ( E10 A ) ; - E38QN3 ( E38A03 QN ) ( E09 B ) ;
- CAX121 ( E38A03 Q ) ( E09 A ) ; - E38QN2 ( E38A02 QN ) ( E08 B ) ;
- CAX111 ( E38A02 Q ) ( E08 A ) ; - E38QN1 ( E38A01 QN ) ( E07 B ) ;
- CAX101 ( E38A01 Q ) ( E07 A ) ;
- SDD111 ( Z38A05 D ) ( Z205 Z ) ( E38A05 D ) ( E05 Z ) ;
- SDD121 ( Z38A04 D ) ( Z204 Z ) ( E38A04 D ) ( E04 Z ) ;
- X0901 ( E36 A ) ( E34 A ) ( E31 A ) ( E29 A ) ( E27 A ) ( E26 Z )
  ( D93 A ) ;
- VIH21 ( Z192 A ) ( E37 PO ) ( D92 A ) ;
- STRDENB0 ( Z206 B ) ( Z202 B ) ( Z201 B ) ( Z189 B ) ( Z188 B )
  ( F12 A ) ( E06 B ) ( E02 B ) ( E01 B ) ( D89 B ) ( D88 B ) ;
- STRDENA0 ( Z202 A ) ( Z201 A ) ( Z183 B ) ( Z182 B ) ( F12 Z )
  ( F01 H ) ( E02 A ) ( E01 A ) ( D83 B ) ( D82 B ) ;
- DAB151 ( F12 H ) ( D48 Z ) ; - DAA151 ( F08 B ) ( D47 Z ) ;
- X0415 ( D49 A ) ( D48 A ) ( D47 A ) ( D46 Z ) ;
- SDD151 ( Z38A01 D ) ( Z201 Z ) ( E38A01 D ) ( E01 Z ) ( D45 EN ) ;
- X0414 ( Z146 A ) ( D46 A ) ( D45 ZI ) ; - D151 ( E14 C ) ( D45 IO ) ;
- DAB141 ( F12 G ) ( D42 Z ) ; - DAA141 ( F08 A ) ( D41 Z ) ;
- X0413 ( D43 A ) ( D42 A ) ( D41 A ) ( D40 Z ) ;
- SDD141 ( Z38A02 D ) ( Z202 Z ) ( E38A02 D ) ( E02 Z ) ( D39 EN ) ;
- VIH60 ( D45 PI ) ( D39 PO ) ; - X0412 ( Z140 A ) ( D40 A ) ( D39 ZI ) ;
- D141 ( E13 C ) ( D39 IO ) ; - SDI131 ( E16 B ) ( D37 Z ) ;
- DAB131 ( F12 F ) ( D36 Z ) ; - DAA131 ( F07 B ) ( D35 Z ) ;
- X0411 ( D37 A ) ( D36 A ) ( D35 A ) ( D34 Z ) ;
- VIH58 ( Z193 Z ) ( D93 Z ) ( D33 PI ) ;
- SDD131 ( Z38A03 D ) ( Z203 Z ) ( E38A03 D ) ( E03 Z ) ( D33 EN ) ;
- VIH59 ( D39 PI ) ( D33 PO ) ; - X0410 ( Z134 A ) ( D34 A ) ( D33 ZI ) ;
- D131 ( E12 C ) ( D33 IO ) ; - SDI101 ( E15 B ) ( D19 Z ) ; ...
- X0315 ( C60 A ) ( C59 A ) ( C58 A ) ( C57 Z ) ;
- SDD071 ( Z211 Z ) ( E11 Z ) ( C56 EN ) ;
- VIH53 ( Z190 Z ) ( D90 Z ) ( D02 PI ) ( C56 PO ) ;
- X0314 ( Z57 A ) ( C57 A ) ( C56 ZI ) ;
- D071 ( E08 C ) ( C56 IO ) ; - SDI061 ( E11 B ) ( C54 Z ) ;
- DAB061 ( F09 H ) ( C53 Z ) ; - DAA061 ( F04 A ) ( C52 Z ) ;
- X0313 ( C54 A ) ( C53 A ) ( C52 A ) ( C51 Z ) ;
- SDD061 ( Z212 Z ) ( E12 Z ) ( C50 EN ) ;
- VIH52 ( Z189 Z ) ( D89 Z ) ( C56 PI ) ( C50 PO ) ;
- X0312 ( Z51 A ) ( C51 A ) ( C50 ZI ) ;
- D061 ( E07 C ) ( C50 IO ) ; - SDI051 ( E16 A ) ( C48 Z ) ;
- DAB051 ( F09 G ) ( C47 Z ) ; - DAA051 ( F01 G ) ( C46 Z ) ;
- X0311 ( C48 A ) ( C47 A ) ( C46 A ) ( C45 Z ) ;
```

## LEF/DEF 5.8 Language Reference Examples

---

```
- SDD051 ( Z213 Z ) ( E13 Z ) ( C44 EN ) ;
- VIH51 ( Z188 Z ) ( D88 Z ) ( C50 PI ) ( C44 PO ) ;
- X0310 ( Z45 A ) ( C45 A ) ( C44 ZI ) ;
- D051 ( E06 C ) ( C44 IO ) ; - SDI041 ( E15 A ) ( C42 Z ) ;
- DAB041 ( F09 F ) ( C41 Z ) ; - DAA041 ( F01 F ) ( C40 Z ) ;
- X0309 ( C42 A ) ( C41 A ) ( C40 A ) ( C39 Z ) ;
- SDD041 ( Z214 Z ) ( E14 Z ) ( C37 EN ) ;
- VIH50 ( Z187 Z ) ( D87 Z ) ( C44 PI ) ( C37 PO ) ;
- X0308 ( Z39 A ) ( C39 A ) ( C37 ZI ) ;
- D041 ( E05 C ) ( C37 IO ) ; - SDI031 ( E14 A ) ( C35 Z ) ;
- DAB031 ( F09 E ) ( C34 Z ) ; - DAA031 ( F01 E ) ( C33 Z ) ;
- X0307 ( C35 A ) ( C34 A ) ( C33 A ) ( C32 Z ) ;
- SDD031 ( Z215 Z ) ( E15 Z ) ( C31 EN ) ;
- VIH49 ( Z186 Z ) ( D86 Z ) ( C37 PI ) ( C31 PO ) ;
- X0306 ( Z32 A ) ( C32 A ) ( C31 ZI ) ;
- D031 ( E04 C ) ( C31 IO ) ; - SDI021 ( E13 A ) ( C29 Z ) ;
- DAB021 ( F09 D ) ( C28 Z ) ; - DAA021 ( F01 D ) ( C27 Z ) ;
- X0305 ( C29 A ) ( C28 A ) ( C27 A ) ( C26 Z ) ;
- SDD021 ( Z216 Z ) ( E16 Z ) ( C25 EN ) ;
- VIH48 ( Z185 Z ) ( D85 Z ) ( C31 PI ) ( C25 PO ) ;
- X0304 ( Z26 A ) ( C26 A ) ( C25 ZI ) ;
- D021 ( E03 C ) ( C25 IO ) ; - SDI011 ( E12 A ) ( C23 Z ) ;
- DAB011 ( F09 C ) ( C22 Z ) ; - DAA011 ( F01 C ) ( C21 Z ) ;
- X0303 ( C23 A ) ( C22 A ) ( C21 A ) ( C20 Z ) ;
- SDD011 ( Z209 Z ) ( E09 Z ) ( C19 EN ) ;
- VIH47 ( Z184 Z ) ( D84 Z ) ( C25 PI ) ( C19 PO ) ;
- X0302 ( Z20 A ) ( C20 A ) ( C19 ZI ) ;
- D011 ( E02 C ) ( C19 IO ) ; - SDI001 ( E11 A ) ( C17 Z ) ;
- DAB001 ( F09 B ) ( C16 Z ) ; - DAA001 ( F01 B ) ( C15 Z ) ;
- X0301 ( Z14 A ) ( C17 A ) ( C16 A ) ( C15 A ) ( C14 Z ) ;
- VIH45 ( Z182 Z ) ( D82 Z ) ( C13 PI ) ;
- SDD001 ( Z210 Z ) ( E10 Z ) ( C13 EN ) ;
- VIH46 ( Z183 Z ) ( D83 Z ) ( C19 PI ) ( C13 PO ) ;
- X0300 ( C14 A ) ( C13 ZI ) ; - D001 ( E01 C ) ( C13 IO ) ;
- CCLKB0 ( Z234 Z ) ( Z189 A ) ( E34 Z ) ( D89 A ) ( C11 A ) ;
- VIH10 ( EE16 PI ) ( C11 PO ) ;
- STRAAA ( Z206 A ) ( E06 A ) ( C11 Z ) ;
- CCLKA0 ( Z233 Z ) ( Z188 A ) ( E33 Z ) ( D88 A ) ( C10 A ) ;
- VIH9 ( C11 PI ) ( C10 PO ) ;
- STRB00 ( Z192 B ) ( D92 B ) ( C10 Z ) ;
- CRLINB1 ( Z232 Z ) ( Z187 A ) ( E32 Z ) ( D87 A ) ( C09 A ) ;
- VIH8 ( C10 PI ) ( C09 PO ) ;
- STRA00 ( Z187 B ) ( D87 B ) ( C09 Z ) ;
- CRLINA1 ( Z231 Z ) ( Z186 A ) ( E31 Z ) ( D86 A ) ( C08 A ) ;
- VIH7 ( C09 PI ) ( C08 PO ) ;
- X10001 ( E38A04 G ) ( E38A02 G ) ( C08 Z ) ;
- NXLINB1 ( Z230 Z ) ( Z185 A ) ( E30 Z ) ( D85 A ) ( C07 A ) ;
- VIH6 ( C08 PI ) ( C07 PO ) ;
- CLRX00 ( Z38A04 CD ) ( Z38A02 CD ) ( E38A04 CD ) ( E38A02 CD )
    ( C07 Z ) ;
- NXLINA1 ( Z229 Z ) ( Z184 A ) ( E29 Z ) ( D84 A ) ( C06 A ) ;
```

## LEF/DEF 5.8 Language Reference Examples

---

```
- VIH5 ( C07 PI ) ( C06 PO ) ;
- STRBB0 ( Z205 B ) ( Z193 B ) ( E05 B ) ( D93 B ) ( C06 Z ) ;
- RPTB1 ( Z228 Z ) ( Z183 A ) ( E28 Z ) ( D83 A ) ( C05 A ) ;
- VIH4 ( C06 PI ) ( C05 PO ) ;
- STRAA0 ( Z205 A ) ( Z186 B ) ( E05 A ) ( D86 B ) ( C05 Z ) ;
- RPTA1 ( Z227 Z ) ( Z182 A ) ( E27 Z ) ( D82 A ) ( C04 A ) ;
- VIH3 ( C05 PI ) ( C04 PO ) ;
- STRB0 ( Z204 B ) ( Z203 B ) ( Z191 B ) ( Z190 B ) ( E04 B )
    ( E03 B ) ( D91 B ) ( D90 B ) ( C04 Z ) ;
- CNTENB0 ( Z236 Z ) ( Z191 A ) ( E36 Z ) ( D91 A ) ( C02 A ) ;
- VIH2 ( C04 PI ) ( C02 PO ) ;
- STRA0 ( Z204 A ) ( Z203 A ) ( Z185 B ) ( Z184 B ) ( E04 A )
    ( E03 A ) ( D85 B ) ( D84 B ) ( C02 Z ) ;
- CNTENA0 ( Z235 Z ) ( Z190 A ) ( E35 Z ) ( D90 A ) ( C01 A ) ;
- VIH1 ( C02 PI ) ( C01 PO ) ; - CALCH ( E37 A ) ( C01 Z ) ;
```

#

## Scan Chain Synthesis Example

You define the scan chain in the COMPONENTS and SCANCHAINS sections in your DEF file.

```
COMPONENTS 100 ;
- SIN MUX ;
- SOUT PAD ;
- C1 SDFF ;
- C2 SDFF ;
- C3 SDFF ;
- C4 SDFF ;
- B1 BUF ;
- A1 AND ; ...
END COMPONENTS

NETS 150 ;
- N1 (C1 SO) (C3 SI) ;
- N2 (C3 SO) (A1 A) ; ...
END NETS
```

You do not need to define any scan nets in the NETS section. This portion of the NETS section shows the effect of the scan chain process on existing nets that use components you specify in the SCANCHAINS section.

```
SCANCHAINS 1 ;
- SC
    + COMMONSCANPINS (IN SI) (OUT SO)
    + START SIN Z2
    + FLOATING C1 C2 C3
    + ORDERED C4 B1 (IN A) (OUT Q) ;
    + STOP SOUT A ;
END SCANCHAINS
```

## LEF/DEF 5.8 Language Reference

### Examples

---

Because components C1, C2, and C3 are floating, TROUTE SCANCHAIN can synthesize them in any order in the chain. TROUTE synthesizes ordered components (C4 and B1) in the order you specify.



---

## Optimizing LEF Technology for Place and Route

---

This appendix contains the following information.

- [Overview](#)
- [Guidelines for Routing Pitch](#) on page 794
- [Guidelines for Wide Metal Spacing](#) on page 796
- [Guidelines for Wire Extension at Vias](#) on page 797
- [Guidelines for Default Vias](#) on page 799
- [Guidelines for Stack Vias \(MAR Vias\) and Samenet Spacing](#) on page 801
- [Example of an Optimized LEF Technology File](#) on page 805

### Overview

This appendix provides guidelines for defining the optimized technology section in the LEF file to get the best performance using Cadence® place-and-route tools.

For the following guidelines, the preferred routing direction for `metal1` and all other odd metal layers is horizontal. The preferred routing direction for `metal2` and all other even metal layers is vertical. Standard cells are arranged in horizontal rows.

This appendix discusses the following LEF statements.

```
LAYER layerName
    TYPE ROUTING ;
    PITCH distance ;
    WIDTH defWidth ;
    SPACING minSpacing [RANGE minwidth maxwidth] ;
    WIREEXTENSION value ;

END layerName
```

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

```
VIA viaName DEFAULT
    [TOPSTACKONLY]
    LAYER layerName RECT pt pt ; ...

END viaName

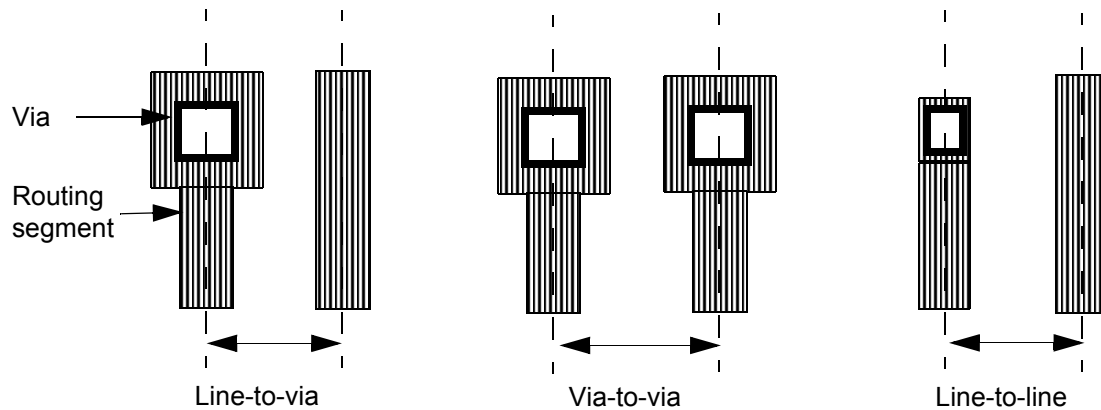
SPACING
    SAMENET
        layerName layerName minSpace [STACK] ;

END SPACING
```

## Guidelines for Routing Pitch

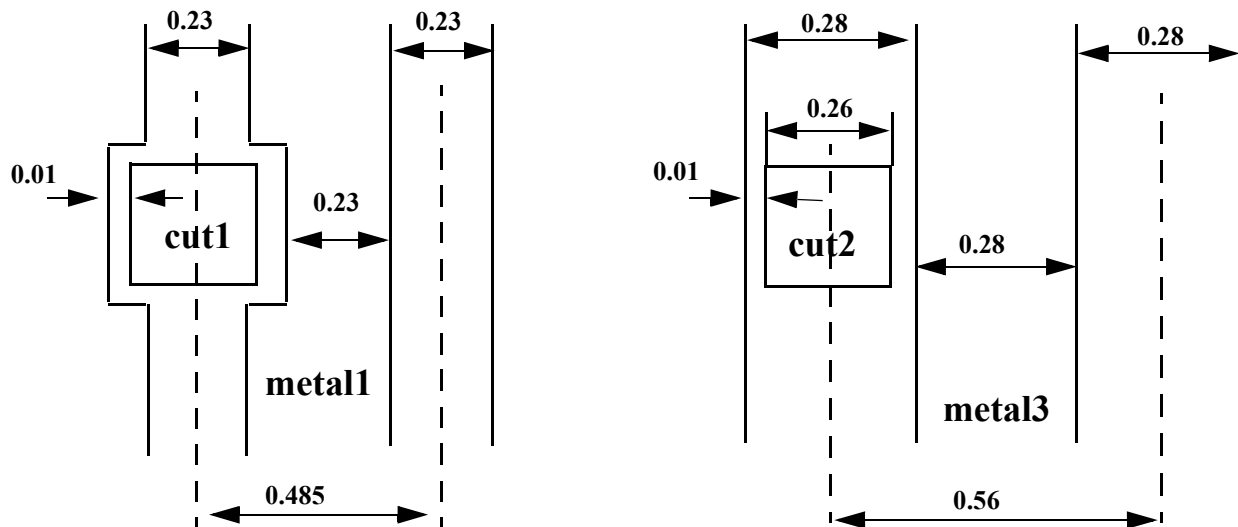
The following is a summary for choosing the right pitch for an existing design library. For detailed information on determining routing pitch, refer to the *Cadence Abstract Generator User Guide*.

### Pitch Measurement



## DESIGN RULE No. 1

W.1	Minimum width of <code>metal1</code> = 0.23 $\mu\text{m}$
S.1	Minimum space between two <code>metal1</code> regions = 0.23 $\mu\text{m}$
W.2	Minimum and maximum width of <code>cut1</code> = 0.26 $\mu\text{m}$
E.1	Minimum extension of <code>metal1</code> beyond <code>cut1</code> = 0.01 $\mu\text{m}$
W.3	Minimum width of <code>metal3</code> = 0.28 $\mu\text{m}$
S.2	Minimum space between two <code>metal3</code> regions = 0.28 $\mu\text{m}$
W.4	Minimum and maximum width of <code>cut2</code> = 0.26 $\mu\text{m}$
E.2	Minimum extension of <code>metal1</code> beyond <code>cut2</code> = 0.01 $\mu\text{m}$



Although the minimum `metal1` routing pitch is 0.485 $\mu\text{m}$  from the design rule, you should use 0.56 $\mu\text{m}$  instead, to match the `metal3` routing pitch in the same preferred direction.

## LEF Construct No. 1

```
LAYER metal1
  TYPE ROUTING ;
  WIDTH 0.23 ;
  SPACING 0.23 ;
  PITCH 0.56 ;
  DIRECTION HORIZONTAL ;
```

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

```
END metal1
```

```
LAYER metal3
    TYPE ROUTING ;
    WIDTH 0.28 ;
    SPACING 0.28 ;
    PITCH 0.56 ;
    DIRECTION HORIZONTAL ;

