

# **LEF/DEF 5.8 Language Reference**

**Product Version 5.8**  
**May 2017**

© 2017 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> .....	42
<u>Layer (Masterslice or Overlap)</u> .....	45
<u>Layer (Routing)</u> .....	50
<u>Library</u> .....	112
<u>Macro</u> .....	114
<u>Manufacturing Grid</u> .....	148
<u>Maximum Via Stack</u> .....	149
<u>Nondefault Rule</u> .....	149
<u>Property Definitions</u> .....	154
<u>Site</u> .....	155
<u>Units</u> .....	158
<u>Use Min Spacing</u> .....	161
<u>Version</u> .....	161

## LEF/DEF 5.8 Language Reference

---

<u>Via</u> .....	162
<u>Via Rule</u> .....	173
<u>Via Rule Generate</u> .....	175

## 2

### ALIAS Statements .....

<u>ALIAS Statements</u> .....	215
<u>ALIAS Definition</u> .....	216
<u>ALIAS Examples</u> .....	216
<u>ALIAS Expansion</u> .....	217

## 3

### Working with LEF .....

<u>Incremental LEF</u> .....	219
<u>Error Checking</u> .....	220
<u>Message Facility</u> .....	221
<u>Error-Checking Facility</u> .....	223

## 4

### DEF Syntax .....

<u>About Design Exchange Format Files</u> .....	226
<u>General Rules</u> .....	227
<u>Character Information</u> .....	227
<u>Order of DEF Statements</u> .....	231
<u>DEF Statement Definitions</u> .....	232
<u>Blockages</u> .....	232
<u>Bus Bit Characters</u> .....	236
<u>Component Mask Shift</u> .....	236
<u>Components</u> .....	237
<u>Design</u> .....	244
<u>Die Area</u> .....	245
<u>Divider Character</u> .....	245
<u>Extensions</u> .....	245
<u>Fills</u> .....	246

## LEF/DEF 5.8 Language Reference

---

<u>GCell Grid</u>	250
<u>Groups</u>	252
<u>History</u>	253
<u>Nets</u>	253
<u>Nondefault Rules</u>	276
<u>Pins</u>	279
<u>Pin Properties</u>	296
<u>Property Definitions</u>	296
<u>Regions</u>	298
<u>Rows</u>	299
<u>Scan Chains</u>	300
<u>Slots</u>	306
<u>Special Nets</u>	307
<u>Styles</u>	324
<u>Technology</u>	336
<u>Tracks</u>	336
<u>Units</u>	338
<u>Version</u>	339
<u>Vias</u>	339

## A

<u>Examples</u>	353
<u>LEF</u>	353
<u>DEF</u>	364
<u>Scan Chain Synthesis Example</u>	369

## B

<u>Optimizing LEF Technology for Place and Route</u>	371
<u>Overview</u>	371
<u>Guidelines for Routing Pitch</u>	372
<u>Guidelines for Wide Metal Spacing</u>	374
<u>Guidelines for Wire Extension at Vias</u>	375
<u>Guidelines for Default Vias</u>	377
<u>Guidelines for Stack Vias (MAR Vias) and Samenet Spacing</u>	379
<u>Example of an Optimized LEF Technology File</u>	383

## C

<u>Calculating and Fixing Process Antenna Violations</u> .....	389
<u>Overview</u> .....	390
<u>What Are Process Antennas?</u> .....	391
<u>What Is the Process Antenna Effect (PAE)?</u> .....	392
<u>What Is the Antenna Ratio?</u> .....	393
<u>What Can Be Done to Improve the Antenna Ratio?</u> .....	393
<u>Using Process Antenna Keywords in the LEF and DEF Files</u> .....	394
<u>Calculating Antenna Ratios</u> .....	395
<u>Calculating the Antenna Area</u> .....	395
<u>Calculating a PAR</u> .....	396
<u>Calculating a CAR</u> .....	401
<u>Calculating Ratios for a Cut Layer</u> .....	409
<u>Checking for Antenna Violations</u> .....	412
<u>Area Ratio Check</u> .....	413
<u>Side Area Ratio Check</u> .....	413
<u>Cumulative Area Ratio Check</u> .....	414
<u>Cumulative Side Area Ratio Check</u> .....	415
<u>Cut Layer Process Antenna Model Examples</u> .....	415
<u>Routing Layer Process Antenna Model Examples</u> .....	416
<u>Example Using the Antenna Keywords</u> .....	422
<u>Using Antenna Diode Cells</u> .....	423
<u>Changing the Routing</u> .....	424
<u>Inserting Antenna Diode Cells</u> .....	424
<u>Using DiffUseOnly</u> .....	424
<u>Calculations for Hierarchical Designs</u> .....	425
<u>LEF and DEF Keywords for Hierarchical Designs</u> .....	426
<u>Design Example</u> .....	426
<u>Top-Down Hierarchical Design Example</u> .....	429
<u>Index</u> .....	431

---

# 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 42
  - ❑ Layer (Masterslice or Overlap) on page 45
  - ❑ Layer (Routing) on page 50
  - ❑ Library on page 112
  - ❑ Macro on page 114
    - Layer Geometries on page 130
    - Macro Obstruction Statement on page 136
    - Macro Pin Statement on page 139

- ❑ [Manufacturing Grid](#) on page 148
- ❑ [Maximum Via Stack](#) on page 149
- ❑ [Nondefault Rule](#) on page 149
- ❑ [Property Definitions](#) on page 154
- ❑ [Site](#) on page 155
- ❑ [Units](#) on page 158
- ❑ [Use Min Spacing](#) on page 161
- ❑ [Version](#) on page 161
- ❑ [Via](#) on page 162
- ❑ [Via Rule](#) on page 173
- ❑ [Via Rule Generate](#) on page 175

## 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 227.

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

For information, see [Name Escaping Semantics for Identifiers](#) on page 228.

## **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 ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[ANTENNAAREADIFFREDUCEPWL
  ( ( diffArea1 diffAreaFactor1 ) ( diffArea2 diffAreaFactor2 ) ... ) ; ]
```

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 53.

The ACCURRENTDENSITY syntax is defined as follows:

```
{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 ...
    ...
} ;
```

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



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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>
CUTAREA	<p>Specifies cut area values, in square microns (<math>\mu\text{m}^2</math>). You can specify more than one cut area. If you specify multiple cut area values, the values must be specified in ascending order.</p> <p>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.</p> <p><i>Type:</i> Float</p>
TABLEENTRIES	<p>Defines the maximum current for each frequency and cut area pair specified in the <code>FREQUENCY</code> and <code>CUTAREA</code> statements, in <math>\text{mA}/\mu\text{m}^2</math>.</p> <p>The pairings define each cut area for the first frequency in the <code>FREQUENCY</code> 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.</p> <p><i>Type:</i> Float</p>

```
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  $\text{PAR}(m_i)$  equation defaults to 1.0.

For more information on the  $\text{PAR}(m_i)$  equation and process antenna models, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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 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

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

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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

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 58.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### 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.

ARRAYCUTS *arrayCuts* SPACING *arraySpacing*

Indicates that a large via array with a size greater than or equal to  $arrayCuts \times arrayCuts$  in both dimensions must use  $N \times N$  cut arrays (where  $N = arrayCuts$ ) 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  $N = arrayCuts$ , and  $M$  can be any value, including one that is larger than  $N$ . (See [Array Spacing Rule Example 2](#).)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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

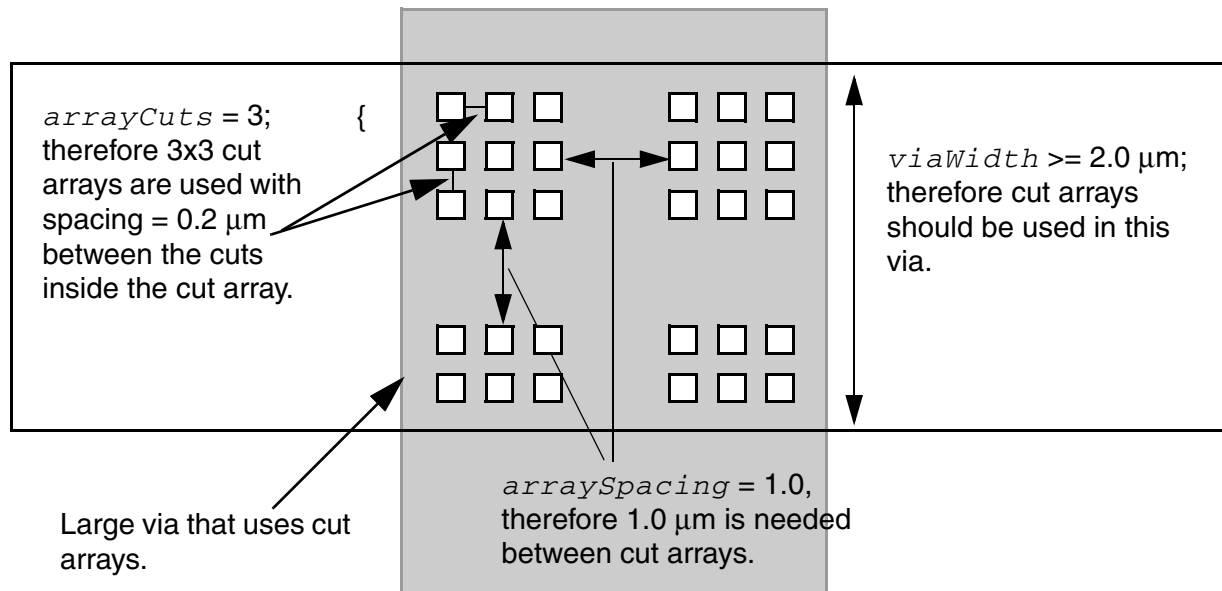
Assume the following array spacing rule exists:

```
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 22.

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 23:

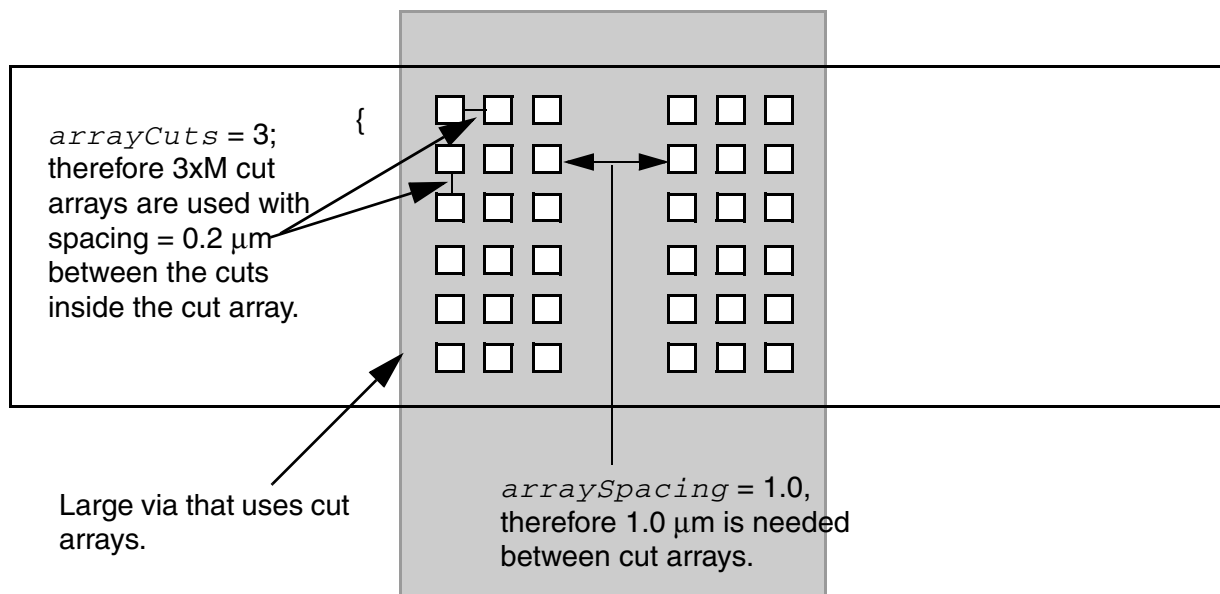
```
ARRAYSPACING LONGARRAY CUTSPACING 0.2 ARRAYCUTS 3 SPACING 1.0 ;
```

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 60.

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.

*value* Specifies a current limit for the layer in mA/μm<sup>2</sup>.  
*Type:* 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.  
*Type:* 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.  
*Type:* 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 26.)

*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 26.)

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 27.)

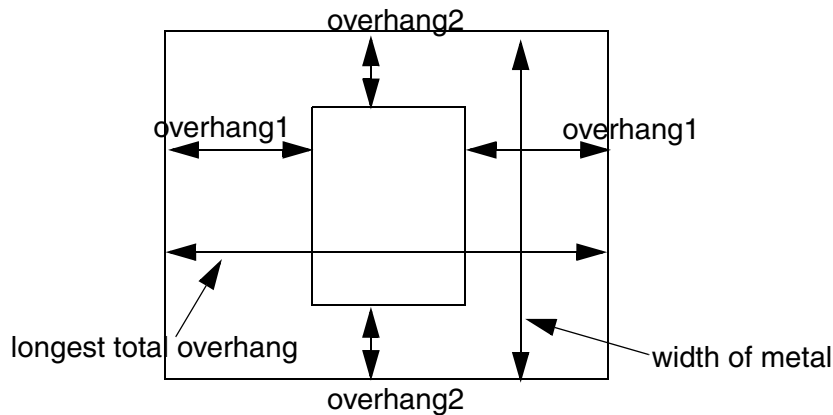
*Type:* Float, specified in microns

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 30).

*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  $\mu\text{m}$ :

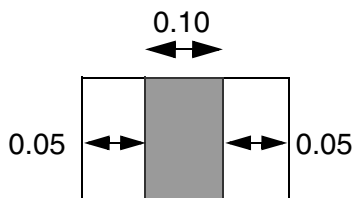
```
LAYER via34
TYPE CUT ;
WIDTH 0.25 ;                               #cuts .25 x .25 squares
ENCLOSURE .05 .01 ;                         #minimum width enclosure rule
ENCLOSURE BELOW .07 .07 WIDTH 1.0 ; #m3 needs .07 on all sides if m3 width >=1.0
ENCLOSURE ABOVE .09 .09 WIDTH 1.0 ; #m4 needs .09 on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

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

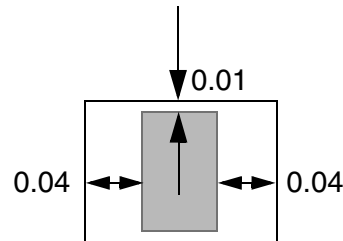
The following definition describes a cut layer that requires an enclosure of either .05  $\mu\text{m}$  on opposite sides and 0.0  $\mu\text{m}$  on the other two sides, or 0.04  $\mu\text{m}$  on opposites sides and 0.01  $\mu\text{m}$  on the other two sides. It also requires an enclosure of 0.03  $\mu\text{m}$  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
```

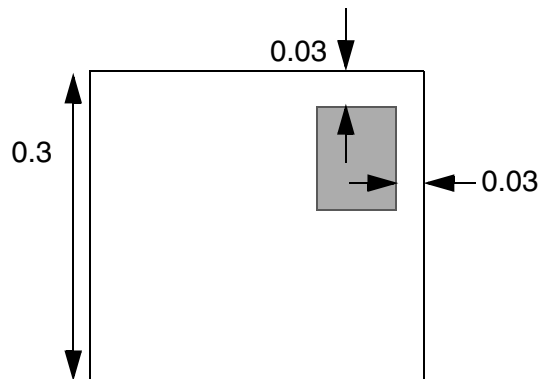
**Figure 1-4 Illustrations of Enclosure Rule With Width and ExceptExtraCut**



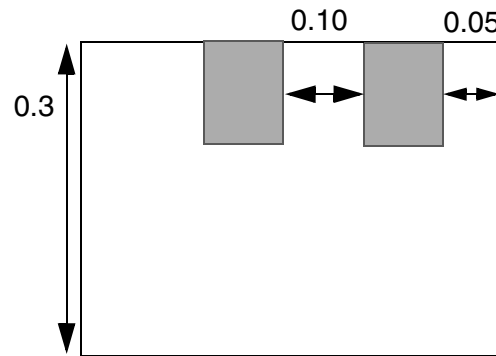
a) Okay; has 0.0 and 0.05 overhang.



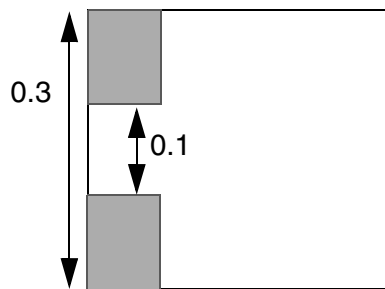
b) Okay; has 0.01 and 0.04 overhang.



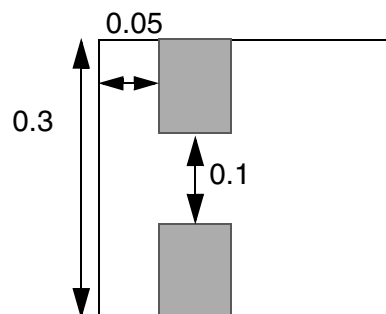
c) Okay; meets wide-wire enclosure rule of 0.03 0.03.



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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

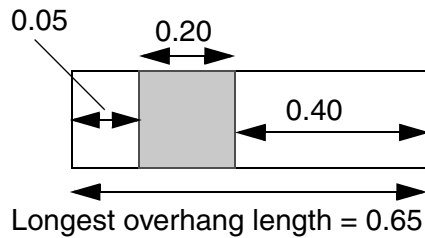
---

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

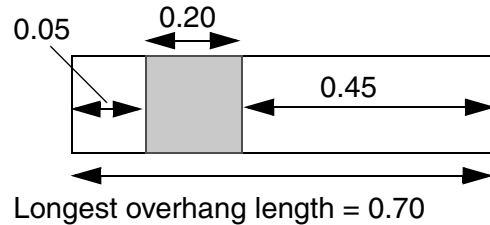
The following definition describes a cut layer that requires an enclosure of .05  $\mu\text{m}$  on opposite sides and 0.0  $\mu\text{m}$  on the other two sides, as long as the total length enclosure on any two opposite sides is greater than or equal to 0.7  $\mu\text{m}$ . Otherwise, it requires 0.05  $\mu\text{m}$  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 30 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
```

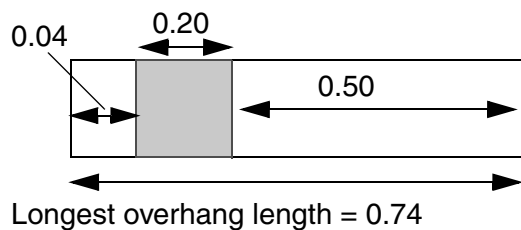
**Figure 1-5 Illustrations of Enclosure Rule With Length and Width**



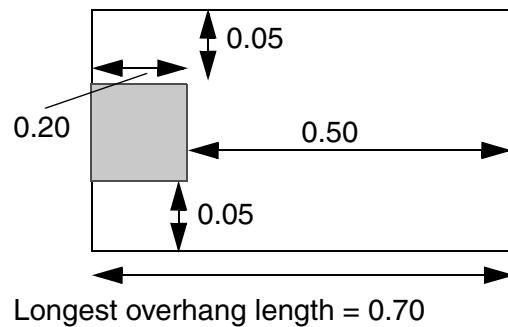
a) Violation. Longest overhang length  $< 0.70$ , and did not meet second rule of 0.05 on all sides.



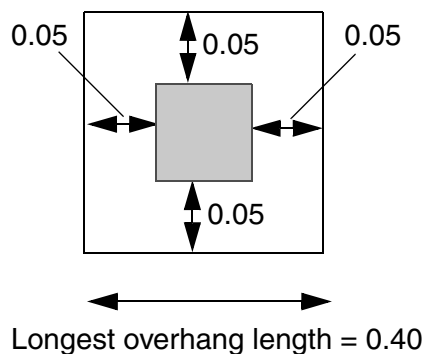
b) Okay. Longest overhang length  $\geq 0.70$ , and has 0.05 on opposite sides, and 0.0 on other sides.



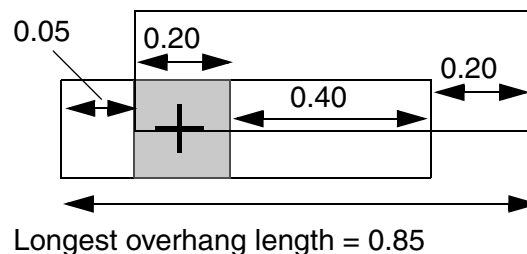
c) Violation. Longest overhang length  $\geq 0.70$ , but does not have 0.05 on opposite sides, and did not meet second rule of 0.05 on all sides.



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.

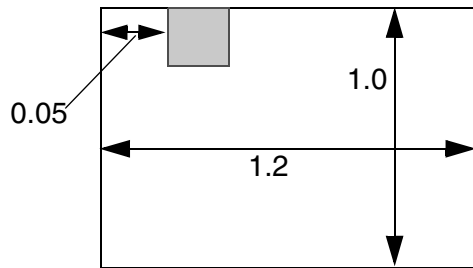


f) Okay. Overhang length  $\geq 0.70$ . (The center of the via cut is where the total overhang length is measured.)

## 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 34.)

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

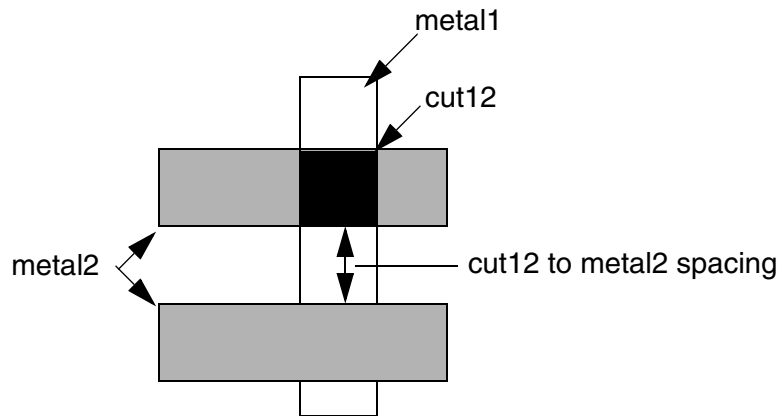
## Example 1-6 Spacing Rule Examples

### ■ 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 36). A cut that is within .25  $\mu\text{m}$  of three other cuts requires spacing that is greater than or equal to 0.22  $\mu\text{m}$ .

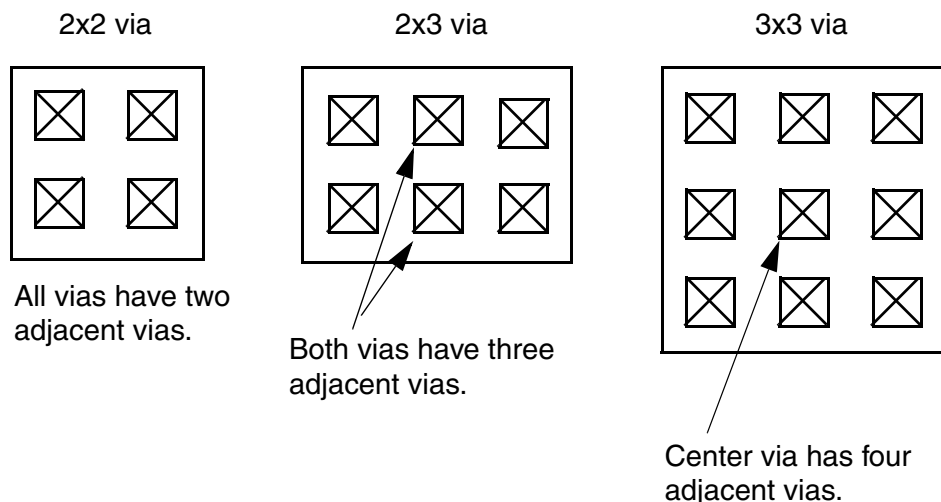
```
LAYER CUT12
    SPACING 0.20 ;                                #default cut spacing
    SPACING 0.22 ADJACENTCUTS 3 WITHIN 0.25 ;
    ...
END CUT12
```

### 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 36 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 1xn vias and 2x2 vias, but requires larger cut spacing on 2x3, 2x4 and 3xn vias. For four adjacent cuts, the rule allows tight cut spacing on 2xn vias, but it requires larger cut spacing on 3xn 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 36). 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

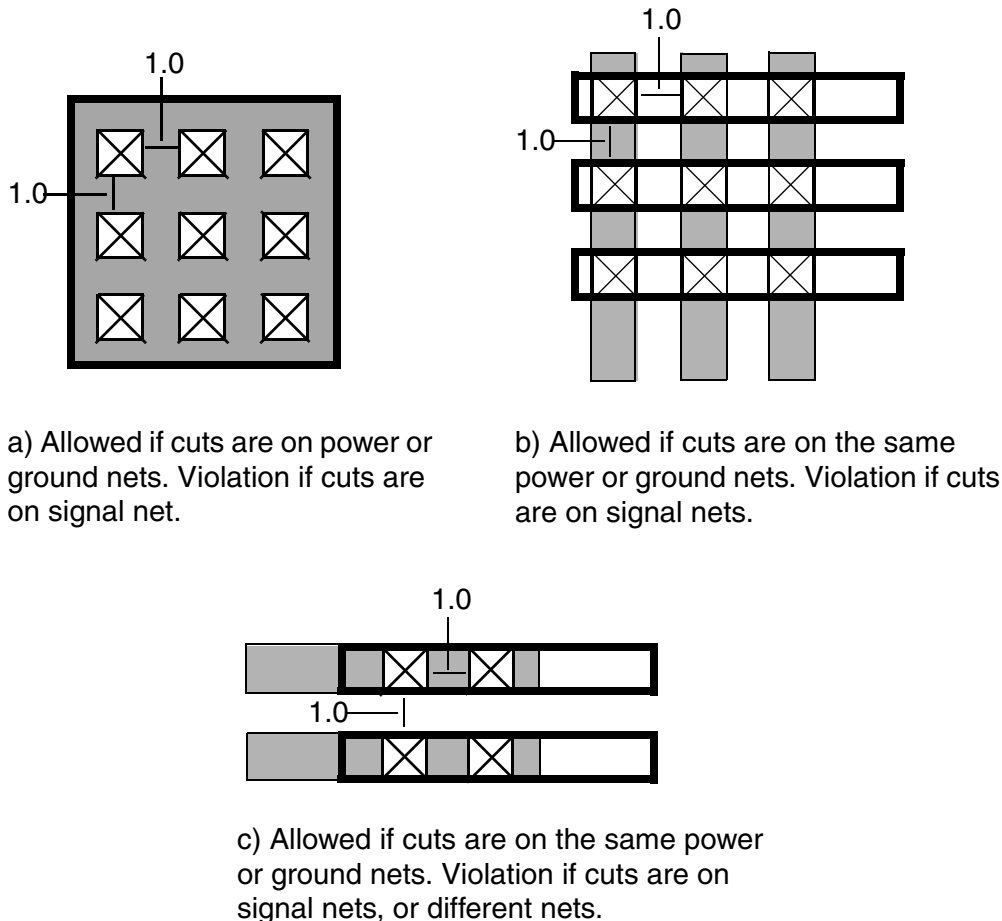
[Figure 1-7](#) on page 37 illustrates the following spacing rule:

## 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

---

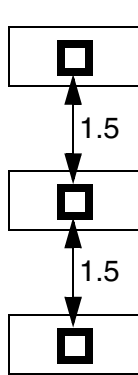
```
LAYER cut23 ;  
SPACING 0.20 SAMENET LAYER cut12 STACK ;
```

#### ■ Spacing Rule Example 8

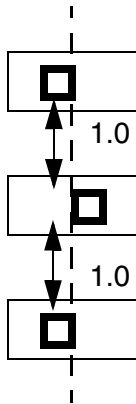
Figure 1-8 on page 39 illustrates the following spacing rule:

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

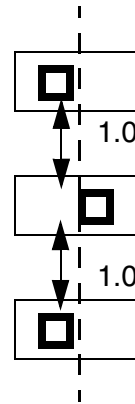
**Figure 1-8 Parallel Overlap Rule**



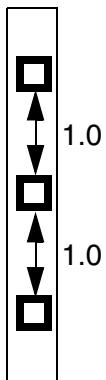
a) Okay. Cuts have parallel overlap > 0; therefore PARALLELOVERLAP rule of 1.5 applies.



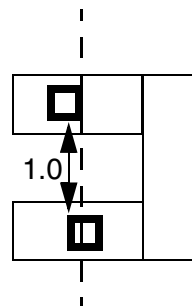
b) Okay. Cuts have parallel overlap = 0; therefore PARALLELOVERLAP rule does not apply, and only 1.0 spacing is needed.



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 34, and [Figure 1-9](#) on page 41.)

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

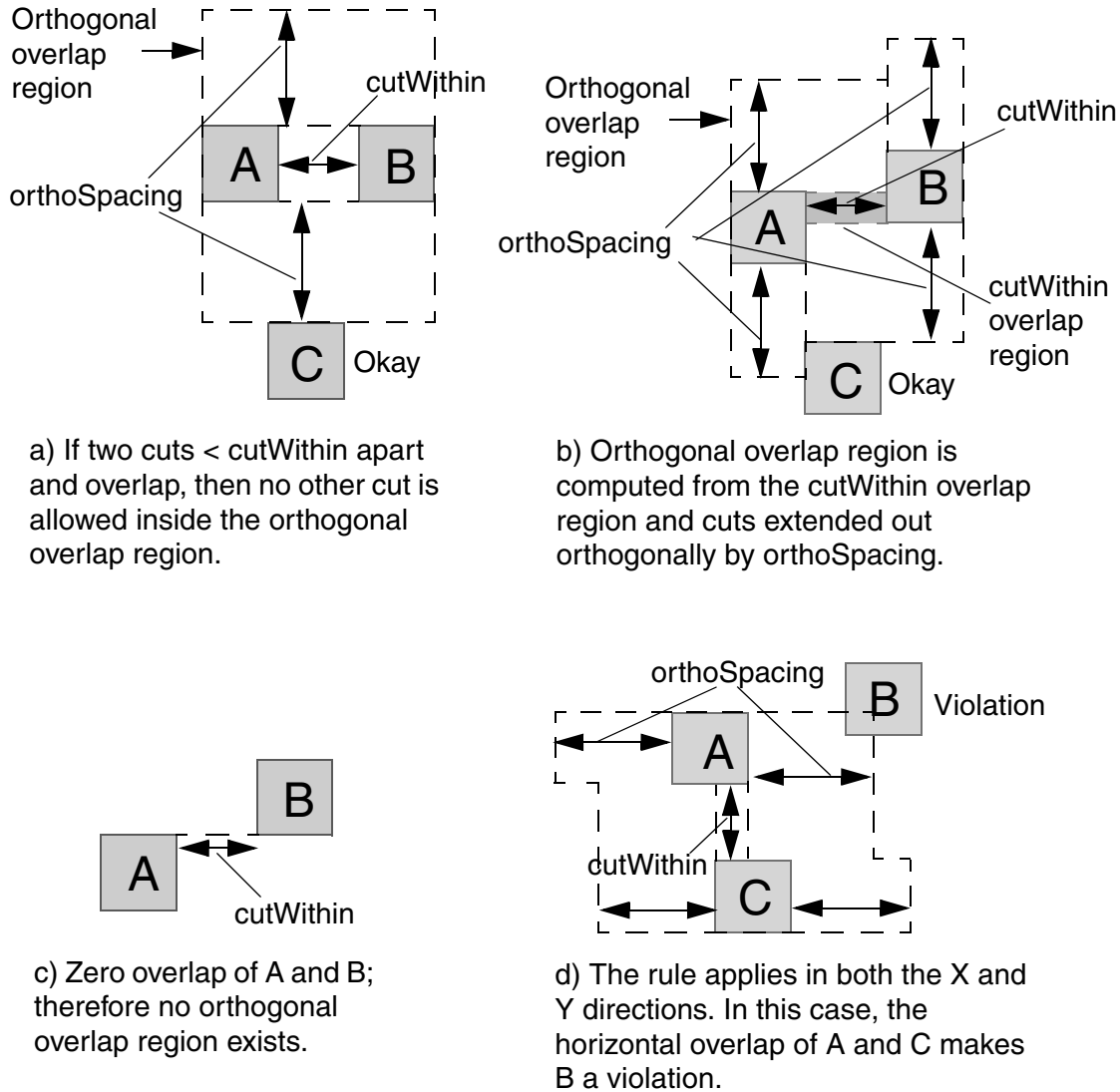
#### Example 1-7 Spacing Table Orthogonal Rule

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**



#### 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

---

#### Layer (Implant)

```
LAYER layerName
    TYPE IMPLANT ;
    [MASK maskNum ;]
    [WIDTH minWidth ;]
    [SPACING minSpacing [LAYER layerName2] ;] ...
    [PROPERTY propName propVal ;] ...
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.
PROPERTY <i>propName</i> <i>propVal</i>	Specifies a numerical or string value for a layer property defined in the PROPERTYDEFINITIONS statement. The <i>propName</i> you specify must match the <i>propName</i> listed in the PROPERTYDEFINITIONS statement.
SPACING <i>minSpacing</i>	Specifies the minimum spacing for the layer. This value affects the legal cell placement. <i>Type:</i> Float, specified in microns
TYPE IMPLANT	Identifies the layer as an implant layer.
WIDTH <i>minWidth</i>	Specifies the minimum width for this layer. This value affects the legal cell placement. <i>Type:</i> Float, specified in microns

#### Example 1-8 Implant Layer

## LEF/DEF 5.8 Language Reference

### LEF Syntax

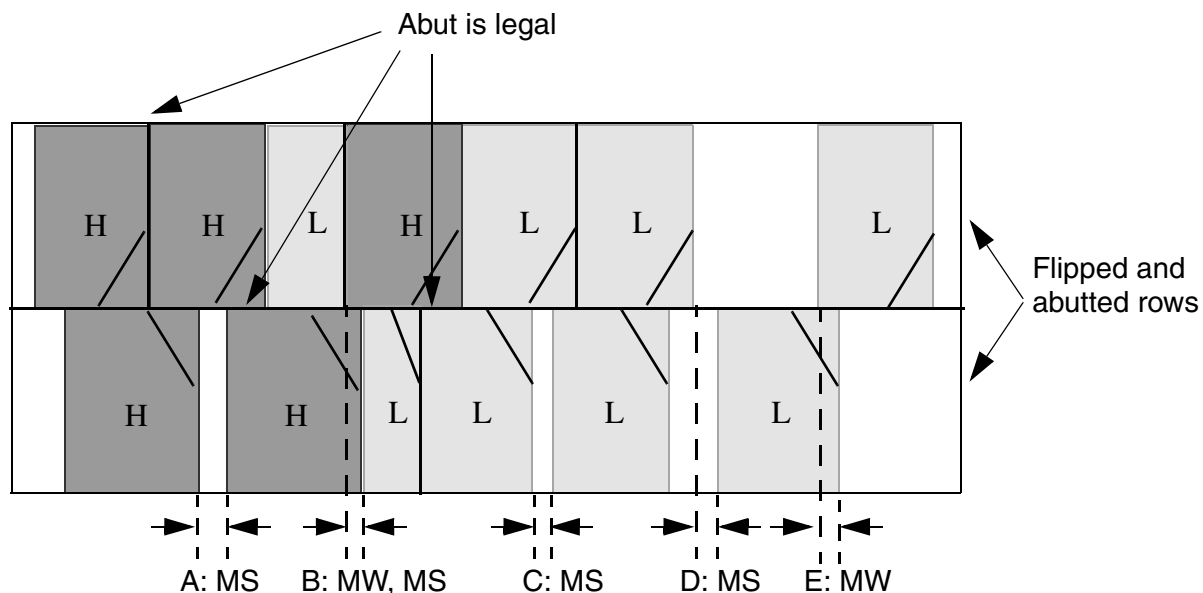
Typically, you define high-drive cells on one implant layer and low-drive cells on another implant layer. The following example defines high-drive cells on *implant1* and low-drive cells on *implant2*. Both implant layers cover the entire cell. The placer and filler cell creation attempt to legalize 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 1-10](#) on page 43.

**Figure 1-10**



MS = Minimum spacing error  
MW = Minimum width error

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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.

## Layer (Masterslice or Overlap)

```
LAYER layerName
    TYPE {MASTERSLICE | OVERLAP} ;
    [MASK maskNum ;]
    [PROPERTY propName propVal ;] ...
    [PROPERTY LEF58 TYPE
        "TYPE [NWELL | PWELL | ABOVEEDGE | BELOWDIEEDGE | DIFFUSION | TRIMPOLY
            | TRIMMETAL | REGION]
        ];" ;
    [PROPERTY LEF58 TRIMMEDMETAL
        "TRIMMEDMETAL metalLayer [MASK maskNum]
        ];" ;
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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### 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.

#### 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.

### 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 [NWELL | PWELL | ABOVEEDGE | BELOWDIEEDGE | DIFFUSION | TRIMPOLY  
      | TRIMMETAL | REGION]  
    ;" ;
```

Where:

ABOVEEDGE | BELOWDIEEDGE

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

	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.
DIFFUSION	Defines a diffusion layer.
NWELL	Indicates that the layer is a nwell layer.
PWELL	Indicates that the layer is a pwell layer.
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 <code>REGION</code> property would be applied.
TRIMMETAL	<p>Defines a trim metal layer. This layer type is only used along with metal layers manufactured with self-aligned double patterning (SADP) technology. The <code>TRIMMETAL</code> 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 <code>TRIMMETAL</code> layer.</p> <p>The <code>TRIMMETAL</code> layer can have the following property to indicate which metal layer or which mask of the metal layer can be trimmed by the <code>TRIMMETAL</code> shapes. The metal layer should have the <code>MASK</code> construct with value of 2 or larger to indicate that it is SADP (or DPT) layer. The <code>TRIMMETAL</code> layer could also have the <code>MASK</code> construct to indicate number of masks on that layer.</p> <pre>PROPERTY LEF58_TRIMMEDMETAL     "TRIMMEDMETAL metalLayer [MASK maskNum]     ; " ;</pre>
TRIMPOLY	Defines a trim poly layer for self-aligned double patterning (SADP) technology. Only cells would have shapes on that layer.

### 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 ;" ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
END TOPDIE

MACRO A
...
  OBS
    LAYER TOPDIE
    RECT 0.000 0.000 100.000 100.000 ;
  END
...
END A
```

### Trimmed Metal Rule

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 ; " ;
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
...  
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
```

```
LAYER M2  
    TYPE ROUTING ;  
    MASK 2 ;  
...  
END M2
```

## 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 ;]
[ANTENNA_MODEL {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_SPANLENGHTHABLE
    "SPANLENGHTHABLE {spanLength}... [WRONGDIRECTION]
    [ORTHOGONAL length] [EXCEPTOTHERSPAN otherSpanlength]
    ; " ;]
[PROPERTY LEF58_TYPE
    "TYPE {POLYROUTING}} ;" ;
[PROPERTY LEF58_WIDTHTABLE
    "WIDTHTABLE {width}...[WRONGDIRECTION] [ORTHOGONAL]
    ;" ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[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/μm).

**Note:** 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 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 ...
    ...
} ;
```

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. Type: Float

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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 <code>FREQUENCY</code> and <code>WIDTH</code> statements, in mA/<math>\mu</math>m.</p> <p>The pairings define each width for the first frequency in the <code>FREQUENCY</code> 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.

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  $\mu$ m is  $9.0 + (7.5 - 9.0) \times (0.8 - 0.7) / (0.8 - 0.4) = 8.625$  mA/ $\mu$ m at frequency 300Mhz. Therefore, a 0.7  $\mu$ m wide wire can carry  $8.625 \times 0.7 = 6.035$  mA of `RMS` current.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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
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 ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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 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.”](#)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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 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.”](#)



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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.”](#)

ANTENNADIFFAREARATIO {*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.”](#))

*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.”](#)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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, see [Appendix C, “Calculating and Fixing Process Antenna Violations.”](#)

**Type:** Integer

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### 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 68.

*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.
---------	---

<i>value</i>	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. <i>Type:</i> 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  $7.5 \times .4 = 3.0$  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 65.)

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 1-12](#) on page 67.)

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 1-11](#) on page 66). 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 1-11](#) on page 66 illustrates this example.

```
LAYER metal4
```

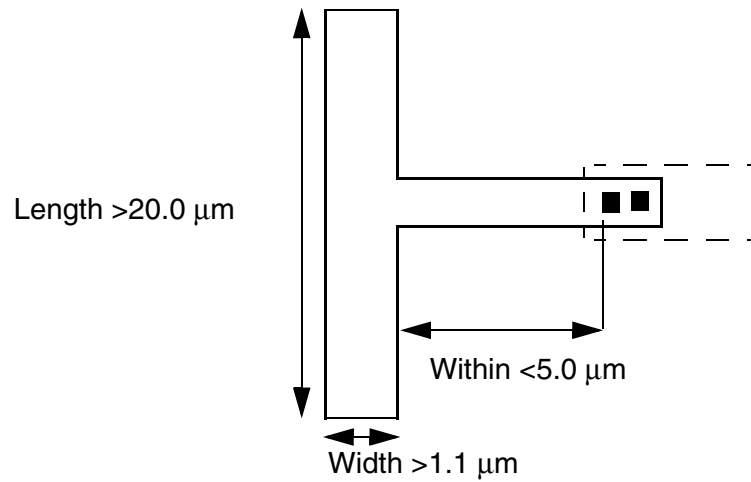
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
MINIMUMCUT 2 WIDTH 1.1 LENGTH 20.0 WITHIN 5.0 ;
```

**Figure 1-11 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 1-12](#) on page 67 illustrates this example.

```
MINIMUMCUT 2 WIDTH 1.0 WITHIN 0.3 ;
```

**Figure 1-12 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 63.

**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 1-13](#) on page 69). 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
    ....
```

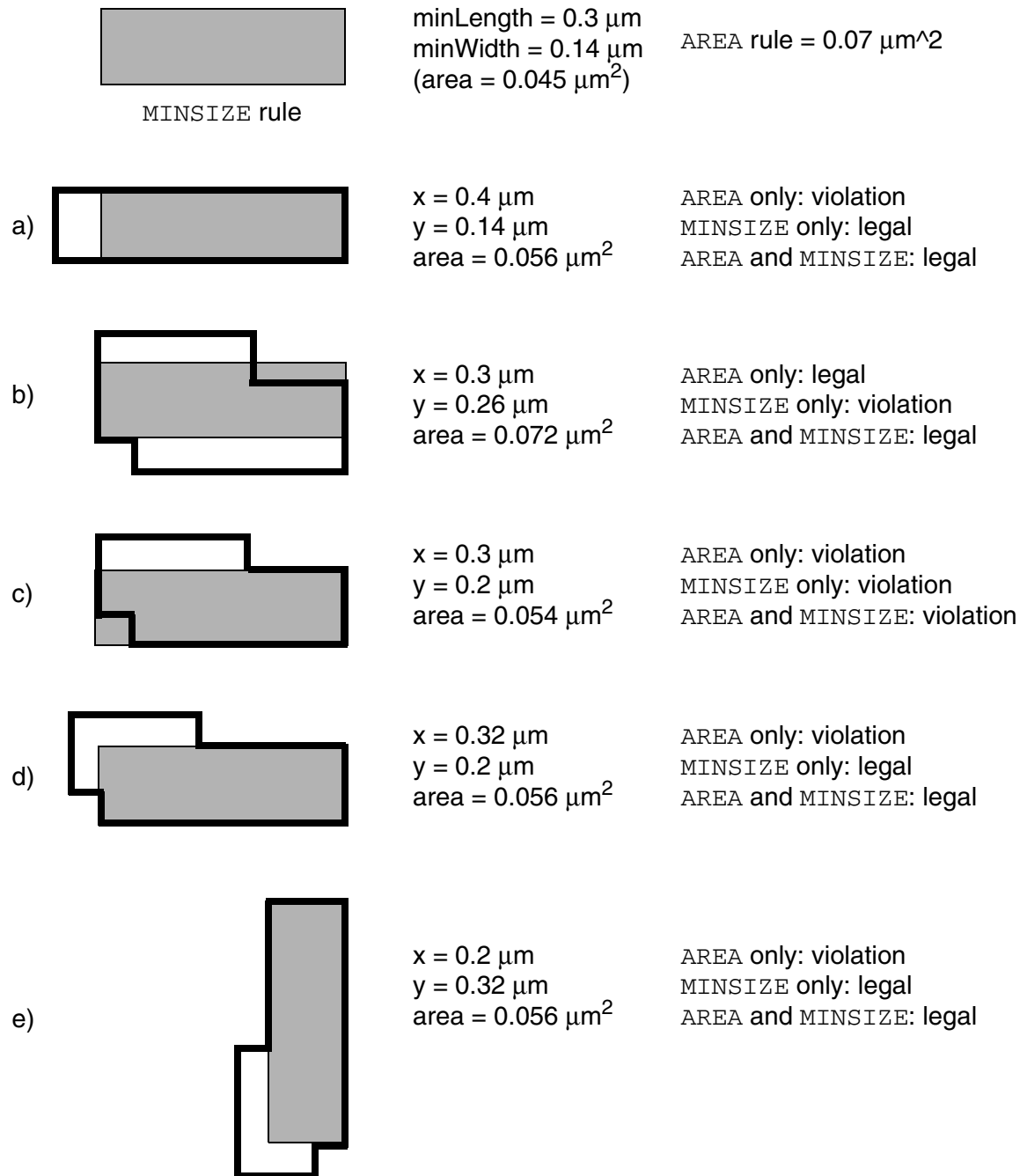
Figure 1-13 on page 69 illustrates how these rules behave when one or both of the rules are present in the `LAYER` statement:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**Figure 1-13 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 metal1

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 1-14](#) on page 74. For an example, see [Example 1-16](#) on page 72.

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 1-14](#) on page 74 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 1-14</a> on page 74 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 1-14</a> on page 74 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 1-14](#) on page 74. 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 $\mu\text{m}$ .



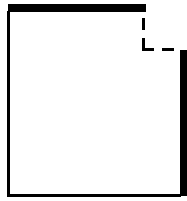
## LEF/DEF 5.8 Language Reference

### LEF Syntax

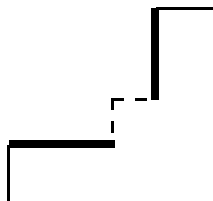
MINSTEP Rule	Result
MINSTEP 0.05 INSIDECORNER ;	Shapes <b>b</b> and <b>e</b> are violations because their consecutive edges are less than 0.05 $\mu\text{m}$ . Shapes <b>a</b> , <b>c</b> , <b>d</b> , and <b>f</b> are not inside corner checks.
MINSTEP 0.05 INSIDECORNER LENGTHSUM 0.15 ;	Shape <b>b</b> is 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 1-14**

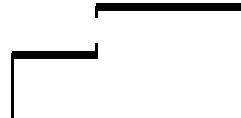
**Note:** All dashed edges are 0.04  $\mu\text{m}$  in length.



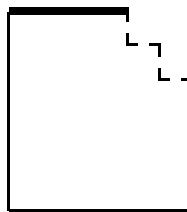
a) OUTSIDECORNER  
0.08  $\mu\text{m}$  LENGTHSUM



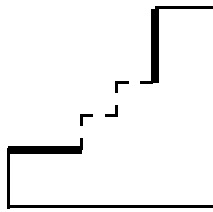
b) INSIDECORNER  
0.08  $\mu\text{m}$  LENGTHSUM



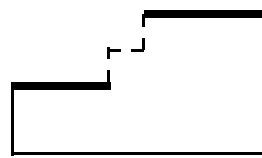
c) STEP  
0.04  $\mu\text{m}$



d) OUTSIDECORNER  
0.16  $\mu\text{m}$  LENGTHSUM



e) INSIDECORNER  
0.16  $\mu\text{m}$  LENGTHSUM

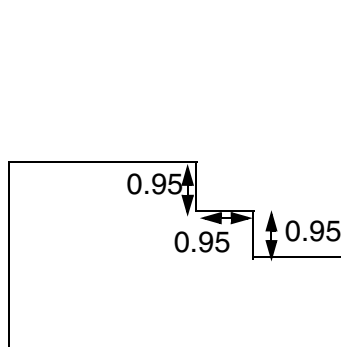


f) STEP  
0.12  $\mu\text{m}$  LENGTHSUM

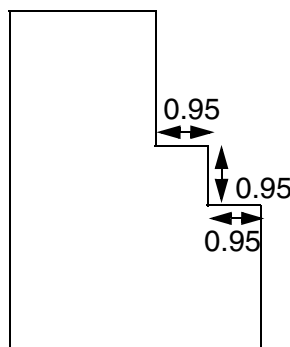
■ Figure 1-15 on page 74 shows the results of the following MINSTEP MAXEDGES rule:

MINSTEP 1.0 MAXEDGES 2 ;

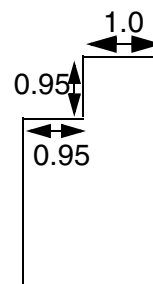
**Figure 1-15**



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 111.

**Type:** Float, specified in microns

*distance*                      Specifies one pitch value that is used for both the x and y pitch.

*xDistance yDistance*

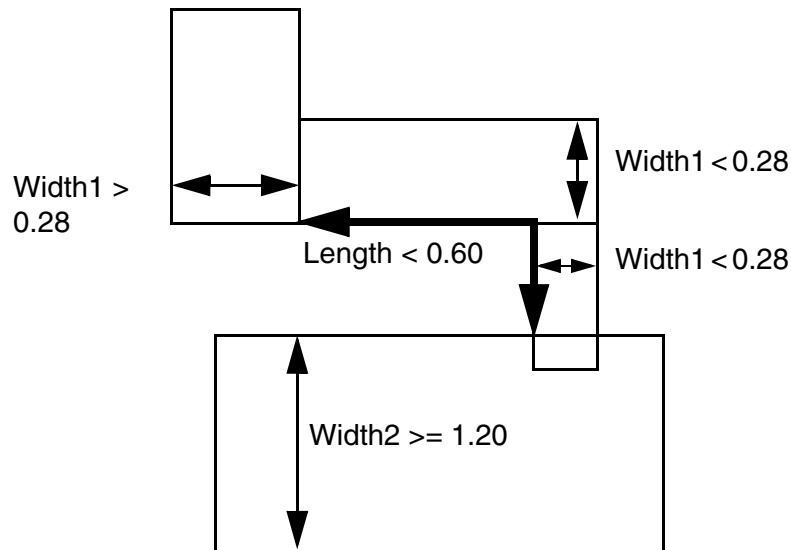
Specifies the x pitch (the space between each vertical routing track), and the y pitch (the space between each horizontal routing track).



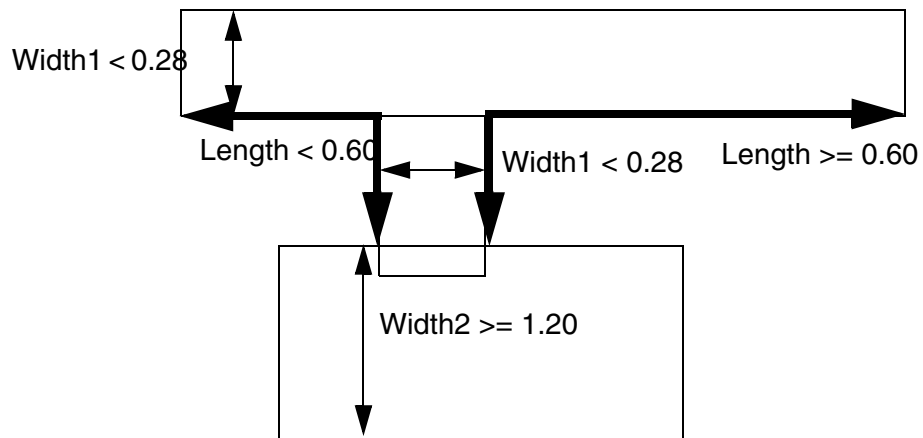
## LEF/DEF 5.8 Language Reference

### 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.

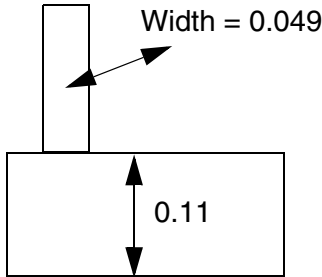
If the given value of `LENGTH` in `PROTRUSIONWIDTH` is zero, then the length of the protrusion wire is irrelevant. In this case, the width of the protrusion wire should always be checked independent of the length of the wire. The following example illustrates this rule:

```
PROTRUSIONWIDTH 0.05 LENGTH 0 WIDTH 0.11 ; " ;
```

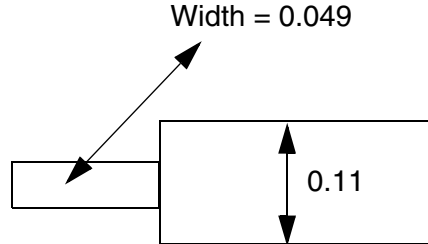
## LEF/DEF 5.8 Language Reference

### 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 84.

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 84.

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

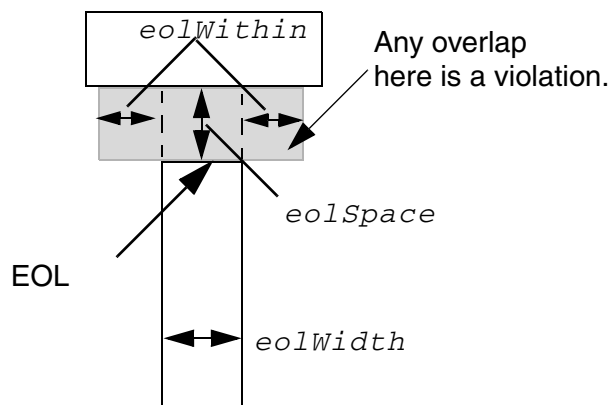
**Note:** If you specify multiple RANGE rules, the range values should not overlap.

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 1-17](#) on page 81).

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 1-17**



a) EOL width < *eolWidth* requires *eolSpace* beyond EOL to either side by < *eolWithin* distance.

## LEF/DEF 5.8 Language Reference

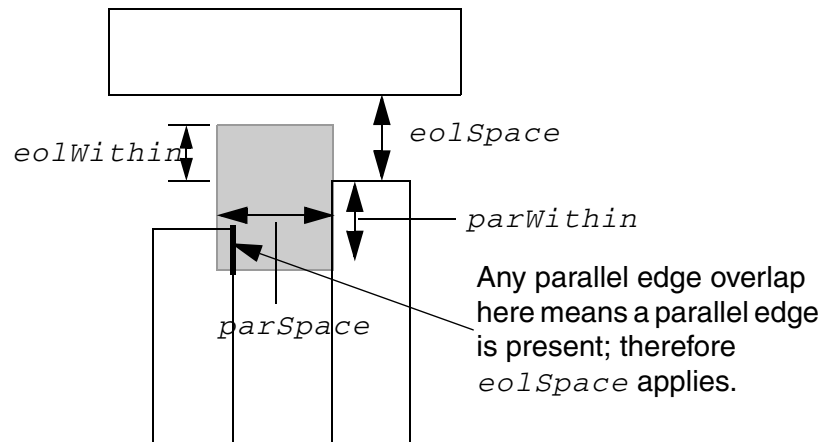
### LEF Syntax

---

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 1-18](#) on page 82).

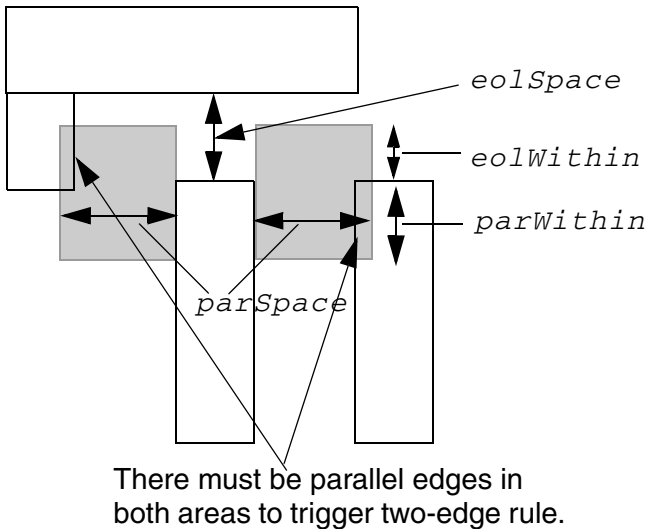
**Figure 1-18**



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 1-19](#) on page 83).

Figure 1-19



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 91.)

## 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 1-26](#) on page 92.)

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  $\mu\text{m}$ .

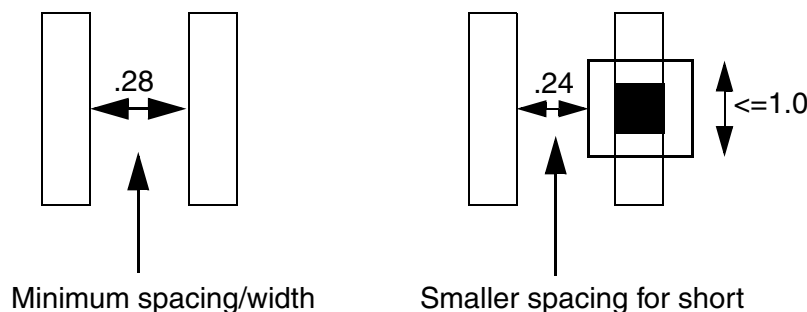
```
SPACING 0.3 ;
SPACING 0.4 RANGE 0.5 1.0 ;
SPACING 0.5 RANGE 1.01 2.0 ; #The RANGE begins at 1.01 and not 1.0 because
                              #RANGE values should not overlap.
```

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 1-20](#) on page 85).

```
SPACING 0.28 ;
SPACING 0.24 LENGTHTHRESHOLD 1.0 ;
```

**Figure 1-20**



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 .0.28  $\mu\text{m}$  spacing is applied (see [Figure 1-21](#) on page 86).

```
SPACING 0.28 ; #Default minimum spacing is >=0.28 um.
SPACING 0.28 LENGTHTHRESHOLD 1.0 ; #For short parallel lengths of <= 1.0 um,
```

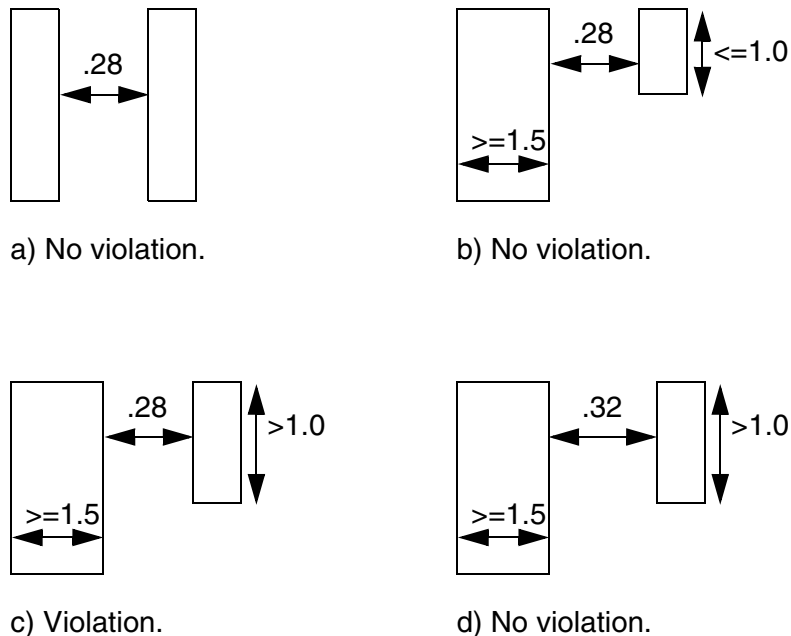
## LEF/DEF 5.8 Language Reference

### 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.
```

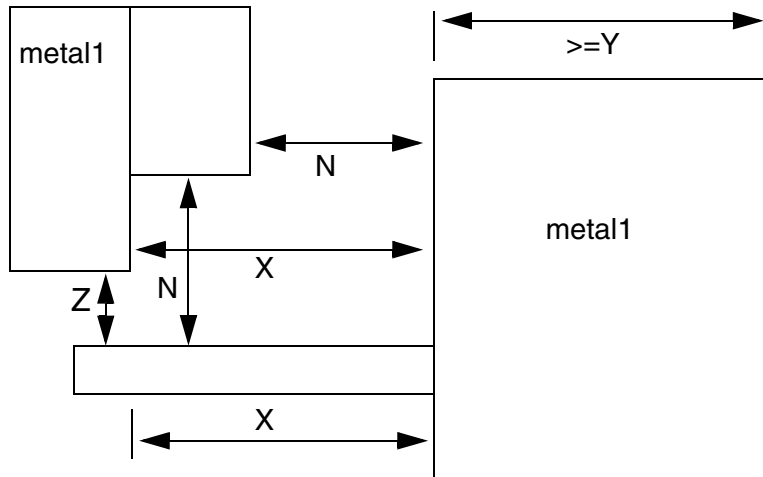
**Figure 1-21**



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 1-22](#) on page 87, 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 1-22**



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:

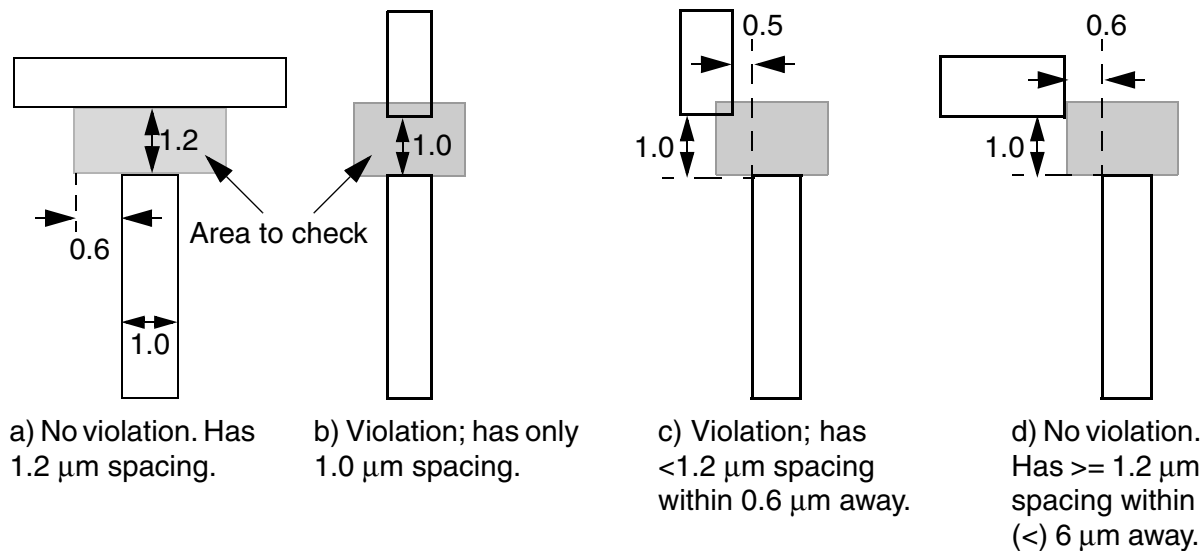
```
SPACING 1.0 ;                #minimum spacing is 1.0 um
SPACING 1.2 ENDOFLINE 1.3 WITHIN 0.6 ;
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

Any EOL that is less than  $1.3\text{ }\mu\text{m}$  wide requires spacing that is greater than or equal to  $1.2\text{ }\mu\text{m}$  beyond the EOL, within  $0.6\text{ }\mu\text{m}$  to either side. [Figure 1-23](#) on page 88 includes examples of legal spacing for, and violations of, this rule.