END metal3
```

### Recommendations

- Use line-to-via spacing for both the horizontal and vertical direction.
- Allow diagonal vias with the routing pitch.
- Align the routing pitch for `metal1` and `metal2`, with the pins inside the standard cells.
- Have uniform routing pitch in the same preferred direction. The pitch ratio should be 2 - 3 or 1 - 2. It is better to define the `metal1` pitch larger than necessary in order to achieve a 1 - 1 ratio because the `metal1` width is usually smaller the `metal2` and `metal3` widths.

### Pitch Recommendations for Library Development

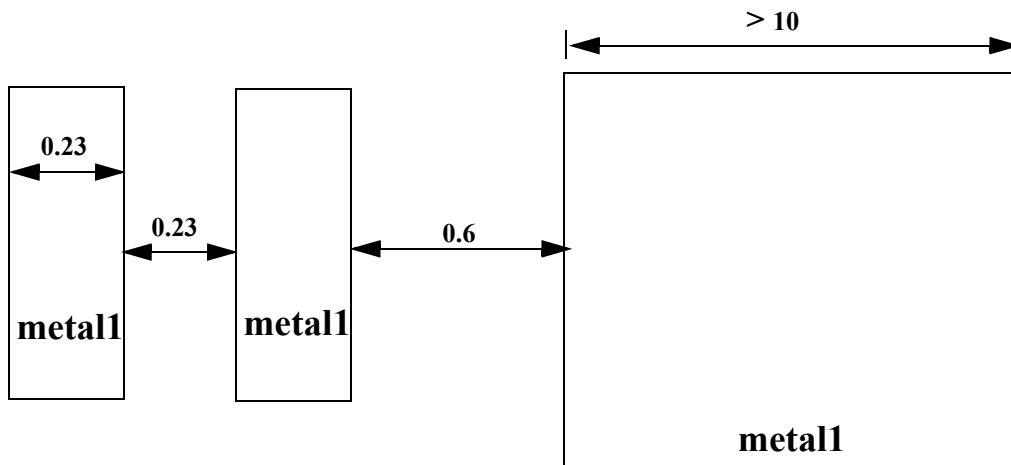
- All pins should be on the grid, and only those portions of the pins that are accessible to the router should be modeled as pins. For example, 45 degree pin geometry.
- The height of the cell should be the even multiple of the `metal1` pitch, and the width of the cell should be the even multiple of the `metal2` pitch.
- The blockage modeling, especially for `metal1`, should be simplified as much as possible. For example, it is very common for the entire area within the cell boundary to be obstructed in `metal1`, so use a single rectangular blockage instead of many small blockages.

## Guidelines for Wide Metal Spacing

The `SPACING` statement in the LEF `LAYER` section is applied to both regular and special wires. You can use the Cadence® ultra router option `frouteUseRangeRule` to determine which objects to check against the `SPACING RANGE` statement. The default checks both pin and obstruction.

## DESIGN RULE No. 2

- S.1 Minimum space between two `metal1` regions = 0.23  $\mu\text{m}$
- S.2 Minimum space between metal lines with one or both metal line width and length are greater than 10 $\mu\text{m}$  = 0.6  $\mu\text{m}$



## LEF CONSTRUCT No. 2

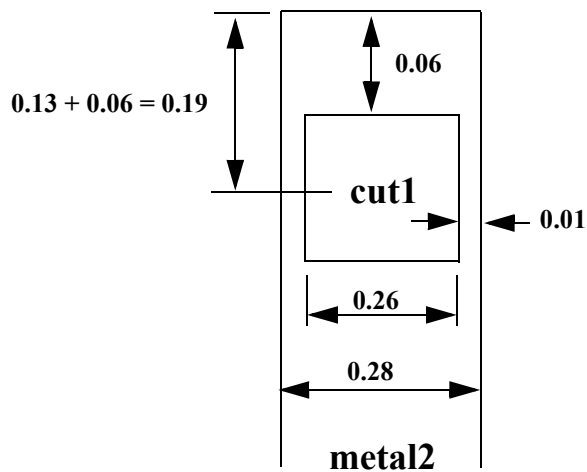
```
LAYER metal1
    WIDTH 0.23 ;
    SPACING 0.23 ;
    SPACING 0.6 RANGE 10.002 1000 ;
END metal1
```

## Guidelines for Wire Extension at Vias

The following guidelines are for wire extension at vias.

### DESIGN RULE No. 3

- W.1            Minimum and maximum width of `cut1` = 0.26  $\mu\text{m}$
- W.2            Minimum width of `metal2` = 0.28  $\mu\text{m}$
- E.1            Minimum extension of `metal2` beyond `cut1` = 0.01  $\mu\text{m}$
- E.2            Minimum extension of `metal2` end-of-line region beyond `cut1` = 0.06  $\mu\text{m}$



### LEF CONSTRUCT No. 3

```

LAYER metal2
    TYPE ROUTING ;
    WIDTH 0.28 ;
    SPACING 0.28 ;
    PITCH 0.56 ;
    WIREEXTENSION 0.19 ;
    DIRECTION VERTICAL ;

END metal2

VIA via23 DEFAULT
    LAYER metal2 ;
    RECT -0.14 -0.14 0.14 0.14 ;           # Use square via
    LAYER cut2 ;
    RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal3 ;
    RECT -0.14 -0.14 0.14 0.14 ;           # Use square via
    
```

END via23

## Recommendations

- Use the `WIREEXTENSION` statement instead of defining multiple vias because the width of the `metal2` in `cut1` is the same as the default routing width of the `metal2` layer.
- The `WIREEXTENSION` statement only extends wires and not vias. For 65nm and below, `WIREEXTENSION` is no longer recommended because it may generate some advance rule violations if wires and vias have different widths.
- Define the `DEFAULT VIA` as a square via.

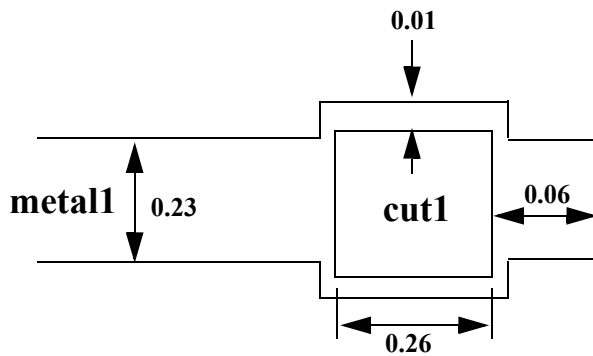
## Guidelines for Default Vias

The following guidelines are for default vias.

#### DESIGN RULE No. 4

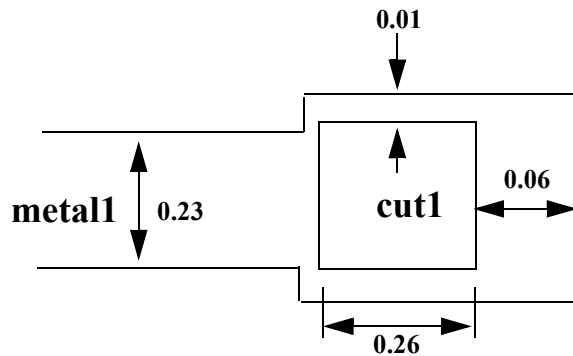
- W.1            Minimum width of `metal1` = 0.23  $\mu\text{m}$   
W.2            Minimum and maximum width of `cut1` = 0.26  $\mu\text{m}$   
E.1            Minimum extension of `metal1` beyond `cut1` = 0.01  $\mu\text{m}$   
E.2            Minimum extension of `metal1` end-of-line region beyond `cut1` = 0.06  $\mu\text{m}$

**Case A:**



Use WIREEXTENSION and  
square DEFAULT VIA

**Case B:**



Use Horizontal and Vertical DEFAULT VIAS

#### LEF CONSTRUCT No. 4 (Case B)

```
LAYER metal1
  TYPE ROUTING ;
  WIDTH 0.23 ;
  SPACING 0.23 ;
  PITCH 0.56 ;
  DIRECTION HORIZONTAL ;

END metal1

VIA via12_H DEFAULT
  LAYER metal1 ;
  RECT -0.19 -0.14 0.19 0.14 ;          # metal1 end-of-line
```



## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

```
        extension 0.6 in both directions
    LAYER cut1 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal2 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via12_H

VIA via12_V DEFAULT
    LAYER metall ;
        RECT -0.14 -0.19 0.14 0.19 ;           # metall end-of-line
        extension 0.6 in both directions
    LAYER cut1 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal2 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via12_V
```

### Recommendations

- If the width of the end-of-line metal extension is the same as the default metal routing width, as in Case A, use the `WIREEXTENSION` statement in the LEF `LAYER` section, and define a square via in the `DEFAULT VIA` section.
- If the width of the end-of-line metal extension is the same as the width of the via metal, as in Case B, define one horizontal `DEFAULT VIA` and one vertical `DEFAULT VIA` to cover the required metal extension area in both preferred and non-preferred routing directions. Do not use the `WIREEXTENSION` statement in the LEF `LAYER` section.

## Guidelines for Stack Vias (MAR Vias) and Samenet Spacing

The following guidelines are for stack vias (minimum area rule) and `SAMENET SPACING`.

**DESIGN RULE No. 5**

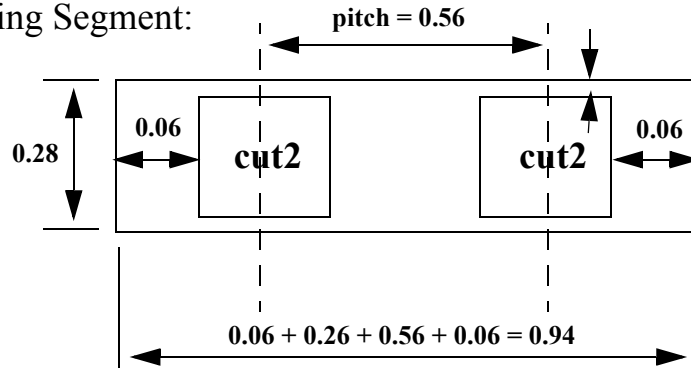
- |     |                                                                                                                           |
|-----|---------------------------------------------------------------------------------------------------------------------------|
| W.1 | Minimum width of <code>metal2</code> = 0.28 $\mu\text{m}$                                                                 |
| W.2 | Minimum and maximum width of <code>cut2</code> = 0.26 $\mu\text{m}$                                                       |
| E.1 | Minimum extension of <code>metal2</code> beyond <code>cut2</code> = 0.01 $\mu\text{m}$                                    |
| A.1 | Minimum area of <code>metal2</code> = 0.2025 $\mu\text{m}$                                                                |
| C.1 | <code>Cut2</code> can be fully or partially stacked on <code>cut1</code> , contact or any combination                     |
| W.1 | Minimum width of <code>metal3</code> = 0.28 $\mu\text{m}$                                                                 |
| W.2 | Minimum and maximum width of <code>cut3</code> = 0.26 $\mu\text{m}$                                                       |
| E.1 | Minimum extension of <code>metal2</code> beyond <code>cut3</code> = 0.01 $\mu\text{m}$                                    |
| A.1 | Minimum area of <code>metal3</code> = 0.2025 $\mu\text{m}$                                                                |
| C.1 | <code>Cut3</code> can be fully or partially stacked on <code>cut2</code> , <code>cut1</code> , contact or any combination |

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

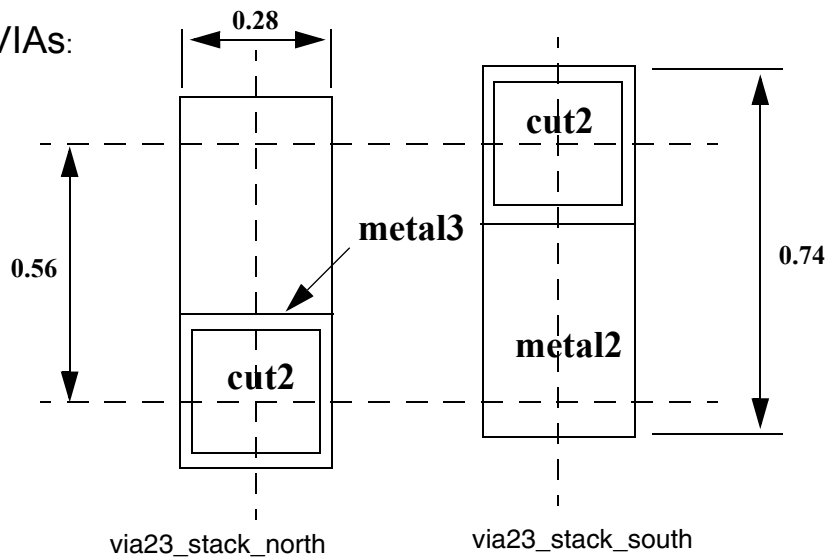
---

Default Routing Segment:



Minimum routing area of metal3 =  $0.28 \times 0.94 = 0.2632 > 0.25$  (MAR)

MAR VIAs:



## LEF CONSTRUCT No. 5

```
VIA via23_stack_north DEFAULT
  LAYER metal2 ;
  RECT -0.14 -0.14 0.14 0.6 ;    # MAR = 0.28 x 0.74
  LAYER cut2 ;
```

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

```
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal3 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via23_stack_north

VIA via23_stack_south DEFAULT
    LAYER metal2 ;
        RECT -0.14 -0.6 0.14 0.14 ;    # MAR = 0.28 x 0.74
    LAYER cut2 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal3 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via23_stack_south

VIA via34_stack_east DEFAULT
    LAYER metal3 ;
        RECT -0.14 -0.14 0.6 0.14 ;    # MAR = 0.28 x 0.74
    LAYER cut3 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal4 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via34_stack_east

VIA via34_stack_west DEFAULT
    LAYER metal3 ;
        RECT -0.6 -0.14 0.14 0.14 ;    # MAR = 0.28 x 0.74
    LAYER cut3 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal4 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via34_stack_west
```

## Recommendations

- The minimum metal routing segment (two vias between one pitch grid) with or without end-of-line metal extension should automatically satisfy the minimum area rule.
- If vias are stackable, create the `TOPSTACKONLY` vias with a rectangular shape blocking only one neighboring grid for both sides of the preferred routing direction. In other words, one north oriented and one south oriented for vertical-preferred routing layers, and one east oriented and one west oriented for horizontal-preferred routing layers.

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

- Use slightly larger dimensions for the via size to make them an even number, so they snap to the manufacturing grids.
- The `STACK` keyword in the `SAMENETSPACING` statements only allows vias to be fully overlapped (stacked) by `SROUTE` commands. To allow vias to be partially overlapped, set the environment variable `SROUTE.ALLOWOVERLAPINSTACKVIA` to `TRUE`.
- The `metal1` layer does not require a MAR via because all `metal1` pins should satisfy the minimum area rules.

## Example of an Optimized LEF Technology File

```
VERSION 5.8 ;

BUSBITCHARS "[]" ;

UNITS
    DATABASE MICRONS 100 ;
END UNITS

LAYER metal1
    TYPE ROUTING ;
    WIDTH 0.23 ;
    SPACING 0.23 ;
    SPACING 0.6 RANGE 10.02 1000 ;
    PITCH 0.56 ;
    DIRECTION HORIZONTAL ;
END metal1

LAYER cut1
    TYPE CUT ;
END cut1

LAYER metal2
    TYPE ROUTING ;
    WIDTH 0.28 ;
    SPACING 0.28 ;
    SPACING 0.6 RANGE 10.02 1000 ;
    PITCH 0.56 ;
    WIREEXTENSION 0.19 ;
    DIRECTION VERTICAL ;
END metal2
```

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

```
LAYER cut2
    TYPE CUT ;
END cut2
```

```
LAYER metal3
    TYPE ROUTING ;
    WIDTH 0.28 ;
    SPACING 0.28 ;
    SPACING 0.6 RANGE 10.02 1000 ;
    PITCH 0.56 ;
    WIREEXTENSION 0.19 ;
    DIRECTION HORIZONTAL ;
END metal3
```

```
LAYER cut3
    TYPE CUT ;
END cut3
```

```
LAYER metal4
    TYPE ROUTING ;
    WIDTH 0.28 ;
    SPACING 0.28 ;
    SPACING 0.6 RANGE 10.02 1000 ;
    PITCH 0.56 ;
    WIREEXTENSION 0.19 ;
    DIRECTION VERTICAL ;
END metal4
```

```
LAYER cut4
    TYPE CUT ;
END cut4
```

```
LAYER metal5
    TYPE ROUTING ;
    WIDTH 0.28 ;
    SPACING 0.28 ;
    SPACING 0.6 RANGE 10.02 1000 ;
```

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

```
PITCH 0.56 ;
WIREEXTENSION 0.19 ;
DIRECTION HORIZONTAL ;

END metal5
```

```
LAYER cut5
    TYPE CUT ;

END cut5
```

```
LAYER metal6
    TYPE ROUTING ;
    WIDTH 0.44 ;
    SPACING 0.46 ;
    SPACING 0.6 RANGE 10.02 1000 ;
    PITCH 1.12 ;
    DIRECTION VERTICAL ;

END metal6
```

#### ### start DEFAULT VIA ###

```
VIA via12_H DEFAULT
    LAYER metall ;
        RECT -0.19 -0.14 0.19 0.14 ;      # metall end-of-line ext 0.6
    LAYER cut1 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal2 ;
        RECT -0.14 -0.14 0.14 0.14 ;

END via12_H
```

```
VIA via12_V DEFAULT
    LAYER metall ;
        RECT -0.14 -0.19 0.14 0.19 ;      # metall end-of-line ext 0.6
    LAYER cut1 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal2 ;
        RECT -0.14 -0.14 0.14 0.14 ;

END via12_V
```

```
VIA via23 DEFAULT
    LAYER metal2 ;
        RECT -0.14 -0.14 0.14 0.14 ;
    LAYER cut2 ;
```

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

```
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal3 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via23
```

```
VIA via34 DEFAULT
    LAYER metal3 ;
        RECT -0.14 -0.14 0.14 0.14 ;
    LAYER cut3 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal4 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via34
```

```
VIA via45 DEFAULT
    LAYER metal4 ;
        RECT -0.14 -0.14 0.14 0.14 ;
    LAYER cut4 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal5 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via45
```

```
VIA via56_H DEFAULT
    LAYER metal5 ;
        RECT -0.24 -0.19 0.24 0.19 ;
    LAYER cut5 ;
        RECT -0.18 -0.18 0.18 0.18 ;
    LAYER metal6 ;
        RECT -0.27 -0.27 0.27 0.27 ;
END via56_H
```

```
VIA via56_V DEFAULT
    LAYER metal5 ;
        RECT -0.19 -0.24 0.19 0.24 ;
    LAYER cut5 ;
        RECT -0.18 -0.18 0.18 0.18 ;
    LAYER metal6 ;
        RECT -0.27 -0.27 0.27 0.27 ;
END via56_V
```

### end DEFAULT VIA ###



## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

#### ### start STACK VIA ###

```
VIA via23_stack_north DEFAULT
    LAYER metal2 ;
        RECT -0.14 -0.14 0.14 0.6 ;    # MAR = 0.28 x 0.74
    LAYER cut2 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal3 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via23_stack_north
```

```
VIA via23_stack_south DEFAULT
    LAYER metal2 ;
        RECT -0.14 -0.6 0.14 0.14 ;    # MAR = 0.28 x 0.74
    LAYER cut2 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal3 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via23_stack_south
```

```
VIA via34_stack_east DEFAULT
    LAYER metal3 ;
        RECT -0.14 -0.14 0.6 0.14 ;    # MAR = 0.28 x 0.74
    LAYER cut3 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal4 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via34_stack_east
```