**Figure 1-23**



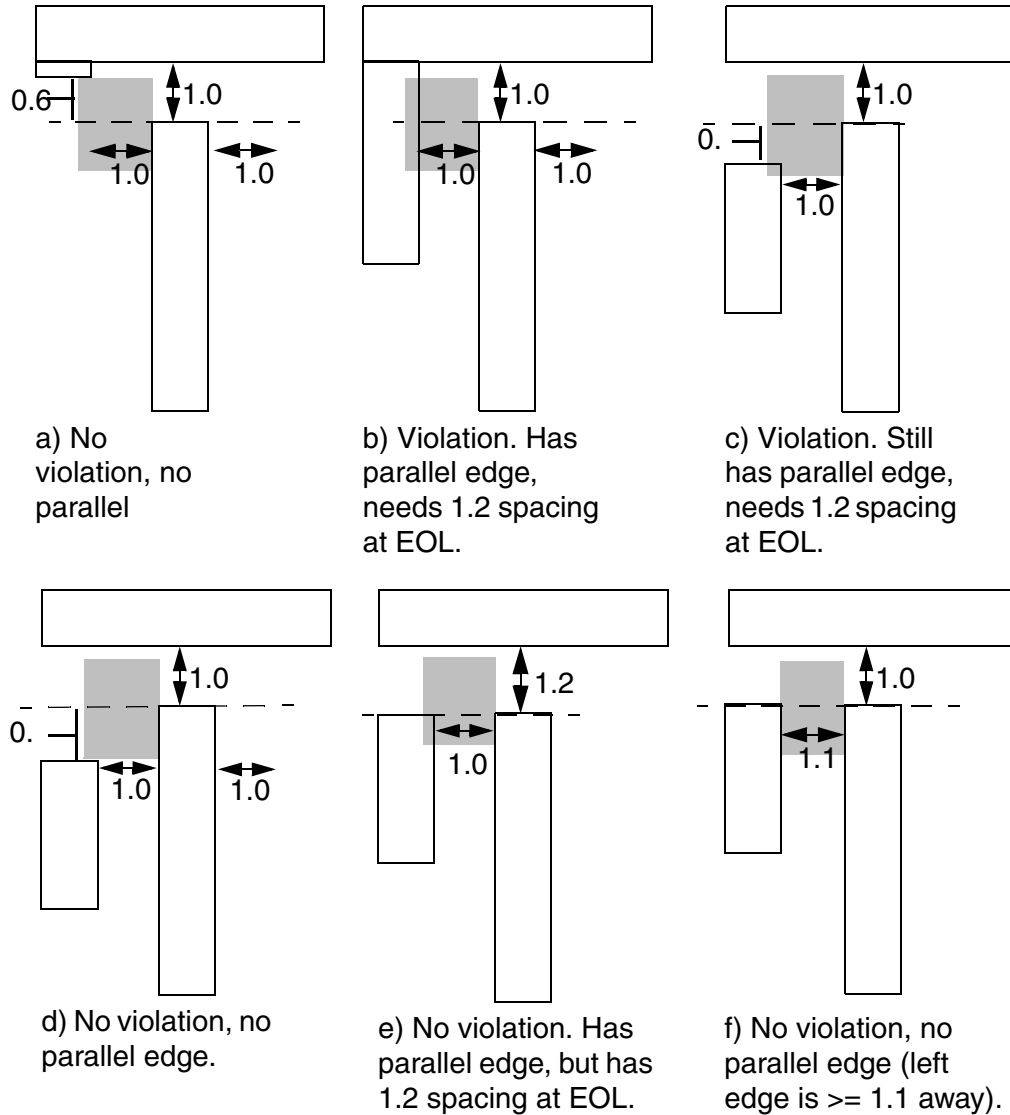
■ If you include the following routing layer rules in your LEF file:

```
SPACING 1.0 ;                               #minimum spacing is  $1.0\text{ }\mu\text{m}$ 
SPACING 1.2 ENDOFLINE 1.3 WITHIN 0.6 PARALLELEDGE 1.1 WITHIN 0.5 ;
```

Any line that is less than  $1.3\text{ }\mu\text{m}$  wide, with a parallel edge that is less than  $1.1\text{ }\mu\text{m}$  away, and is within  $0.5\text{ }\mu\text{m}$  of the EOL, requires spacing greater than or equal to  $1.2\text{ }\mu\text{m}$  beyond the EOL, within  $0.6\text{ }\mu\text{m}$  to either side of the EOL. [Figure 1-24](#) on page 89 includes examples of legal spacing for, and violations of, this rule.



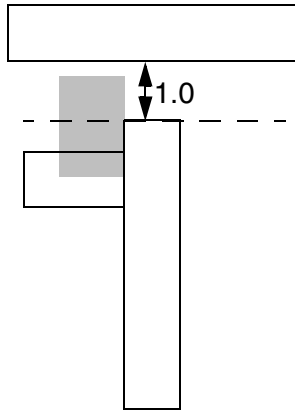
Figure 1-24



## 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:

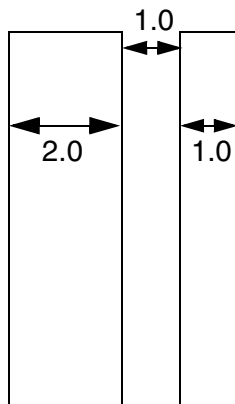
```
SPACING 1.0 ;                #minimum spacing is 1.0 μm
SPACING 1.2 ENDOFLINE 1.3 WITHIN 0.6 PARALLELEDGE 1.1 WITHIN 0.5 TWOEDGES ;
```

### Example 1-19 Same Net Spacing Rule

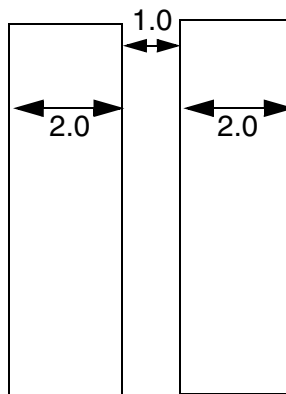
If you include the following routing layer rules in your LEF file, same-net power or ground nets can use 1.0  $\mu\text{m}$  spacing, even if they are 2  $\mu\text{m}$  to 5  $\mu\text{m}$  wide, as shown in [Figure 1-25](#) on page 91:

```
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 1-25**



a) Okay if both wires are the same net and are either a power or ground net.



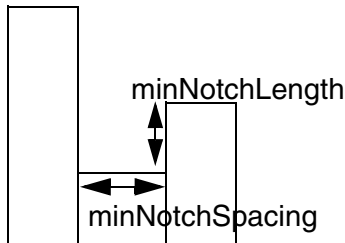
b) Okay if both wires are the same net and are either a power or ground net.

### Example 1-20 Notch Length Spacing Rule

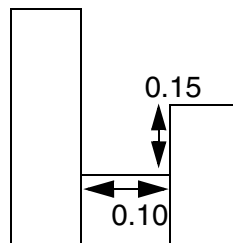
The figure below illustrates the following routing layer rules:

```
SPACING 0.10 ;  
SPACING 0.12 NOTCHLENGTH 0.15 ;
```

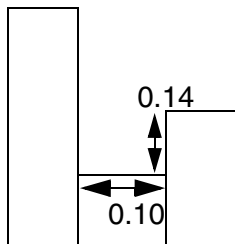
**Figure 1-26 Notch Length Rule Definitions**



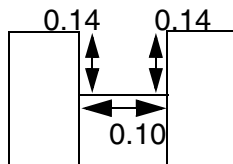
a) Illustration of notch spacing rule.



b) Okay; notchLength is not < 0.15.



c) Violation



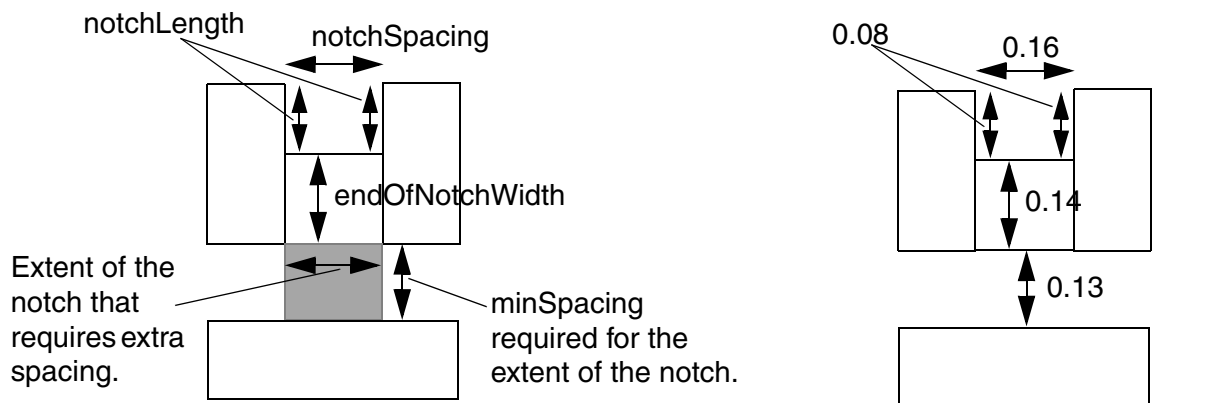
d) Violation

### 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 1-27](#) on page 93 for different layout examples for these rules.

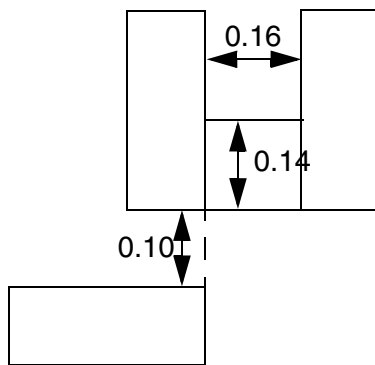
```
SPACING 0.10 ;           #default spacing  
SPACING 0.14 ENDOFNOTCHWIDTH 0.15 NOTCHSPACING 0.16 NOTCHLENGTH 0.08 ;
```

**Figure 1-27 End Of Notch Width Rule Definitions**

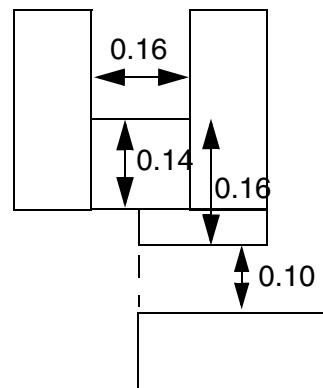


a) ENDOFNOTCHWIDTH rule definitions. The gray box shows where extra space is required.

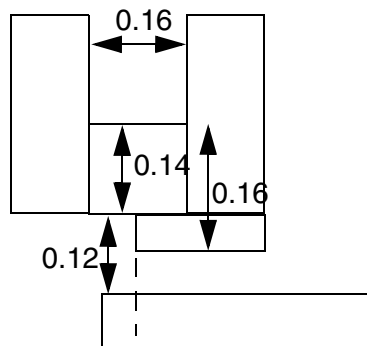
b) Violation; at least 0.14  $\mu\text{m}$  spacing required.



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 95.

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 1-28](#) on page 96, 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

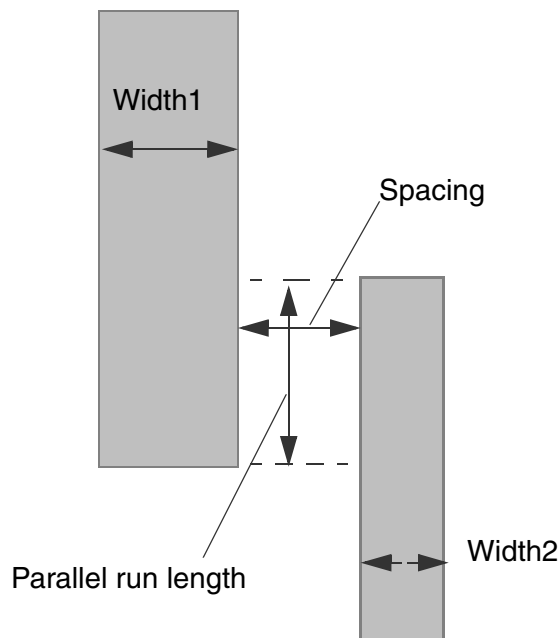
## LEF/DEF 5.8 Language Reference

### 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 1-28**



These rules translate into the following SPACINGTABLE PARALLEL RUNLENGTH statement:

```
LAYER metall
...
SPACINGTABLE
  PARALLEL RUNLENGTH 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 PARALLEL RUNLENGTH 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

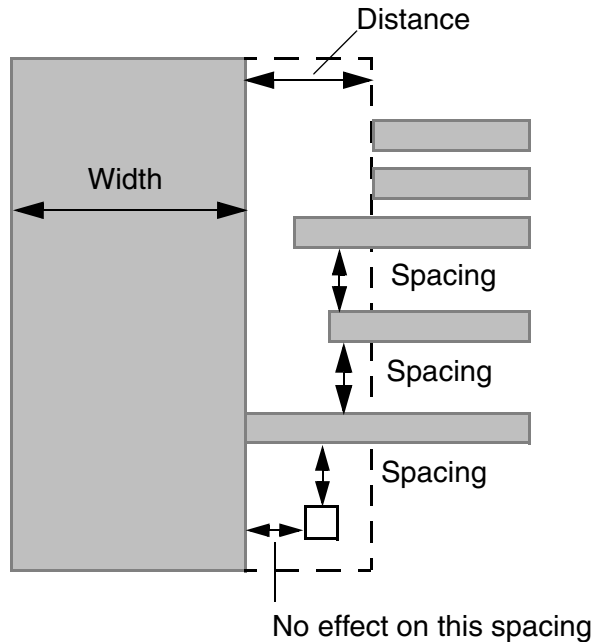
Processes often require a second spacing table to enforce the wide wire spacing rules between nearby perpendicular wires, even if the wires are narrow. [Figure 1-29](#) on page 98 illustrates this situation. Use the following SPACINGTABLE INFLUENCE syntax to describe this table:

```
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 1-29**



The wide wire rules often match the larger width and spacing values in the `SPACINGTABLE PARALLELRUNLENGTH` 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 $\mu\text{m}$ spacing
Either width>0.25 $\mu\text{m}$ and parallel length>0.0 $\mu\text{m}$	0.20 $\mu\text{m}$ spacing
Both width>0.25 $\mu\text{m}$ and parallel length>0.0 $\mu\text{m}$	0.25 $\mu\text{m}$ spacing
Either width>1.50 $\mu\text{m}$ and parallel length>1.50 $\mu\text{m}$	0.50 $\mu\text{m}$ spacing
Both width>1.50 $\mu\text{m}$ and parallel length>1.50 $\mu\text{m}$	0.60 $\mu\text{m}$ spacing
Either width>3.00 $\mu\text{m}$ and parallel length>3.00 $\mu\text{m}$	1.00 $\mu\text{m}$ spacing
Both width>3.00 $\mu\text{m}$ and parallel length>3.00 $\mu\text{m}$	1.20 $\mu\text{m}$ 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.

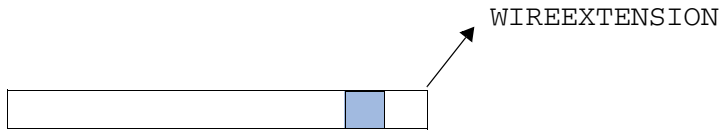
*Default:* Wires are extended half of the routing width

*Type:* Float

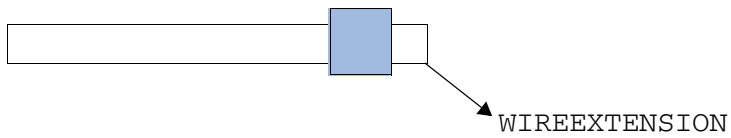
**Note:** The WIREEXTENSION statement only extends wires and not vias. For 65nm and below, WIREEXTENSION is no longer recommended because it may generate some advance rule violations if wires and vias have different widths.

The following figure shows WIREEXTENSION with same and different wire and via widths:

**Figure 1-30 Illustration of WIREEXTENSION**



a) Wire and via with same width



b) Wire and via with different widths

## 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};" ;
```

Where:

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.
-------------	---

## 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_SPANLENGHTHABLE  
    "SPANLENGHTHABLE {spanLength} ... [WRONGDIRECTION]  
        [ORTHOGONAL length] [EXCEPTOTHERSPAN otherSpanlength]  
    ; " ;
```

Where:

ORTHOGONAL <i>length</i>	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
--------------------------	--

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

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`SPANLENGHTHABLE {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 `SPANLENGHTHABLE` spacing tables can be defined - one with `WRONGDIRECTION` and the other without.

*Type:* Float, specified in microns

`WRONGDIRECTION`

Specifies all of the allowable legal span lengths in the direction parallel to the specified direction in `DIRECTION` on a Manhattan routing layer.

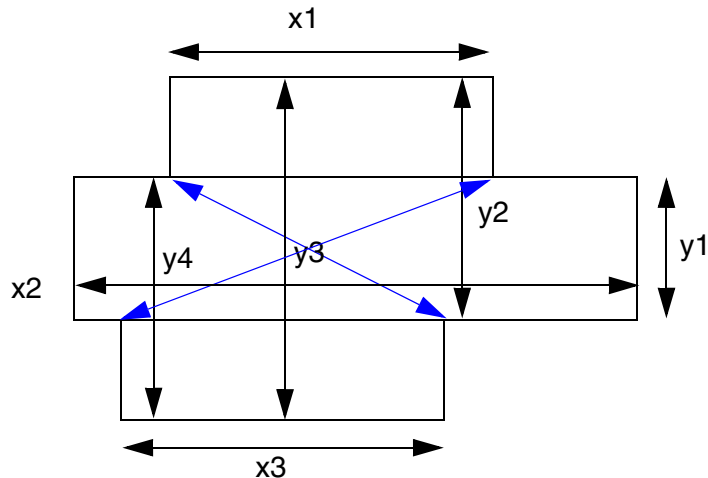
Note that using `WRONGDIRECTION` changes the interpretation of any wrong-way routing widths in the `DEF NETS` section. If `WRONGDIRECTION` is specified, then any wrong-way routing in the `DEF NETS` section will use the `WRONGDIRECTION` width for that layer unless the net or route has a `NONDEFAULTRULE` 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 `SPANLENGHTHABLE {spanLength}... ORTHOGONAL length` and `SPANLENGHTHABLE {spanLength}... WRONGDIRECTION` with preferred direction being horizontal:

**Figure 1-31 Illustration of SPANLENGHTHABLE**



y1, y2, y3 & y4 should be checked against the given span lengths in SPANLENGHTHABLE without WRONGDIRECTION while x1, x2 & 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.

## 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_WIDTHTABLE
    "WIDTHTABLE {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 WIDTHTABLE in the corresponding direction, if WIDTHTABLE WRONGDIRECTION is specified. Otherwise, the width between the corners must be greater than or equal to the first width value in the WIDTHTABLE (for both directions) either vertically or horizontally.

*Type:* Float, specified in microns



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`WIDTHTABLE {width}...`

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  $\mu\text{m}$ .
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

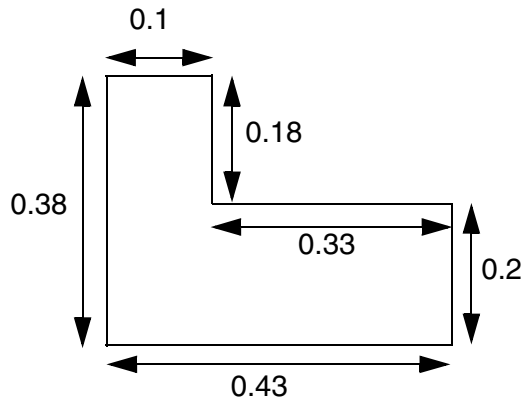
---

```
PROPERTY LEF58_WIDHTHABLE
```

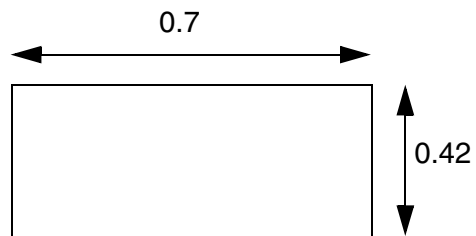
```
"WIDHTHABLE 0.10 0.15 0.20 0.25 0.30 0.40 ;" ;
```

**Figure 1-32 Illustration of WIDHTHABLE Rule**

```
PROPERTY LEF58_WIDHTHABLE "WIDHTHABLE 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.

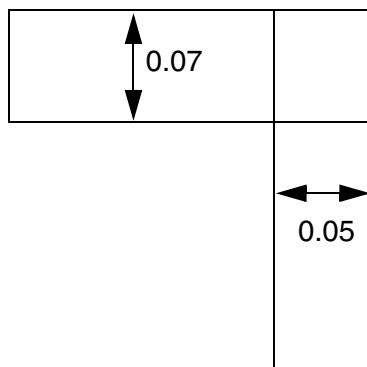
```
DIRECTION VERTICAL ;
```

```
PROPERTY LEF58_WIDHTHABLE
```

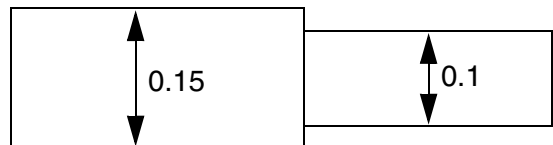
```
"WIDHTHABLE 0.05 0.1 0.15 WRONGDIRECTION ; " ;
```

**Figure 1-33 Illustration of Width Table Rule with ORTHOGONAL and WRONGDIRECTION**

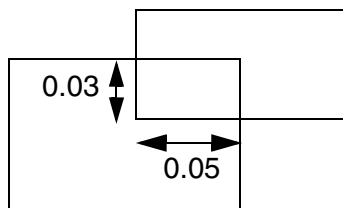
```
DIRECTION VERTICAL ;  
PROPERTY LEF58_WIDHTHABLE  
"WIDHTHABLE 0.05 0.07 0.1 0.12 0.15 ORTHOGONAL ;  
WIDHTHABLE 0.1 0.15 WRONGDIRECTION ; " ;
```



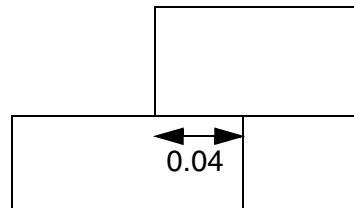
a) Violation, 0.07 is not a legal width of a horizontal wire



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 ;" ;
```

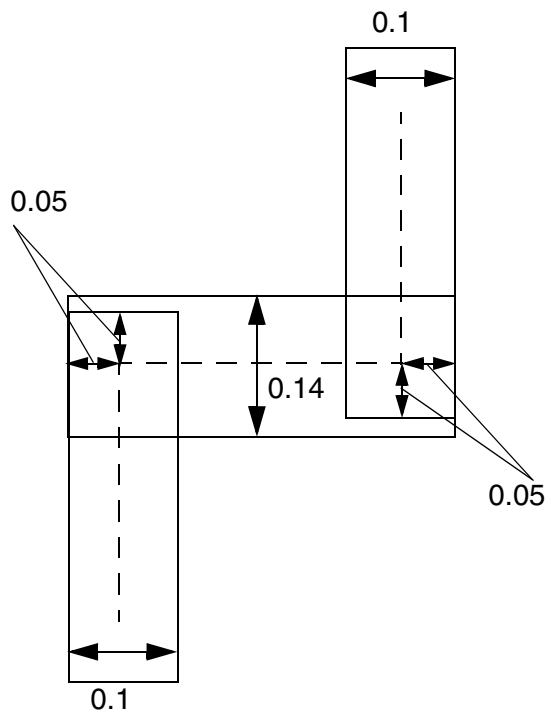
In the DEF, a vertical default route in the NETS section will have a width of 0.10  $\mu\text{m}$  with extension of 0.05  $\mu\text{m}$ , while a horizontal route will have a width of 0.14  $\mu\text{m}$  with extension of 0.05  $\mu\text{m}$ .

## LEF/DEF 5.8 Language Reference

### LEF Syntax

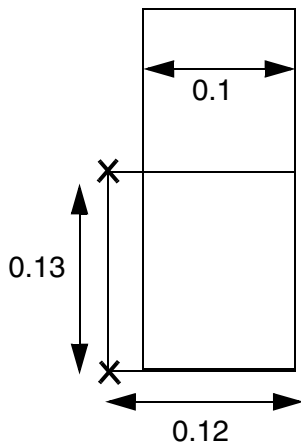
---

In more advanced nodes, wider widths are required for non-preferred direction. The following example shows route with wrong-way segment:

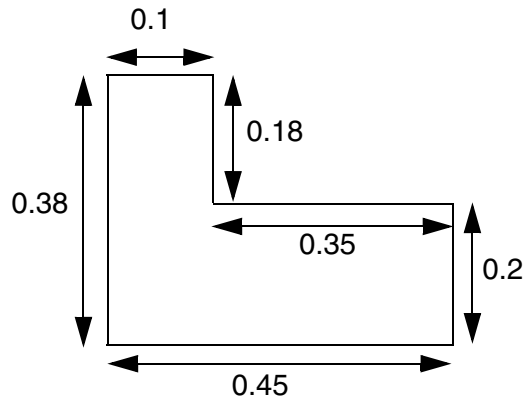


**Figure 1-34 Illustration of WIDTH Rule with WRONGDIRECTION**

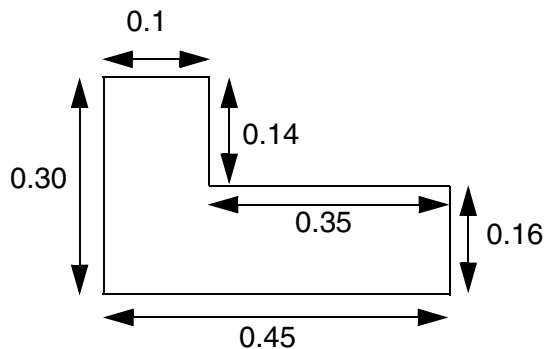
```
DIRECTION VERTICAL;  
WIDTH 0.1;  
PROPERTY LEF58_WIDTH "WIDTH 0.2 WRONGDIRECTION ;" ;
```



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.

#### ***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_OALAYERMAP STRING ;
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`:

```
PROPERTYDEFINITIONS
LIBRARY LEF58_OALAYERMAP STRING "
    OALAYERMAP Metal1A LAYER M1 MASK 1 ;
    OALAYERMAP Metal1B LAYER M1 MASK 2 ; " ;
END PROPERTYDEFINITIONS
```

## 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 ;] ...
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.

### 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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

RING	<p>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 <code>CLASS RING</code> 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 <code>RING</code> macro.</p>
BLOCK	<p>Predefined macro used in hierarchical design.</p> <p>A block macro can have one of the following sub-classes:</p> <p><b>BLACKBOX</b>—A block that sometimes only contains a <code>SIZE</code> 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 <code>LEF MACRO</code> pin names.</p> <p><b>SOFT</b>—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 <code>BLACKBOX</code> has no sub-block implementation available.</p>
PAD	<p>I/O pad. A pad can be one of the following types: <code>INPUT</code>, <code>OUTPUT</code>, <code>INOUT</code>, <code>POWER</code>, or <code>SPACER</code>, for I/O rows; <code>INPUT</code>, <code>OUTPUT</code>, <code>INOUT</code>, or <code>POWER</code>, for I/O corner pads; <code>AREAIO</code> 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 <code>CLASS COVER BUMP</code> macro for a connection to the IC package).</p> <p>For an example of a macro pad cell, see <a href="#">Example 1-22</a> on page 116.</p>

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

CORE	<p>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.</p> <p>A core macro can be one of the following types:</p> <p>FEEDTHRU—Used for connecting to another cell.</p> <p>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.</p> <p>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'.</p> <p>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.</p> <p>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.</p>
ENDCAP	<p>A macro placed at the ends of core rows (to connect with power wiring).</p> <p>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:</p> <p>PRE—A left-end macro</p> <p>POST—A right-end macro</p> <p>TOPLEFT—A top left I/O corner cell</p> <p>TOPRIGHT—A top right I/O corner cell</p> <p>BOTTOMLEFT —A bottom left I/O corner cell</p> <p>BOTTOMRIGHT—A bottom right I/O corner cell</p> <p>The ENDCAP sub-class is required. The PRE and POST are CORE area cells, whereas the other four are PAD CLASS.</p>

### Example 1-22 Macro Pad Cell

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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
    END

# 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
```

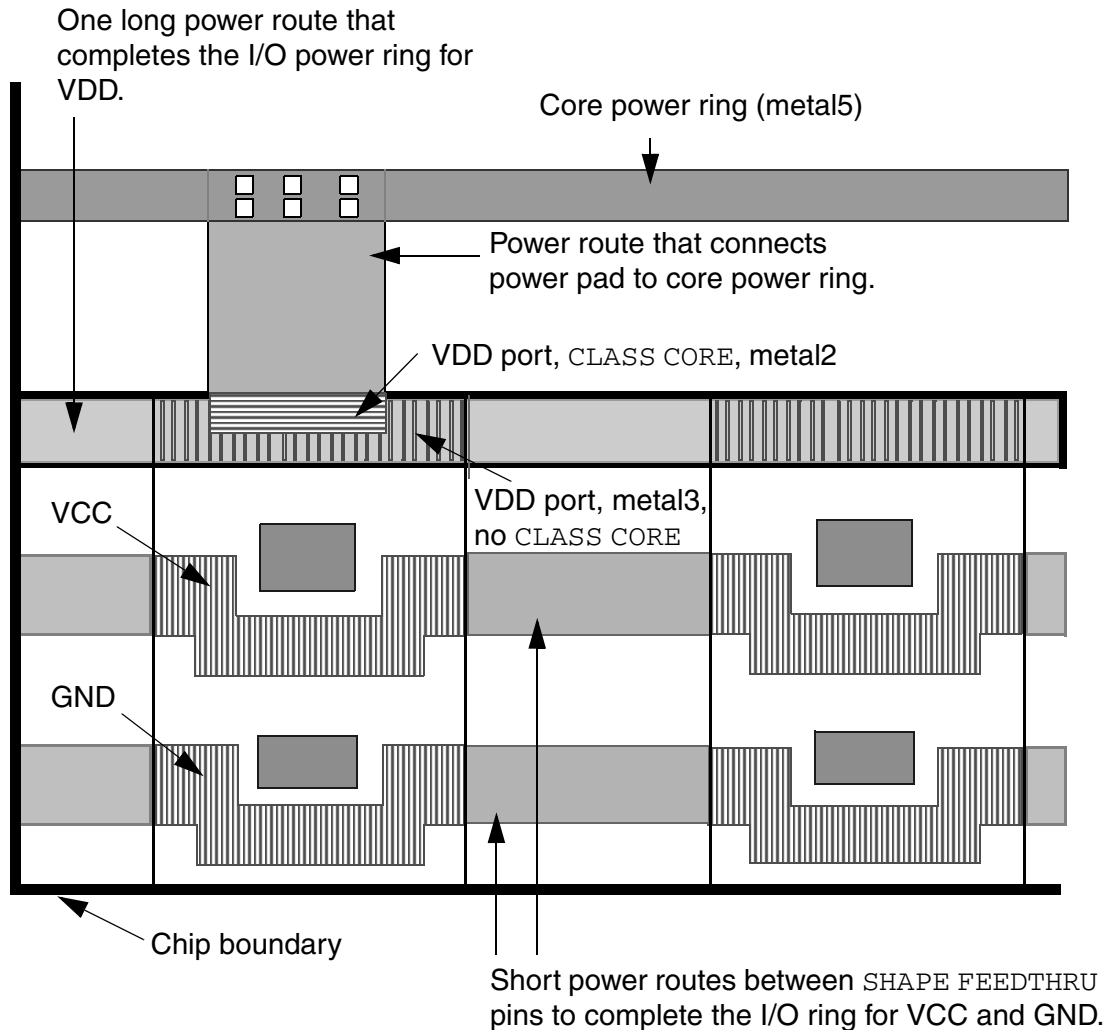
## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
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
    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 1-35 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 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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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 136 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 139 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 156:

```
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

---

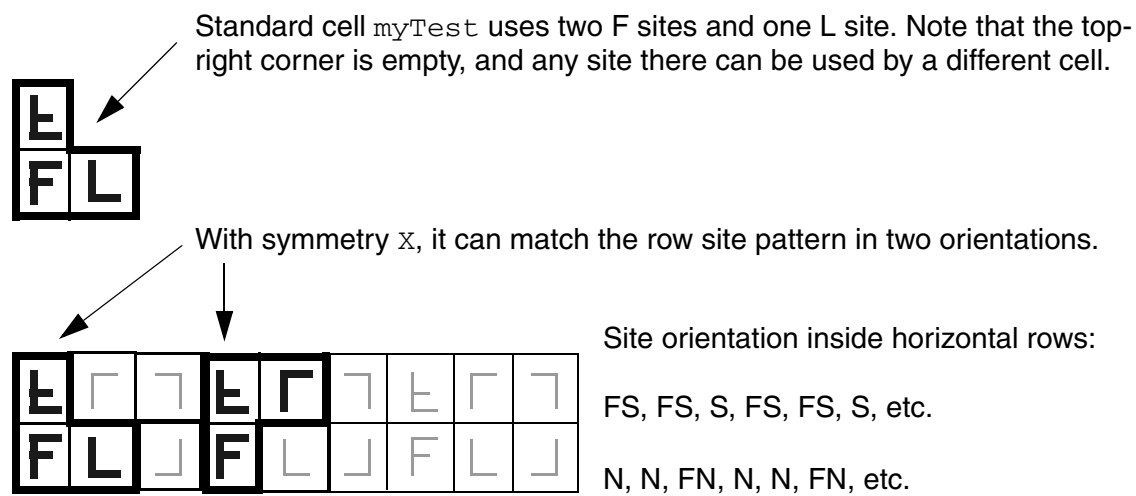
```

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 1-36 on page 124 illustrates the placement results of this definition.

**Figure 1-36**



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 1-37 on page 124.

**Figure 1-37**



## 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 126.

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.

**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 Cover Macros**

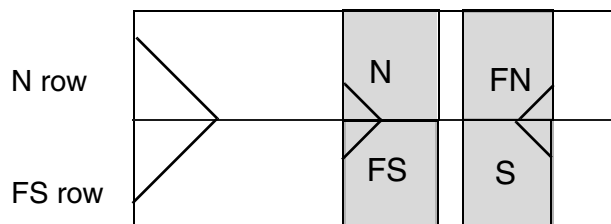
If you define a cover macro with its actual size, some place-and-route tools cannot place the rest of the cells in your design because it uses the cell boundary to check for overlaps. You can resolve this in two ways:

- 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 1-38](#) on page 126 illustrates the normal orientations for single-height, flipped and abutted rows with standard cells and sites.

**Figure 1-38 Normal Orientations for Single-Height Rows**

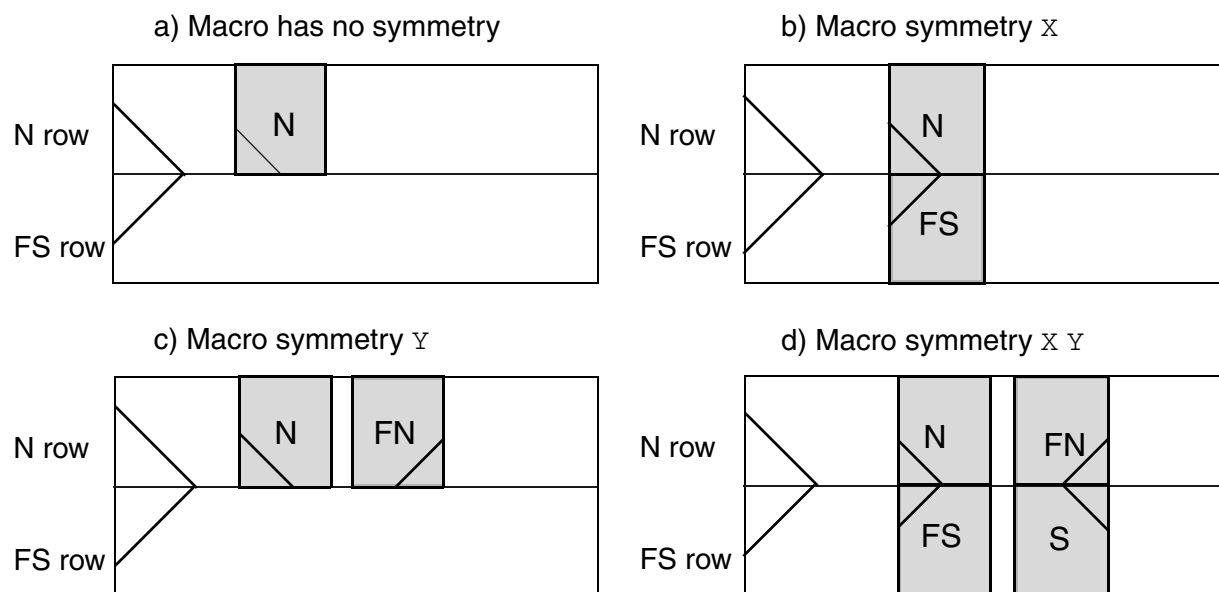


The following examples describe typical combinations of orientations for standard cells. Applications typically create only N (or FS for flipped) row orientations for horizontal standard cell rows; therefore, the examples describe these two rows.

### **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 1-39](#) on page 127. 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.

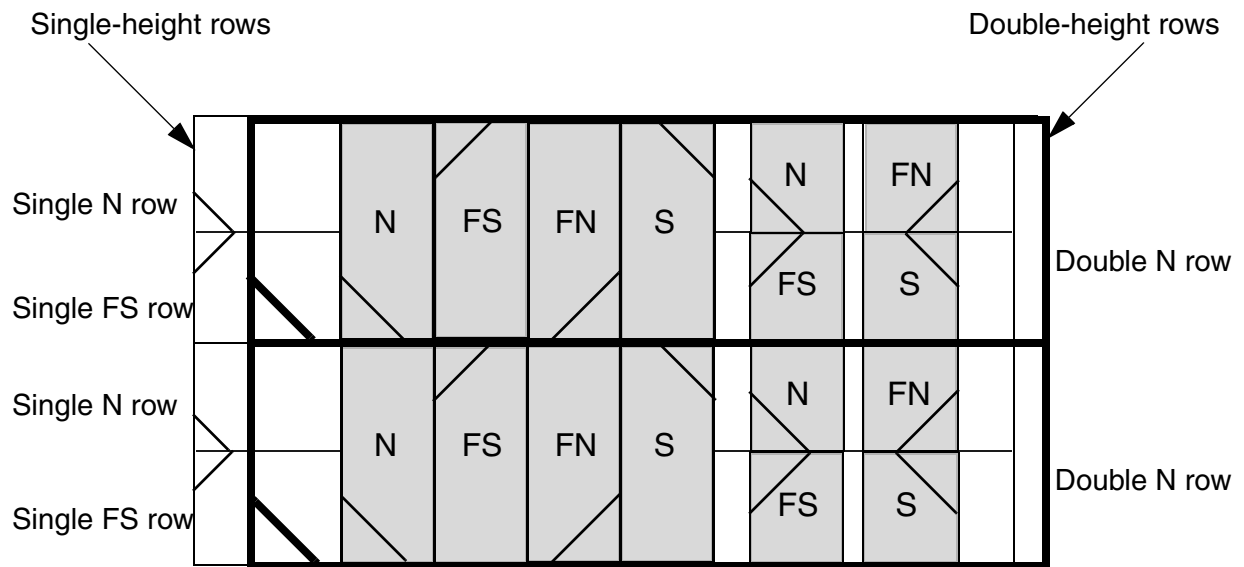
**Figure 1-39 Legal Placements for Row Sites with Symmetry Y**



### 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 1-40](#) on page 128). 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 1-40 Legal Placements for Single-Height Row Sites with Symmetry Y and Double-Height Row Sites with Symmetry X Y**



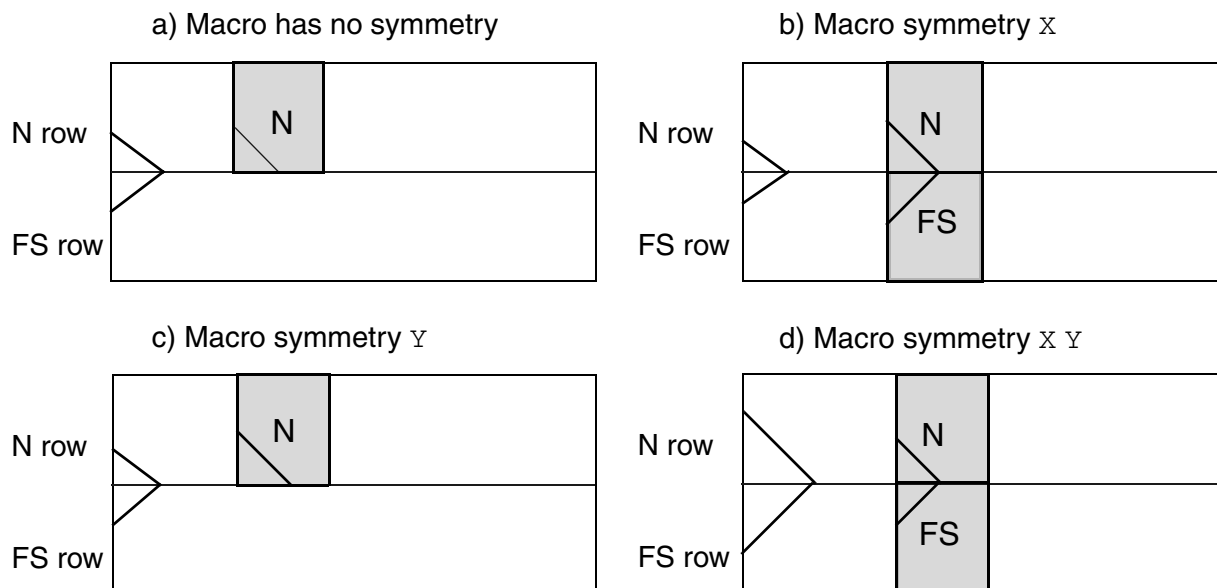
**Note:** The single-height rows are shifted slightly to the left of the double-height rows in the above figure for illustration purposes. In a real design, they should be aligned.

### 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 1-41](#) on page 129 for the different combinations when the `SITE` has no symmetry.



**Figure 1-41 Legal Placements for Row Sites with No Symmetry**

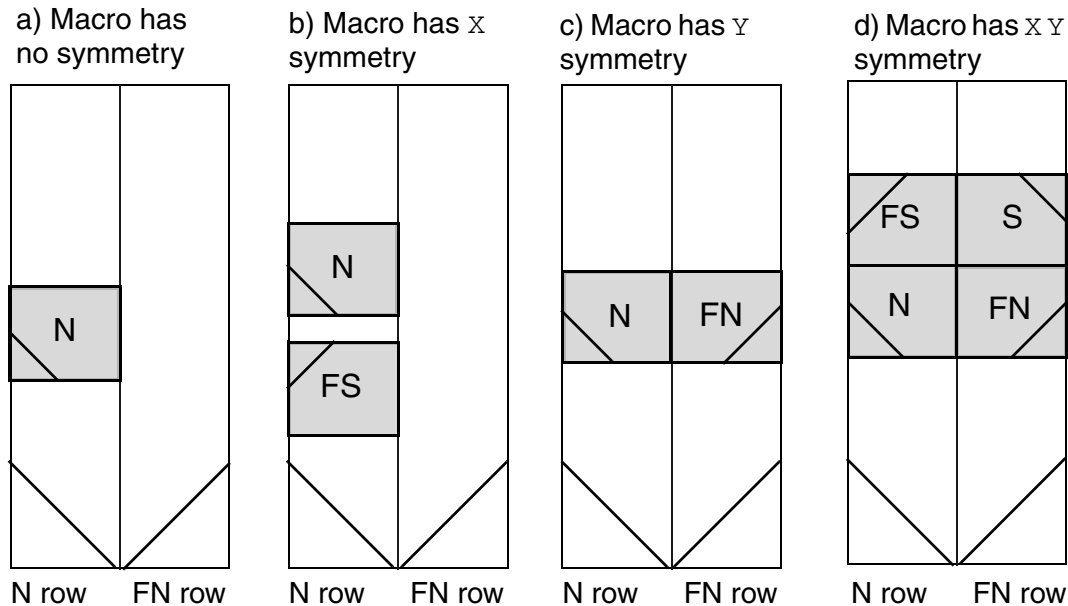


### Example 1-29 Vertical Rows

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 1-42](#) on page 130 shows the legal placement for a site with symmetry  $\times$ , and the typical standard cell `MACRO` symmetry  $\times \gamma$ .

**Figure 1-42 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 135 .

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.

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 1-43](#) on page 136.

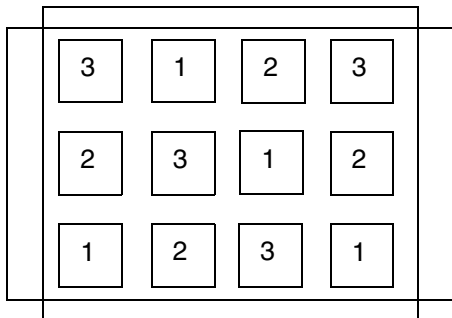
```
LAYER M1 ;
    RECT MASK 2 10 10 11 11 ;
LAYER M2 ;
```

## LEF/DEF 5.8 Language Reference

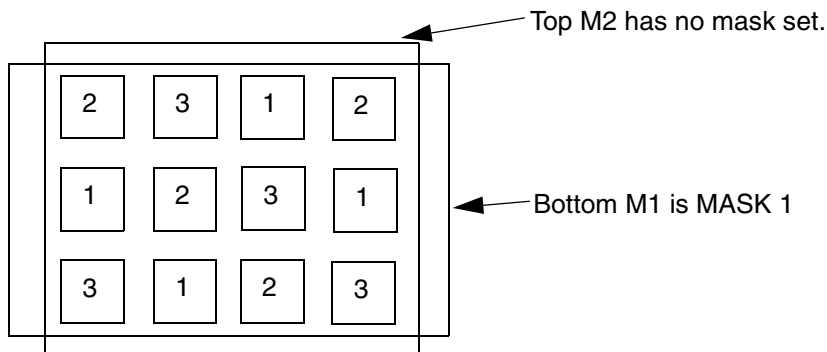
### LEF Syntax

```
RECT 10 10 11 11 ;
VIA 5 5 VIA1_1 ;
VIA MASK 031 15 15 VIA1_2 ;
```

**Figure 1-43 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 pt ... ;
      | PATH ITERATE pt ... stepPattern ;
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
| RECT pt pt ;  
| RECT ITERATE pt pt stepPattern ;  
| POLYGON pt pt pt pt ... ;  
| POLYGON ITERATE pt pt pt pt ... stepPattern ;  
} ...  
| VIA pt viaName ;  
| VIA ITERATE pt viaName stepPattern ;  
} ...
```

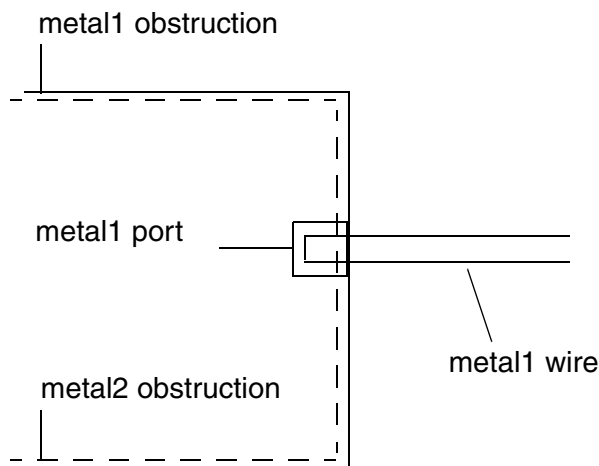
END]

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 130 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 1-44](#) on page 137, 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.

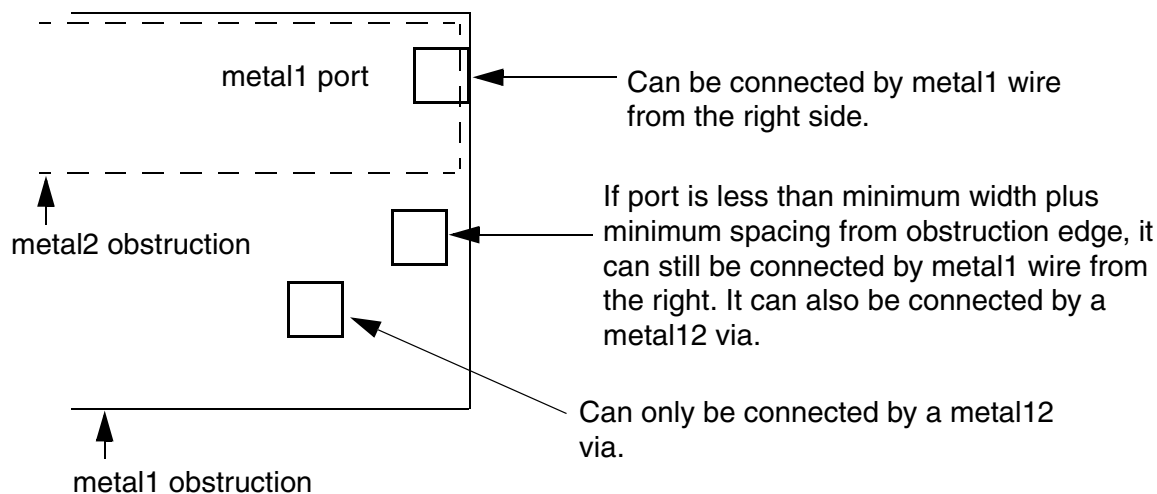
**Figure 1-44**



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 1-45](#) on page 138).

**Figure 1-45**



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.

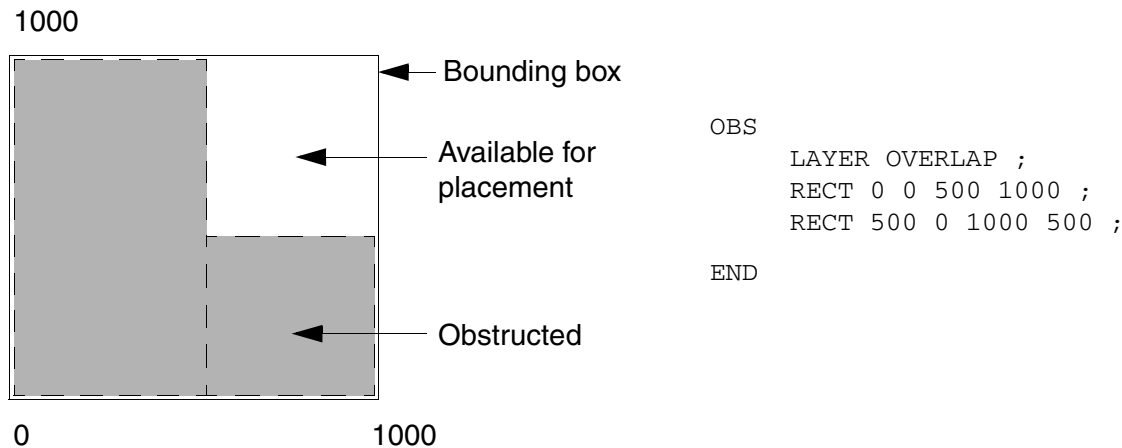
### Rectilinear Blocks

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 1-46](#) on page 139.

**Figure 1-46**



### 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
    [CLASS {NONE | CORE | BUMP} ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
{layerGeometries} ...
END} ...
[PROPERTY propName propVal ;] ...
[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.”](#)

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.”](#)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

**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

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
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
                                #practice
ANTENNAGATEAREA 1.0 ;          #OXIDE1 gate area
ANTENNAMAXAREACAR 50.0 LAYER m1 ; #metall1 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.

**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.”](#)

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`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 144.

**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.

**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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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
...
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
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
...
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).

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} ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
{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 130 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.

### 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:

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### VIAINPINONLY

Specifies that vias must be dropped inside the original pin shapes to connect to the pin, and planar connection to the pin is not allowed. In some advanced nodes, the pin shapes can be extended for metal alignment purpose. However, via insertion is not allowed in that extended portion.

#### SHAPE

Specifies a pin with special connection requirements because of its shape.

*Value:* Specify one of the following:

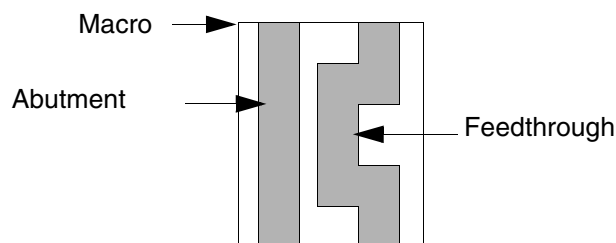
ABUTMENT	Pin that goes straight through cells with a regular shape and connects to pins on adjoining cells without routing.
RING	Pin on a large block that forms a ring around the block to allow connection to any point on the ring. Cover macro special pins also typically have shape RING.
FEEDTHRU	Pin with an irregular shape with a jog or neck within the cell.

Figure 1-47 on page 147 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 1-47**



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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 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 144.

**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:

ANALOG	Pin is used for analog connectivity.
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.
SIGNAL	Pin is used for regular net connectivity.

## 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-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 ;]
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
[SPACING minSpacing ;]
[WIREEXTENSION value ;]
END layerName} ...
[VIA viaStatement] ...
[USEVIA viaName ;] ...
[USEVIARULE viaRuleName ;] ...
[MINCUTS cutLayerName numCuts ;] ...
[PROPERTY propName propValue ;] ...

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

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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 `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 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.

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.

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 162. 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 101.

### Example 1-35 Nondefault Rule Statement

Assume two default via rules were defined:

```
VIARULE via12rule GENERATE DEFAULT
    LAYER metal1 ;
    ...
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:

```
NONDEFAULTRULE wide1_5x
    LAYER metal1
        WIDTH 1.5 ; #metal1 has a 1.5 um width
    END metal1
    LAYER metal2
        WIDTH 1.5 ;
    END metal2
    LAYER metal3
```



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
        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 metal1
        WIDTH 3.0 ; #metal1 has 3.0 um width
    END metal1
    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 metal1
        WIDTH 1.5 ;    #metal1 has 1.5 um width
        SPACING 3.0 ;  #extra spacing of 3.0 um
    END metal1
    LAYER metal2
        WIDTH 1.5
        SPACING 3.0
    END metal2
    LAYER metal3
        WIDTH 1.5
        SPACING 3.0
    END metal3
```

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

```
#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

### *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

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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
```

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*

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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.

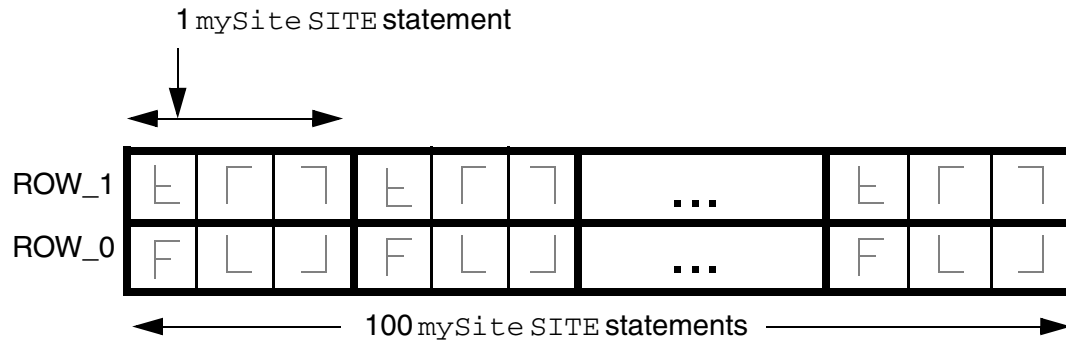
#### Example 1-37 Site Row Pattern Statement

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
```

[Figure 1-48](#) on page 157 illustrates some DEF rows made up of mySite sites.

**Figure 1-48**



```
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 126.

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.
- 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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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 PICOFARADS 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 159. 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 PICOFARADS *convertFactor*

Interprets one LEF capacitance unit as 1 picofarad.

CURRENT MILLIAMPS *convertFactor*

Interprets one LEF current unit as 1 milliamp.

DATABASE MICRONS *LEFconvertFactor*

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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

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

### 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

## LEF/DEF 5.8 Language Reference

### LEF Syntax

LEFconvertFactor	LEF	Database Units
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
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.



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## 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). <i>Default:</i> 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.

## 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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## Via

```
VIA viaName [DEFAULT]
    { VIARULE viaRuleName ;
      CUTSIZE xSize ySize ;
      LAYERS botMetallLayer cutLayer topMetallLayer ;
      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 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

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

#### 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*.

#### 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 1-54](#) on page 173 ). So a writer may write out `MASK 1` for the metal shapes even if they are read in with no `MASK` value.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`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:

*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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

*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 1-52](#) on page 170. The other two patterns supported are alternating rows, and alternating columns, see [Figure 1-53](#) on page 171.

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.

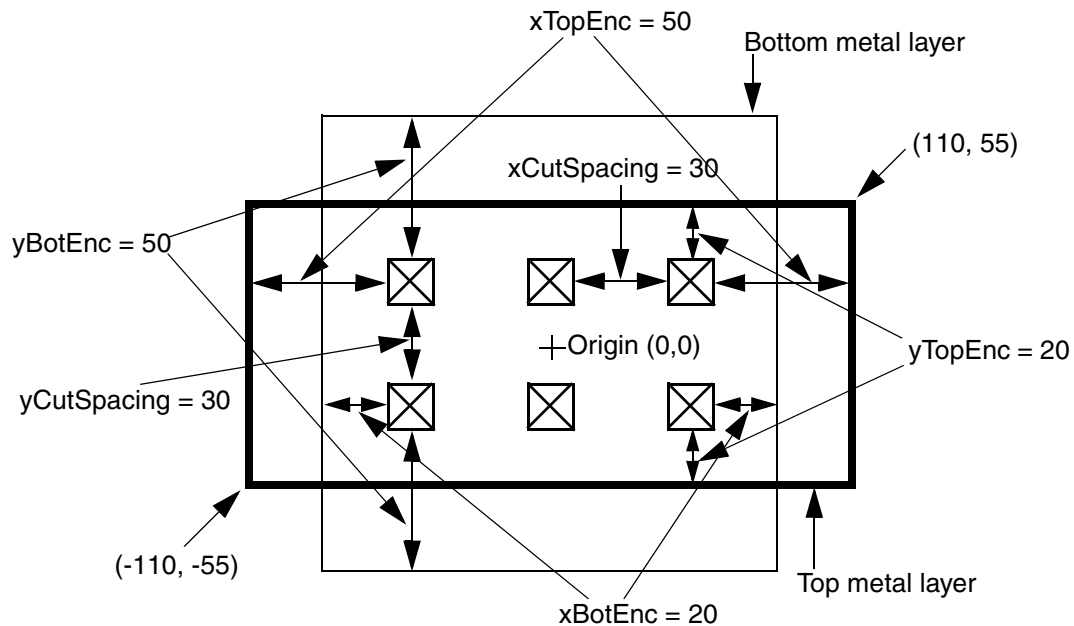
Type: Float, specified in microns

### Example 1-38 Via Rules

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 1-49](#) on page 166 illustrates this via rule.

```
VIA myVia
  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 ;
END myVia
```

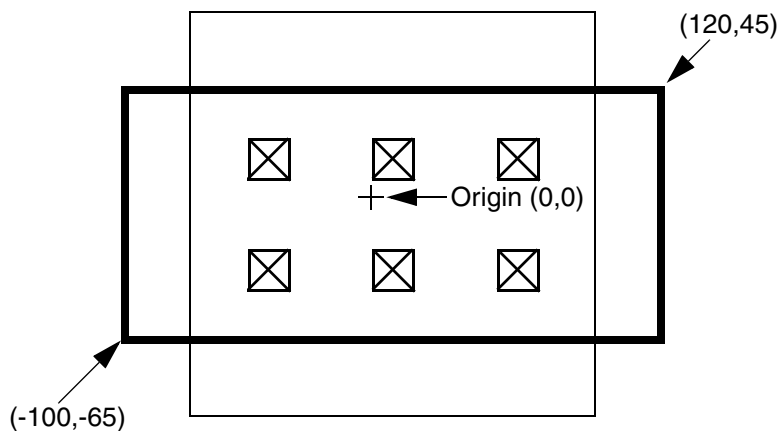
**Figure 1-49 Via Rule**



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 1-50](#) on page 166):

```
ORIGIN 10 -10 ;
```

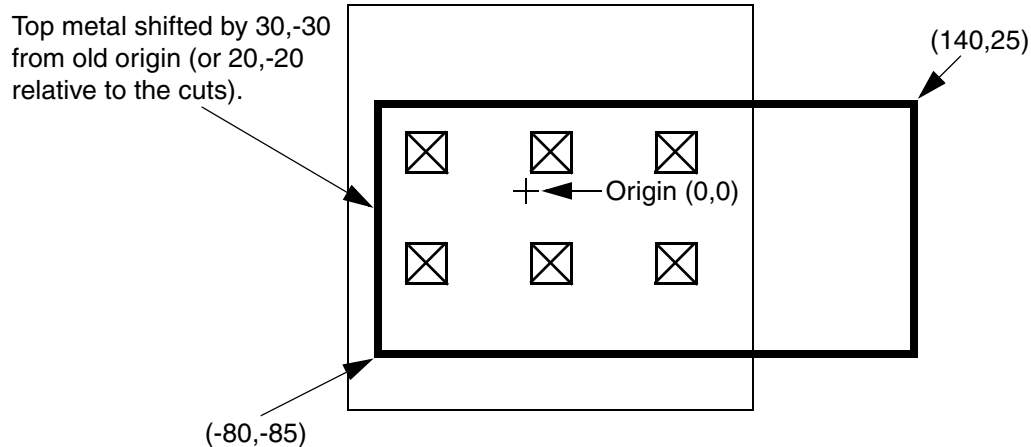
**Figure 1-50 Via Rule With Origin**



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 1-51 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 ;
  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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

`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 via12 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:

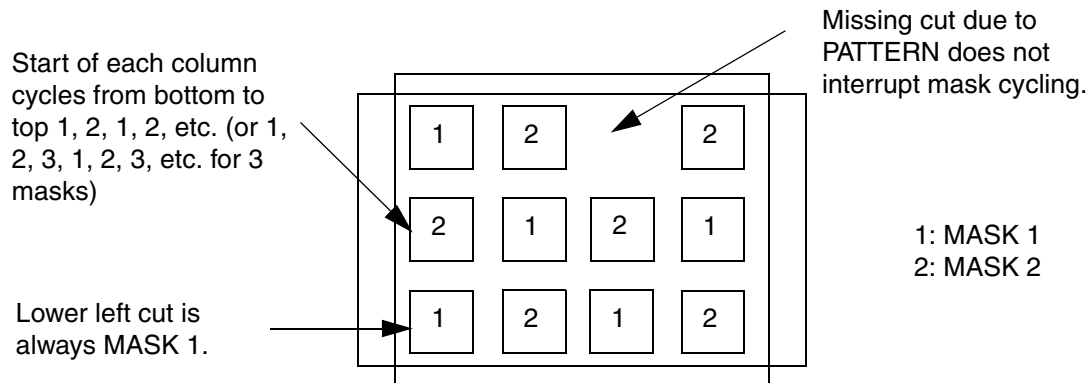
```
VIA myParamVia1
  VIARULE myGenVia1                                CUTSIZE 0.4 0.4
  LAYERS M1 VIA1 M2                                CUTSPACING 0.4 0.4
  ENCLOSURE 0.4 0 0 0.4                            ROWCOL 3 4                #3 rows, 4 columns
  PATTERN 2_F_1_D;                                  #1 cut in top row is missing
```

## LEF/DEF 5.8 Language Reference

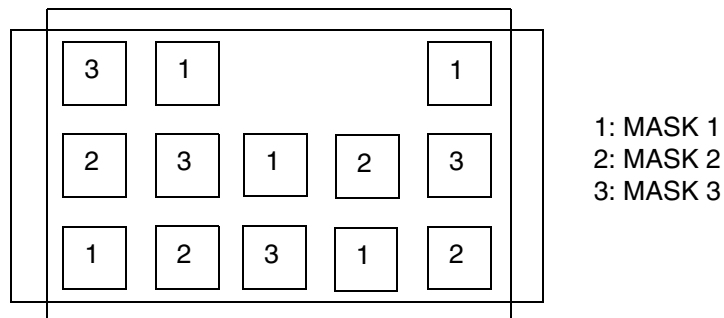
### LEF Syntax

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 1-52 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.