```
VIA via34_stack_west DEFAULT
    LAYER metal3 ;
        RECT -0.6 -0.14 0.14 0.14 ;    # MAR = 0.28 x 0.74
    LAYER cut3 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal4 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via34_stack_west
```

```
VIA via45_stack_north DEFAULT
    LAYER metal4 ;
        RECT -0.14 -0.14 0.14 0.6 ;    # MAR = 0.28 x 0.74
    LAYER cut4 ;
```

## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

```
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal5 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via45_stack_north

VIA via45_stack_south DEFAULT
    LAYER metal4 ;
        RECT -0.14 -0.6 0.14 0.14 ;    # MAR = 0.28 x 0.74
    LAYER cut4 ;
        RECT -0.13 -0.13 0.13 0.13 ;
    LAYER metal5 ;
        RECT -0.14 -0.14 0.14 0.14 ;
END via45_stack_south

VIA via56_stack_east DEFAULT
    LAYER metal5 ;
        RECT -0.19 -0.19 0.35 0.19 ;    # MAR = 0.38 x 0.54
    LAYER cut5 ;
        RECT -0.18 -0.18 0.18 0.18 ;
    LAYER metal6 ;
        RECT -0.27 -0.27 0.27 0.27 ;
END via56_stack_east

VIA via56_stack_west DEFAULT
    LAYER metal5 ;
        RECT -0.35 -0.19 0.19 0.19 ;    # MAR = 0.38 x 0.54
    LAYER cut5 ;
        RECT -0.18 -0.18 0.18 0.18 ;
    LAYER metal6 ;
        RECT -0.27 -0.27 0.27 0.27 ;
END via56_stack_west