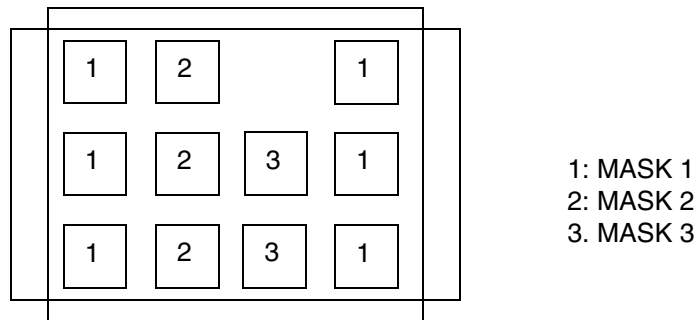
Example of a parameterized via checker-board cut-mask pattern for a 3-mask layer with 2 missing cuts. For parameterized vias (with `VIARULE . . .`), the mask of the cuts are pre-defined as an alternating pattern starting with `MASK 1` at the bottom left. The mask cycles from left-to-right and bottom-to-top 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 1-53 Parameterized via cut-mask pattern using Suffixes**

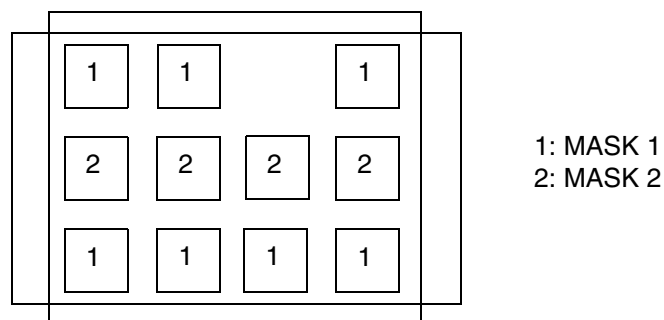
Mask color pattern for a parameterized via with 3-mask cut-layer:

```
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 via1 ;
  RECT MASK 1 -0.2 -0.2 0.2 0.2 ;    #first cut on mask 1
```

## LEF/DEF 5.8 Language Reference

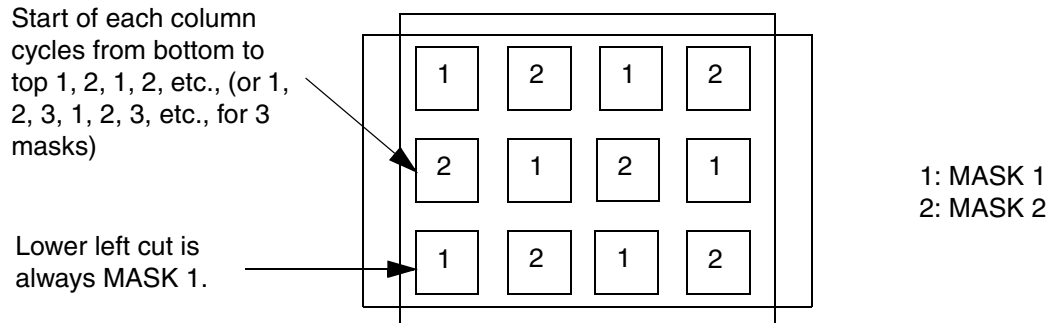
### LEF Syntax

---

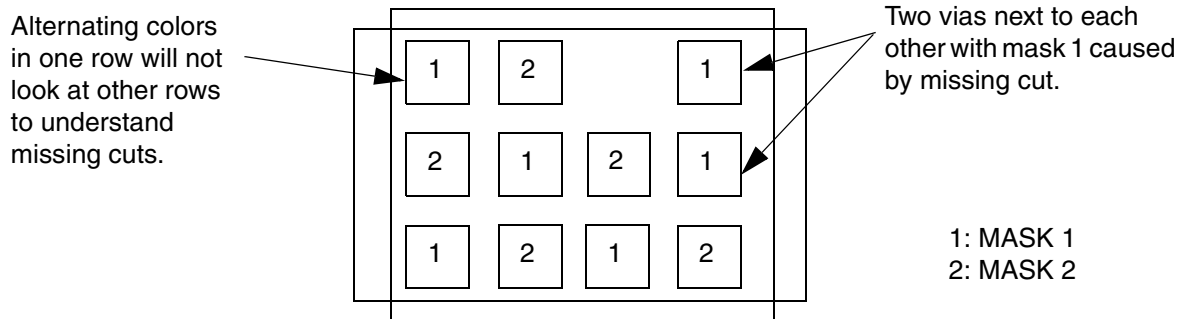
```
RECT MASK 2 0.6 -0.2 1.0 0.2 ;    #second cut on mask 2
LAYER m2 ;
RECT -0.2 -0.4 1.0 0.4 ;          #no mask, some readers will set to mask 1
END myVia1
```

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 1-54](#) on page 173). 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 1-54 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.

DEFAULT	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

---

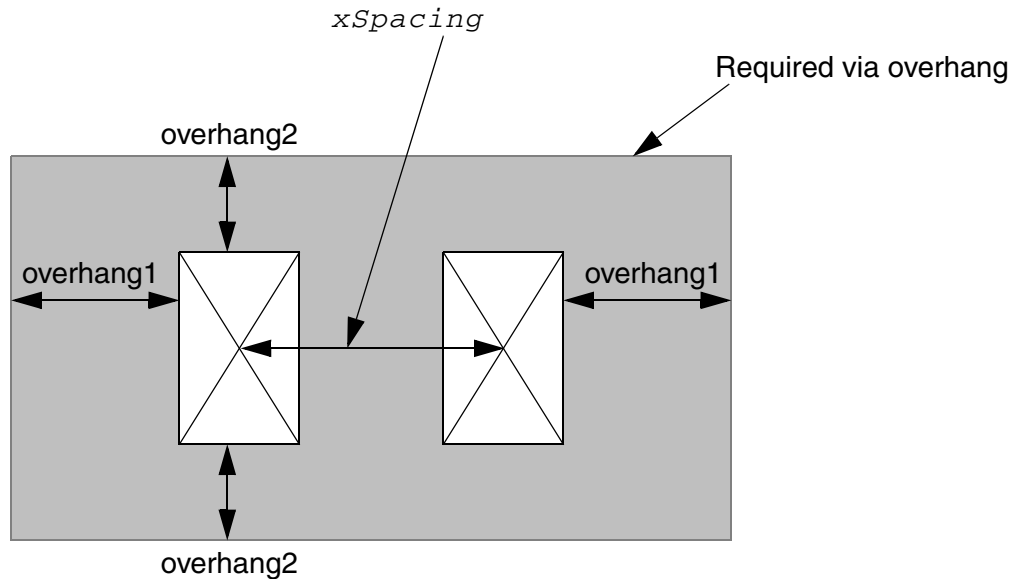
Specifies that the via must be covered by metal on two opposite sides by at least *overhang1*, and on the other two sides by at least *overhang2* (see [Figure 1-55](#) on page 178). The via generation code then chooses the direction of overhang that best maximizes the number of cuts that can fit in the 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 1-55 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.

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---



## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---





## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

## LEF/DEF 5.8 Language Reference

### LEF Syntax

---

---

# ALIAS Statements

---

This chapter contains information about the following topics.

- [ALIAS Statements](#) on page 215
  - ❑ [ALIAS Definition](#) on page 216
  - ❑ [ALIAS Examples](#) on page 216
  - ❑ [ALIAS Expansion](#) on page 217

## 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 219
- [Error Checking](#) on page 220

### 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 &msgs = "ROM size = 256" ;  
&MESSAGE &msgs = &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 226
  - ❑ General Rules on page 227
  - ❑ Character Information on page 227
    - Name Escaping Semantics for Identifiers on page 228
    - Escaping Semantics for Quoted Property Strings on page 229
  - ❑ Order of DEF Statements on page 231
- DEF Statement Definitions on page 232
  - ❑ Blockages on page 232
  - ❑ Bus Bit Characters on page 236
  - ❑ Components on page 237
  - ❑ Design on page 244
  - ❑ Die Area on page 245
  - ❑ Divider Character on page 245
  - ❑ Extensions on page 245
  - ❑ Fills on page 246
  - ❑ GCell Grid on page 250
  - ❑ Groups on page 252
  - ❑ History on page 253
  - ❑ Nets on page 253
    - Regular Wiring Statement on page 260

- ❑ [Nondefault Rules](#) on page 276
- ❑ [Pins](#) on page 279
- ❑ [Pin Properties](#) on page 296
- ❑ [Property Definitions](#) on page 296
- ❑ [Regions](#) on page 298
- ❑ [Rows](#) on page 299
- ❑ [Scan Chains](#) on page 300
- ❑ [Slots](#) on page 306
- ❑ [Special Nets](#) on page 307
  - [Special Wiring Statement](#) on page 311
- ❑ [Styles](#) on page 324
- ❑ [Technology](#) on page 336
- ❑ [Tracks](#) on page 336
- ❑ [Units](#) on page 338
- ❑ [Version](#) on page 339
- ❑ [Vias](#) on page 339

## 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 231.

### 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 <code>blockage</code> 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 <code>blockage</code> 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 240.

### 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
...
```

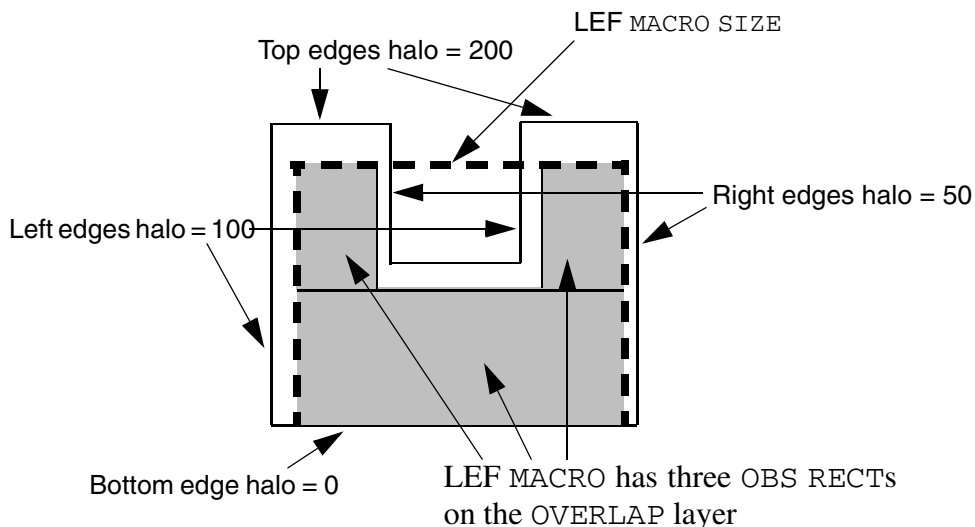
If an application shifts the layers M1, VIA1, and M2, then the above example indicates that the instance of AND2 cell shifts the M1 layer masks by 2. Since M1 is a three-mask layer, this shows that the cell-master M1 layer mask 1 shifts to 3, mask 2 shifts to 1, and mask 3 shifts to 2. The other layer 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 241:

```
- 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 243.

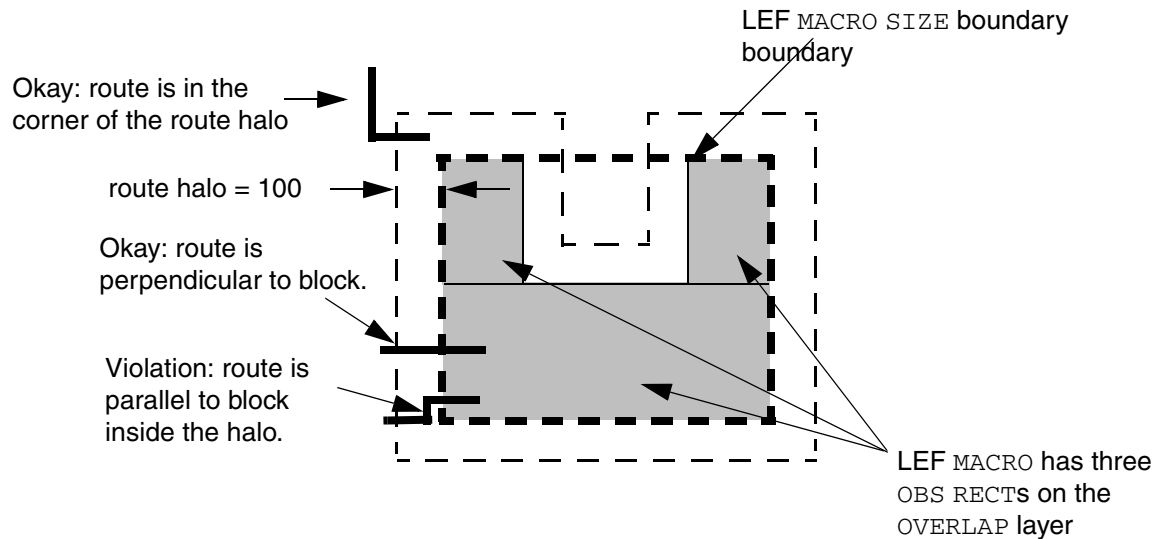
```
- il/i2
+ PLACED ( 0 0 ) N
+ ROUTEHALO 100 metal1 metal3 ;
```

## 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 249.

`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:</i> ( <i>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) ;
```



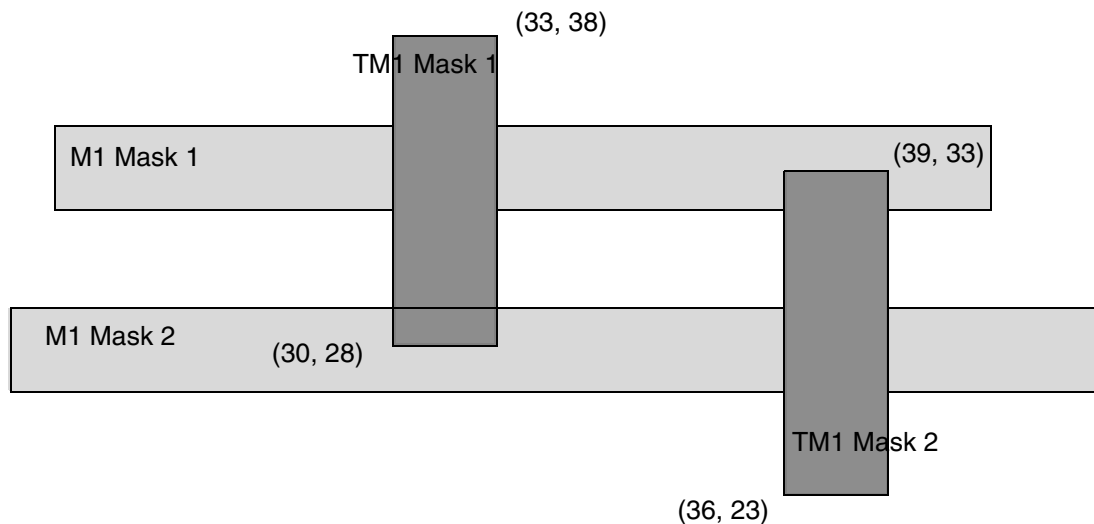
## LEF/DEF 5.8 Language Reference

### 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 318.

## 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.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

- 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 308.

`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 269](#)" 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*

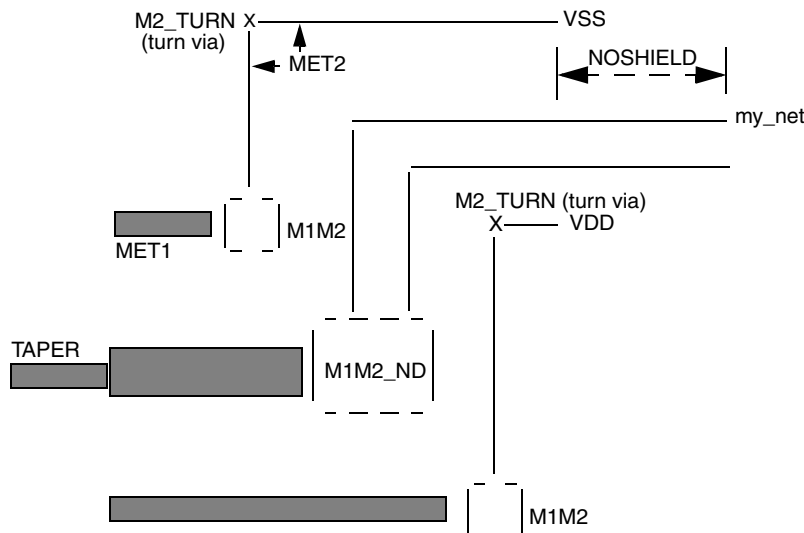
Specifies a numerical or string value for a net property defined in the `PROPERTYDEFINITIONS` statement. The *propName* you specify must match the *propName* listed in the `PROPERTYDEFINITIONS` statement.

*regularWiring*

Specifies the regular physical wiring for the net or subnet. For regular wiring syntax, see [“Regular Wiring Statement”](#) on page 260.

`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 271.

## 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 268.

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 269"](#) 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  
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 265.

## 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 244).

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 267.

*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 267.

( *x* *y* )

Specifies the route coordinates. You cannot specify a route with zero length.

For more information, see [“Specifying Coordinates”](#) on page 269.

*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

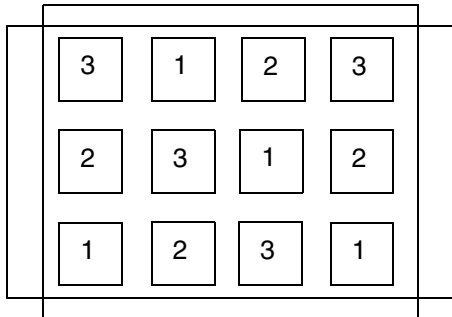
## LEF/DEF 5.8 Language Reference

### DEF Syntax

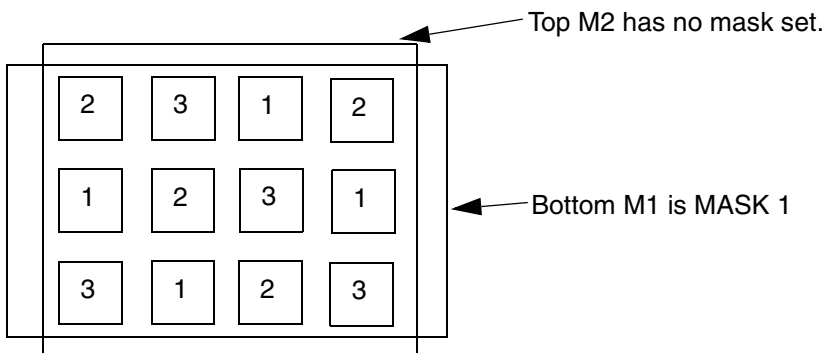
---

- 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 267.

**Figure 4-4 Via-master multi-mask patterns**



Via-master cut masks for VIA1\_2.



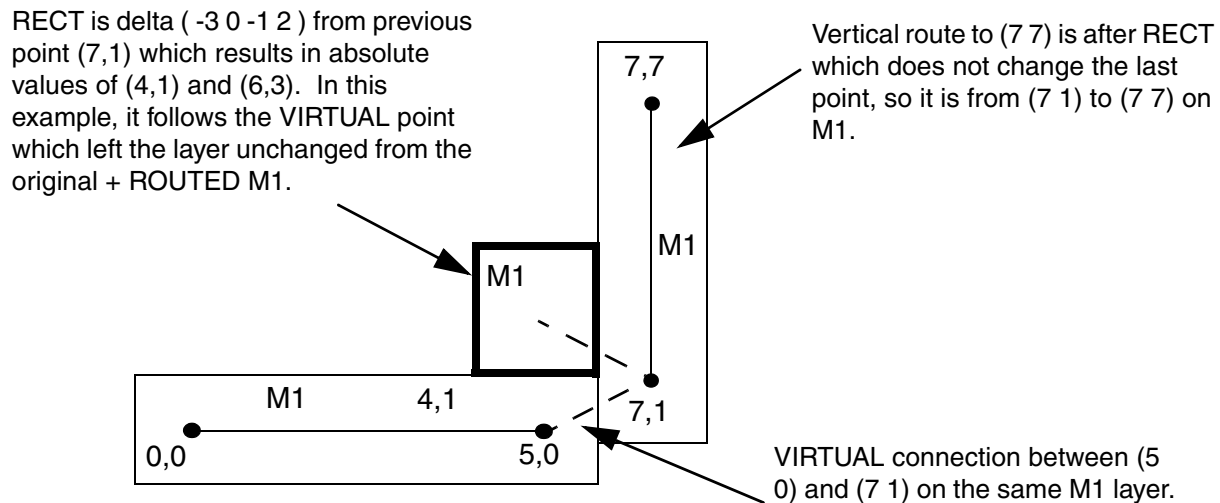
Masks for via-instance: ... ( 20 20 ) MASK 031 VIA1\_2;  
Bottom M1 is MASK 1. Top M2 has no mask set. Lower-left cut is MASK 3, and other via-instance cut shape masks are derived from via-master cut-masks as shown above by "shifting" the via-master masks to match: 1->3, 2->1, 3->2.

### **Example 4-12 Routing Points - Usage of Virtual and Rect**

Figure 4-5 on page 268 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).

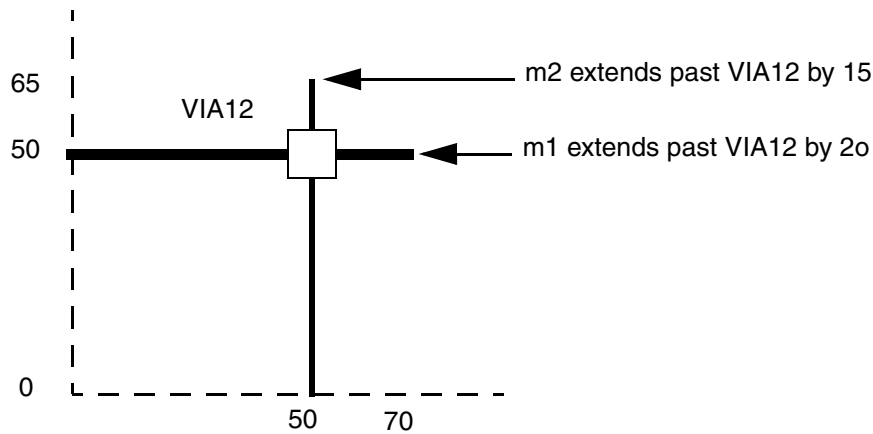
If the wire segment is a 45-degree edge, and no *STYLE* is specified, the default octagon style is used for the endpoints. The routing rule width must be an even multiple of the manufacturing grid in order to keep all of the coordinates of the resulting outer wire boundary on the manufacturing grid.

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

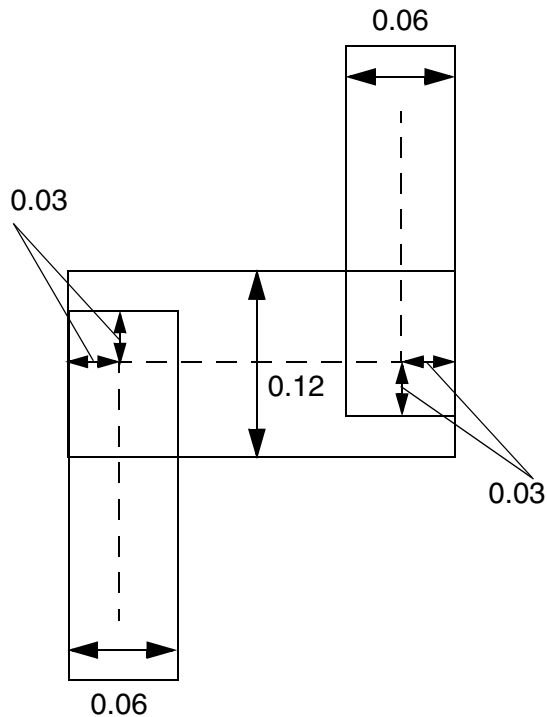
For example, the default rule in the table shows that a vertical route in the NETS section will have a width of 0.06  $\mu\text{m}$  with extension of 0.03  $\mu\text{m}$ , while a horizontal route will have



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 ) ;
```

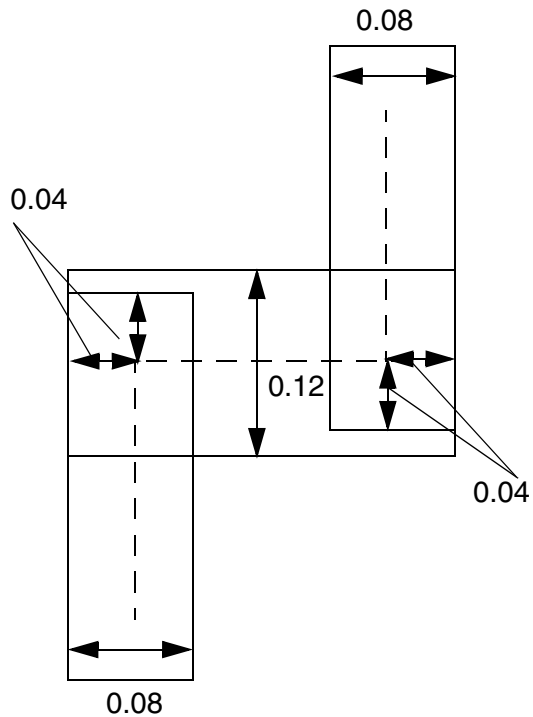
**Figure 4-6 Default rule on wrong way segment**



- 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 ) ;
```

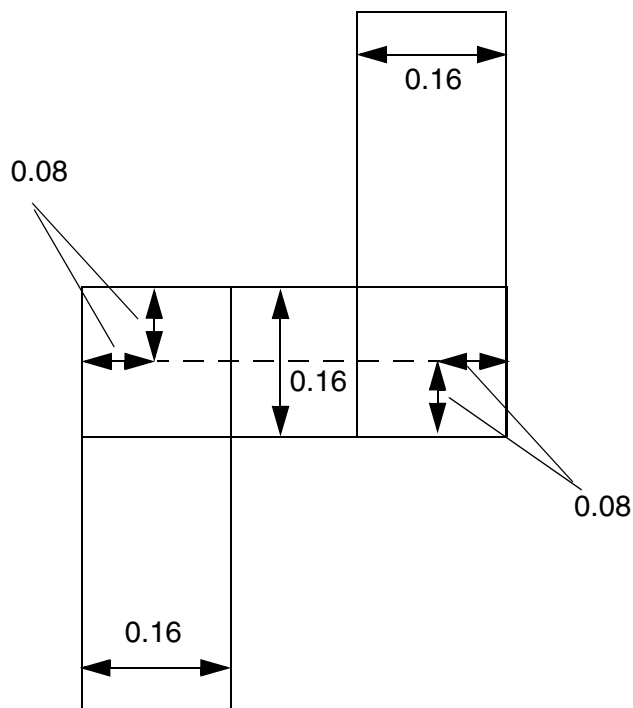
**Figure 4-7 Non-Default rule on wrong way segment**



- 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} ...] ...
  ;} ...
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.

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

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

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).

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

*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

`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
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
+ 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 ;
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]] ...
    [+ ANTENNAPINDIFFAREA value [LAYER layerName]] ...
    [+ ANTENNAMODEL {OXIDE1 | OXIDE2 | OXIDE3 | OXIDE4}] ...
    [+ ANTENNAPINGATEAREA value [LAYER layerName]] ...
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
[+ 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.

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.

#### Example 4-16 Antenna Model Statement



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

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
...
```

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,

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

or the layer above it.

*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

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

*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.”](#)

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.”](#)

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.

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.

FIXED *pt orient*

Specifies the pin’s location, orientation, and that it’s 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.

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.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

*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. For an example, see [Example 4-18](#) on page 287.

**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).

#### 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).

#### 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 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, 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 4-17](#) on page 285.

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

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

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.

#### Example 4-17 Multi-Mask Patterns for Pins

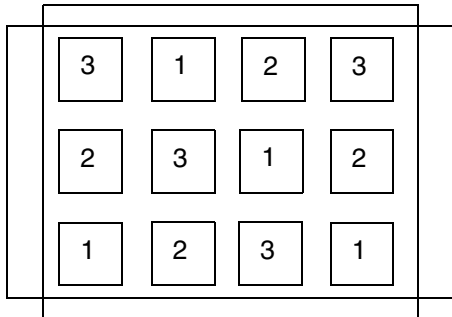
The following example shows via master masks:

```
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
...
```

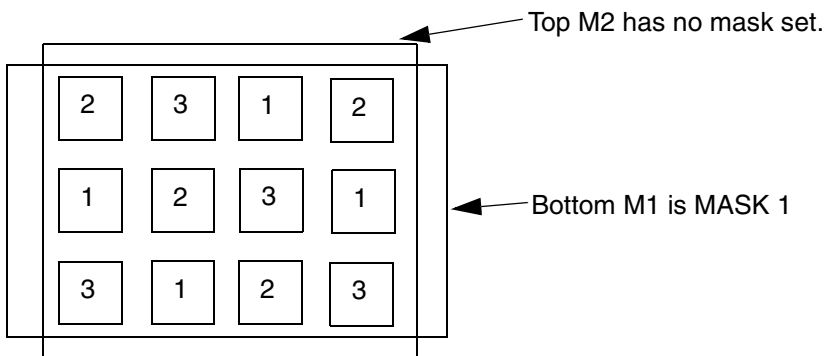
The VIA1 via will have:

- 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 286.