### end STACK VIA ###
```

---

## Calculating and Fixing Process Antenna Violations

---

This appendix describes process antenna violations and how you can use the router to correct them. It includes the following sections:

- [Overview](#) on page 812
- [Using Process Antenna Keywords in the LEF and DEF Files](#) on page 816
- [Calculating Antenna Ratios](#) on page 817
- [Checking for Antenna Violations](#) on page 834
- [Using Antenna Diode Cells](#) on page 845
- [Using DiffUseOnly](#) on page 846
- [Calculations for Hierarchical Designs](#) on page 847

## Overview

During deep submicron wafer fabrication, gate damage can occur when excessive static charges accumulate and discharge, passing current through a gate. If the area of the layer connected directly to the gate or connected to the gate through lower layers is large relative to the area of the gate and the static charges are discharged through the gate, the discharge can damage the oxide that insulates the gate and cause the chip to fail. This phenomenon is called the *process antenna effect* (PAE).

To determine the extent of the PAE, the router calculates the area of the layer relative to the area of the gates connected to it, or connected to it through lower layers. The number it calculates is called the *antenna ratio*. Each foundry sets a maximum allowable antenna ratio for the chips it fabricates.

For example, assume a foundry sets a maximum allowable antenna ratio of 500. If a net has two input gates that each have an area of 1 square micron, any metal layers that connect to the gates and have an area larger than 1,000 square microns have process antenna violations because they would cause the antenna ratio to be higher than 500:

$$\text{Antenna Ratio} = \frac{\text{Area of metal layer}}{\text{Area of gates}} \quad 500 = \frac{1000}{1 + 1}$$

To tell the router the values to use when it calculates the antenna ratio, you set antenna keywords in the LEF and DEF files. The router measures potential damage caused by PAE by checking the ratio it calculates against the values specified by the antenna keywords. When it finds a net whose antenna ratio for a specified layer exceeds the maximum allowed value for that layer, it finds a *process antenna violation* and attempts to fix it using one or both of the following methods:

- Changing the routing so the routing layers connected to a gate or connected to a gate through lower layers are not so large that they build enough static charge to damage the gate
- Inserting diodes that protect the gate by providing an alternate path to discharge the static charge

LEF can specify several types of antenna ratios, including ratios for PAE damage on one layer only and ratios calculated by adding accumulated damage on several layers. In addition, LEF can specify ratios based on the area of the metal wires or the cut area of vias.

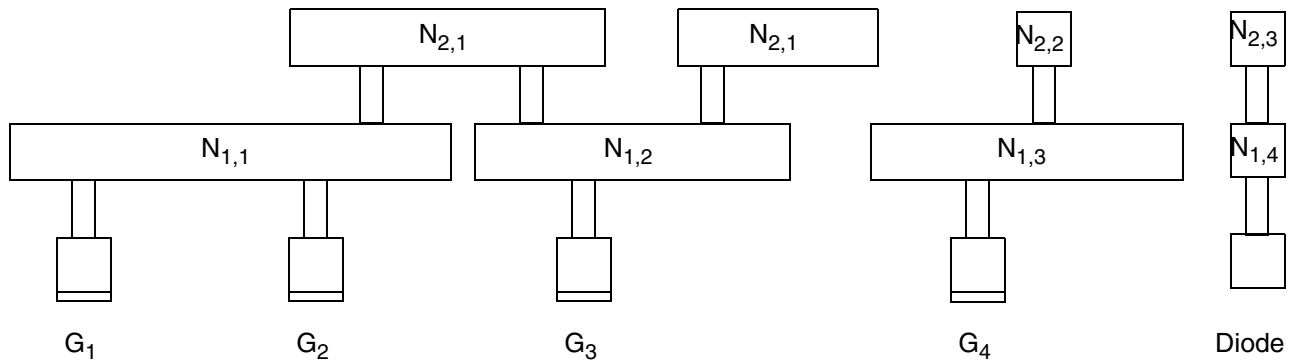
## What Are Process Antennas?

In a chip manufacturing process, metal layers are built up, layer by layer, starting with the first-level metal layer (usually referred to as *metal1*). Next, the *metal1-metal2* vias are created, then the second-level metal layer, then *metal2-metal3* vias, and so on.

On each metal layer, metal is initially deposited so it covers the entire chip. Then, the unneeded portions of the metal are removed by etching, typically in plasma (charged particles).

Figure C-1 on page 813 shows a section of an imaginary chip after the unneeded metal from *metal2* is removed.

**Figure C-1**



In the figure,

- Gate areas for transistors are labelled  $G_k$ , where  $k$  is a sequential number starting with 1.
- Wire segments are labelled  $N_{i,j}$ 
  - $N$  signifies that the wire segment is an electrically connected node
  - $i$  specifies the metal layer to which the node belongs
  - $j$  is a sequential number for the node on that metal layer
- Nodes are labelled so that all pieces of the metal geometry on layer *metal<sub>i</sub>* that are electrically connected by conductors at layers below *metal<sub>i</sub>* belong to the same node. For example, the two *metal2* wire segments that belong to node  $N_{2,1}$  are electrically connected to gates  $G_1$ ,  $G_2$ , and  $G_3$  by a piece of wire on *metal1* (labelled  $N_{1,2}$ ).

Thick oxide insulates the already-fabricated structures below *metal2*, preventing them from direct contact with the plasma. The *metal2* geometries, however, are exposed to the plasma,

and collect charge from it. As the metal geometries collect charge, they build up voltage potential.

Because the metal geometries collect charge during the metallization process, they are referred to as process antennas. In general, the more area covered by the metal geometries that are exposed to the plasma (that is, the larger the process antennas), the more charge they can collect.

In [Figure C-1](#) on page 813, note the following:

- Node  $N_{1,1}$  is electrically connected to gates  $G_1$  and  $G_2$ .
- Node  $N_{1,2}$  is electrically connected to gate  $G_3$ .
- Node  $N_{2,1}$  (node  $N_{2,1}$  has two pieces of metal) is electrically connected to gates  $G_1$ ,  $G_2$ , and  $G_3$ .
- Node  $N_{1,3}$  and node  $N_{2,2}$  are electrically connected to gate  $G_4$ .
- Node  $N_{1,4}$  and node  $N_{2,3}$  are electrically connected to the diffusion (diode).

## What Is the Process Antenna Effect (PAE)?

If the voltage potential across the gate oxide becomes large enough to cause current to flow across the gate oxide, from the process antennas to the gates to which the process antennas are electrically connected, the current can damage the gate oxide. The process antenna effect (PAE) is the term used to describe the build-up of charge and increase in voltage potential. The larger the total gate area that is electrically connected to the process antennas on a specific layer, the more charge the connected gates can withstand.

In the imaginary chip in [Figure C-1](#) on page 813, if the current were to flow, the following would happen, as a result of the node-gate connections:

- The charge collected by process antennas on nodes  $N_{1,1}$ ,  $N_{1,2}$ , and  $N_{2,1}$  would be discharged through one or more of gates  $G_1$ ,  $G_2$ , and  $G_3$ .
- The charge collected by process antennas on nodes  $N_{1,3}$  and  $N_{2,2}$  would be discharged through gate  $G_4$ .
- The charge collected by process antennas on node  $N_{1,4}$  and  $N_{2,3}$  would be discharged through the diode.

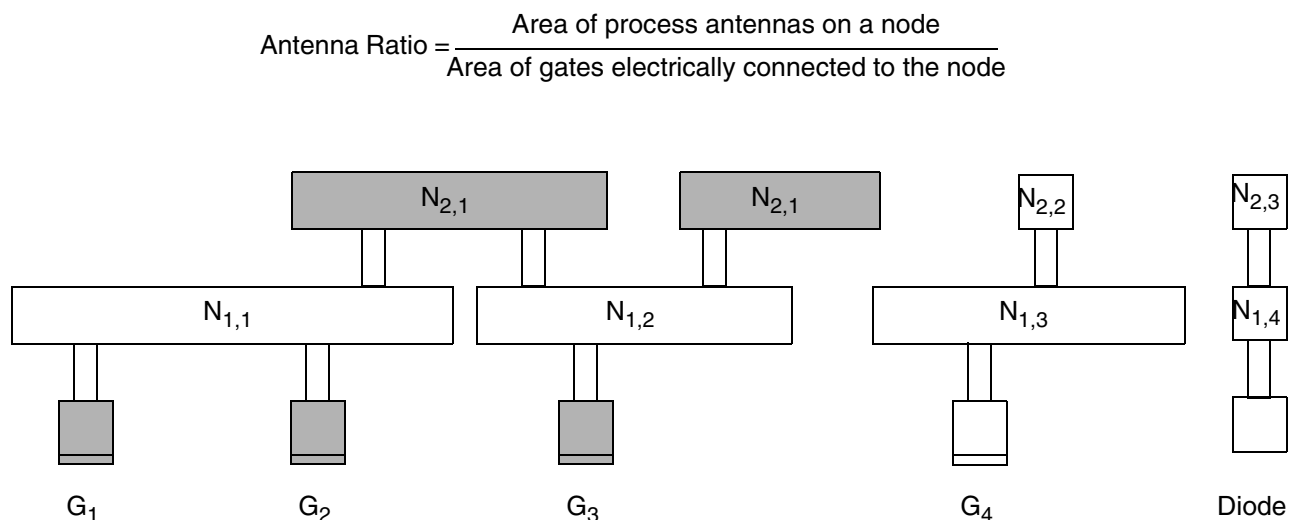
## What Is the Antenna Ratio?

Because the total gate area that is electrically connected to a node (and therefore connected to the process antennas) determines the amount of charge from the process antennas the electrically connected gates can withstand, and because the size of the process antennas connected to the node determines how much charge the antennas collect, it is useful to calculate the ratio of the size of the process antennas on a node to the size of the gate area that is electrically connected to the node. This is the antenna ratio. The greater the antenna ratio, the greater the potential for damage to the gate oxide.

If you check a chip and obtain an antenna ratio greater than the threshold specified by the foundry, gate damage is likely to occur.

Figure C-2 on page 815 shows the same section of the imaginary chip as the previous figure. The shaded areas in this figure represent the process antennas on node  $N_{2,1}$  and the gates to which they connect: gates  $G_1$ ,  $G_2$ , and  $G_3$ . The shaded gates discharge the electricity collected by the process antennas on node  $N_{2,1}$ .

**Figure C-2**



## What Can Be Done to Improve the Antenna Ratio?

If there is an alternate path for the current to flow, the charge on the node can be discharged through the alternate path before the voltage potential reaches a level that damages the gate. For example, a Zener diode, which allows current to flow in the reverse direction when the reverse bias reaches a specified breakdown voltage, provides an alternate path, and helps avoid building up so much charge at the node that the charge is discharged through the gate

oxide. Diffusion features that form the output of a logic gate (source and drain of transistors) can provide such an alternate discharge path.

Routers typically use two methods to decrease the antenna ratio:

- Changing the routing by breaking the metal layers into smaller pieces
- Inserting antenna diode cells to discharge the current

Both of these methods supply alternate paths for the current. For details about how to specify antenna diode cells, see [“Using Antenna Diode Cells”](#) on page 845.

## Using Process Antenna Keywords in the LEF and DEF Files

You tell the router the values to use for the gate, diffusion, and metal areas by setting values for process antenna keywords in the LEF and DEF files for your design. You also tell the router the values to use for the threshold process antenna ratios by setting the keywords.

The following table lists LEF version 5.5 antenna keywords.

If the keyword ends with ...	It refers to ...	Examples
area	Area of the gates or diffusion  Measured in square microns	ANTENNADIFFAREA ANTENNAGATEAREA
factor	Area multiplier used for the metal nodes	ANTENNAAREAFactor ANTENNASIDEAREAFactor
<b>Note:</b> Use <code>DIFFUSEONLY</code> if you want the multiplier to apply only when connecting to diffusion. For more information, see <a href="#">“Using DiffUseOnly”</a> on page 846.		



If the keyword ends with ...	It refers to ...	Examples
ratio	Relationship the router is calculating  Cum is used in keywords for cumulative antenna ratio.	ANTENNAAREARATIO ANTENNASIDEAREARATIO ANTENNADIFFAREARATIO ANTENNADIFFSIDEAREARATIO ANTENNACUMAREARATIO ANTENNACUMSIDEAREARATIO ANTENNACUMDIFFAREARATIO ANTENNACUMDIFFSIDEAREARATIO

## Calculating Antenna Ratios

Tools should calculate antenna ratios using one of the following models:

- The partial checking model

Using this model, you calculate damage to gates by process antennas on one layer. For example, if you use the partial checking model to calculate the PAE referred to a gate from *metal3*, you do not consider any potential damages referred to that gate from metallization steps on *metal1* or *metal2*.

You use this model to calculate a partial antenna ratio (PAR). A PAR tells you if any single metallization step is likely to inflict damage to a gate.

- The cumulative checking model

This model is more conservative than the partial checking model. It adds damage to a gate caused by the PAE referred to the gate from each metallization step, starting from *metal1* up to the layer that is being checked. For example, if you use the cumulative checking model to calculate the PAE referred to a gate from *metal3*, you add the PAR from the relevant antenna areas on *metal1*, *metal2*, and *metal3*.

You use this model to calculate a cumulative antenna ratio (CAR). A CAR adds the damages on successive layers together to accumulate them as the layers are built up.

## Calculating the Antenna Area

The area used to model the charge-collecting ability of a node is called the *antenna area*. The router calculates the antenna area for one of the following areas:

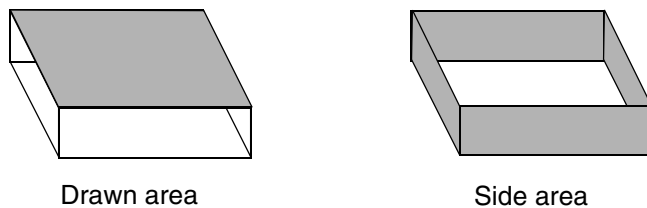
- The drawn area (the top surface area of the metal shape)

- The side area (the area of the sides of the metal shape)

The height of each side is taken from the `THICKNESS` statement for that layer.

Figure C-3 on page 818 shows drawn and side areas.

**Figure C-3**



## Antenna Area Factor

You can increase or decrease the calculated antenna area by specifying an antenna area factor in the LEF file.

- Use `ANTENNAAREAFactor` to adjust the calculation of the drawn area.
- Use `ANTENNASIDEAREAFactor` to adjust the calculation of the side area.

The default value of both factors is 1.

The final ratio check can be scaled (that is, made more or less pessimistic) by using the `ANTENNAAREAFactor` or `ANTENNASIDEAREAFactor` values that are used to multiply the final PAR and CAR values.

**Note:** The LEF and DEF `ANTENNA` values are always unscaled values; only the final ratio-check is affected by the scale factors.

## Calculating a PAR

The general  $PAR(m_1)$  equation for a single layer is calculated as:

$$PAR(m_1) = \frac{\{(metalFactor \times metal\_area) \times diffMetalReduceFactor - minusDiffFactor \times diff\_area\}}{(gate\_area + plusDiffFactor \times diff\_area)}$$

The existing `ANTENNAAREAFactor` statement is shown as *metalFactor* for the metal area. It has no effect on the `diff_area`, `gate_area`, or `cut_area` shown. Likewise, the `ANTENNAAREADIFFREDUCEPWL` statement is shown as *diffMetalReduceFactor*, the

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

ANTENNAAREAMINUSDIFF statement is shown as *minusDiffFactor*, and the ANTENNAGATEPLUSDIFF statement is shown as *plusDiffFactor*. For cut layer, the ratio equation illustrates the effect of an ANTENNAAREAFACOR *cutFactor* statement as *metalFactor*. If there is no preceding ANTENNAAREAFACOR statement, the *metalFactor* value defaults to 1.0.

For single layer rules, the PAR value is compared to ANTENNA[*SIDE*]AREARATIO and/or ANTENNADIFF[*SIDE*]AREARATIO, as appropriate. For cumulative layer rules, the CAR values is compared to ANTENNACUM[*SIDE*]AREARATIO and/or ANTENNACUMDIFF[*SIDE*]AREARATIO, as appropriate.

The following example uses a simplified formula to calculate a PAR, without including the various area factors:

$$\text{PAR}(N_{i,j}, G_k) = \frac{\text{Area}(N_{i,j})}{\sum_{G_k \in C(N_{i,j})} \text{Area}(G_k)}$$

$\text{PAR}(N_{i,j}, G_k)$  is the partial antenna ratio for node  $j$  on *metal<sub>i</sub>* with respect to gate  $G_k$ , where  $G_k$  is electrically connected to node  $N_{i,j}$  by layer  $i$  or below.

$\text{Area}(N_{i,j})$  is the drawn or side area of node  $N_{i,j}$ .

$C(N_{i,j})$  is the set of gates  $G_k$  that are electrically connected to  $N_{i,j}$  through the layers below *metal<sub>i</sub>*.

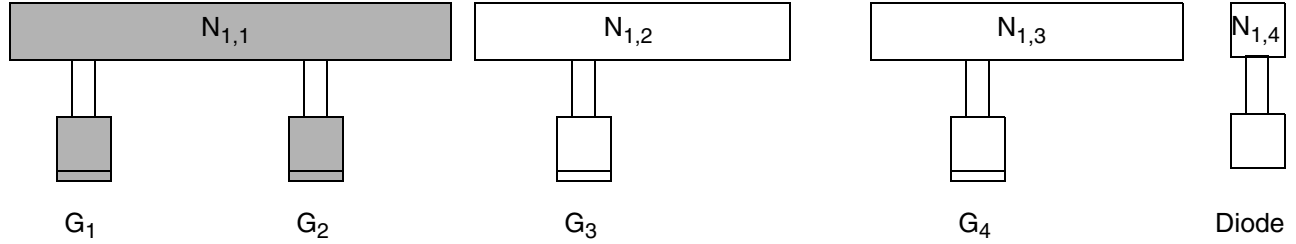
$\text{Area}(G_k)$  is the drawn or side area of gate  $G_k$ . (The reason to include the  $G_k$  parameter for PAR is to maintain uniformity with the notation for CAR.)

**Note:** For a specified node  $N_{i,j}$ , the  $\text{PAR}(N_{i,j}, G_k)$  for all gates  $G_k$  that are connected to the node  $N_{i,j}$  using *metal<sub>i</sub>* or below are identical.

### Calculations for PAR on the First Metal Layer

Figure C-4 on page 820 shows a section of an imaginary chip after the first metal layer is processed.

**Figure C-4**



The shaded areas in the figure represent the wire segment and the gates whose areas you must compute to evaluate the formula below.

To calculate  $PAR(N_{i,j}, G_k)$  for node  $N_{1,1}$ , a node on the first metal layer, with respect to gate  $G_1$ , use the following formula:

$$PAR(N_{1,1}, G_1) = \frac{Area(N_{1,1})}{Area(G_1) + Area(G_2)}$$

Because gates  $G_1$  and  $G_2$  both connect to node  $N_{1,1}$ , the following statement is true:

$$PAR(N_{1,1}, G_1) = PAR(N_{1,1}, G_2)$$

To calculate  $PAR$  for node  $N_{1,2}$ , another node on the first metal layer, with respect to gate  $G_3$ , use the following formula:

$$PAR(N_{1,2}, G_3) = \frac{Area(N_{1,2})}{Area(G_3)}$$

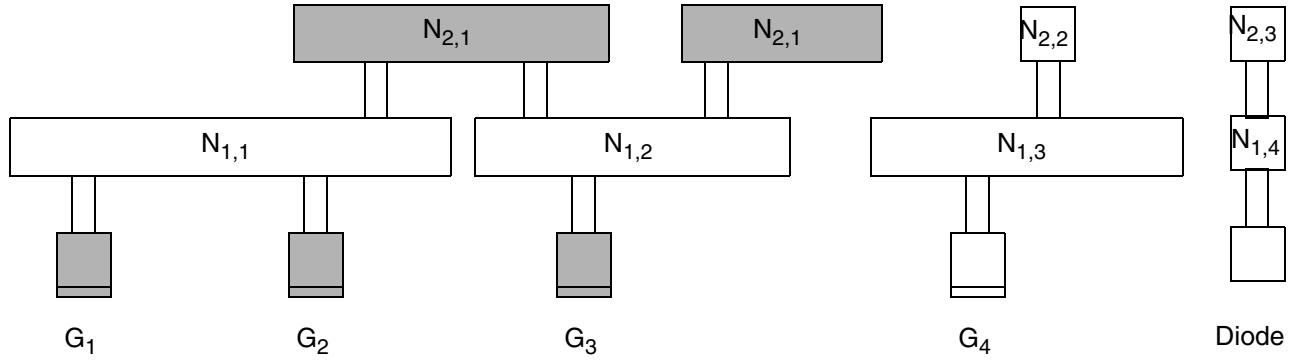
To calculate  $PAR(N_{i,j}, G_k)$  for node  $N_{1,3}$ , another node on the first metal layer, with respect to gate  $G_4$ , use the following formula:

$$PAR(N_{1,3}, G_4) = \frac{Area(N_{1,3})}{Area(G_4)}$$

### Calculations for PAR on the Second Metal Layer

Figure C-5 on page 821 shows the chip after the second metal layer is processed.

**Figure C-5**



The shaded areas in the figure represent the wire segments and the gates whose areas you must compute to evaluate the formula below.

$N_{2,1}$  consists of two pieces of metal on the second layer that are electrically connected at this step in the fabrication process. Therefore, to calculate  $\text{PAR}(N_{2,1}, G_1)$ , you must add the area of both pieces together.

To calculate  $\text{PAR}(N_{i,j}, G_k)$  for node  $N_{2,1}$ , a node on the second metal layer, with respect to gate  $G_1$ , use the following formula:

$$\text{PAR}(N_{2,1}, G_1) = \frac{\text{Area}(N_{2,1})}{\text{Area}(G_1) + \text{Area}(G_2) + \text{Area}(G_3)}$$

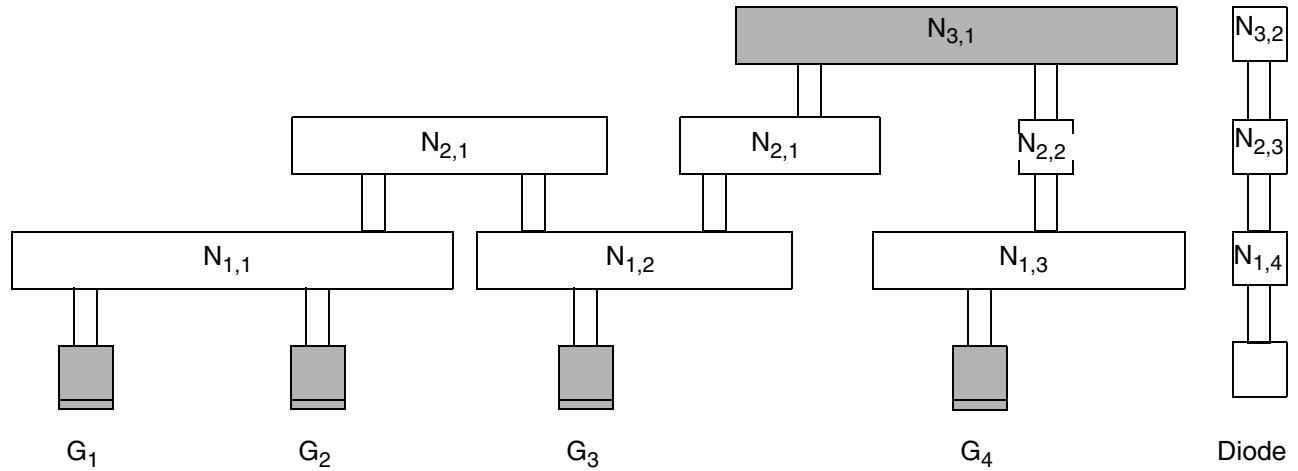
As on the first layer,

$$\text{PAR}(N_{2,1}, G_1) = \text{PAR}(N_{2,1}, G_2) = \text{PAR}(N_{2,1}, G_3)$$

### Calculations for PAR on the Third Metal Layer

[Figure C-6](#) on page 822 shows the chip after the third metal layer is processed.

**Figure C-6**



The shaded areas in the figure represent the wire segment and the gates whose areas you must compute to evaluate the formula below.

To calculate  $PAR(N_{i,j}, G_k)$  for node  $N_{3,1}$ , a node on the third metal layer, with respect to gate  $G_1$ , use the following formula:

$$PAR(N_{3,1}, G_1) = \frac{Area(N_{3,1})}{Area(G_1) + Area(G_2) + Area(G_3) + Area(G_4)}$$

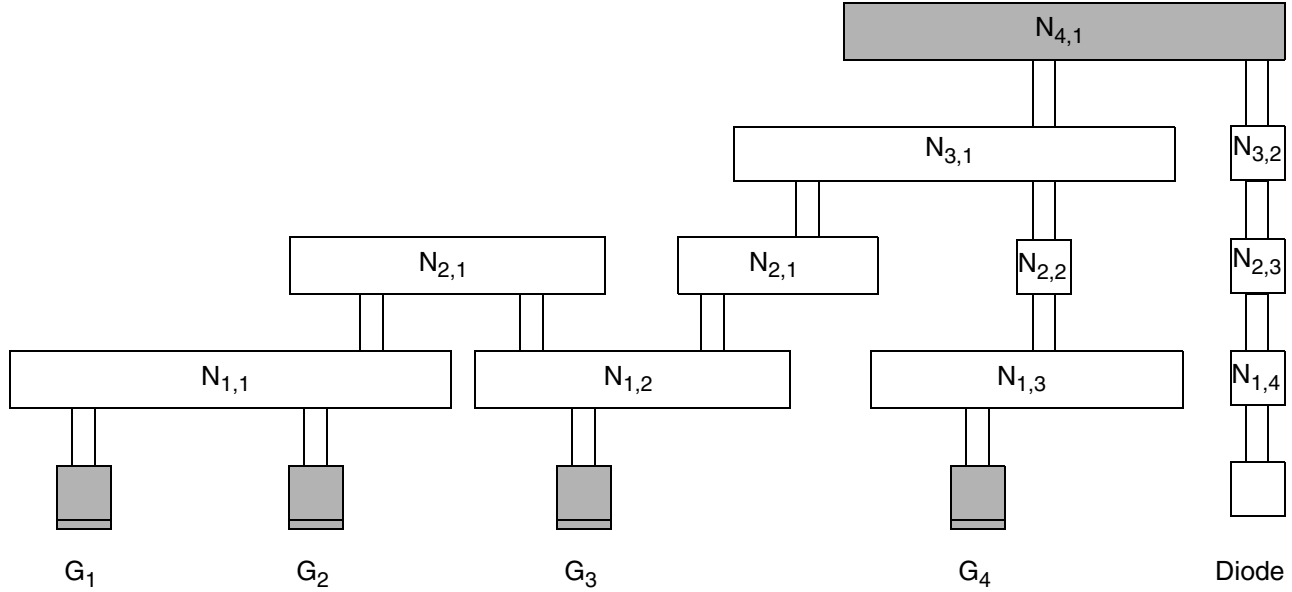
As on the prior layers,

$$PAR(N_{3,1}, G_1) = PAR(N_{3,1}, G_2) = PAR(N_{3,1}, G_3) = PAR(N_{3,1}, G_4)$$

### Calculations for PAR on the Fourth Metal Layer

Figure C-7 on page 823 shows the chip after the fourth metal layer is processed.

**Figure C-7**



To calculate  $PAR(N_{i,j}, G_k)$  for the fourth metal layer, use the following formula:

$$PAR(N_{4,1}, G_1) = \frac{Area(N_{4,1})}{Area(G_1) + Area(G_2) + Area(G_3) + Area(G_4)}$$

As on the prior layers,

$$PAR(N_{4,1}, G_1) = PAR(N_{4,1}, G_2) = PAR(N_{4,1}, G_3) = PAR(N_{4,1}, G_4)$$