**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.

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.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

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.

#### Example 4-18 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
    ...
    + 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
```

*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).

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

`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.

`PORT`

Indicates that the following `LAYER`, `POLYGON`, and `VIA` statements are all part of one `PORT` connection, until another `PORT` statement occurs. If this statement is missing, all of the `LAYER`, `POLYGON`, and `VIA` statements are 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
    [ SPACING minSpacing
      | DESIGNRULEWIDTH effectiveWidth]
    pt pt
  | + POLYGON layerName
    [ SPACING minSpacing
      | DESIGNRULEWIDTH effectiveWidth]
    pt pt pt
  | + VIA viaName pt
  ] ...
]
```

`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.

`POLYGON layerName pt pt pt`



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

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 4-20](#) on page 291.)

`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 4-21](#) on page 291.)

*Type:* Integer, specified in DEF database units

`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 4-21](#) on page 291.)

*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

### Example 4-19 Port Example

## LEF/DEF 5.8 Language Reference

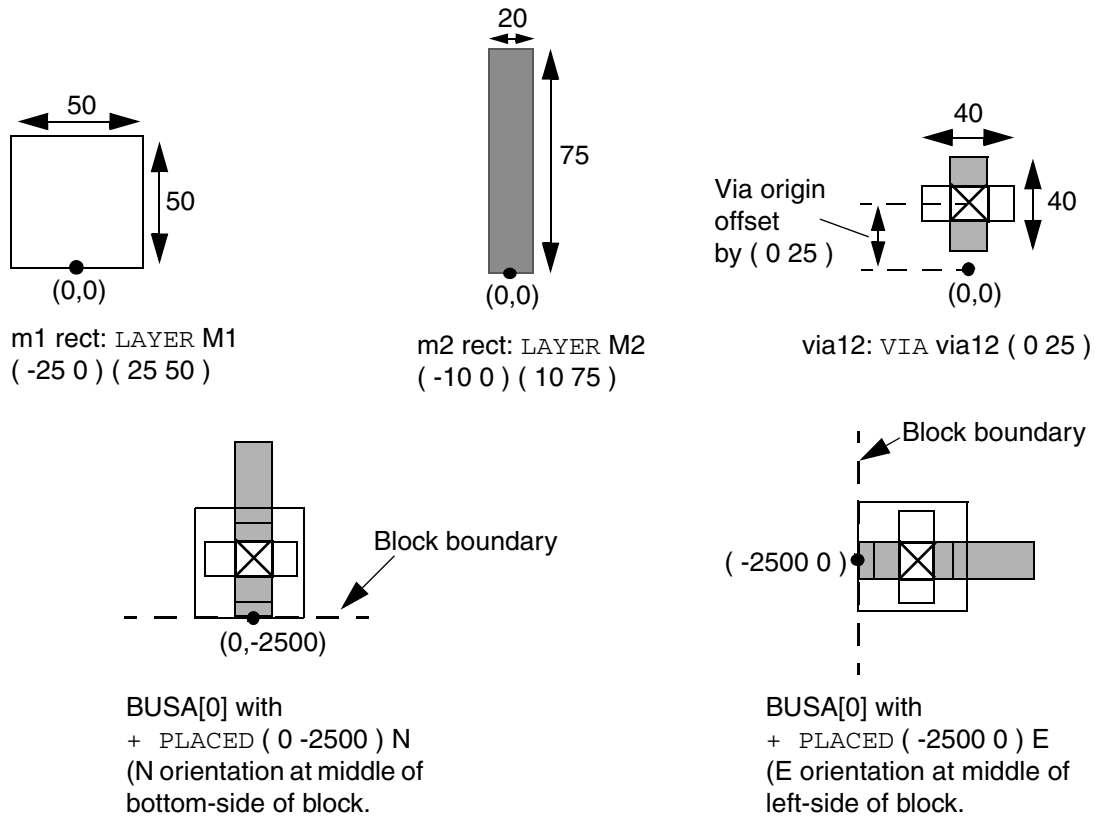
### DEF Syntax

---

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 291 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
  + 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 4-20 Port Statement With Polygon

The following PINS statement creates a polygon with a 45-degree angle:

```
PINS 2 ;
- myPin3 + NET myNet1 + DIRECTION INPUT
+ PORT
+ POLYGON metal1 ( 0 0 ) ( 100 100 ) ( 200 100 ) ( 200 0 ) #45-degree angle
+ FIXED ( 10000 5000 ) N ;
...
END PINS
```

### Example 4-21 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
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
+ 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 ;
END PINS
```

#### 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 4-18](#) on page 287.

**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.

#### ***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.









## LEF/DEF 5.8 Language Reference

### DEF Syntax

#### ***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.

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
```

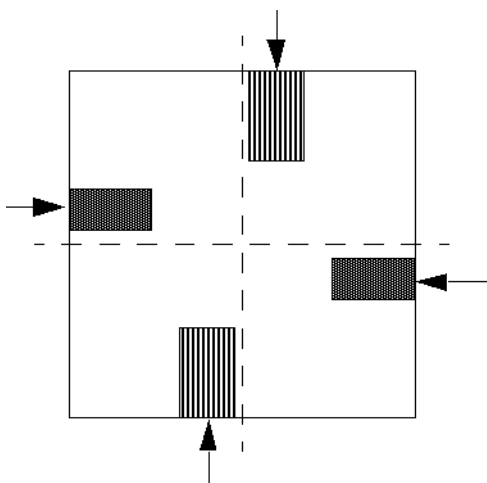
## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
- 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
      + 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]
```

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" ;  
    - comp1 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.



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

#### *objectType*

Specifies the object type being defined. You can define properties for the following object types:

COMPONENT  
COMPONENTPIN  
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 propName} ...] ...  
  ;] ...  
END REGIONS]
```

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 propName*

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 )
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
+ PROPERTY REGIONORDER 1 ;
```

## Rows

```
[ROW rowName siteName origX origY siteOrient  
    [DO numX BY numY [STEP stepX stepY]]  
    [+ PROPERTY {propName propVal} ...] ... ;] ...
```

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 244.

STEP *stepX* *stepY*

Specifies the spacing between sites in horizontal and vertical rows.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

#### 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,
                                         #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.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

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.

<i>floatingComp</i>	Specifies the component name.
( IN <i>pin</i> )	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.
( OUT <i>pin</i> )	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.
BITS <i>numBits</i>	<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 MAXBITS 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.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

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.

<i>fixedComp</i>	Specifies the component name.
( IN <i>pin</i> )	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.
( OUT <i>pin</i> )	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.
BITS <i>numBits</i>	<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 MAXBITS 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.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

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

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.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
- chain3_clock2
+ PARTITION clock2
  MAXBITS 1200
+ START block1/LV_testpoint_0_Q_reg Q
+ 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.



## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

- 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.
- 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 ) ...
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
        + 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 ) ;
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
- LAYER MET2
    RECT ( 1000 2000 ) ( 1500 4000 )
    RECT ( 1000 4500 ) ( 1500 6500 )
    RECT ( 1000 7000 ) ( 1500 9000 )
    RECT ( 1000 9500 ) ( 1500 11500 ) ;
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

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

message appears. On output, the writer outputs the netlist in either format, depending on the command arguments of the output command.

*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 ;
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

```
- bumpa2 bumpcell + COVER ( 200 100 ) N ;
```

The pin is assigned in the PIN statement.

```
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.

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

#### *specialWiring*

Specifies the special wiring for the net. For syntax information, see “[Special Wiring Statement](#)” on page 311.

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 321.

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 321.

*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 321. For an example of special wiring with routing points, see [Example 4-31](#) on page 317.

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 316.

*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 317.

## 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 244).

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.

*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.

( *x y* )

Specifies the route coordinates. You cannot specify a route with zero length.

For more information, see [“Specifying Coordinates”](#) on page 322.

*Type:* Integer, specified in database units

### 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

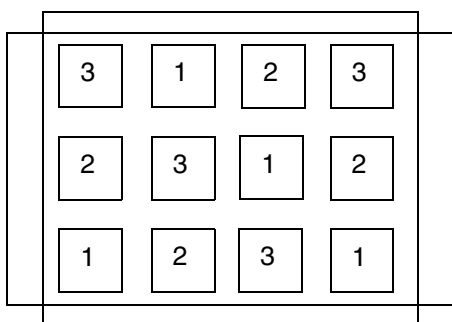
## LEF/DEF 5.8 Language Reference

### DEF Syntax

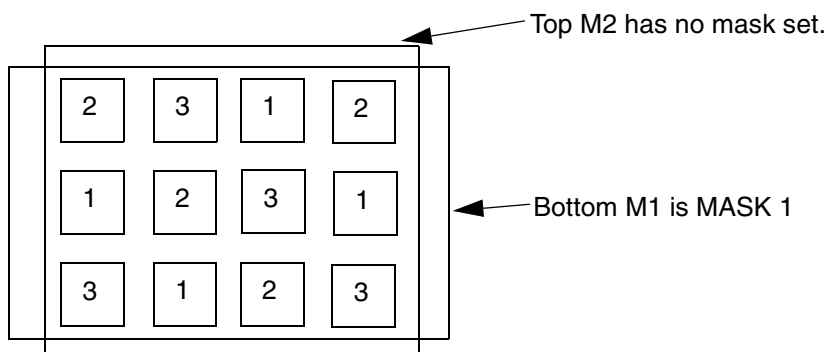
- ❑ *bottomMaskNum* is 1 in the 031 value, so mask 1 is used for the bottom metal (M1) shape
- ❑ *cutMaskNum* is 3 in the 031 value, so the bottom-most, then left-most cut of the via-instance is mask 3. The mask for the other cuts of the via-instance are derived from the via-master by "shifting" the 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 318.

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

---

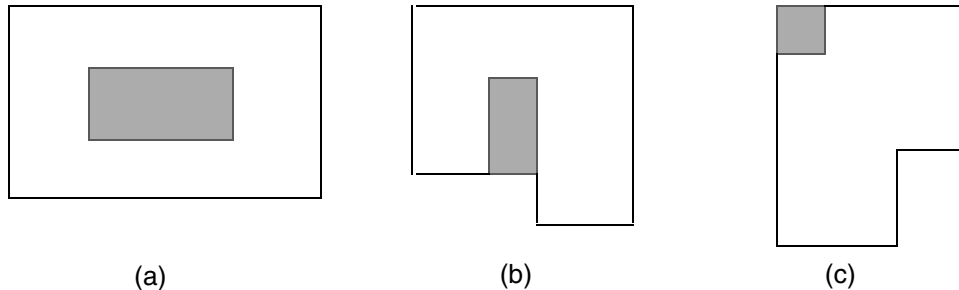
#### SHAPE

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 320.)

**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

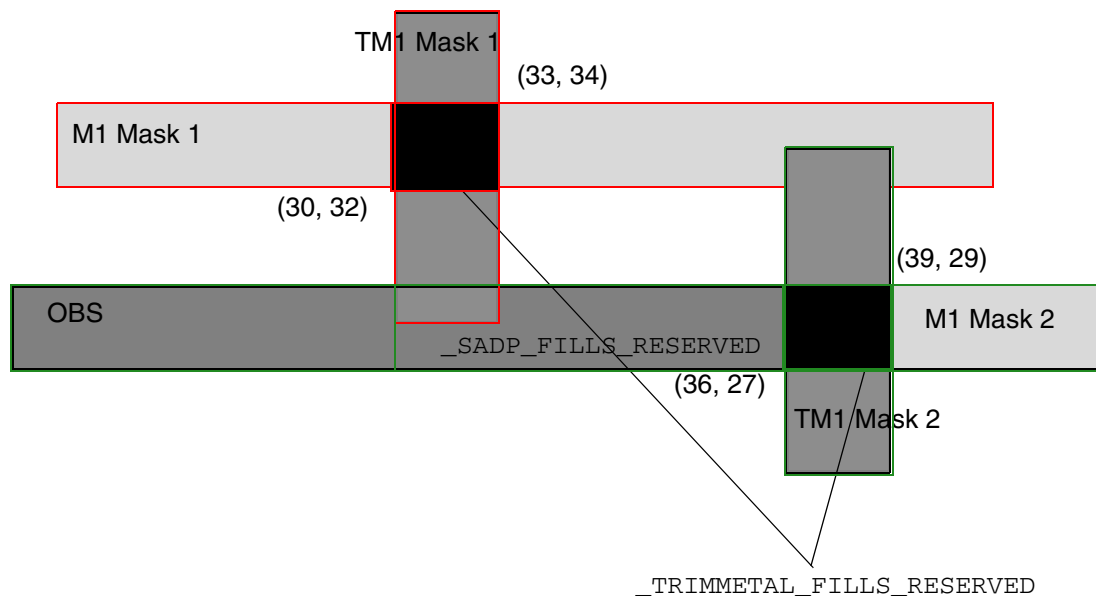
As trim metal shapes need to be aligned and merged, dummy patches are often added even on OBS and unconnected pins. Those patches would be written out in `_SADP_FILLS_RESERVED` with the `DRCFILL` tag in the `SPECIALNETS` section in the following format:

```
- _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 327 through [Figure 4-24](#) on page 336 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

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

at a 45-degree angle, and must enclose the point (0 0).

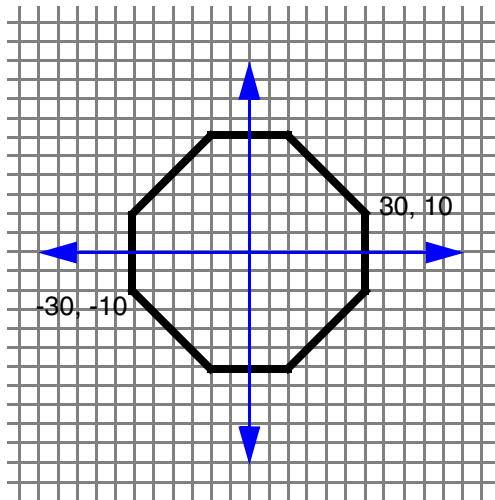
Type: Integer, specified in DEF database units

#### Example 4-34 Styles Statement

The following `STYLES` statement defines the basic octagon shown in [Figure 4-14](#) on page 325.

```
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))

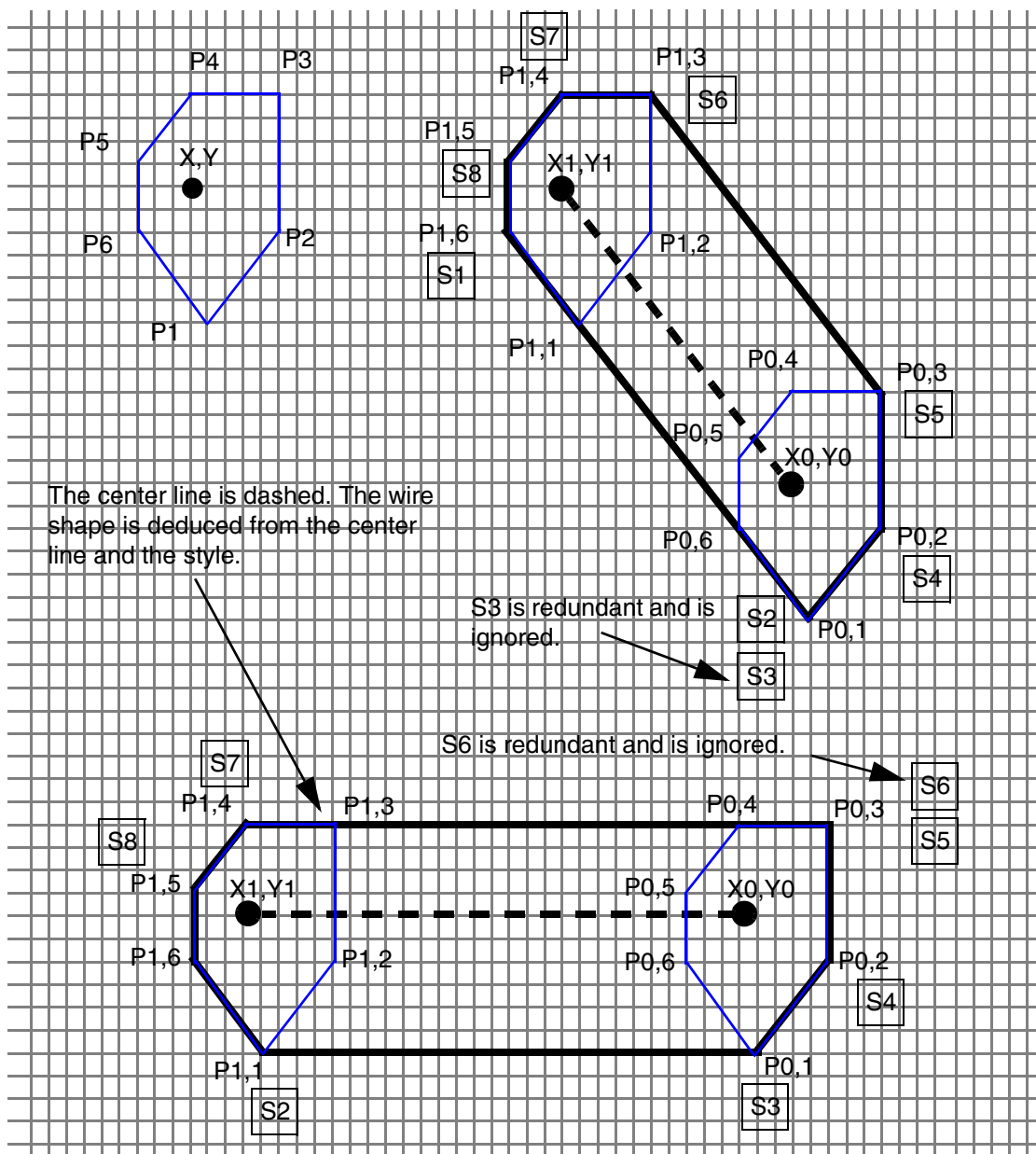
S6 = right-most point in (highest points in (P0,1 through P0,n P1,1 through P1,n))

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 327. 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

---

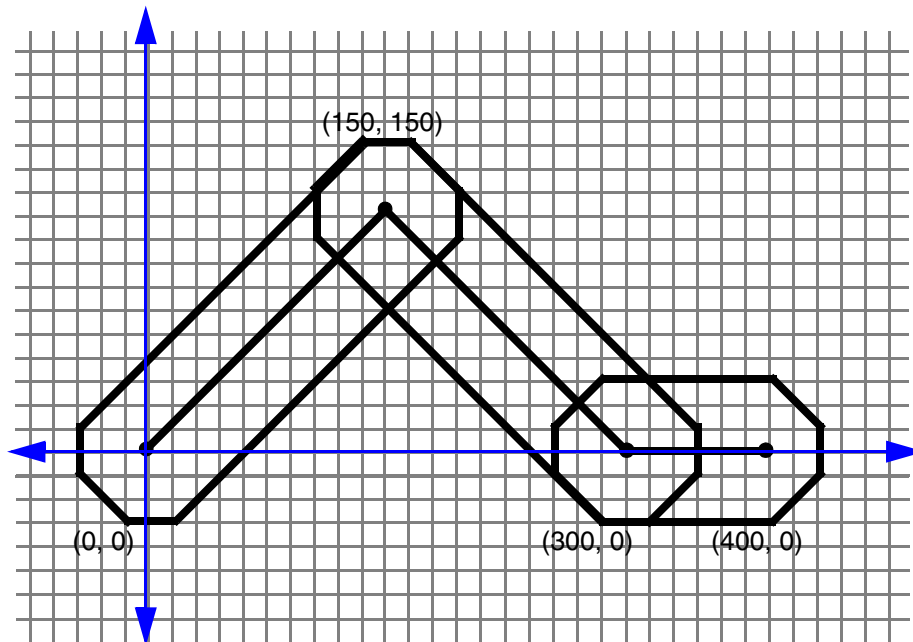
The following statements define an X wire with octagonal ends, as shown in [Figure 4-16](#) on page 328.

```
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

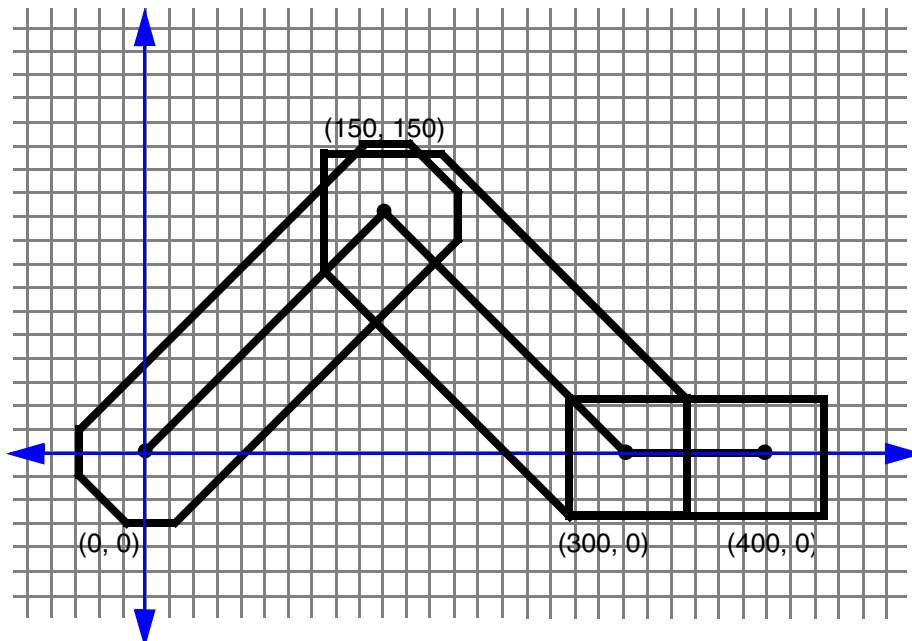
The following statements define the same X wire with mixed octagonal and manhattan styles, as shown in [Figure 4-17](#) on page 329.

```
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

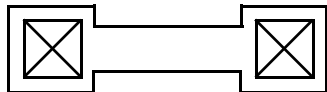
The following statements define a manhattan wire with a width of 70, as shown in [Figure 4-18](#) on page 330.

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 metal1 100 + STYLE 0 ( 0 0 ) via12 ( 600 * ) via12 ;
END SPECIALNETS
```

**Figure 4-18**



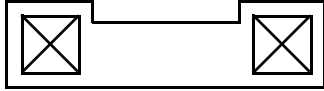
#### Example 4

The following statements define a similar wire that is offset from the center, as shown in [Figure 4-19](#) on page 331. 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 metal1 100 + STYLE 0 ( 0 0 ) via12 ( 600 * ) via12 ;
END SPECIALNETS
```

**Figure 4-19**



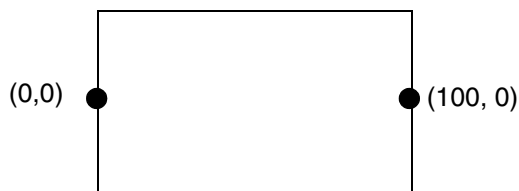
### Example 5

The following statements define a wire that uses a “2-point line” style, as shown in [Figure 4-20](#) on page 331.

**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 metal1 20 + STYLE 0 ( 0 0 ) ( 100 0 ) ;  
END SPECIALNETS
```

**Figure 4-20**

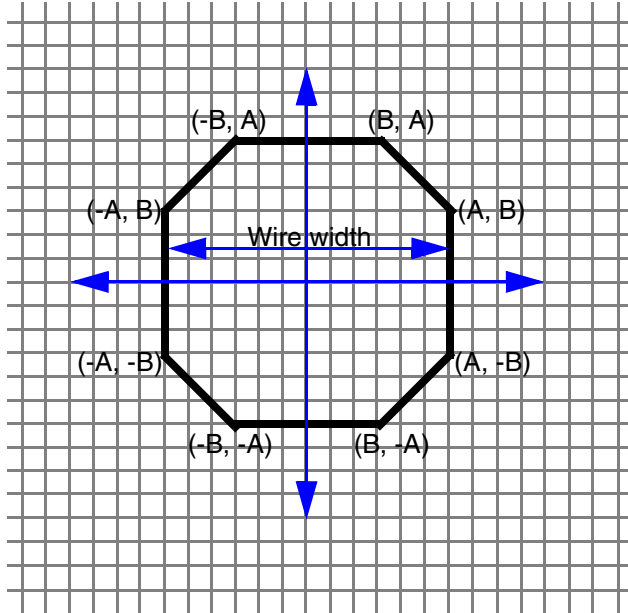


### 45-Degree Routing Without Styles

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 332) 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:

$$D/\text{sqrt}(2) = A + B$$

$$B = D/\text{sqrt}(2) - A \text{ or } W/\text{sqrt}(2) - A$$

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:

$$B = [\text{ceiling}(W/(\text{sqrt}(2) * M) * M) - A$$

The following table lists examples coordinate computations:

**Table 4-1**

W = Width ( $\mu\text{m}$ )	M = mgrid ( $\mu\text{m}$ )	D = W/(sqrt(2)*M)	ceiling (D)	A ( $\mu\text{m}$ )	B ( $\mu\text{m}$ )	Diagonal width ( $\mu\text{m}$ )
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 334.

```
SPECIALNETS 1 ;
- POWER (* power)
```

## LEF/DEF 5.8 Language Reference

### DEF Syntax

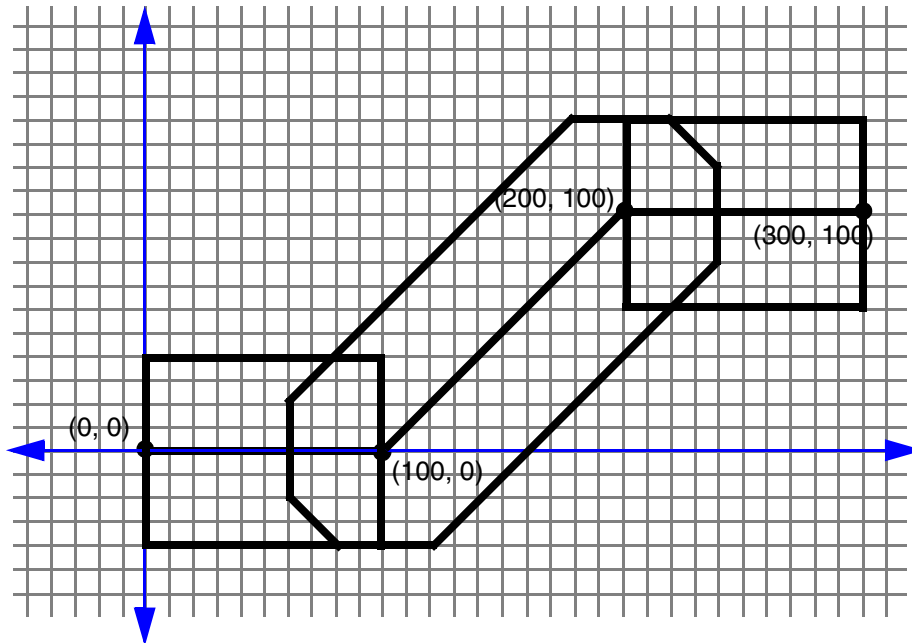
```

+ 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 335.

```

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