**Note:** Node  $N_{4,1}$  is connected to the diffusion layer through the output diode. After the router calculates the antenna ratio, it compares its calculations to the area of the diffusion, instead of the area of the gates.

## Calculating a CAR

To calculate a CAR, the router adds the PARs for all the relevant nodes on the specified or lower metal layers that are electrically connected to a gate. Therefore,  $CAR(N_{i,j}, G_k)$  designates the cumulative damage to gate  $G_k$  by metallization steps up to the current level of metal,  $i$ .

To create a single accumulative model that combines both metal and cut damage into one model, specify the `ANTENNACUMROUTINGPLUSCUT` statement for the layer, so that:

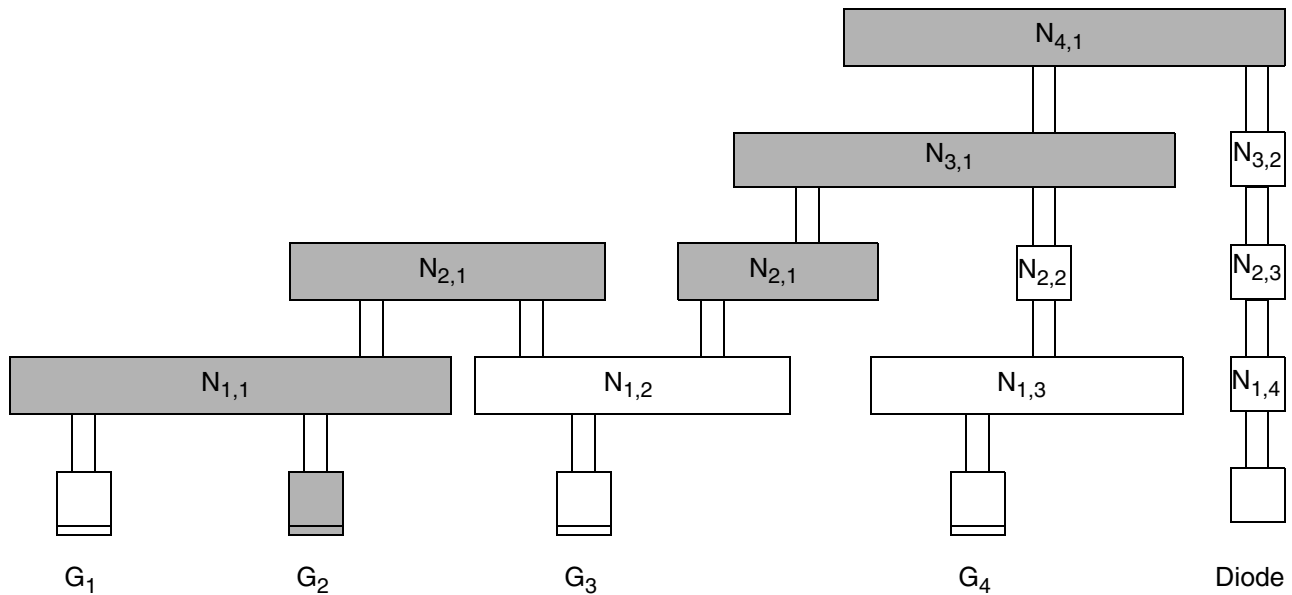
$$\text{CAR}(m_i) = \text{PAR}(m_i) + \text{CAR}(v_{i-1})$$

This means that the CAR from the cut layer below this metal layer is accumulated, instead of the CAR from the metal layer below this metal layer.

**Note:** In practice, the router only needs to keep track of the worst-case CAR; however, the CARs for all of the gates shown in [Figure C-8](#) on page 824 are described here.

The router calculates an antenna ratio with respect to a node-gate pair. To find the CAR for the node  $N_{i,j}$  - gate  $G_k$  pair, you trace the path of the current between gate  $G_k$  and node  $N_{i,j}$  and add the PAR with respect to gate  $G_k$  for the all nodes in the path between the first metal layer and layer  $i$  that you can trace back to  $G_k$ .

**Figure C-8**



The path of the current between gate  $G_2$  and node  $N_{4,1}$  is shaded.

### **Important**

In [Figure C-8](#) on page 824, node  $N_{1,2}$  is not shaded because it was not electrically connected to  $G_2$  when *metal1* was processed. That is, because the charge accumulated on  $N_{1,2}$  when *metal1* was processed cannot damage gate  $G_1$ , the router does not include it in the calculations for  $\text{CAR}(N_{2,1}, G_1)$ .

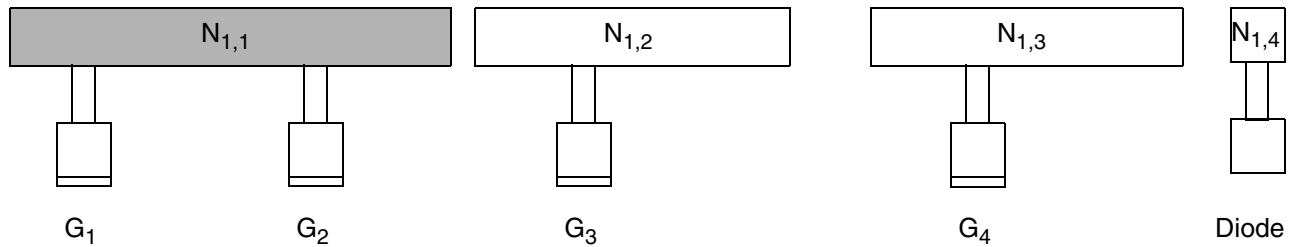
Another way to explain this is to say that the PAE from node  $N_{1,2}$  with respect to gate  $G_2$  is 0.



### Calculations for CAR on the First Metal Layer

Figure C-9 on page 825 shows the chip after the first metal layer is processed.

**Figure C-9**



In the figure above,

$$\text{CAR}(N_{1,1}, G_1) = \text{PAR}(N_{1,1}, G_1)$$

$$\text{CAR}(N_{1,1}, G_2) = \text{PAR}(N_{1,1}, G_2)$$

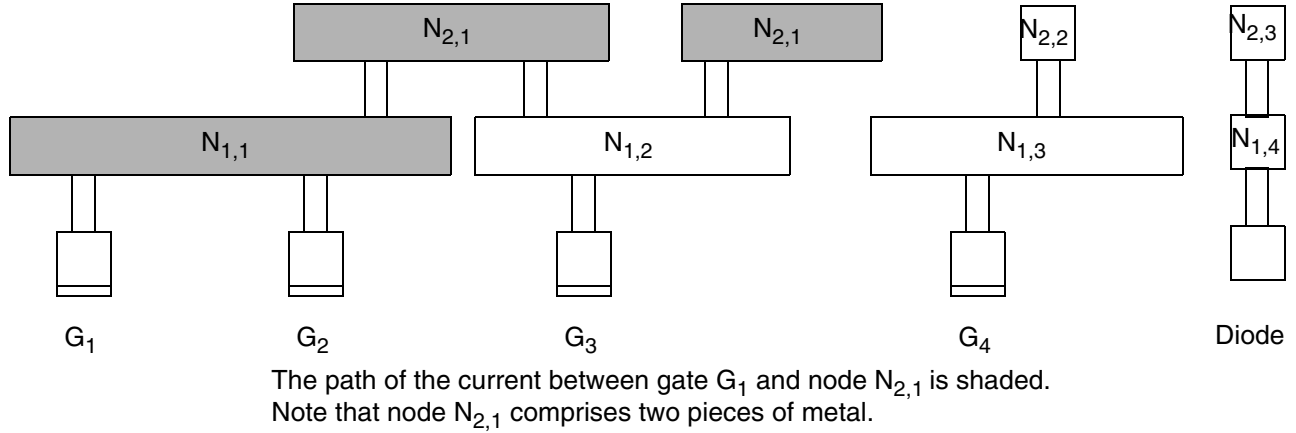
Because  $\text{PAR}(N_{1,1}, G_1)$  equals  $\text{PAR}(N_{1,1}, G_2)$ ,  $\text{CAR}(N_{1,1}, G_1)$  equals  $\text{CAR}(N_{1,1}, G_2)$ .

**Note:** In general,  $\text{CAR}(N_{i,j}, G_k)$  equals  $\text{CAR}(N_{i,j}, G_{k'})$  if the two gates  $G_k$  and  $G_{k'}$  are electrically connected to the same node on *metal1*, the lowest layer that is subject to the process antenna effect.

### Calculations for CAR on the Second Metal Layer

Figure C-10 on page 826 shows the chip after the second metal layer is processed.

**Figure C-10**



**Important**

In the figure above,  $N_{1,2}$  is not included in the calculations for  $CAR(N_{2,1}, G_1)$  because it was not electrically connected to  $G_1$  when *metal1* was processed. That is, because the charge accumulated on  $N_{1,2}$  when *metal1* was processed cannot damage gate  $G_1$ , the router does not include it in the calculations for  $CAR(N_{2,1}, G_1)$ .

In the figure above,

$$CAR(N_{2,1}, G_1) = PAR(N_{1,1}, G_1) + PAR(N_{2,1}, G_1)$$

$$CAR(N_{2,1}, G_2) = PAR(N_{1,1}, G_2) + PAR(N_{2,1}, G_2)$$

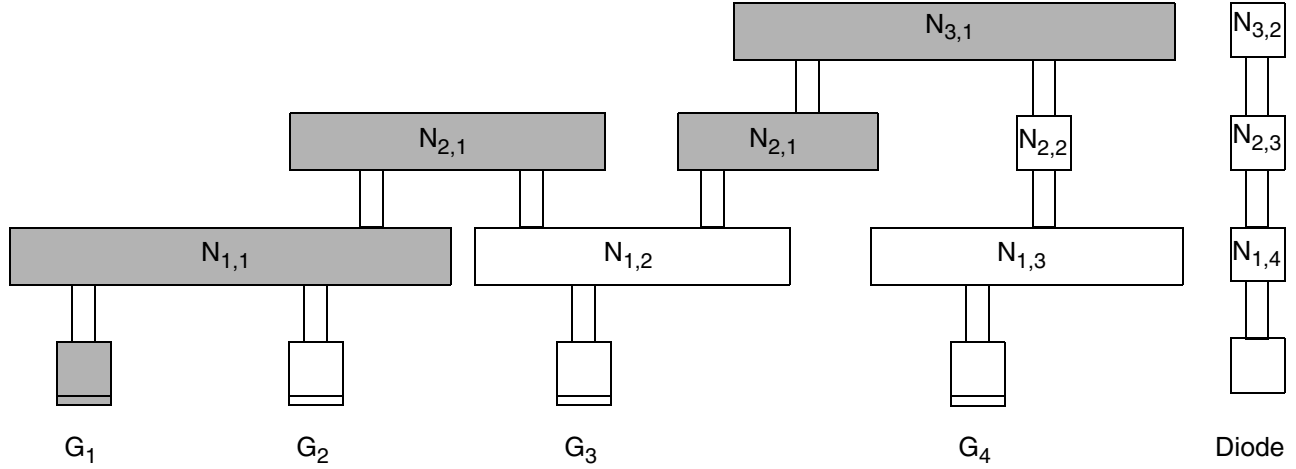
Gates  $G_1$  and  $G_2$  have the same history with regard to PAE because they are connected to the same piece of *metal1*, so they have the same CAR for any node on a specified layer:

$$CAR(N_{2,1}, G_1) = CAR(N_{2,1}, G_2)$$

**Calculations for CAR on the Third Metal Layer**

Figure C-11 on page 827 shows the chip after the third metal layer is processed.

**Figure C-11**



The path of the current between gate G<sub>1</sub> and node N<sub>3,1</sub> is shaded.

### **Gate G<sub>1</sub>**

In the figure above,

$$\text{CAR}(N_{3,1}, G_1) = \text{PAR}(N_{1,1}, G_1) + \text{PAR}(N_{2,1}, G_1) + \text{PAR}(N_{3,1}, G_1)$$

### **Gate G<sub>2</sub>**

In the figure above,

$$\text{CAR}(N_{3,1}, G_2) = \text{PAR}(N_{1,1}, G_2) + \text{PAR}(N_{2,1}, G_2) + \text{PAR}(N_{3,1}, G_2)$$

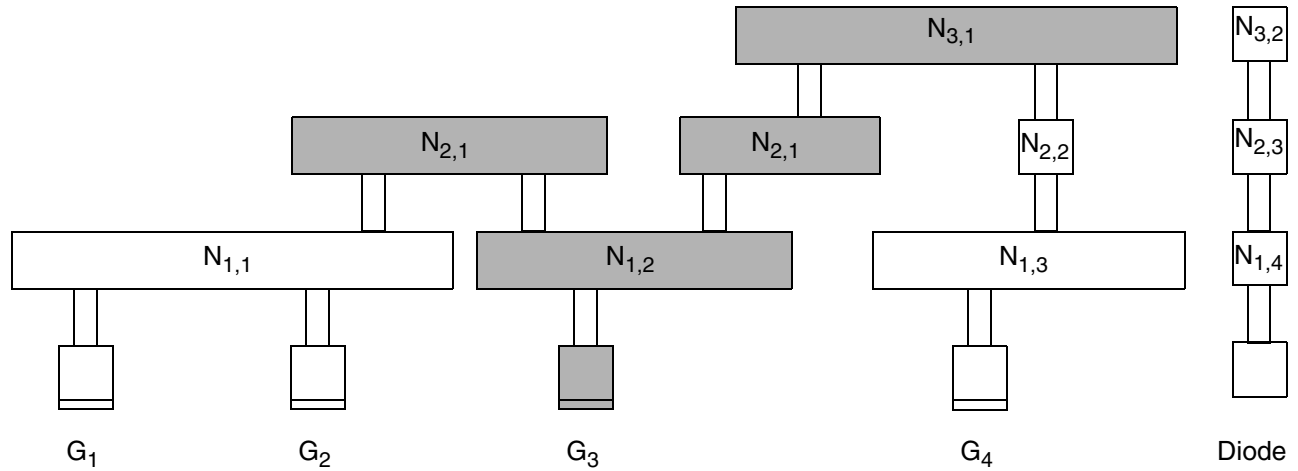
$\text{CAR}(N_{3,1}, G_1)$  equals  $\text{CAR}(N_{3,1}, G_2)$  because gates G<sub>1</sub> and G<sub>2</sub> are both electrically connected to the same node, N<sub>1,1</sub>, on *metal1* and therefore have the same history with regard to PAE. Therefore, the formula for  $\text{CAR}(N_{3,2}, G_2)$  is  $\text{CAR}(N_{3,1}, G_1) = \text{CAR}(N_{3,1}, G_2)$

### **Gates G<sub>3</sub> and G<sub>4</sub>**

Gates G<sub>3</sub> and G<sub>4</sub> are not connected to the same node on *metal1* and therefore do not have the same history with regard to PAE. Therefore, the  $\text{CAR}(N_{3,1}, G_3)$  and  $\text{CAR}(N_{3,1}, G_4)$  do not necessarily equal  $\text{CAR}(N_{3,1}, G_1)$  or  $\text{CAR}(N_{3,1}, G_2)$ .

In [Figure C-12](#) on page 828, the relevant areas for calculating CAR for gate G<sub>3</sub> are shaded.

**Figure C-12**

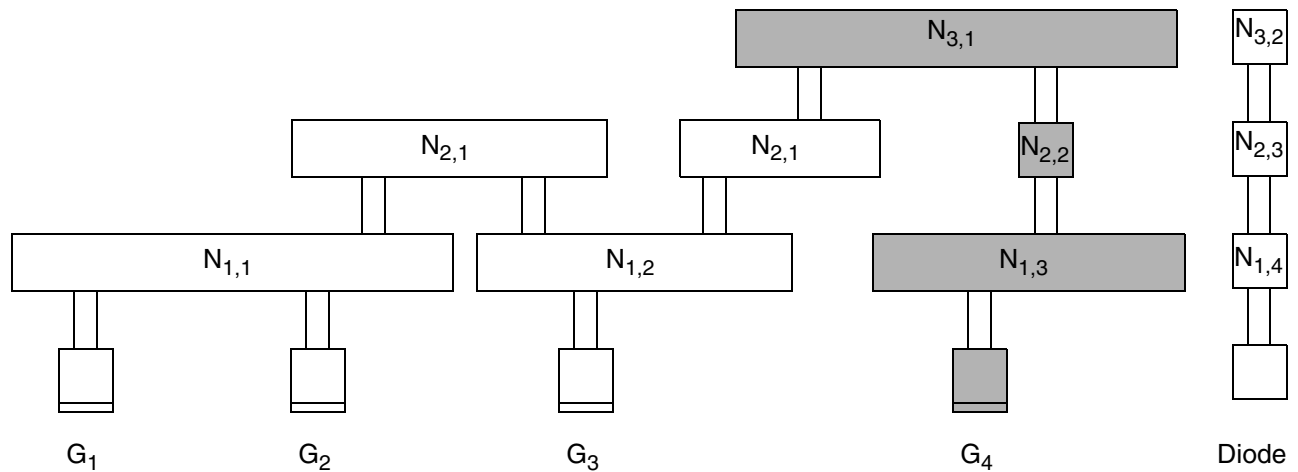


In the figure above,

$$\text{CAR}(N_{3,1}, G_3) = \text{PAR}(N_{1,2}, G_3) + \text{PAR}(N_{2,1}, G_3) + \text{PAR}(N_{3,1}, G_3)$$

In [Figure C-13](#) on page 828, the relevant areas for calculating CAR for gate  $G_4$  are shaded.

**Figure C-13**



In the figure above,

$$\text{CAR}(N_{3,1}, G_4) = \text{PAR}(N_{1,3}, G_4) + \text{PAR}(N_{2,2}, G_4) + \text{PAR}(N_{3,1}, G_4)$$

### Calculations for CAR on the Fourth Metal Layer

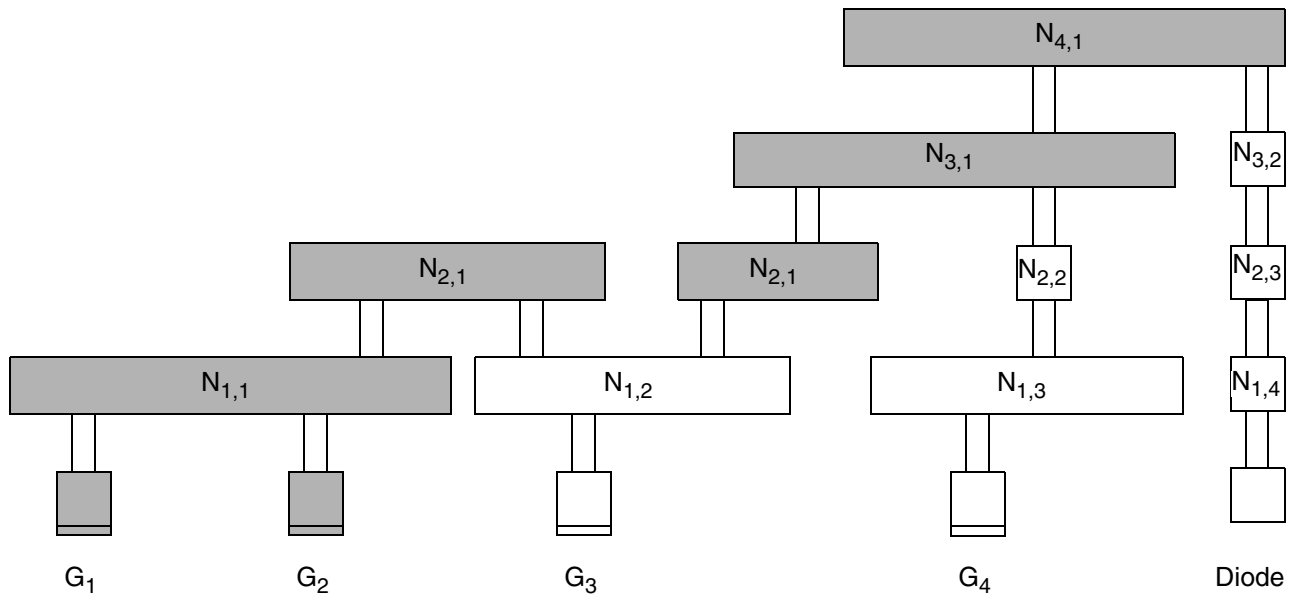
The following figure shows the chip after the fourth metal layer is processed.

**Note:** Node  $N_{4,1}$  is connected to the diffusion layer through the output diode. After the router calculates the antenna ratio, it compares its calculations to the area of the diffusion, instead of the area of the gates.

#### Gates $G_1$ and $G_2$

In [Figure C-14](#) on page 829, the relevant areas for calculating  $CAR(N_{4,1}, G_1)$  and  $CAR(N_{4,1}, G_2)$  are shaded.

**Figure C-14**



In the figure above,

$$CAR(N_{4,1}, G_1) = PAR(N_{1,1}, G_1) + PAR(N_{2,1}, G_1) \\ + PAR(N_{3,1}, G_1) + PAR(N_{4,1}, G_1)$$

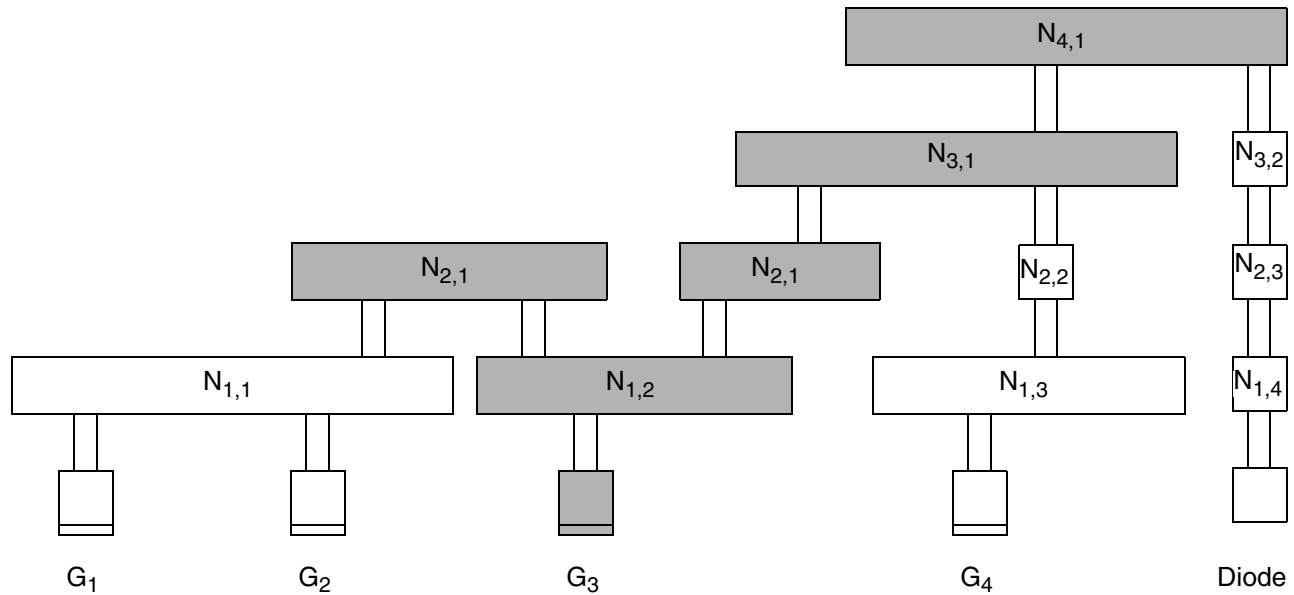
$$CAR(N_{4,1}, G_2) = PAR(N_{1,1}, G_2) + PAR(N_{2,1}, G_2) \\ + PAR(N_{3,1}, G_2) + PAR(N_{4,1}, G_2)$$

$$CAR(N_{4,1}, G_1) = CAR(N_{4,1}, G_2)$$

### Gate $G_3$

In [Figure C-15](#) on page 830, the relevant areas for calculating  $CAR(N_{4,1}, G_3)$  are shaded.

**Figure C-15**



In the figure above,

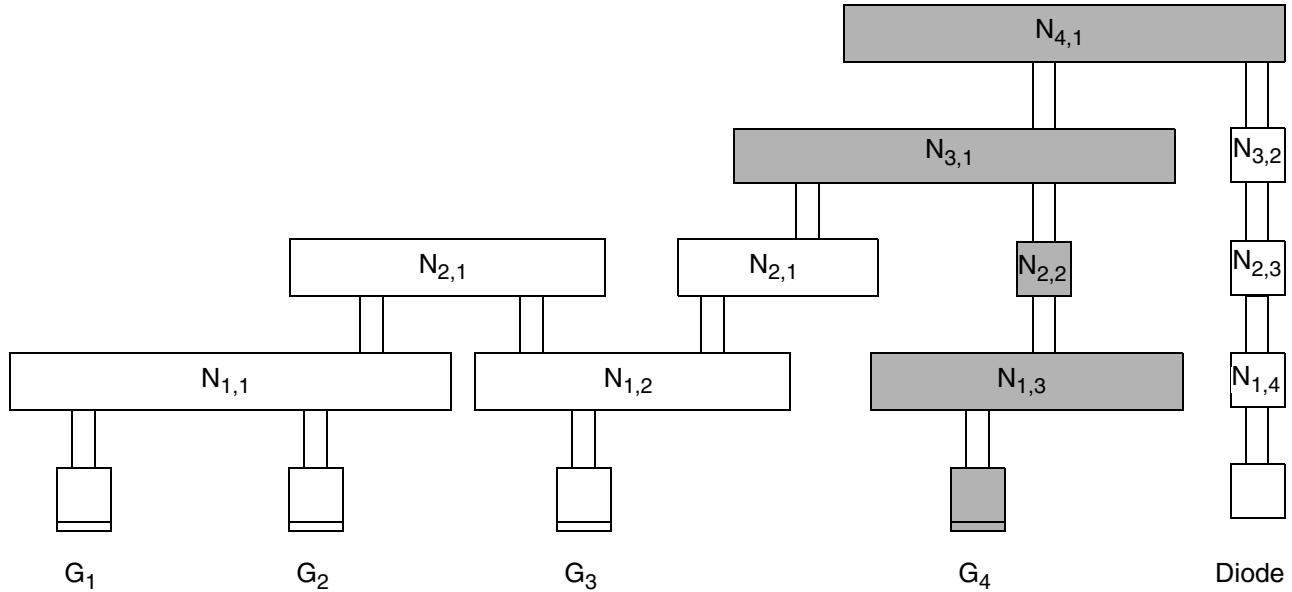
$$CAR(N_{4,1}, G_3) = PAR(N_{1,2}, G_3) + PAR(N_{2,1}, G_3) \\ + PAR(N_{3,1}, G_3) + PAR(N_{4,1}, G_3)$$

$CAR(N_{4,1}, G_3)$  does not equal  $CAR(N_{4,1}, G_1)$  or  $CAR(N_{4,1}, G_2)$  because it is not connected to the same node on *metal1*.

### Gate $G_4$

In [Figure C-16](#) on page 831, the relevant areas for calculating  $CAR(N_{4,1}, G_4)$  are shaded.

**Figure C-16**



In the figure above,

$$\begin{aligned} \text{CAR}(N_{4,1}, G_4) = & \text{PAR}(N_{1,3}, G_4) + \text{PAR}(N_{2,2}, G_4) \\ & + \text{PAR}(N_{3,1}, G_4) + \text{PAR}(N_{4,1}, G_4) \end{aligned}$$

$\text{CAR}(N_{4,1}, G_4)$  does not equal  $\text{CAR}(N_{4,1}, G_1)$ ,  $\text{CAR}(N_{4,1}, G_2)$ , or  $\text{CAR}(N_{4,1}, G_3)$  because it is not connected to the same node on *metal1*.

## Calculating Ratios for a Cut Layer

The router calculates damage from a cut layer separately from damage from a metal layer. Calculations for the cut layers do not use side area modelling.

### Calculating a PAR on a Cut Layer

The general  $\text{PAR}(c_i)$  equation for a single layer is calculated as:

$$\text{PAR}(c_i) = \frac{((\text{cutFactor} \times \text{cut\_area}) \times \text{diffAreaReduceFactor}) - (\text{minusDiffFactor} \times \text{diff\_area})}{\text{gate\_area} + (\text{plusDiffFactor} \times \text{diff\_area})}$$

The existing `ANTENNAAREAFactor` statement is shown as *cutFactor* for the metal area. Likewise, the `ANTENNAAREADIFFREDUCEPWL` statement is shown as *diffAreaReduceFactor*, the `ANTENNAAREAMINUSDIFF` statement is shown as

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

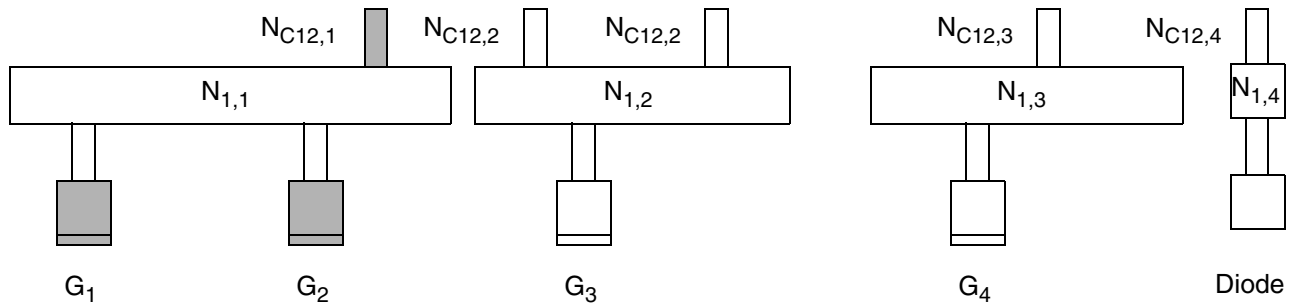
*minusDiffFactor*, and the `ANTENNAGATEPLUSDIFF` statement is shown as *plusDiffFactor*. For cut layer, the ratio equation illustrates the effect of an `ANTENNAAREAFAC` *cutFactor* statement as *metalFactor*. If there is no preceding `ANTENNAAREAFAC` statement, the *metalFactor* value defaults to 1.0.

In the figures and text that follow,

- $C_{ij}$  is the cut layer between  $metal_i$  and  $metal_j$ .
- $N_{C_{ij},k}$  specifies an electrically connected node on  $C_{ij}$ .
- The nodes are numbered sequentially, from left to right.

Figure C-17 on page 832 shows the chip after the C12 process step.

**Figure C-17**



In the figure above,

$$PAR(N_{C12,1}, G_1) = \frac{Area(N_{C12,1})}{Area(G_1) + Area(G_2)}$$

As in calculations on the metal layers,

$$PAR(N_{C12,1}, G_1) = PAR(N_{C12,1}, G_2)$$

### Calculating a CAR on a Cut Layer

As explained in “[Calculating Antenna Ratios](#)”:

$$CAR(c_i) = PAR(c_i) + CAR(c_{i-1})$$

To create a single accumulative model that combines both metal and cut damage into one model, specify the `ANTENNACUMROUTINGPLUSCUT` statement for the layer, so that:

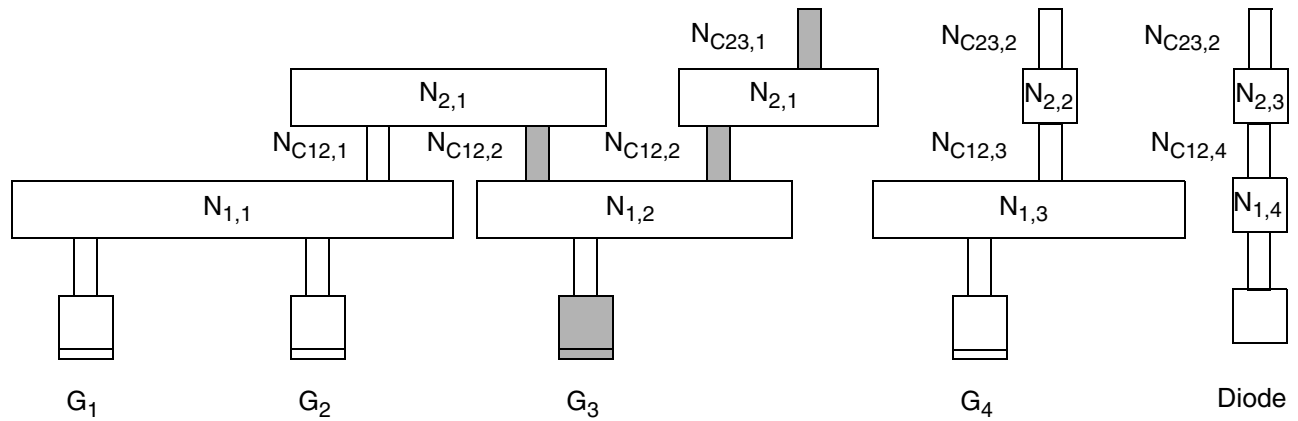


$$\text{CAR}(c_i) = \text{PAR}(c_i) + \text{CAR}(m_{i-1})$$

This means that the CAR from the *metal* layer below this cut layer is accumulated, instead of the CAR from the cut layer below this cut layer.

Figure C-18 on page 833 shows the chip after the C23 process step.

**Figure C-18**

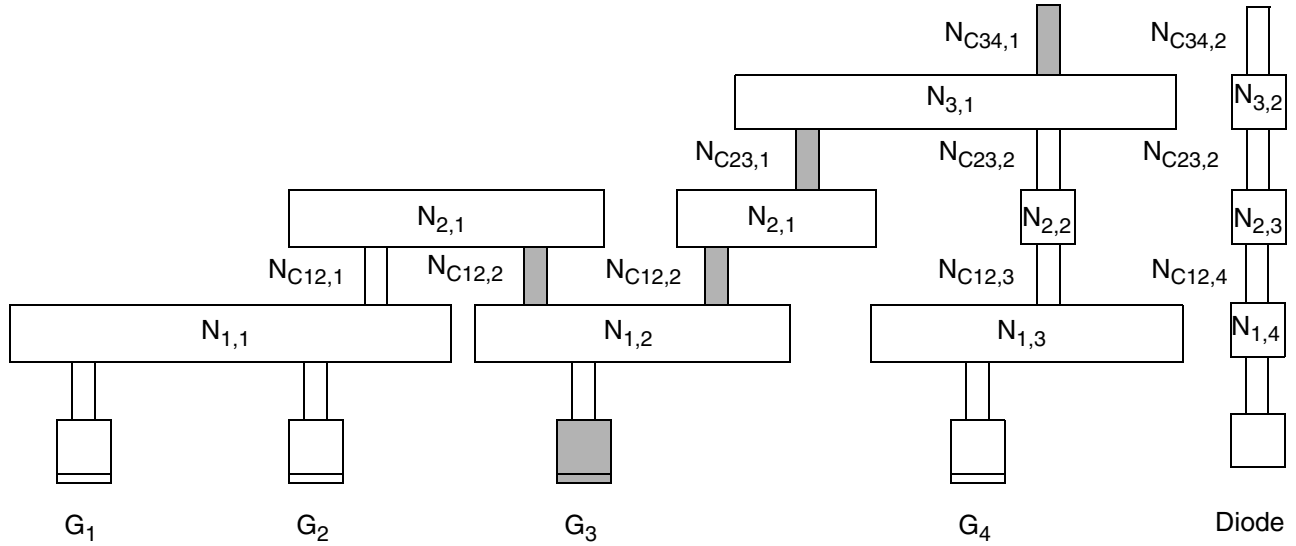


The router calculates the CAR with respect to gate  $G_3$  after the cut C23 process step as follows:

$$\text{CAR}(N_{C23,1}, G_3) = \frac{\text{Area}(N_{C12,2})}{\text{Area}(G_3)} + \frac{\text{Area}(N_{C23,1})}{\text{Area}(G_1) + \text{Area}(G_2) + \text{Area}(G_3)}$$

Figure C-19 on page 834 shows the chip after the C34 process step.

**Figure C-19**



The router calculates the CAR with respect to gate  $G_3$  after the cut C34 process step as follows:

$$\text{CAR}(N_{C34,1}, G_3) = \frac{\text{Area}(N_{C12,2})}{\text{Area}(G_3)} + \frac{\text{Area}(N_{C23,1})}{\text{Area}(G_1) + \text{Area}(G_2) + \text{Area}(G_3)} + \frac{\text{Area}(N_{C34,1})}{\text{Area}(G_1) + \text{Area}(G_2) + \text{Area}(G_3) + \text{Area}(G_4)}$$

## Checking for Antenna Violations

For each metal layer, the router performs several antenna checks, using the keywords and values specified in the LEF or DEF file. The router can perform the following four types of antenna checks, depending on the keywords you set in the LEF file:

- Area Ratio Check
- Side Area Ratio Check
- Cumulative Area Ratio Check
- Cumulative Side Area Ratio Check

## Area Ratio Check

The area ratio check compares the PAR for each layer to the value of the `ANTENNAAREARATIO` or `ANTENNADIFFAREARATIO`.

The router calculates the PAR as follows:

$$\text{PAR}(N_{i,j}, G_k) = \frac{\text{Drawn area of } N_{i,j}}{\Sigma \text{ Area of gates connected below } N_{i,j}}$$

According to the formula above, the area ratio check finds the PAR for node  $N_{i,j}$  with respect to gate  $G_k$  by dividing the drawn area of the node by the area of the gates that are electrically connected to it. The final PAR is multiplied by the `ANTENNAAREAFACOR` (the default value for the factor is 1) and compared to the `ANTENNAAREARATIO` or `ANTENNADIFFAREARATIO`. If the PAR is greater than the `ANTENNAAREARATIO` or `ANTENNADIFFAREARATIO` specified in the LEF file, the router finds a process antenna violation and attempts to fix it.

The link between  $\text{PAR}(N_{i,j}, G_k)$  and a PAE violation at node  $N_{i,j}$  depends on whether node  $N_{i,j}$  is connected to a piece of diffusion, as follows:

- If there is no connection from node  $N_{i,j}$  to a diffusion area through the current and lower layers, a violation occurs when the PAR is greater than the `ANTENNAAREARATIO`.
- If there is a connection from node  $N_{i,j}$  to a diffusion area through current and lower layers, a violation occurs when the PAR is greater than the `ANTENNADIFFAREARATIO`.
- If there is a connection from node  $N_{i,j}$  to a diffusion area through current and lower layers, and `ANTENNADIFFAREA` is not specified for an output or inout pin, the value is 0.

## Side Area Ratio Check

The side area ratio check compares the PAR computed based on the side area of the nodes for each layer to the value of the `ANTENNASIDEAREARATIO` or `ANTENNADIFFSIDEAREARATIO`.

The router calculates the PAR as follows:

$$\text{PAR}(N_{i,j}, G_k) = \frac{\text{Side area of } N_{i,j}}{\Sigma \text{ Area of gates connected below } N_{i,j}}$$

According to the formula above, the area ratio check finds the PAR for node  $N_{i,j}$  with respect to gate  $G_k$  by dividing the side area of the node by the area of the gates that are electrically

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

connected to  $N_{i,j}$ . The final PAR is multiplied by the `ANTENNASIDEAREAFACOR` (the default value for the factor is 1) and compared to the `ANTENNASIDEAREARATIO` or `ANTENNADIFFSIDEAREARATIO`. If the PAR is greater than the `ANTENNASIDEAREARATIO` or `ANTENNADIFFSIDEAREARATIO` specified in the LEF file, the router finds a process antenna violation and attempts to fix it.

The link between  $PAR(N_{i,j}, G_k)$  and a PAE violation at node  $N_{i,j}$  depends on whether node  $N_{i,j}$  is connected to a piece of diffusion, as follows:

- If there is no connection to the diffusion area through the current and lower layers, a violation occurs when the PAR is greater than the `ANTENNASIDEAREARATIO`.
- If there is a connection to the diffusion area through current and lower layers, a violation occurs when the PAR is greater than the `ANTENNADIFFSIDEAREARATIO`.
- If there is a connection to the diffusion area through current and lower layers, and `ANTENNADIFFAREA` is not specified for an output or inout pin, the value is 0.

## Cumulative Area Ratio Check

The cumulative area ratio check compares the CAR to the value of `ANTENNACUMAREARATIO` or `ANTENNACUMDIFFAREARATIO`. The CAR is equal to the sum of the PARs of all nodes on the same or lower layers that are electrically connected to the gate.

**Note:** When you use CARs, you can ignore metal layers by not specifying the CAR keywords for those layers. For example, if you want to check *metal1* using a PAR and the remaining metal layers using a CAR, you can define `ANTENNAAREARATIO` or `ANTENNASIDEAREARATIO` for *metal1*, and `ANTENNACUMAREARATIO` or `ANTENNACUMSIDEAREARATIO` for the remaining metal layers.

The cumulative area ratio check finds the CAR for node  $N_{i,j}$  with respect to gate  $G_k$  by adding the PARs for all layers of metal, from the current layer down to *metal1*, for all nodes that are electrically connected  $G_k$ . The final CAR is multiplied by the `ANTENNAAREAFACOR` (the default value for the factor is 1) and compared to the `ANTENNACUMAREARATIO` or `ANTENNACUMDIFFAREARATIO`. If the CAR is greater than the `ANTENNACUMAREARATIO` or `ANTENNACUMDIFFAREARATIO` specified in the LEF file, the router finds a process antenna violation and attempts to fix it.

The link between  $CAR(N_{i,j}, G_k)$  and a PAE violation at node  $N_{i,j}$  depends on whether node  $N_{i,j}$  is connected to a piece of diffusion, as follows:

- If there is no connection to a diffusion area through the current and lower layers, a violation occurs when the CAR is greater than the `ANTENNACUMAREARATIO`.

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

- If there is a connection to a diffusion area through current and lower layers, a violation occurs when the CAR is greater than the `ANTENNACUMDIFFAREARATIO`.
- If there is a connection to a diffusion area through current and lower layers, and `ANTENNADIFFAREA` is not specified for an output or inout pin, the value is 0.

## Cumulative Side Area Ratio Check

The cumulative side area ratio check compares the CAR to the value of the `ANTENNACUMSIDEAREARATIO` or `ANTENNACUMDIFFAREARATIO`.

**Note:** When you use CARs, you can ignore metal layers by not specifying the CAR keywords for those layers. For example, if you want to check *metal1* using a PAR and the remaining metal layers using a CAR, you can define `ANTENNAAREARATIO` or `ANTENNASIDEAREARATIO` for *metal1*, and `ANTENNACUMAREARATIO` or `ANTENNACUMSIDEAREARATIO` for the remaining metal layers.

The cumulative side area ratio check finds the CAR for node  $N_{i,j}$  with respect to gate  $G_k$  by adding the PARs for all layers of metal, from the current layer down to *metal1*, for all nodes that are electrically connected  $G_k$ . The final CAR is multiplied by the `ANTENNASIDEAREAFACOR` (the default value for the factor is 1) and compared to the `ANTENNACUMSIDEAREARATIO` or `ANTENNACUMDIFFAREARATIO`. If the CAR is greater than the `ANTENNACUMSIDEAREARATIO` or `ANTENNACUMDIFFAREARATIO` specified in the LEF file, the router finds a process antenna violation and attempts to fix it.

- If there is no connection to a diffusion area through the current and lower layers, a violation occurs when the CAR is greater than the `ANTENNACUMSIDEAREARATIO`.
- If there is a connection to a diffusion area through current and lower layers, a violation occurs when the CAR is greater than the `ANTENNACUMSIDEAREARATIO`.
- If there is a connection to a diffusion area through current and lower layers, and `ANTENNACUMDIFFAREA` is not specified for an output or inout pin, the value is 0.

## Cut Layer Process Antenna Model Examples

### ■ Example 1

To create the following process antenna rule for a cut layer *via1*:

$$\text{cut\_area} / (\text{gate\_area} + 2.0 \times \text{diff\_area}) \leq 10$$

Cut layers should include the following information:

```
ANTENNAGATEPLUSDIFF 2.0 ;
ANTENNADIFFAREARATIO 10 ;
```

## ■ Example 2

Assume the following process antenna rule:

$$\text{cut\_area} \times \text{PWL}(\text{diff\_area}) / \text{gate\_area} \leq 10$$

This rule uses a cumulative model with diffusion area reduction function, where:

- ❑  $\text{PAR} = (\text{cut\_area} \times \text{diffReduceFactor}) / \text{gate\_area} \leq 10$
- ❑  $\text{diffReduceFactor} = 1.0$  for  $\text{diff\_area} < 0.1 \mu\text{m}^2$
- ❑  $\text{diffReduceFactor} = 0.2$  for  $\text{diff\_area} \geq 0.1 \mu\text{m}^2$

Cut layers should include the following information:

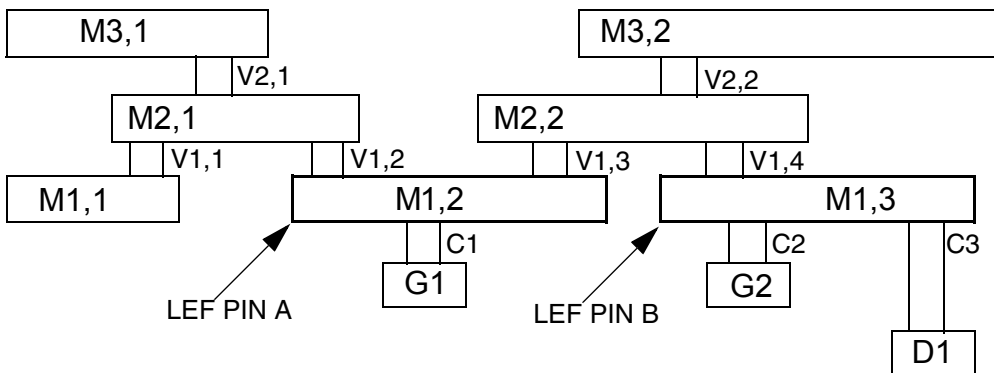
```
ANTENNAAREADIFFREDUCEPWL ( ( 0.0 1.0 ) ( 0.0999 1.0 ) ( 0.1 0.2 )
    ( 1000.0 0.2 ) ) ;
ANTENNACUMDIFFAREARATIO 10 ;
```

For examples of models that use the `ANTENNACUMROUTINGPLUSCUT` and the `ANTENNAAREAMINUSDIFF` rules, see the examples below in “Routing Layer Process Antenna Models.”

## Routing Layer Process Antenna Model Examples

The following process antenna rule examples use the topology shown in [Figure C-20](#) on page 838. In this figure, there are two polysilicon gates (G1, G2), one diffusion connection (D1), contacts (C), and via (V1, V2) and metal (M1, M2, M3) shapes. Note that M1,2 is one LEF PIN, and M1,3 is a different LEF PIN. The other metal is routing.

**Figure C-20**



## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

The following area values are also used for the examples:

G1 = 1.0	D1 = 0.5	M2,1 = 4.0
G2 = 2.0	M1,1 = 1.0	M2,2 = 5.0
All Cs = 0.1	M1,2 = 2.0	M3,1 = 6.0
All Vs = 0.1	M1,3 = 3.0	M3,2 = 9.0

#### Example 1

The following process antenna rule combines cut area and metal area into one cumulative rule:

$$\text{ratio} = (\text{metal\_area} + 10 \times \text{cut\_area}) / \text{gate\_area}$$

- The cumulative ratio  $\leq 1000$  for diffusion  $< 0.1$ , and  $\leq 4000$  for diffusion  $\geq 0.1$
- The single layer ratio  $\leq 500$  for diffusion  $< 0.1$ , and  $\leq 1500$  for diffusion  $\geq 0.1$

Every routing layer should include the following information:

```
ANTENNACUMROUTINGPLUSCUT ;
ANTENNACUMDIFFAREARATIO ( ( 0.0 1000 ) ( 0.0999 1000 ) ( 0.1 4000 )
    ( 1000.0 4000 ) ) ;
ANTENNADIFFAREARATIO ( ( 0.0 5000 ) ( 0.0999 500 ) ( 0.1 1500 )
    ( 1000.0 1500 ) ) ;
```

Every cut layer should include the following information:

```
ANTENNAAREAFACOR 10 ; #10.0 x cut area
ANTENNACUMROUTINGPLUSCUT ;
ANTENNACUMDIFFAREARATIO ( ( 0.0 1000 ) ( 0.0999 1000 ) ( 0.1 4000 )
    ( 1000.0 4000 ) ) ;
ANTENNADIFFAREARATIO ( ( 0.0 5000 ) ( 0.0999 500 ) ( 0.1 1500 )
    ( 1000.0 1500 ) ) ;
```

**Note:** ANTENNAAREARATIO and ANTENNACUMAREARATIO are not required because the \*DIFFAREARATIO statements are checked, even if diff\_area is equal to 0.

For gate G1, the PARs and CARs are computed as follows:

1.  $\text{CAR}(\text{C}, \text{G1}) = 10 \times \text{area}(\text{C1}) / \text{area}(\text{G1}) = 10 \times 0.1 / 1.0 = 1.0$

The polysilicon and contact cut layer and shapes are not normally visible in LEF and DEF. If the contact cut area should be included, its CAR value should be included with LEF PIN A, using appropriate ANTENNA statements. The M1 PIN area should not be

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

included because *M1* area is a PIN shape in the LEF and will be added in by tools reading LEF. Therefore, there should be two antenna statements for LEF PIN A, either:

```
ANTENNAGATEAREA 1.0 LAYER M1 ;  
ANTENNAMAXCUTCAR 1.0 LAYER C ;
```

or:

```
ANTENNAGATEAREA 1.0 LAYER M1 ;  
ANTENNAMAXAREACAR 1.0 LAYER M1 ;
```

Because the *M1* PIN area is not included in the MAXAREACAR value, both of sets of statements give the same results. For more details, see [“Calculations for Hierarchical Designs.”](#)

Similarly, the LEF PIN B should have values, such as either:

```
ANTENNAGATEAREA 2.0 LAYER M1 ;  
ANTENNADIFFFAREA 0.5 LAYER M1 ;  
ANTENNAMAXCUTCAR 1.0 LAYER C ; #only C2 affects G2; C3 does not
```

or:

```
ANTENNAGATEAREA 2.0 LAYER M1 ;  
ANTENNADIFFFAREA 0.5 LAYER M1 ;  
ANTENNAMAXAREACAR 1.0 LAYER M1 ; #only C2 affects G2; C3 does not
```

2.  $PAR(M1, G1) = \text{area}(M1, 2) / \text{area}(G1) = 2 / 1 = 2.0$
3.  $CAR(M1, G1) = PAR(M1, G1) + \text{PIN A's } CAR(C, G1)$   
 $\text{PIN A's } CAR(C, G1) = \text{ANTENNAMAXCUTCAR for LAYER C} = 1.0$   
 $= 2.0 + 1.0 = 3.0$
4. diode\_area = 0, single-layer PWL(0) = 500, check  $PAR(M1, G1) = 2.0 \leq 500$ , cum\_layer PWL(0) = 1000, therefore check  $CAR(M1, G1) = 3.0 \leq 1000$
5.  $PAR(V1, G1) = 10 \times \text{area}(V1, 2 + V1, 3) / \text{area}(G1) = 10 \times 0.2 / (1) = 2.0$
6.  $CAR(V1, G1) = PAR(V1, G1) + CAR(M1, G1) = 2.0 + 3.0 = 5.0$
7. diode\_area = 0, single-layer PWL(0) = 500, check  $PAR(V1, G1) = 2.0 \leq 500$ , cum\_layer PWL(0) = 1000, therefore check  $CAR(V1, G1) = 5.0 \leq 1000$
8.  $PAR(M2, G1) = \text{area}(M2, 1 + M2, 2) / \text{area}(G1 + G2) = (4+5) / (1 + 2) = 3.0$
9.  $CAR(M2, G1) = PAR(M2, G1) + CAR(V1, G1) = 3.0 + 5.0 = 8.0$
10. diode\_area = 0.5, single-layer PWL(0.5) = 1500, check  $PAR(M2, G1) = 3.0 \leq 1500$ , cum\_layer PWL(0.5) = 4000, therefore check  $CAR(M2, G1) = 8.0 \leq 4000$
11.  $PAR(V2, G1) = 10 \times \text{area}(V2, 1 + V2, 2) / \text{area}(G1 + G2) = 10 \times 0.2 / (1 + 2) = 0.67$



## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

12.  $CAR(V2,G1) = PAR(V2,G1) + CAR(M2,G1) = 0.67 + 8.0 = 8.67$
13. diode\_area = 0.5, single-layer PWL(0.5) = 1500, check  $PAR(V2,G1) = 0.67 \leq 1500$ , cum\_layer PWL(0.5) = 4000, therefore check  $CAR(V2, G1) = 8.67 \leq 4000$
14.  $PAR(M3,G1) = area(M3,1 + M3,2) / area(G1 + G2) = (6 + 9) / (1 + 2) = 5$
15.  $CAR(M3,G1) = PAR(M3,G1) + CAR(V2,G1) = 5 + 8.67 = 12.34$
16. diode\_area = 0.5, single-layer PWL(0.5) = 1500, check  $PAR(M3,G1) = 5 \leq 1500$ , cum\_layer PWL(0.5) = 4000, therefore check  $CAR(M3,G1) = 13.67 \leq 4000$

#### Example 2

The following cumulative rule is the same as the rule in Example 1, except it also subtracts the diff\_area factor. Only the cumulative model is used.

$$ratio = [(metal\_area + 10 \times cut\_area) - (100 \times diff\_area)] / gate\_area$$

Every routing layer should include the following information:

```
ANTENNACUMROUTINGPLUSCUT ;
ANTENNAAREAMINUDIFF 100.0 ;
ANTENNACUMDIFFAREARATIO 1000 ;
```

Every cut layer should include the following information:

```
ANTENNAAREAFACOR 10 ; #10.0 x cut area
ANTENNACUMROUTINGPLUSCUT ;
ANTENNAAREAMINUDIFF 100.0 ;
ANTENNACUMDIFFAREARATIO 1000 ;
```

For gate G1, the PARs and CARs are computed as follows:

1.  $CAR(C,G1) = 10 \times area(C1) / area(G1) = 10 \times 0.1 / 1.0 = 2.0$

This value is on the LEF PIN, as mentioned in Example 1.

2.  $PAR(M1,G1) = area(M1,2) / area(G1) - (100 \times diff\_area) = (2 / 1) - (100 \times 0) = 2.0$
3.  $CAR(M1,G1) = PAR(M1,G1) + PIN A's CAR(C,G1)$   
PIN A's  $CAR(M1) = ANTENNAMAXAREACAR$  for LAYER M1 = 1.0  
 $= 2.0 + 1.0 = 3.0$
4. Check  $CAR(M1,G1) = 3.0 \leq 1000$
5.  $PAR(V1,G1) = [10 \times area(V1,2 + V1,3) - (100 \times diff\_area)] / area(G1)$   
 $= [(10 \times .2) - (100 \times 0)] / (1) = 2.0$
6.  $CAR(V1,G1) = PAR(V1,G1) + CAR(M1,G1) = 2.0 + 3.0 = 5.0$

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

7. Check  $CAR(V1, G1) = 5.0 \leq 1000$
8.  $PAR(M2, G1) = [area(M2,1 + M2,2) - (100 \times area(D1))] / area(G1 + G2)$   
 $= [(4 + 5) - (100 \times 0.5)] / (1 + 2) = -13.67$
9.  $CAR(M2, G1) = PAR(M2, G1) + CAR(V1, G1) = -13.67 + 5.0 = -8.67$ , truncate to 0
10. Check  $CAR(M2, G1) = 0 \leq 1000$
11.  $PAR(V2, G1) = [(10 \times area(V2,1 + V2,2)) - (100 \times area(D1))] / area(G1 + G2)$   
 $= [(10 \times 0.2) - (100 \times 0.5)] / (1 + 2) = -16.0$
12.  $CAR(V2, G1) = PAR(V2, G1) + CAR(M2, G1) = -16.0 + 0 = -16.0$ , truncate to 0
13. Check  $CAR(V2, G1) = 0 \leq 1000$
14.  $PAR(M3, G1) = [area(M3,1 + M3,2) - (100 \times area(D1))] / area(G1 + G2)$   
 $= [(6 + 9) - (100 \times 0.5)] / (1 + 2) = -11.67$
15.  $CAR(M3, G1) = PAR(M3, G1) + CAR(V2, G1) = -11.67 + 0 = -11.67$ , truncate to 0
16. Check  $CAR(M3, G1) = 0 \leq 1000$

### Example 3

The following cumulative rule for metal layers includes a diffusion area factor added into the denominator of the ratio:

Single layer:  $metal\_area / (gate\_area + 2.0 \times diff\_area) \leq 1000$

Cumulative for the layer:  $metal\_area / (gate\_area + 2.0 \times diff\_area) \leq 5000$

Every metal layer should include the following information:

```
ANTENNAPLUSGATEDIFF 2.0 ;  
ANTENNADIFFFAREARATIO 1000 ;  
ANTENNACUMDIFFFAREARATIO 5000 ;
```

**Note:** The via area is ignored in this example. If an independent via model is needed, similar statements should be added to the via layers, which would be computed separately.

For gate G1, the PARs and CARs are computed as follows:

1.  $PAR(M1, G1) = area(M1,2) / area(G1) = 2.0 / 1 = 2$
2.  $CAR(M1, G1) = PAR(M1, G1) = 2$
3. Check  $PAR(M1, G1) = 2 \leq 1000$ ,  
check  $CAR(M1, G1) = 2 \leq 5000$

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

4.  $PAR(M2,G1) = \text{area}(M2,1 + M2,2) / [\text{area}(G1 + G2) + 2 \times \text{area}(D1)]$   
 $= (4 + 5) / [(1 + 2) + 2 \times 0.5] = 2.25$
5.  $CAR(M2,G1) = CAR(M1,G1) + PAR(M2,G1) = 2 + 2.25 = 4.25$
6. Check  $PAR(M1,G1) = 2.25 \leq 1000$ ,  
check  $CAR(M1,G1) = 4.25 \leq 5000$
7.  $PAR(M3,G1) = \text{area}(M3,1 + M3,2) / [\text{area}(G1 + G2) + 2 \times \text{area}(D1)]$   
 $= (6 + 9) / [(1 + 2) + 2 \times 0.5] = 3.75$
8.  $CAR(M3,G1) = PAR(M3,G1) + CAR(M2,G1) = 3.75 + 4.25 = 8.0$
9. Check  $PAR(M1,G1) = 3.75 \leq 1000$ ,  
check  $CAR(M1,G1) = 8.0 \leq 5000$

#### Example 4

Assume a cumulative rule that includes a diffusion area reduction value and a routing ratio of 1000. The reduction value is 1.0 if the diff\_area is less than 0.1, 0.2 if the diff\_area equals 0.1, and decreases linearly to 0.1 if the diff\_area equals 1.0. The reduction value remains 0.1 if the diff\_area is greater than 1.0.

Every metal layer should include the following information:

```
ANTENNAAREADIFFREDUCEPWL ( ( 0.0 1.0 ) ( 0.0999 1.0 ) ( 0.1 0.2 ) ( 1.0 0.1 )  
    ( 1000.0 0.1 ) ) ;"  
ANTENNACUMDIFFAREARATIO 1000 ;
```

**Note:** The via area is ignored in this example. If an independent via model is needed, similar statements should be added to the via layers, which would be computed separately.

For gate G1, the PARs and CARs are computed as follows:

1. Initial  $PAR(M1,G1) = \text{area}(M1,2) / \text{area}(G1) = 2.0 / 1 = 2$
2. diode\_area = 0,  $PWL(0) = 1.0$ , therefore initial  $PAR(M1,G1)$  is multiplied by 1.0  
to give  $PAR(M1,G1) = 2 \times 1 = 2$
3.  $CAR(M1,G1) = PAR(M1,G1) = 2$
4. Check  $CAR(M1,G1) \leq 1000$ , therefore check  $2 \leq 1000$
5. Initial  $PAR(M2,G1) = \text{area}(M2,1 + M2,2) / \text{area}(G1 + G2) = (4 + 5) / (1 + 2) = 3$
6. diode\_area = 0.5,  $PWL(0.5) = 0.155$ , therefore initial  $PAR(M2,G1)$  is multiplied by 1.0  
to give  $PAR(M2,G1) = 3 \times 0.155 = 0.465$
7.  $CAR(M2,G1) = CAR(M1,G1) + PAR(M2,G1) = 2 + 0.465 = 2.465$

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

8. Check  $CAR(M2, G1) \leq 1000$ , therefore check  $2.465 \leq 1000$
9. Initial  $PAR(M3, G1) = \text{area}(M3, 1 + M3, 2) / \text{area}(G1 + G2) = (6 + 9) / (1 + 2) = 5$
10.  $\text{diode\_area} = 0.5$ ,  $PWL(0.5) = 0.155$ , therefore initial  $PAR(M3, G1)$  is multiplied by 0.155 to give  $PAR(M3, G1) = 5 \times 0.155 = 0.775$
11.  $CAR(M3, G1) = PAR(M3, G1) + CAR(M2, G1) = 0.775 + 2.465 = 3.24$
12. Check  $CAR(M3, G1) \leq 1000$ , therefore check  $3.24 \leq 1000$

### Example Using the Antenna Keywords

The following example is a portion of a LEF file that shows the antenna keywords for a process that has cumulative area ratio damage for metal and cut layers.

Assume you have the following antenna rules for your process:

1. A maximum cumulative metal to gate area ratio of 1000
2. If a diode of greater than .1 microns is connected to the metal, the maximum metal ratio is:  $\text{ratio} = \text{diode\_area} \times 2000 + 5000$
3. A maximum cumulative via to gate area ratio of 20
4. If a diode of greater than .1 microns is connected to the via, the maximum via ratio is:  $\text{ratio} = \text{diode\_area} \times 200 + 100$

The corresponding LEF file would include:

```
LAYER M1
  TYPE ROUTING ;
  ...
  ANTENNACUMAREARATIO 1000 ;
  ANTENNACUMDIFFAREARATIO
    PWL ( ( 0 1000 ) ( 0.099 1000 ) ( 0.1 5200 ) ( 100 205000 ) ) ;
END M1
```

```
LAYER VIA1
  TYPE CUT ;
  ...
  ANTENNACUMAREARATIO 20 ;
  ANTENNACUMDIFFAREARATIO
    PWL ( ( 0 20 ) ( 0.099 20 ) ( 0.1 120 ) ( 100 20100 ) ) ;
```

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

```
END VIA1
```

A typical standard cell that has only *M1* pins and routing inside of it would have:

```
MACRO INV1X
  CLASS CORE ;
  ...
  PIN IN
    DIRECTION INPUT ;
    ANTENNAGATEAREA .5 LAYER M1 ; # connects to 0.5  $\mu\text{m}^2$  poly gate
    ANTENNAPARTIALMETALAREA 1.0 LAYER M1 ; # has 1.0  $\mu\text{m}^2$  M1 area.
    # Note that it should not include the M1 pin area, just the M1 routing
    # area that is not included in the PIN shapes. In many cases, all of the
    # M1 routing is included in the PIN, so this value is 0, and not in the
    # LEF at all.
    ANTENNAMAXAREACAR 10.0 LAYER M1 ; # has 10.0 cumulative ratio so far.
    # This value can include area from internal poly routing if poly routing
    # damage is accumulated with the metal layers. It does not include
    # the area of the M1 pin area, just the M1 routing area that is not
    # included in the PIN shapes. If poly damage is not included, and all
    # of the M1 routing is included in the PIN, this value will be 0, and
    # not in the LEF at all.
    ...
  END IN
  PIN OUT
    DIRECTION OUTPUT ;
    ANTENNADIFFFAREA .2 LAYER M1 ; # connects to 0.2  $\mu\text{m}^2$  diffusion area
    ANTENNAPARTIALMETALAREA 1.0 LAYER M1 ; # has 1.0  $\mu\text{m}^2$  M1 area
    # No ANTENNAMAXAREACAR value because no internal poly gate is connected
    ...
  END OUT
END INV1X
```

## Using Antenna Diode Cells

Routers generally use one of two methods to fix process antenna violations:

- Change the routing by breaking the metal layers into smaller pieces
- Insert antenna diode cells to discharge the current

## Changing the Routing

One method routers use to fix antenna violations is to limit the charge that is collected through the metal nodes exposed to the plasma. To do this, it goes up one layer or pushes the routing down one layer whenever the process antenna ratio exceeds the ratio set in the LEF file.

The router changes the routing by disconnecting nets with antenna violations and making the connections to higher metal layers instead. It does not make the connections to lower layers. This method works because the top metal layer always completes the connection from the gate to the output drain area of the driver, which is a diode that provides a discharge path.

## Inserting Antenna Diode Cells

The second method routers use to repair antenna violations is to insert antenna diode cells in the design. The electrical charges on the metal that connects to the diodes is then discharged through the diode diffusion layer and substrate. The router inserts the diode cells automatically.

The following example shows a LEF definition of an antenna diode cell, with the `CLASS CORE ANTENNACELL` and `ANTENNADIFFAREA` defined:

```
MACRO antennal
    CLASS CORE ANTENNACELL ;
    ...
    PIN ANT1
        AntennaDiffArea 1.0 ;
        PORT
            LAYER metall ;
            RECT 0.190 2.380 0.470 2.660 ;
        END
    END ANT1
END antennal
```

## Using DiffUseOnly

LEF defines only one value for `ANTENNAAREAFactor` and one value for `ANTENNASIDEAREAFactor`, with or without `DIFFUSEONLY`, per layer. If you specify more than one antenna area or side area factor for a layer, only the last one is used. The `AREAFactor` value lets you scale the value of the metal area. If you use the `DIFFUSEONLY` keyword, only metal attached to diffusion is scaled.

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

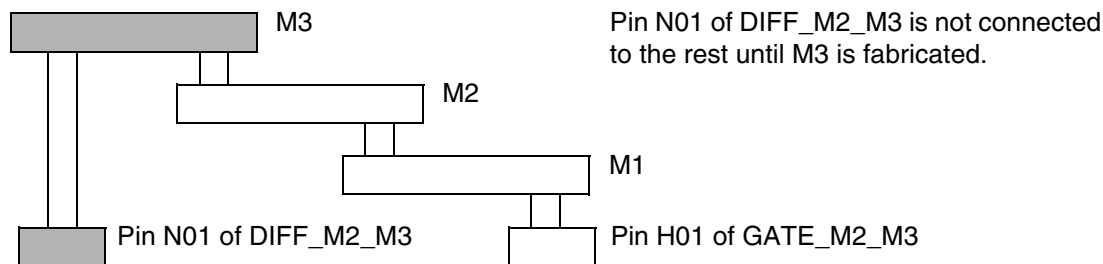
---

Suppose you have the following LEF file:

```
Antenna.lef
-----

LAYER M3
TYPE ROUTING ;
PITCH 0.56 ;
DIRECTION HORIZONTAL ;
WIDTH 0.28 ; SPACING 0.28 ;
SPACING 0.36 RANGE 1.0 250.0 ;
CAPACITANCE CPERSQDIST 0.0009762 ;
RESISTANCE RPERSQ 0.129 ;
THICKNESS 0.60 ;
AntennaAreaRatio 10000 ;
AntennaDiffAreaRatio 10000 ;
AntennaAreaFactor 1.2 DiffUseOnly ;
AntennaSideAreaRatio 5000 ;
AntennaDiffSideAreaRatio 5000 ;
AntennaSideAreaFactor 1.4 DiffUseOnly ;
END M3
```

**Figure C-21**



In the figure,

- The input pin H01 of GATE\_M2\_M3 connects the metal wires to *metal1*, *metal2*, and *metal3* in sequence.
- The ANTENNAAREAFACTOR 1.2 DIFFUSEONLY and ANTENNASIDEAREAFACTOR 1.4 DIFFUSEONLY apply to *metal3* routing.
- Prior to *metal3* fabrication, there is no path to the diffusion diode. This causes the default factor of 1.0 to apply to the *metal1* and *metal2* segments shown when calculating PARs.

## Calculations for Hierarchical Designs

The following section illustrates computation of antenna ratios for hierarchical designs.

## LEF and DEF Keywords for Hierarchical Designs

If the keyword ends with ...	It refers to ...	Examples
area sideArea	Drawn area or side area of the metal wires. Measured in square microns.	ANTENNAPARTIALCUTAREA ANTENNAPARTIALMETALAREA ANTENNAPARTIALMETALSIDEAREA ANTENNAPINDIFFAREA ANTENNAPINGATEAREA ANTENNAPINPARTIALCUTAREA
CAR	Relationship the router is calculating  CAR is used in keywords for cumulative antenna ratio.	ANTENNAMAXAREACAR ANTENNAMAXCUTCAR ANTENNAMAXSIDEAREACAR ANTENNAPINMAXAREACAR ANTENNAPINMAXCUTCAR ANTENNAPINMAXSIDEAREACAR

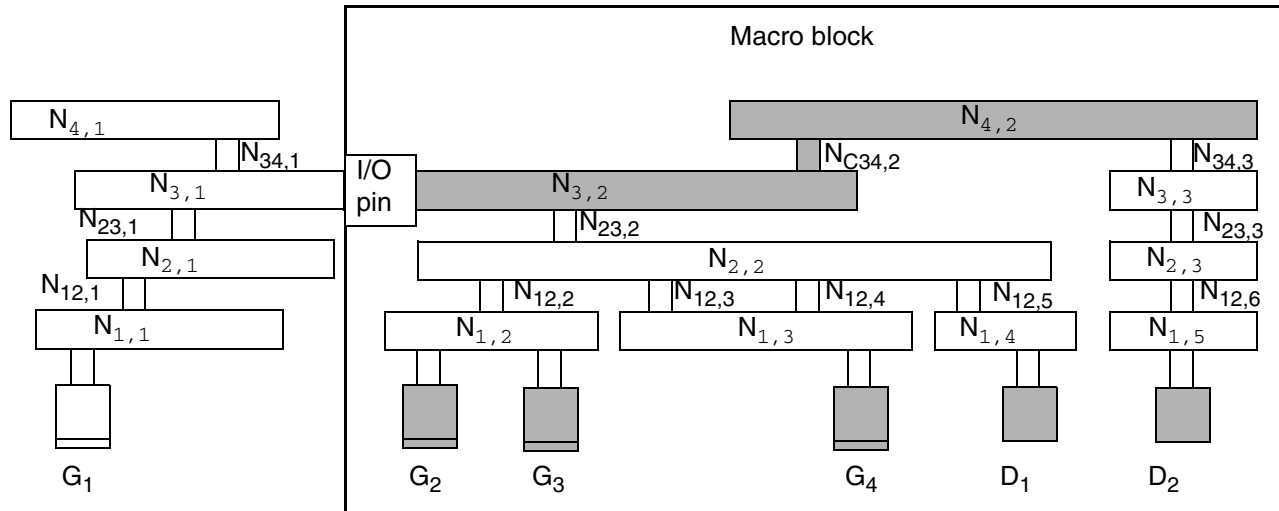
## Design Example

Figure C-22 on page 849 represents a macro block. This block can be a custom hard block or part of a bottom-up hierarchical flow. The resulting PAE values will be the same in either case. In the example,

- Gates  $G_1$ ,  $G_2$ ,  $G_3$ , and  $G_4$  are the same size.
- Node  $N_{1,3}$  is larger than node  $N_{1,2}$ .
- Vias (cuts) are all the same size.
- The I/O pin is on *metal3*.
- The area of diffusion for  $D_1$  is `area(Diff1)`.
- The area of diffusion for  $D_2$  is `area(Diff2)`.
- The area of the cut layer that connects node  $N_{3,1}$  and node  $N_{4,2}$  is `area(NC34,1)`.
- Any damage from the poly layer or poly-to-metal1 via is ignored.



**Figure C-22**



### Relevant Metal Areas

- The relevant metal area for PAE calculations is the partial metal drawn area and side area connected directly to the I/O pin on the inside of the macro on the specified layer.
- Only the same metal layer as the I/O pin or above is needed for PAR calculations in hierarchical designs.

### Important

Do not include the drawn area or side area of the I/O pin in the area calculations for the block, because the router includes these areas in the calculations for the upper level. Only the internal routing area that is not part of the I/O pin should be included.

For the design in the figure above, you must specify values for the following metal areas in the LEF file:

```
ANTENNAPARTIALMETALAREA area(N3,2) LAYER Metal3 ;
ANTENNAPARTIALMETALAREA area(N4,2) LAYER Metal4 ;
ANTENNAPARTIALMETALSIDEAREA sideArea(N3,2) LAYER Metal3 ;
ANTENNAPARTIALMETALSIDEAREA sideArea(N4,2) LAYER Metal4 ;
```

You do not need to specify an `ANTENNAPARTIALMETALAREA` or `ANTENNAPARTIALSIDEMETALAREA` for any layer lower than *metal3* because the I/O pin is on *metal3*; that is, there is no connection outside the block until *metal3* is processed.

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

#### **Relevant Gate, Diffusion, and Cut Areas**

- The relevant gate and diffusion areas are the gate and diffusion areas that connect directly to the I/O pin on the specified layer or are electrically connected to the pin through lower layers.
- The relevant partial cut area is above the current pin layer and inside the macro on the specified layer.

For the design in the figure above, you must specify values for the following gate, diffusion, and cut areas in the LEF file:

```
ANTENNAGATEAREA area(G2 + G3 + G4) LAYER Metal3 ;
ANTENNADIFFAREA area(Diff1) LAYER Metal3 ;
ANTENNADIFFAREA area(Diff1 + Diff2) LAYER Metal4 ;
ANTENNAPARTIALCUTAREA area(N34,2) LAYER Via34 ;
```

#### **Calculating the CAR**

Use the following keywords to calculate the actual CAR on the I/O pin layer or above.

- The relevant maximum CAR value of the drawn and side areas are from the metal layer that is on or below the I/O pin layer.
- The relevant maximum CAR value of the cut layer is from the cut layer that is immediately above the I/O pin layer.

For the example in [Figure C-22](#) on page 849, the keywords and calculations for *metal3* and *via34* would be:

$$\text{ANTENNAMAXAREACAR} \left( \left| \frac{N_{1,3}}{G_4} + \frac{N_{2,2}}{G_2 + G_3 + G_4} + \frac{N_{3,2}}{G_2 + G_3 + G_4} \right| \right) \text{ LAYER Metal3};$$
$$\text{ANTENNAMAXSIDEAREACAR} \left( \left| \frac{N_{1,3}}{G_4} + \frac{N_{2,2}}{G_2 + G_3 + G_4} + \frac{N_{3,1}}{G_2 + G_3 + G_4} \right| \right) \text{ LAYER Metal3};$$
$$\text{ANTENNAMAXCUTCAR} \left( \left| \frac{N_{12,3} + N_{12,4}}{G_4} + \frac{N_{23,2}}{G_2 + G_3 + G_4} + \frac{N_{34,2}}{G_2 + G_3 + G_4} \right| \right) \text{ LAYER Via34};$$

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

#### **Sample LEF File for a Bottom-Up Hierarchical Design**

For a macro block like that shown in [Figure C-22](#) on page 849, you should have the following pin information in your LEF file, ignoring SIDEAREA values:

PIN example

```
ANTENNAGATEAREA 0.3 LAYER METAL3 ; # area of G2 + G3 + G4
ANTENNADIFFAREA 1.0 LAYER METAL3 ; # area of D1
ANTENNAPARTIALMETALAREA 10.0 LAYER METAL3 ; # area of N3,2
ANTENNAMAXAREACAR 100.0 LAYER METAL3 ; # max CAR of N3,2

ANTENNAPARTIALCUTAREA 0.1 LAYER VIA34 ; # area of N34,2
ANTENNAMAXCUTCAR 5.0 LAYER VIA34 ; # max cut CAR of N34,2

ANTENNAGATEAREA 0.3 LAYER METAL4 ; # area of G2 + G3 + G4
ANTENNADIFFAREA 2.0 LAYER METAL4 ; # area of D1 + D2
ANTENNAPARTIALMETALAREA 12.0 LAYER METAL4 ; # area of N4,2
ANTENNAMAXAREACAR 130.0 LAYER METAL4 ; # max CAR of N4,2
```

END example

#### **Top-Down Hierarchical Design Example**

In a top-down design, the router uses the top-level antenna values to check for process antennas inside the block. If the top level is routed first, the top-level routing CAR and PAR values can be passed down into the DEF for the sub-block. This method can also be used to pass down estimated “budgets” for PAR and CAR values.

Set the following keywords in the DEF file for the design. In a top-down design you assign a value to the I/O pin that indicates how much routing, CAR, and PAR occurred outside the block already.

MACRO *macroName*

CLASS BLOCK ;

PIN *pinName*

DIRECTION OUTPUT ;

[ANTENNAPINPARTIALMETALAREA *value* [LAYER *layerName*] ;] ...

[ANTENNAPINPARTIALMETALSIDEAREA *value* [LAYER *layerName*] ;] ...

[ANTENNAPINGATEAREA *value* [LAYER *layerName*] ;] ...

[ANTENNAPINDIFFAREA *value* [LAYER *layerName*] ;] ...

[ANTENNAPINMAXAREACAR *value* [LAYER *layerName*] ;] ...

[ANTENNAPINMAXSIDEAREACAR *value* [LAYER *layerName*] ;] ...

[ANTENNAPINPARTIALCUTAREA *value* [LAYER *cutlayerName*] ;] ...

[ANTENNAPINMAXCUTCAR *value* LAYER *cutlayerName*] ;] ...

## LEF/DEF 5.8 Language Reference

### Calculating and Fixing Process Antenna Violations

---

```
END Z
END macroName
```

### Sample DEF File for a Top-Down Hierarchical Design

An example of the DEF keywords for [Figure C-22](#) on page 849 would be:

```
PINS 100 ;
- example + NET example1
  + ANTENNAPINPARTIALMETALAREA (N3,1) LAYER Metal3 ;
  + ANTENNAPINPARTIALMETALSIDEAREA (N3,1) LAYER Metal3 ;
  + ANTENNAPINGATEAREA (G1) LAYER Metal3 ;
  # No ANTENNAPINDIFFAREA for this example

  + ANTENNAPINMAXAREACAR  $\left( \frac{N_{1,1} + N_{2,1} + N_{3,1}}{G_1} \right)$  LAYER Metal3 ;

  + ANTENNAPINMAXSIDEAREACAR  $\left( \frac{N_{1,1} + N_{2,1} + N_{3,1}}{G_1} \right)$  LAYER Metal3 ;

  + ANTENNAPINPARTIALCUTAREA (N34,1) LAYER via34 ;

  + ANTENNAPINMAXCUTCAR  $\left( \frac{N_{34,1} + N_{23,1} + N_{12,1}}{G_1} \right)$  LAYER Metal3 ;

  + ANTENNAPINGATEAREA (G1) LAYER Metal4 ;
  + ANTENNAPINPARTIALMETALAREA (N4,1) LAYER Metal4 ;
  + ANTENNAPINPARTIALMETALSIDEAREA (N4,1) LAYER Metal4 ;
  ...
END PINS
```

# Index

## Symbols

,... in syntax [8](#)  
 ... in syntax [8](#)  
 [] in syntax [8](#)  
 {} in syntax [8](#)  
 | in syntax [8](#)

## A

abutment pins [599](#)  
 alias  
   expansion in DEF and LEF [637](#)  
   names in DEF and LEF [636](#)  
   statements in LEF and DEF [635](#)

## B

blockages, simplified [590](#)  
 braces in syntax [8](#)  
 brackets in syntax [8](#)  
 BUSBITCHARS statement  
   description, DEF [656](#)

## C

capacitance  
   peripheral [269](#)  
   wire-to-ground [266](#)  
 cell modeling  
   combining blockages [590](#)  
 characters  
   escape [648](#)  
   information [647](#)  
 COMPONENTS statement  
   description, DEF [657](#)  
 conventions  
   user-defined arguments [7](#), [8](#)  
   user-entered text [7](#)  
 COVER model, definition in LEF [547](#)

## D

database  
   converting LEF values to integer  
     values [613](#)  
 debugging  
   DEF files [640](#)  
   LEF libraries [640](#)  
   parametric macros [640](#)  
 DEF  
   example [786](#)  
   syntax overview [646](#)  
 DEF syntax  
   PINS [700](#)  
 DEF syntax and description  
   BUSBITCHARS [656](#)  
   COMPONENTS [657](#)  
   DESIGN [664](#)  
   DIEAREA [665](#)  
   DIVIDERCHAR [13](#), [665](#)  
   GCELLGRID [670](#)  
   GROUPS [672](#)  
   HISTORY [673](#)  
   NETS [673](#)  
   PINPROPERTIES [717](#)  
   PROPERTYDEFINITIONS [718](#)  
   REGIONS [719](#)  
   ROW [721](#)  
   SPECIALNETS [729](#)  
   TECHNOLOGY [758](#)  
   TRACKS [759](#)  
   UNITS DISTANCE MICRONS [760](#)  
   VERSION [761](#)  
   VIAS [761](#)  
 DESIGN statement  
   description, DEF [664](#)  
 diagonal vias, recommendation for  
   RGrid [509](#)  
 DIEAREA statement  
   description, DEF [665](#)  
 DIVIDERCHAR statement  
   description, DEF [13](#), [665](#)

## E

edge capacitance [269](#)  
 EEQ statement, LEF syntax [553](#)  
 electrically equivalent models, LEF  
     syntax [553](#)  
 endcap models, definition in LEF [549](#)  
 equivalent models  
     electrically equivalent (EEQ) [553](#)  
 error checking, utilities [640](#)  
 escape character [648](#)

## F

feedthrough pins  
     LEF [599](#)  
 FOREIGN references  
     LEF syntax [553](#)  
     offset between LEF and GDSII [553](#)

## G

GCell grid  
     restrictions [671](#)  
     uniform, in DEF [671](#)  
 GCELLGRID statement  
     description, DEF [670](#)  
 GROUPS statement  
     description, DEF [672](#)

## H

HISTORY statement  
     description, DEF [673](#)

## I

INOUT pins, netlist [592](#)  
 INPUT DEF command  
     error checking [640](#)  
 INPUT GDSII command  
     with incremental LEF [639](#)  
 INPUT LEF command  
     error checking [640](#)  
     incremental capability [639](#)  
 INPUT pins, netlist [592](#)

italics in syntax [7](#)

## L

LAYER (nonrouting) statement  
     description, LEF [15](#), [221](#)  
 layers  
     for LEF via descriptions [617](#)  
     routing order in LEF [259](#)  
 LEF  
     example [775](#)  
     files  
         distance precision [10](#)  
         line length [10](#)  
         overview [10](#)  
         routing layer order [259](#)  
 LEF syntax  
     overview [12](#)  
 LEF syntax and description  
     LAYER, nonrouting [15](#), [221](#)  
     MACRO [545](#)  
     NONDEFAULT rule [603](#)  
     OBS, macro obstruction [588](#)  
     PIN macro [591](#)  
     PROPERTYDEFINITIONS [608](#)  
     SITE [609](#)  
     UNITS [612](#)  
     VERSION [616](#)  
     VIA [616](#)  
     VIARULE [627](#)  
     VIARULE viaRuleName  
         GENERATE [629](#)  
 LEF values converted to integer  
     values [613](#)  
 legal characters [647](#)  
 library design, simplifying blockages [590](#)  
 literal characters [7](#)

## M

macro obstruction, OBS statement  
     description, LEF [582](#), [588](#)  
 macro PIN statement  
     description, LEF [591](#)  
 MACRO statement  
     description, LEF [545](#)  
 models, site orientation [557](#)  
 mustjoin pins [595](#)

### N

netlist pins

    INOUT [592](#)

    INPUT [592](#)

    OUTPUT [592](#)

nets

    mustJoin nets [675](#)

NETS statement

    description, DEF [673](#)

NONDEFAULT rule statement

    description, LEF [603](#)

### O

OBS (macro obstruction) statement

    description, LEF [582](#), [588](#)

obstructions, simplified [590](#)

Or-bars in syntax [8](#)

orientation

    models [557](#)

    pin [664](#), [695](#), [715](#)

OUTPUT pins, netlist [592](#)

overlaps, specifying in LEF [591](#)

### P

peripheral capacitance [269](#)

PIN (macro) statement

    description, LEF [591](#)

PINPROPERTIES statement

    description, DEF [717](#)

pins

    abutment [599](#)

    direction in LEF [595](#)

    external, DEF [700](#)

    feedthrough pins in LEF [599](#)

    INOUT [592](#)

    INPUT [592](#)

    modeling in LEF [592](#)

    mustjoin [595](#)

    netlist [592](#)

    orientation [664](#), [695](#), [715](#)

    OUTPUT [592](#)

    power geometries [599](#)

    ring [599](#)

    using in LEF [600](#)

PINS statement

    syntax, DEF [700](#)

PITCH parameter, ratio in three-layer

    design [509](#)

placement site function, SITE statement in

    LEF [610](#)

ports

    in LEF [597](#)

    multiple pins [597](#)

power pin

    geometries in LEF [599](#)

PROPERTYDEFINITIONS statement

    description, DEF [718](#)

    description, LEF [608](#)

### R

REGIONS statement

    description, DEF [719](#)

regular wiring

    orthogonal paths [689](#), [744](#)

RGrid, description [509](#)

ring pins [599](#)

routing time, diagonal vias [509](#)

routing width, LEF syntax [307](#)

ROW statement

    description, DEF [721](#)

### S

scan chains

    example [791](#)

    rules [726](#)

SI units in LEF [613](#)

SITE statement

    description, LEF [609](#)

sites

    symmetry [611](#)

special wiring

    description [734](#)

    pins and wiring, DEF [745](#)

SPECIALNETS statement

    description, DEF [729](#)

syntax conventions [7](#)

### T

TECHNOLOGY statement

    description, DEF [758](#)

three-layer design, pitch ratio [509](#)  
TRACKS statement  
    description, DEF [759](#)

## U

UNITS DISTANCE MICRONS statement  
    description, DEF [760](#)  
UNITS statement  
    description, LEF [612](#)

## V

values in library database [613](#)  
VERSION statement  
    description, DEF [761](#)  
vertical bars in syntax [8](#)  
VIA statement  
    description, LEF [616](#)  
VIARULE statement  
    description, LEF [627](#)  
VIARULE viaRuleName GENERATE  
    statement  
    description, LEF [629](#)  
vias  
    default vias in LEF [617](#)  
    layers for vias in LEF [617](#)  
VIAS statement  
    description, DEF [761](#)

## W

wide wire signal wire, specifying [603](#)  
wiring, regular  
    orthogonal paths [689](#), [744](#)  
wiring, special  
    description [734](#)  
    pins and wiring [745](#)