```

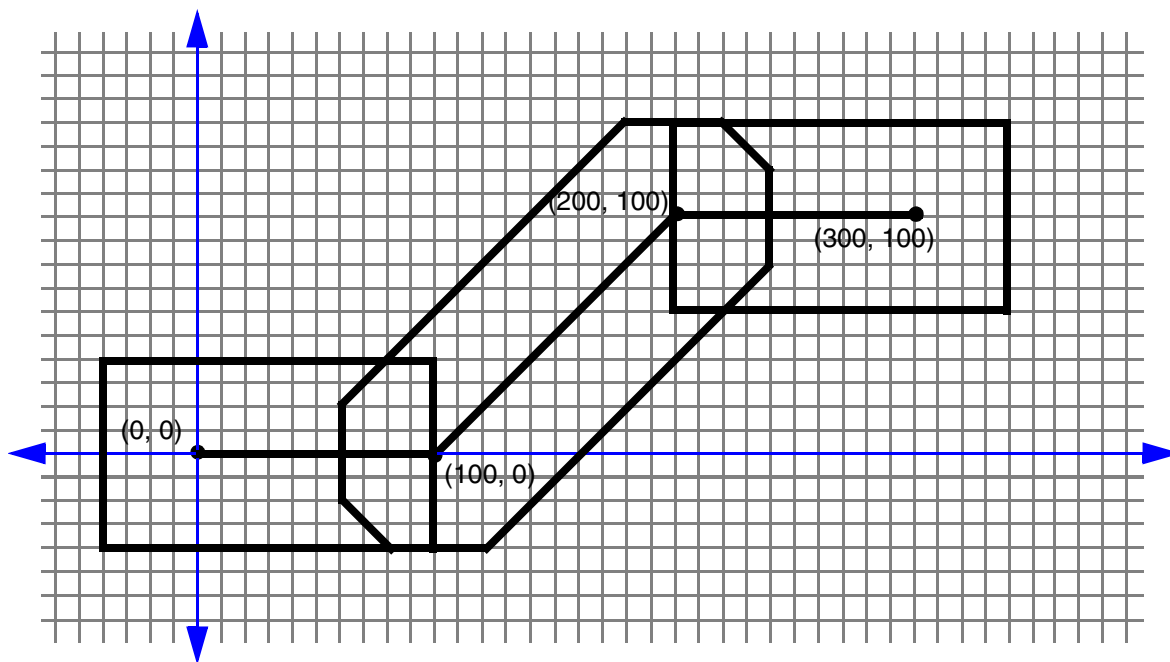
## LEF/DEF 5.8 Language Reference

### 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

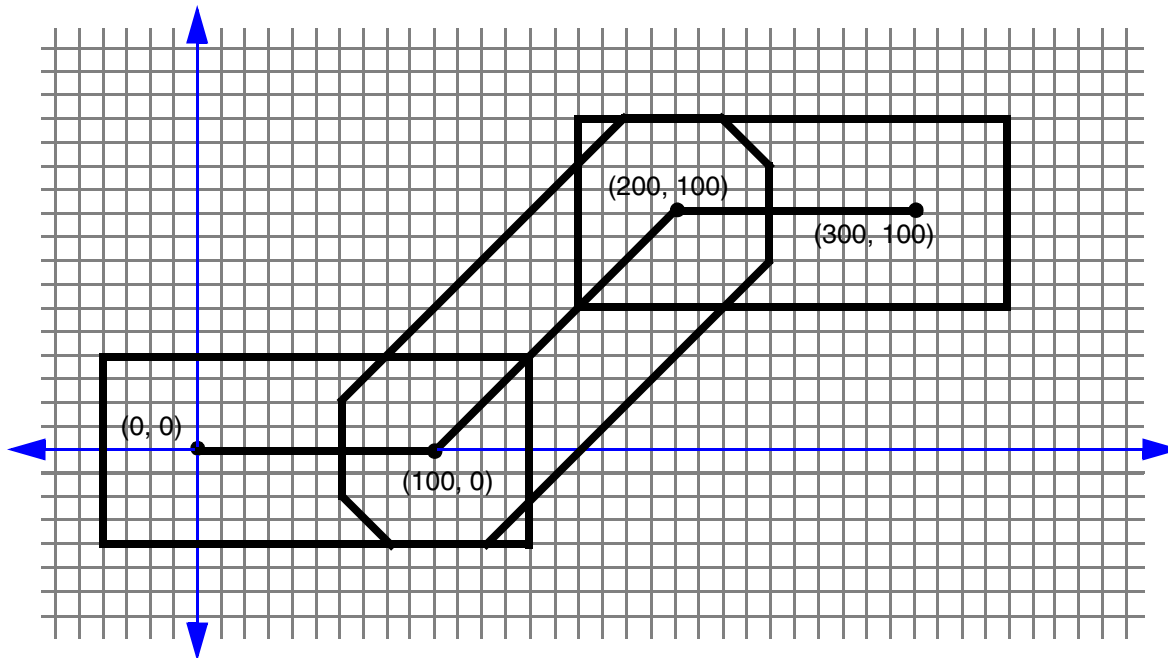
The following definition, using the traditional default wire extension of  $1/2 * width$  for all of the points, produces the route in [Figure 4-24](#) on page 336.

```
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 337.

`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 344).  
*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 348). 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 349.

#### `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/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 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 344.

*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 344.

*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 349.

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 346). The other two patterns supported are alternating rows, and alternating columns (see [Figure 4-29](#) on page 347).

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

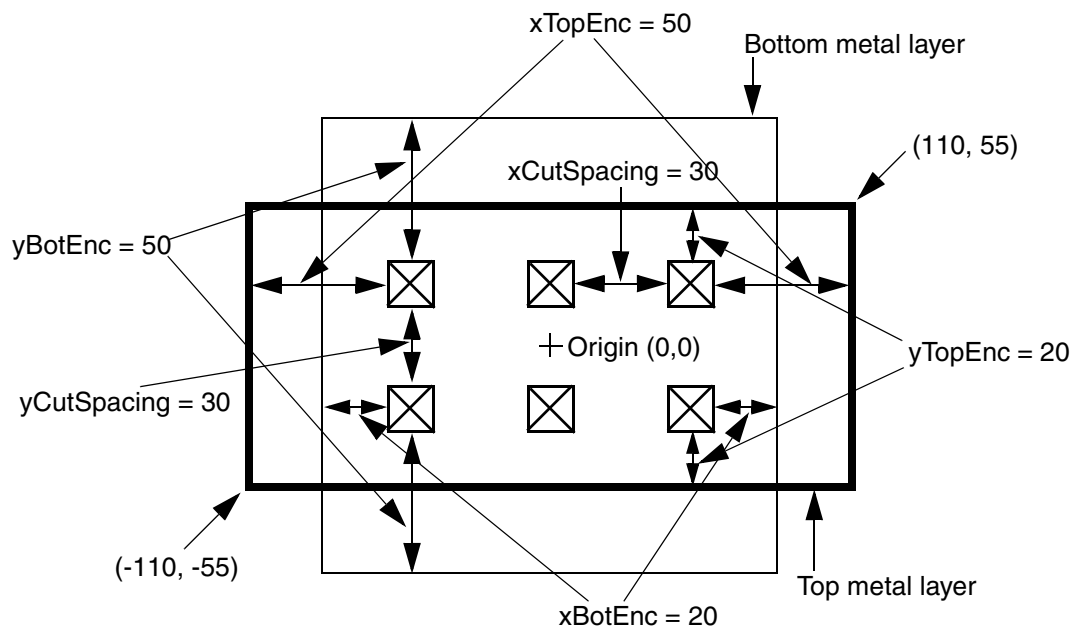
this routing-cut-routing (or masterslice-cut-masterslice) layer combination. This makes it possible for a tool that does not use the LEF `VIARULE` technology section to still generate DEF generated-via parameters by using the default rule.

### 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 344 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**

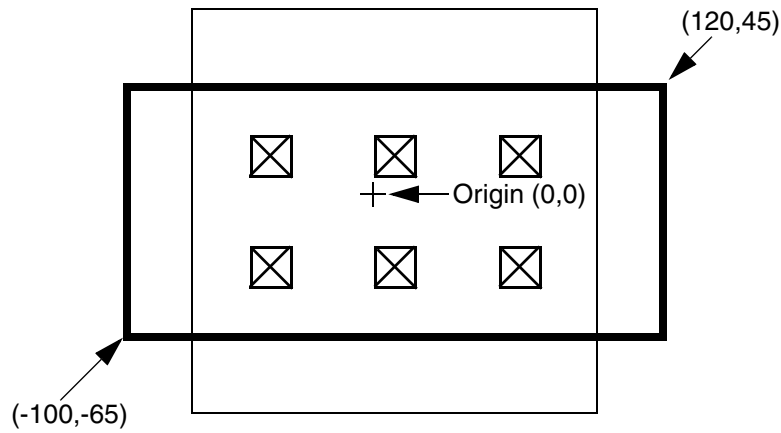


The same via rule with the following `ORIGIN` parameter shifts all of the metal and cut rectangles by 10 in the x direction, and by -10 in the y direction (see [Figure 4-26](#) on page 345):

```
+ ORIGIN 10 -10
```



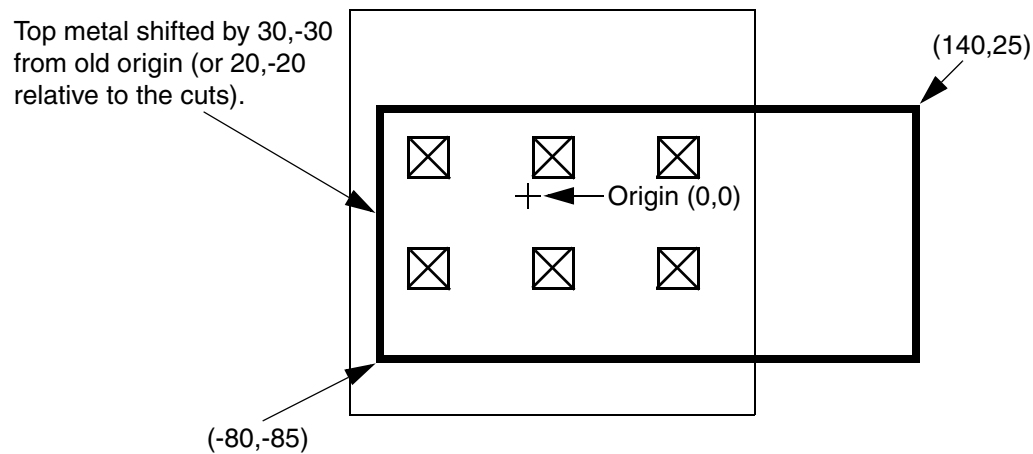
**Figure 4-26 Via Rule With Origin**



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:

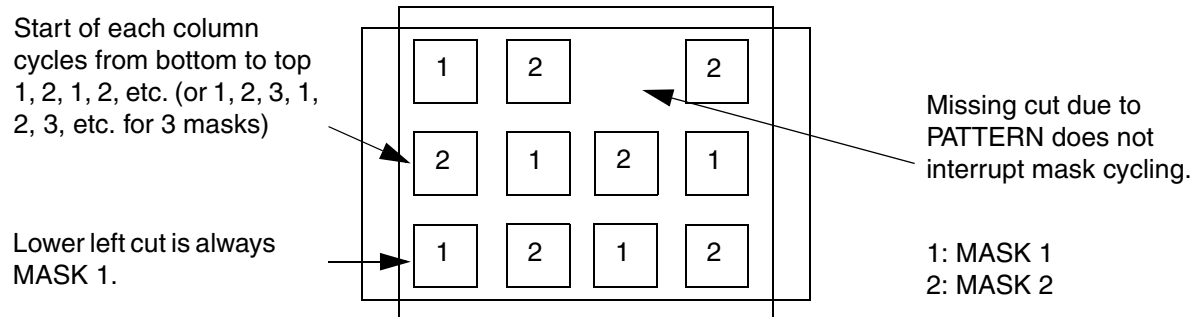
```
- myParamVia1
  + VIARULE myGenVia1      + CUTSIZE 40 40
  + LAYERS M1 VIA1 M2      + CUTSPACING 40 40
```

## LEF/DEF 5.8 Language Reference

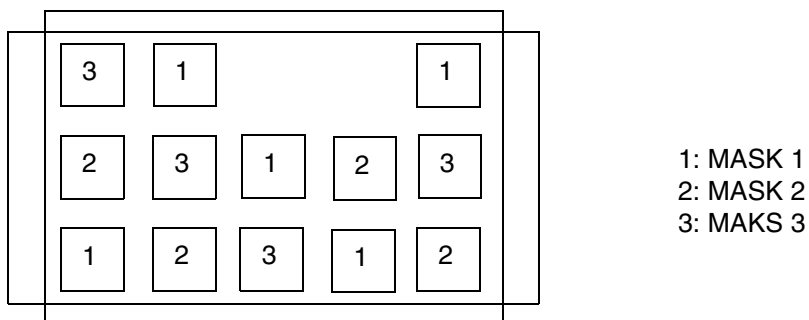
### DEF Syntax

```
+ 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

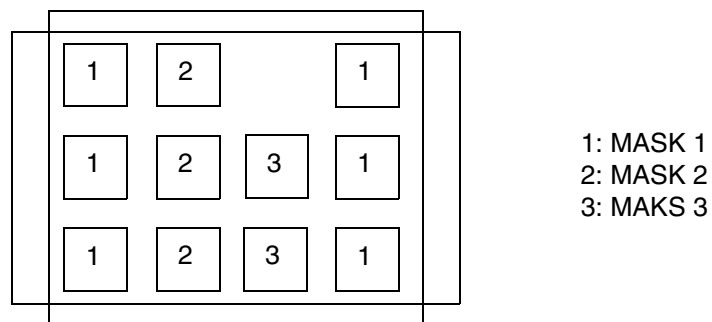
---

**Figure 4-29 Multi-Mask Patterns for Parameterized Vias using Suffixes**

On a 3 mask cut layer

Mask colors for a parameterized via like:

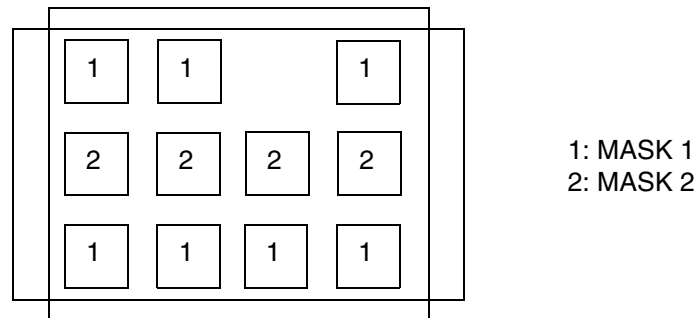
```
- myParamVia1
+ VIARULE myGenVia1      + CUTSIZE 40 40
+ LAYERS M1 VIA1 M2      + CUTSPACING 40 40
+ ENCLOSURE 40 0 0 40    + ROWCOL 3 4
+ PATTERN 2_F_1_D_MC;    #alternating mask color columns due to _MC suffix
```



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 346. 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

---

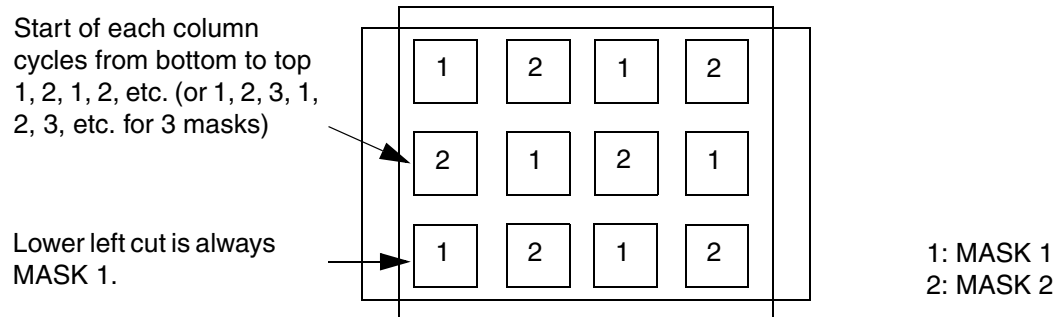
not have a row and column structure or are missing cuts, then this coloring may not be good (see [Figure 4-30](#) on page 349). If the metal layers are not colored, some applications will color them to mask 1 for internal consistency, even though the via master metal shape colors are not really used by LEF or DEF via instances.

#### Example 4-38 Multi-Mask Patterns for Fixed Via

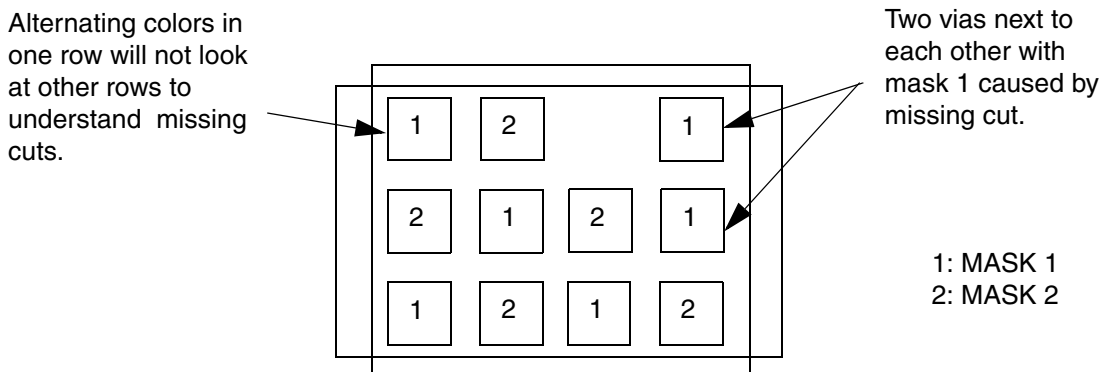
The following example shows a fixed-via with pre-colored cut shapes:

```
- 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,

## LEF/DEF 5.8 Language Reference

### DEF Syntax

---

the last bits are ignored. The number of rows defined must equal the *numCutRows* value in the ROWCOL parameter.

Figure 4-31 on page 350 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**

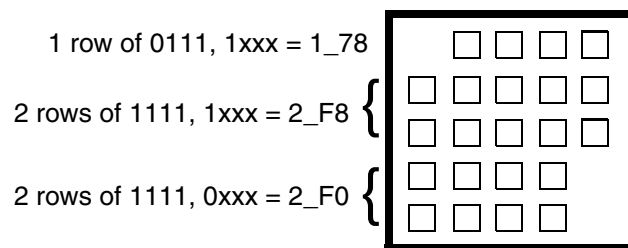


Figure 4-32 on page 351 illustrates the following via cut pattern syntax:

```
- myVia
+ VIARULE myViaRule
...
+ ROWCOL 5 14
+ PATTERN 2_FFE0_3_R4F ;
```

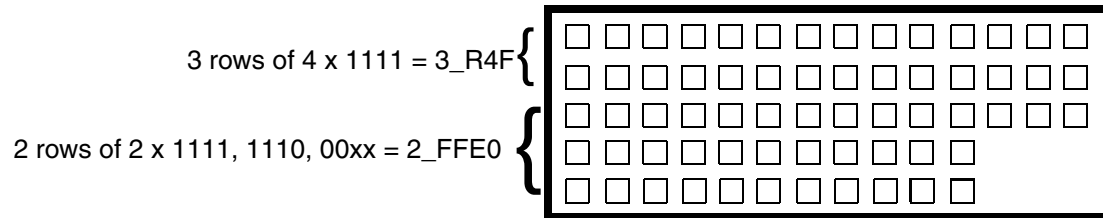
The R4F value indicates a repeat of four Fs. The last two bits of each row definition are ignored because only 14 bits allowed in each row.

## 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 353
- [DEF](#) on page 364
- [Scan Chain Synthesis Example](#) on page 369

### 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 372
- [Guidelines for Wide Metal Spacing](#) on page 374
- [Guidelines for Wire Extension at Vias](#) on page 375
- [Guidelines for Default Vias](#) on page 377
- [Guidelines for Stack Vias \(MAR Vias\) and Samenet Spacing](#) on page 379
- [Example of an Optimized LEF Technology File](#) on page 383

## 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

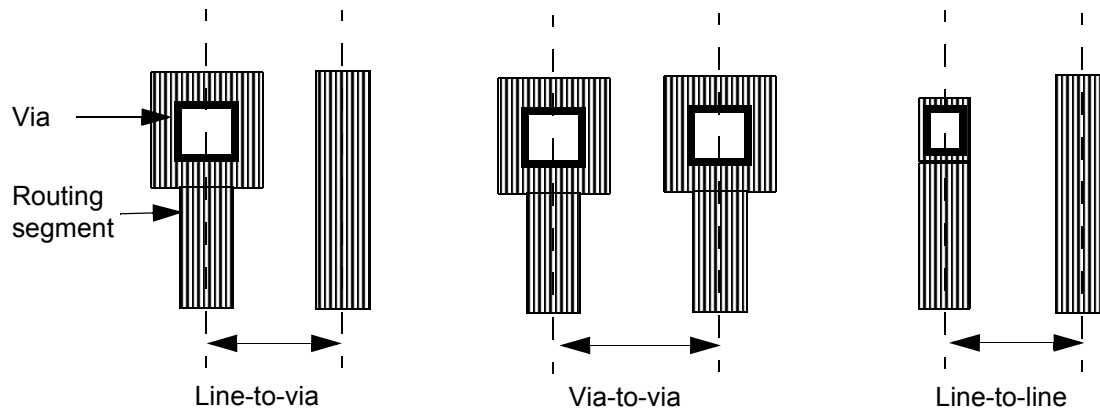
SPACING
    SAMENET
        layerName layerName minSpace [STACK] ;

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



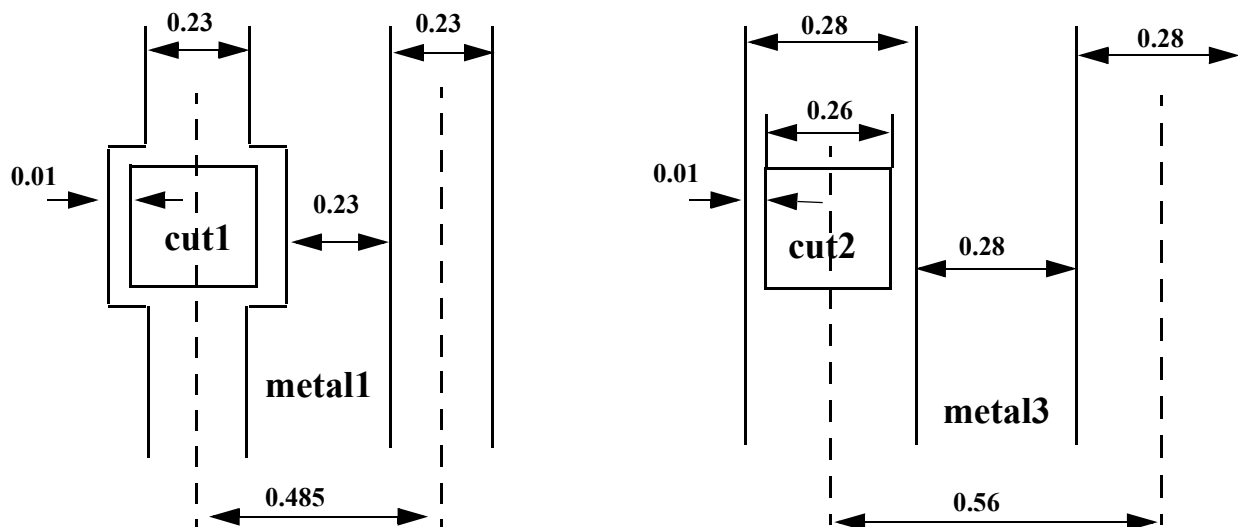
## LEF/DEF 5.8 Language Reference

### Optimizing LEF Technology for Place and Route

---

#### 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

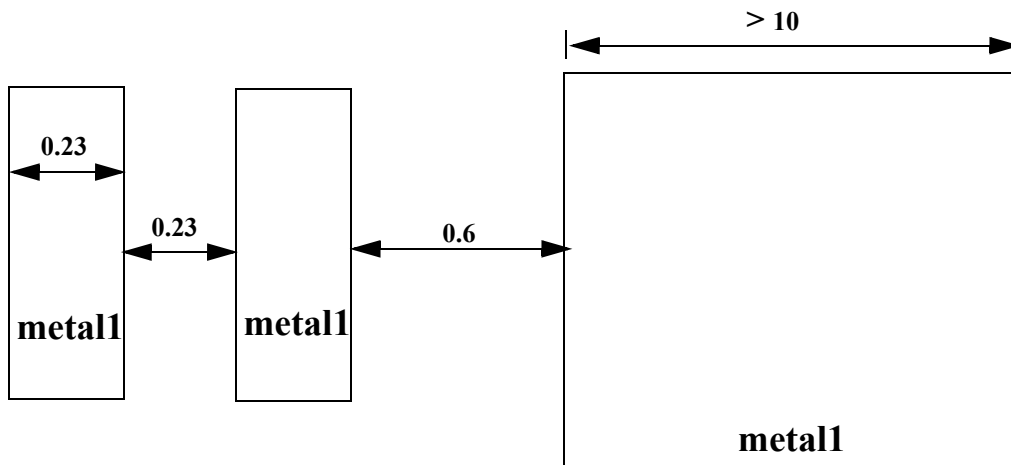
- All pins should be on the grid, and only those portions of the pins that are accessible to the router should be modeled as pins. For example, 45 degree pin geometry.
- The height of the cell should be the even multiple of the `metal1` pitch, and the width of the cell should be the even multiple of 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

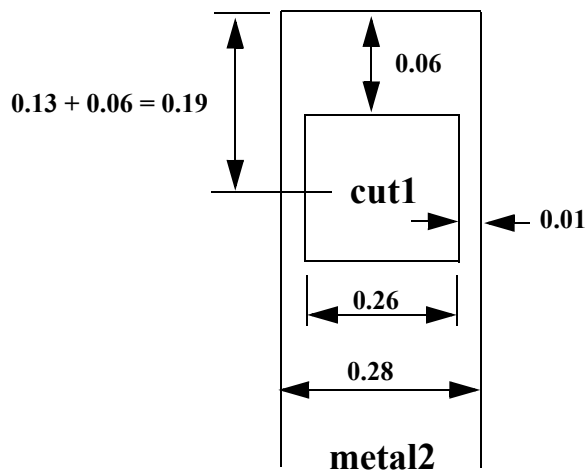
```
LAYER metal1
    WIDTH 0.23 ;
    SPACING 0.23 ;
    SPACING 0.6 RANGE 10.002 1000 ;
END metal1
```

## 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.

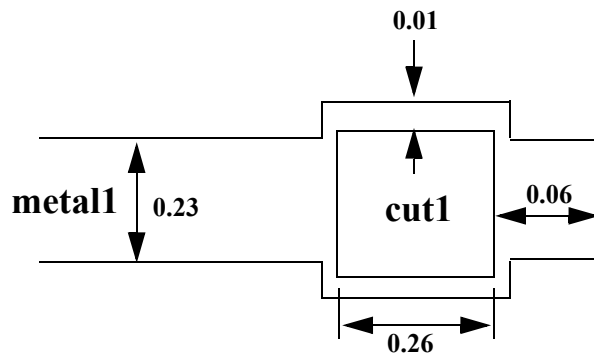
## Guidelines for Default Vias

The following guidelines are for default vias.

## DESIGN RULE No. 4

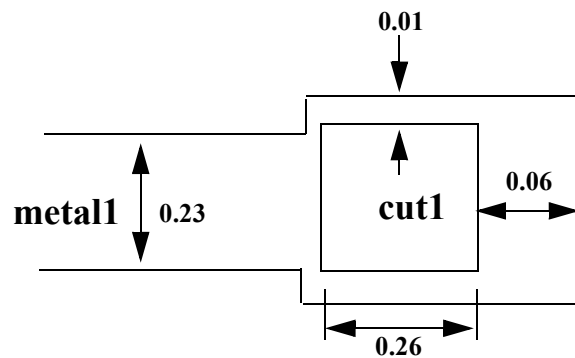
- W.1            Minimum width of `metal1` = 0.23  $\mu\text{m}$
- W.2            Minimum and maximum width of `cut1` = 0.26  $\mu\text{m}$
- E.1            Minimum extension of `metal1` beyond `cut1` = 0.01  $\mu\text{m}$
- E.2            Minimum extension of `metal1` end-of-line region beyond `cut1` = 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**

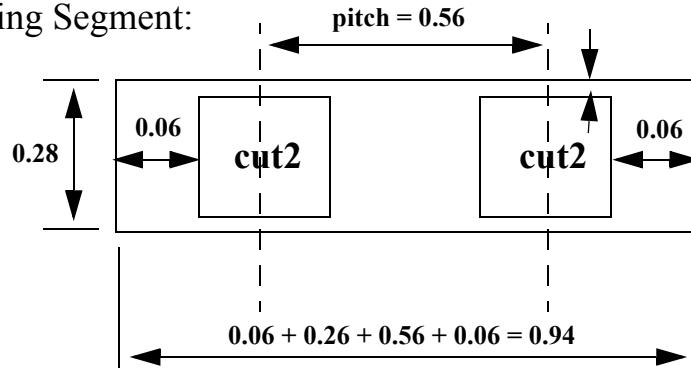
- |     |   |
|-----|---|
| W.1 | Minimum width of <code>metal2</code> = 0.28 $\mu\text{m}$   |
| W.2 | Minimum and maximum width of <code>cut2</code> = 0.26 $\mu\text{m}$   |
| E.1 | Minimum extension of <code>metal2</code> beyond <code>cut2</code> = 0.01 $\mu\text{m}$                                    |
| 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

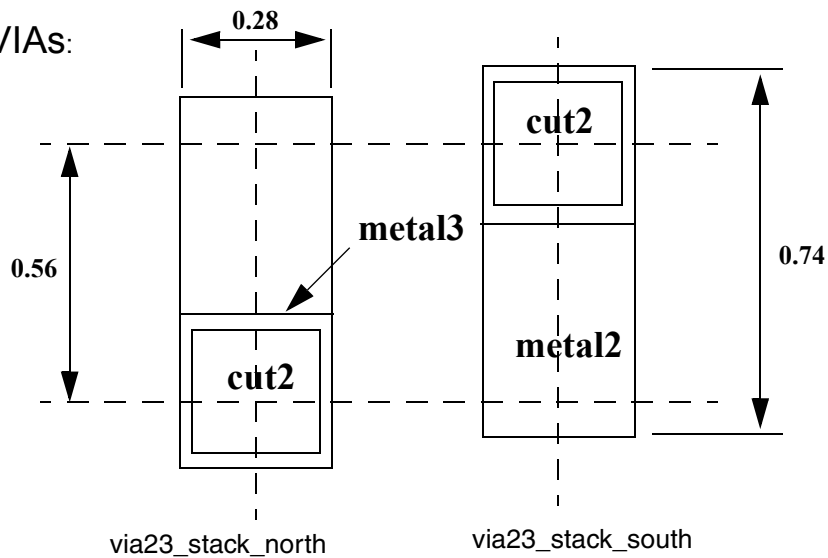
---

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 metall1 ;
        RECT -0.19 -0.14 0.19 0.14 ;      # metall1 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 metall1 ;
        RECT -0.14 -0.19 0.14 0.19 ;      # metall1 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 390
- [Using Process Antenna Keywords in the LEF and DEF Files](#) on page 394
- [Calculating Antenna Ratios](#) on page 395
- [Checking for Antenna Violations](#) on page 412
- [Using Antenna Diode Cells](#) on page 423
- [Using DiffUseOnly](#) on page 424
- [Calculations for Hierarchical Designs](#) on page 425

## 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.

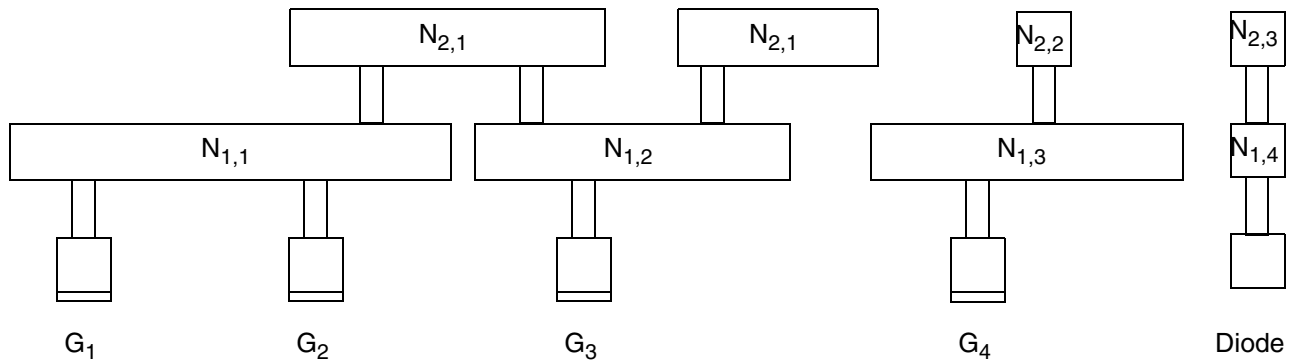
## What Are Process Antennas?

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 391 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 391, 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 391, 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.



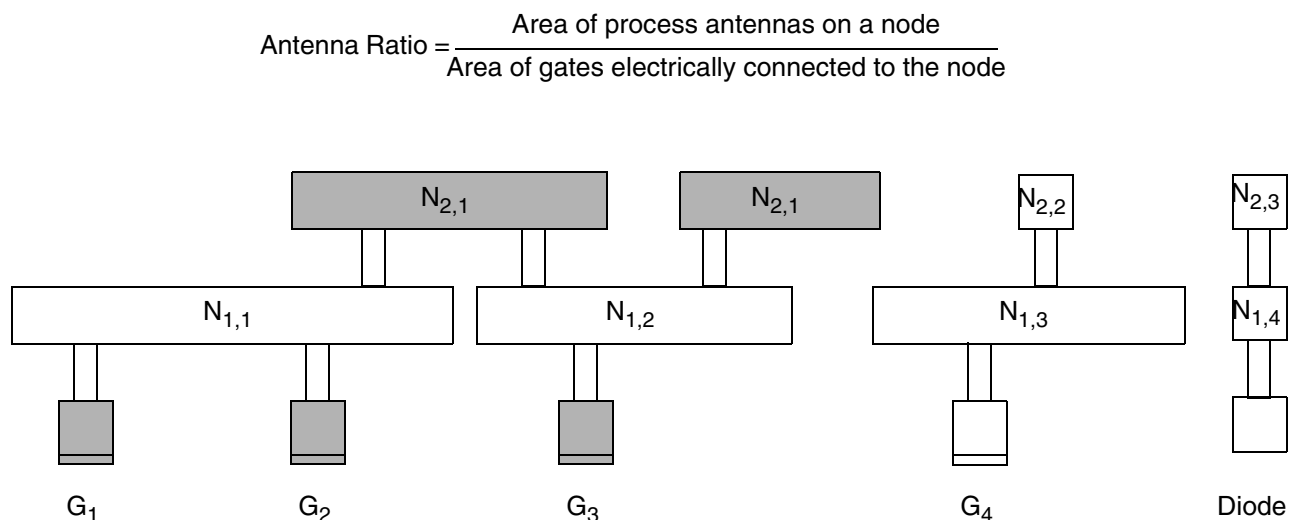
## What Is the Antenna Ratio?

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 393 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 423.

## 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 424.		

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:

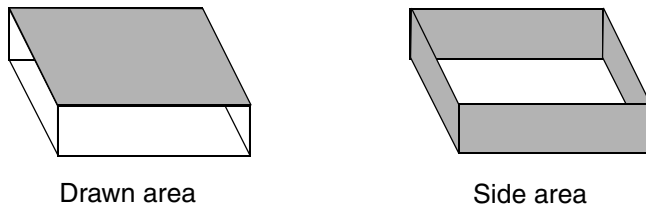
- The drawn area (the top surface area of the metal shape)

- 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 396 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>*.

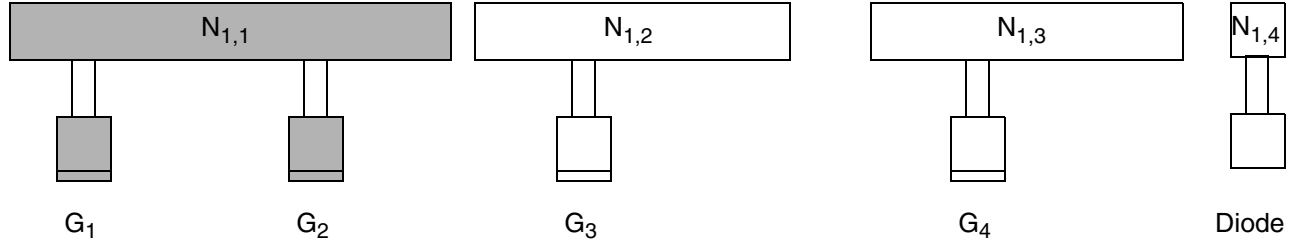
$\text{Area}(G_k)$  is the drawn or side area of gate  $G_k$ . (The reason to include the  $G_k$  parameter for PAR is to maintain uniformity with the notation for CAR.)

**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 398 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:

$$PAR(N_{1,2}, G_3) = \frac{Area(N_{1,2})}{Area(G_3)}$$

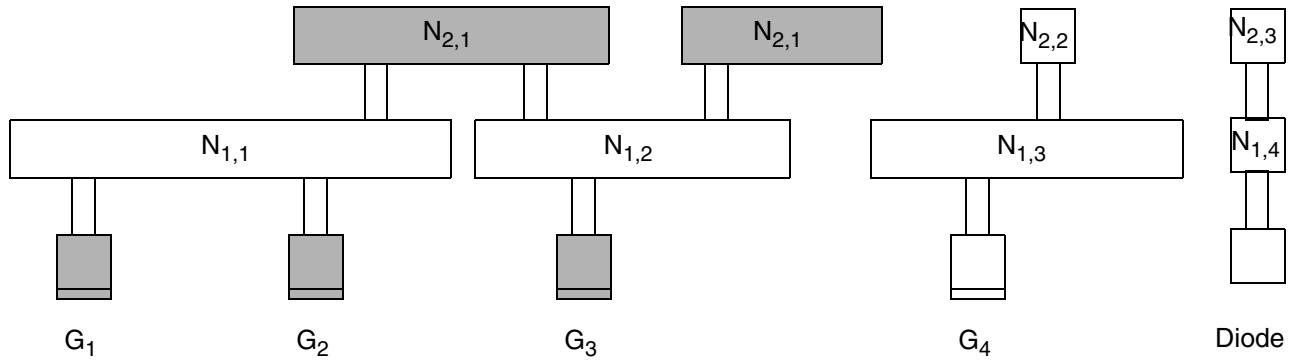
To calculate  $PAR(N_{i,j}, G_k)$  for node  $N_{1,3}$ , another node on the first metal layer, with respect to gate  $G_4$ , use the following formula:

$$PAR(N_{1,3}, G_4) = \frac{Area(N_{1,3})}{Area(G_4)}$$

### Calculations for PAR on the Second Metal Layer

Figure C-5 on page 399 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  $PAR(N_{2,1}, G_1)$ , you must add the area of both pieces together.

To calculate  $PAR(N_{i,j}, G_k)$  for node  $N_{2,1}$ , a node on the second metal layer, with respect to gate  $G_1$ , use the following formula:

$$PAR(N_{2,1}, G_1) = \frac{Area(N_{2,1})}{Area(G_1) + Area(G_2) + Area(G_3)}$$

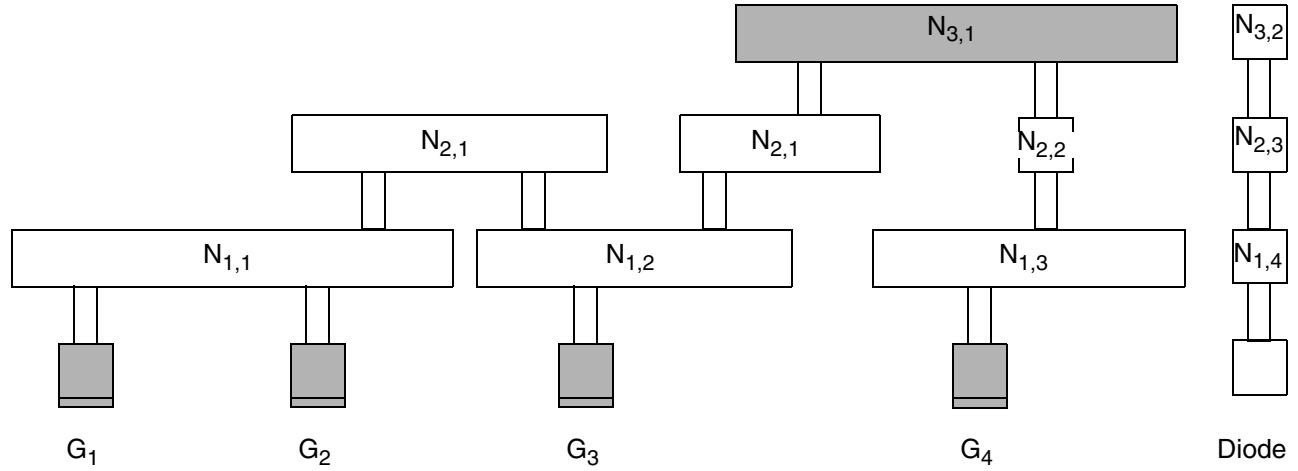
As on the first layer,

$$PAR(N_{2,1}, G_1) = PAR(N_{2,1}, G_2) = PAR(N_{2,1}, G_3)$$

### Calculations for PAR on the Third Metal Layer

[Figure C-6](#) on page 400 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,

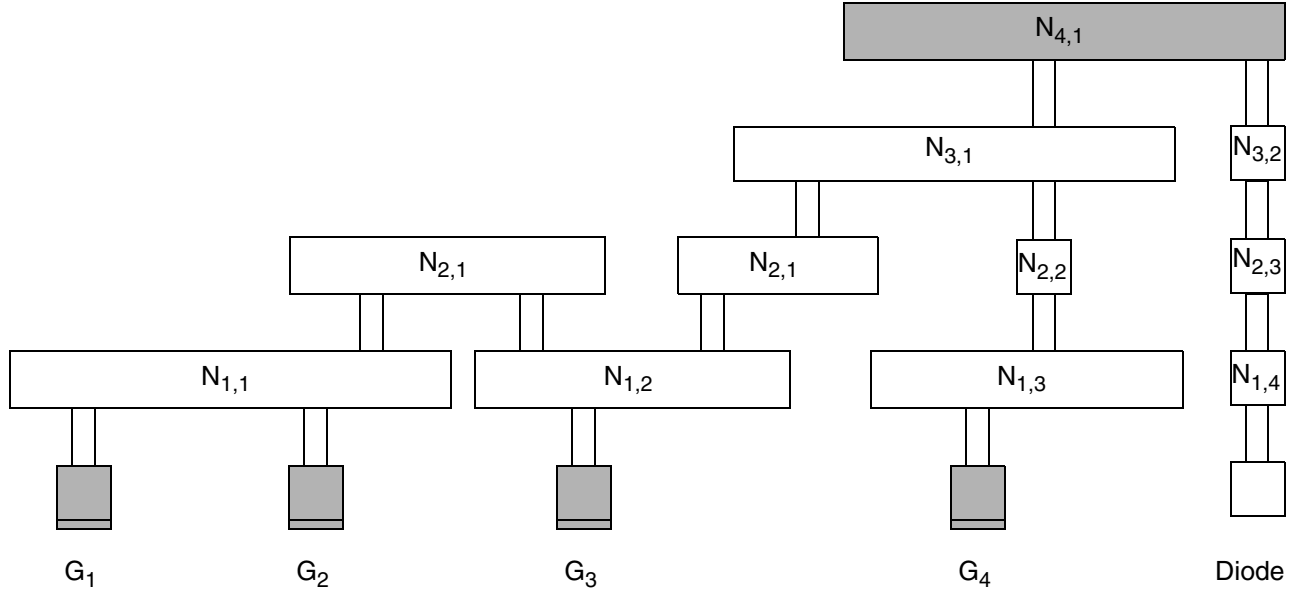
$$PAR(N_{3,1}, G_1) = PAR(N_{3,1}, G_2) = PAR(N_{3,1}, G_3) = PAR(N_{3,1}, G_4)$$

### Calculations for PAR on the Fourth Metal Layer

Figure C-7 on page 401 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:

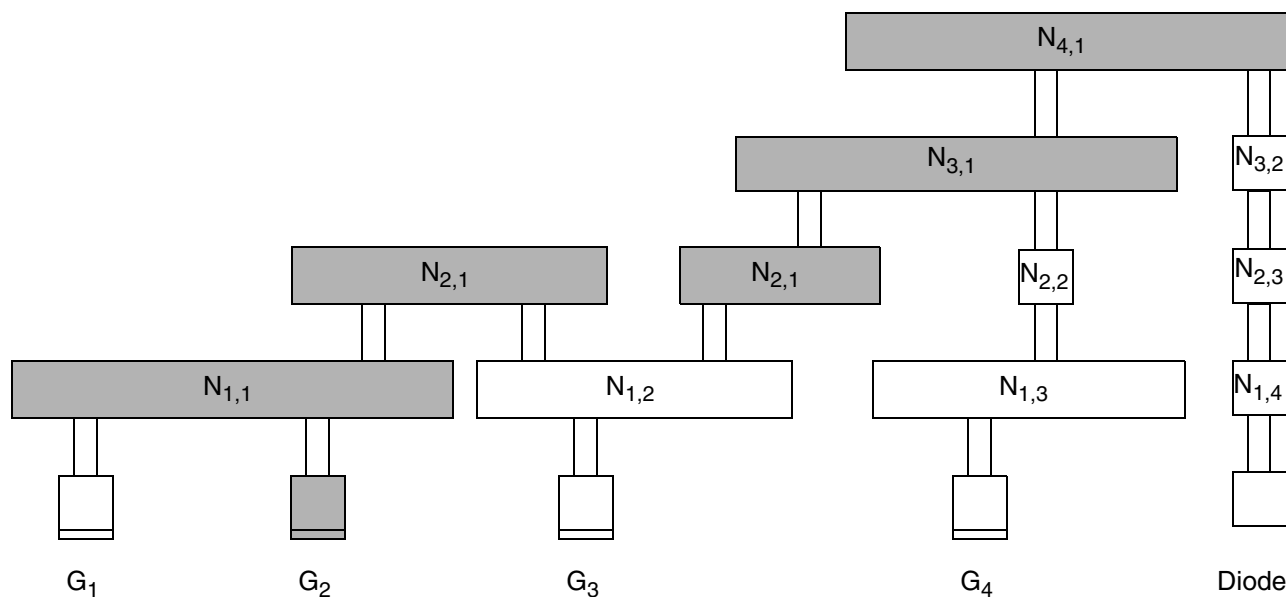
$$\text{CAR}(m_i) = \text{PAR}(m_i) + \text{CAR}(v_{i-1})$$

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 402 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**

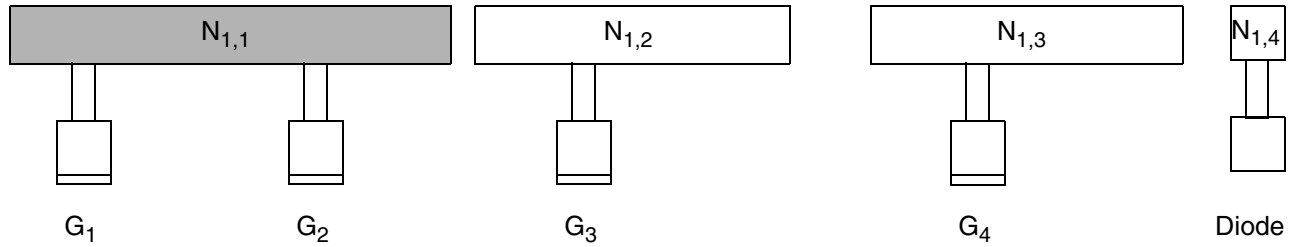
In [Figure C-8](#) on page 402, node  $N_{1,2}$  is not shaded because it was not electrically connected to  $G_2$  when *metal1* was processed. That is, because the charge accumulated on  $N_{1,2}$  when *metal1* was processed cannot damage gate  $G_1$ , the router does not include it in the calculations 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 403 shows the chip after the first metal layer is processed.

**Figure C-9**



In the figure above,

$$\text{CAR}(N_{1,1}, G_1) = \text{PAR}(N_{1,1}, G_1)$$

$$\text{CAR}(N_{1,1}, G_2) = \text{PAR}(N_{1,1}, G_2)$$

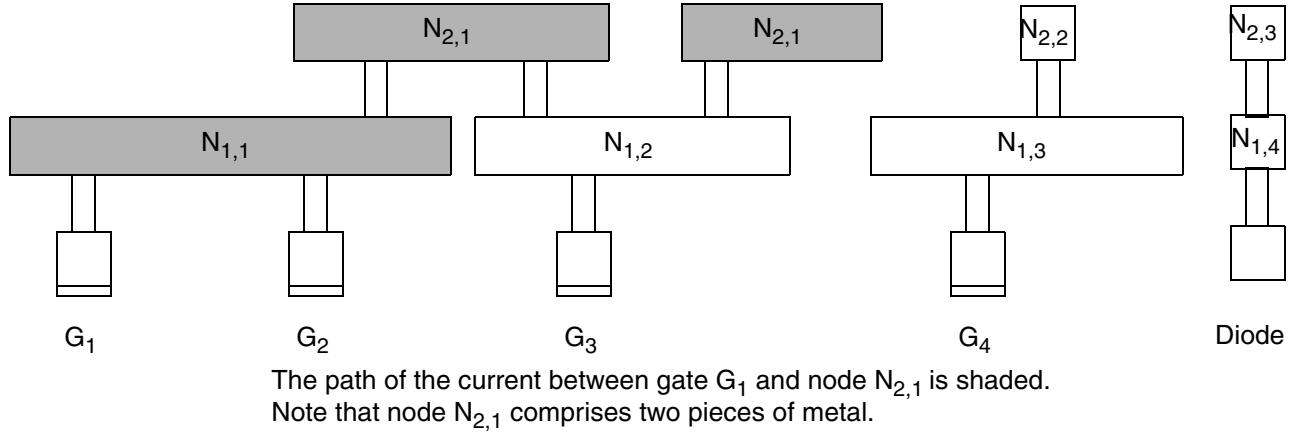
Because  $\text{PAR}(N_{1,1}, G_1)$  equals  $\text{PAR}(N_{1,1}, G_2)$ ,  $\text{CAR}(N_{1,1}, G_1)$  equals  $\text{CAR}(N_{1,1}, G_2)$ .

**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 404 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,

$$CAR(N_{2,1}, G_1) = PAR(N_{1,1}, G_1) + PAR(N_{2,1}, G_1)$$

$$CAR(N_{2,1}, G_2) = PAR(N_{1,1}, G_2) + PAR(N_{2,1}, G_2)$$

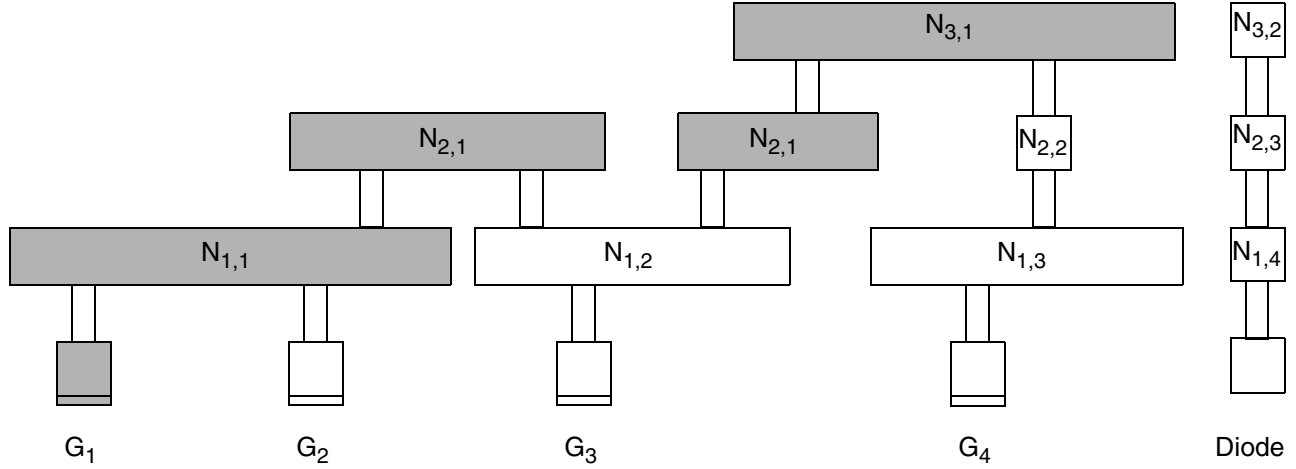
Gates  $G_1$  and  $G_2$  have the same history with regard to PAE because they are connected to the same piece of *metal1*, so they have the same CAR for any node on a specified layer:

$$CAR(N_{2,1}, G_1) = CAR(N_{2,1}, G_2)$$

**Calculations for CAR on the Third Metal Layer**

Figure C-11 on page 405 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,

$$\text{CAR}(N_{3,1}, G_2) = \text{PAR}(N_{1,1}, G_2) + \text{PAR}(N_{2,1}, G_2) + \text{PAR}(N_{3,1}, G_2)$$

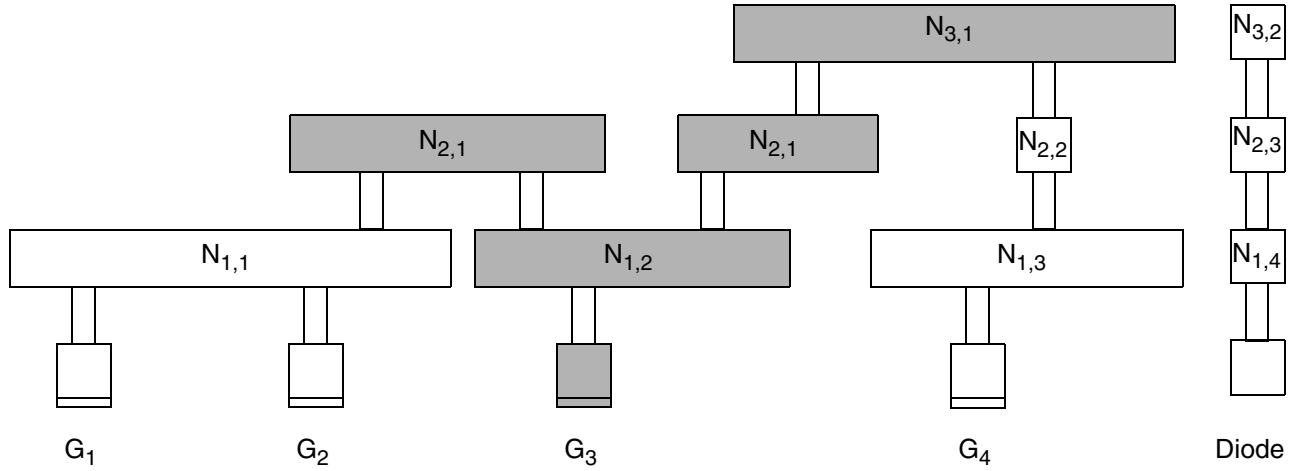
$\text{CAR}(N_{3,1}, G_1)$  equals  $\text{CAR}(N_{3,1}, G_2)$  because gates G<sub>1</sub> and G<sub>2</sub> are both electrically connected to the same node, N<sub>1,1</sub>, on *metal1* and therefore have the same history with regard to PAE. Therefore, the formula for  $\text{CAR}(N_{3,2}, G_2)$  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 406, the relevant areas for calculating CAR for gate G<sub>3</sub> are shaded.

**Figure C-12**

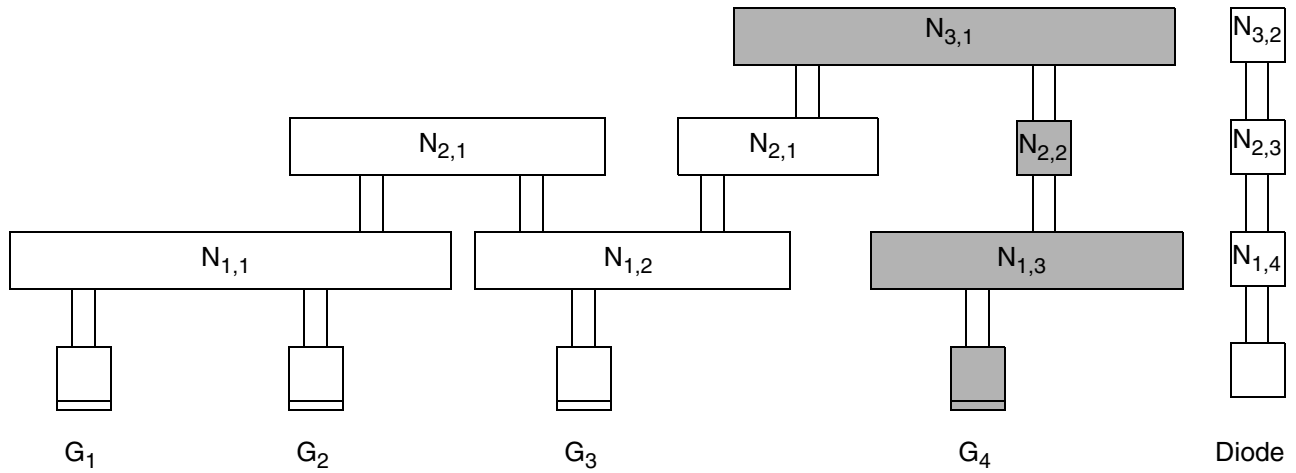


In the figure above,

$$\text{CAR}(N_{3,1}, G_3) = \text{PAR}(N_{1,2}, G_3) + \text{PAR}(N_{2,1}, G_3) + \text{PAR}(N_{3,1}, G_3)$$

In [Figure C-13](#) on page 406, 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

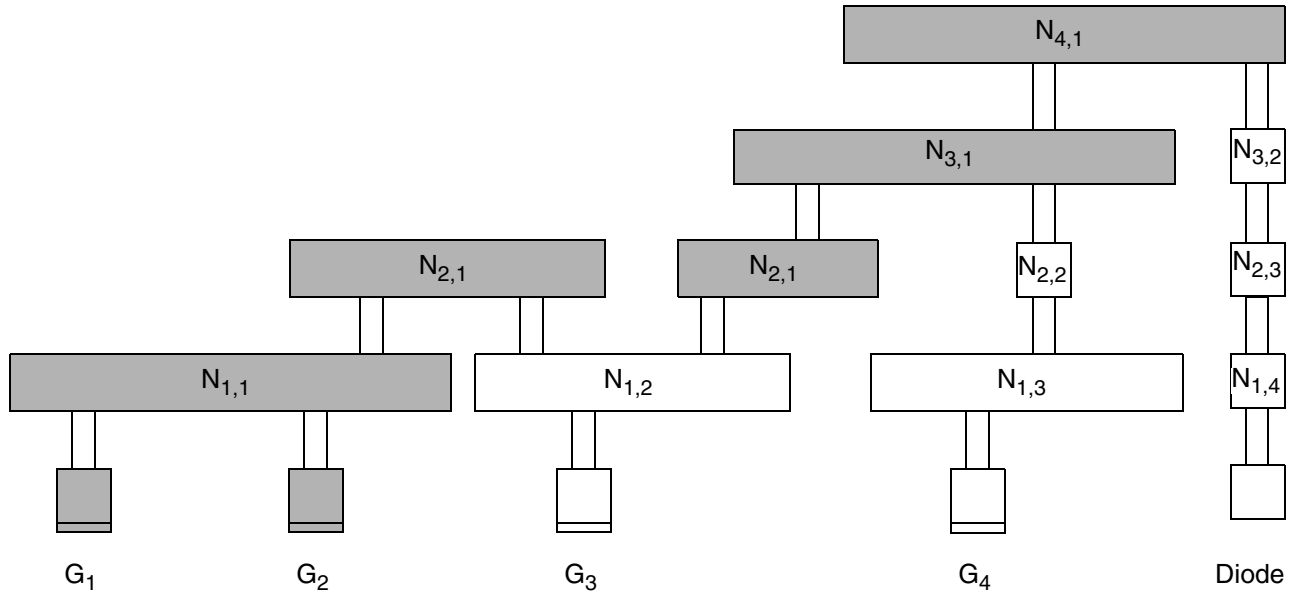
The following figure shows the chip after the fourth metal layer is processed.

**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 407, 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,

$$CAR(N_{4,1}, G_1) = PAR(N_{1,1}, G_1) + PAR(N_{2,1}, G_1) \\ + PAR(N_{3,1}, G_1) + PAR(N_{4,1}, G_1)$$

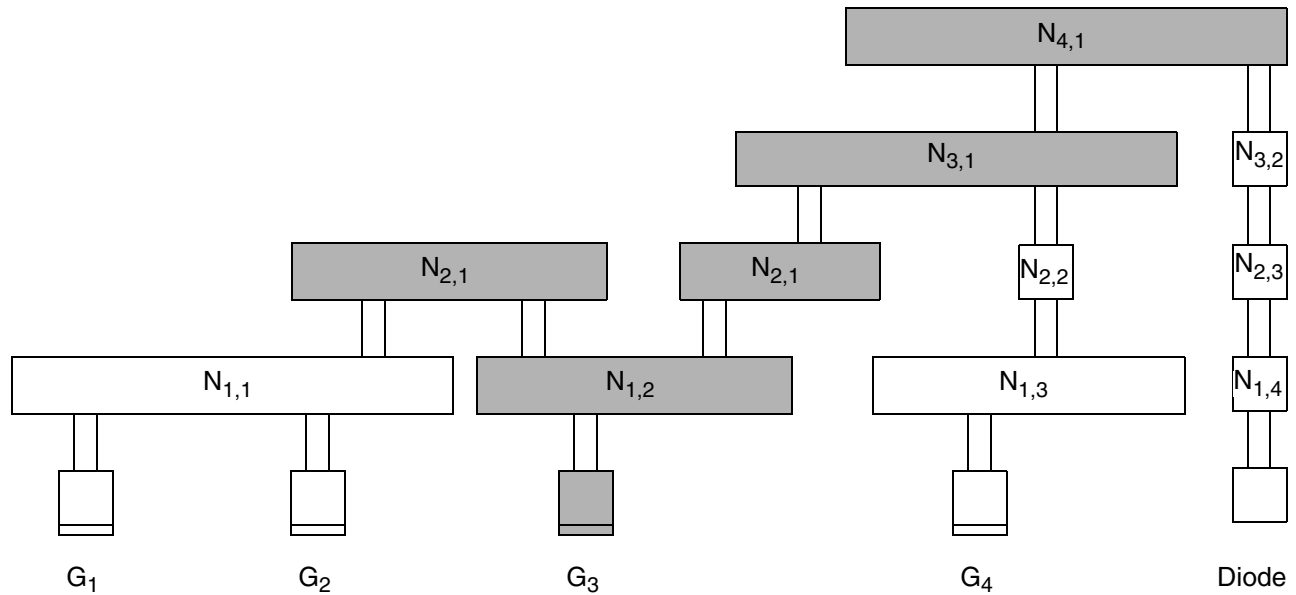
$$CAR(N_{4,1}, G_2) = PAR(N_{1,1}, G_2) + PAR(N_{2,1}, G_2) \\ + PAR(N_{3,1}, G_2) + PAR(N_{4,1}, G_2)$$

$$CAR(N_{4,1}, G_1) = CAR(N_{4,1}, G_2)$$

### Gate $G_3$

In [Figure C-15](#) on page 408, the relevant areas for calculating  $CAR(N_{4,1}, G_3)$  are shaded.

**Figure C-15**



In the figure above,

$$CAR(N_{4,1}, G_3) = PAR(N_{1,2}, G_3) + PAR(N_{2,1}, G_3) \\ + PAR(N_{3,1}, G_3) + PAR(N_{4,1}, G_3)$$

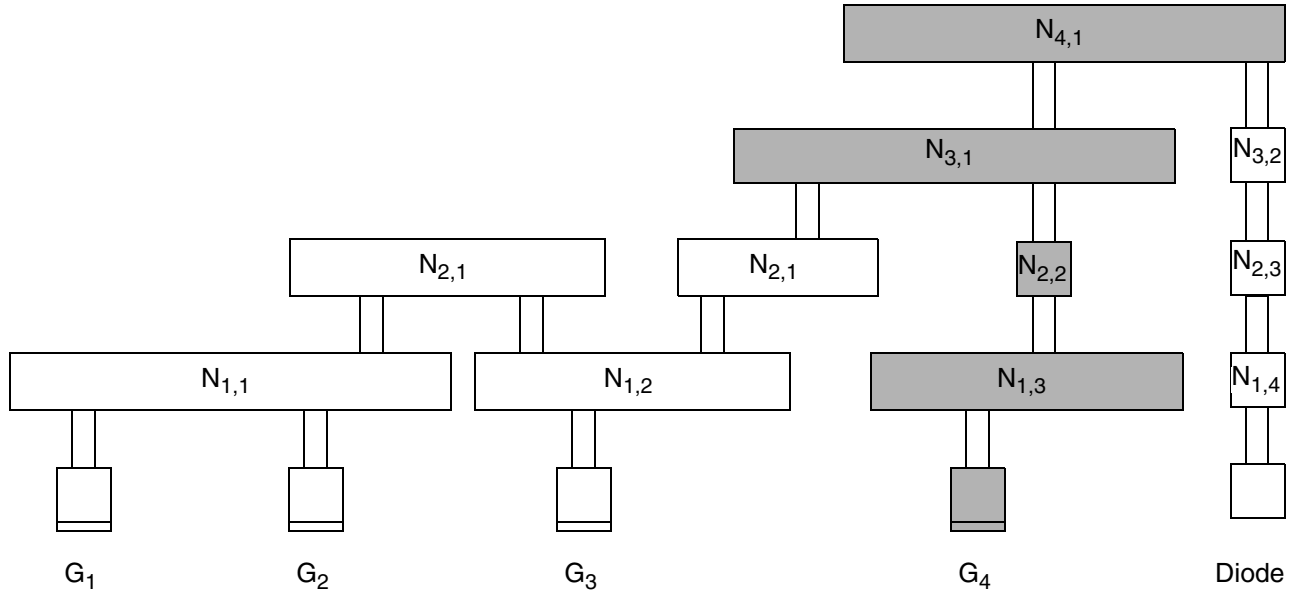
$CAR(N_{4,1}, G_3)$  does not equal  $CAR(N_{4,1}, G_1)$  or  $CAR(N_{4,1}, G_2)$  because it is not connected to the same node on *metal1*.

### Gate $G_4$

In [Figure C-16](#) on page 409, 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

---

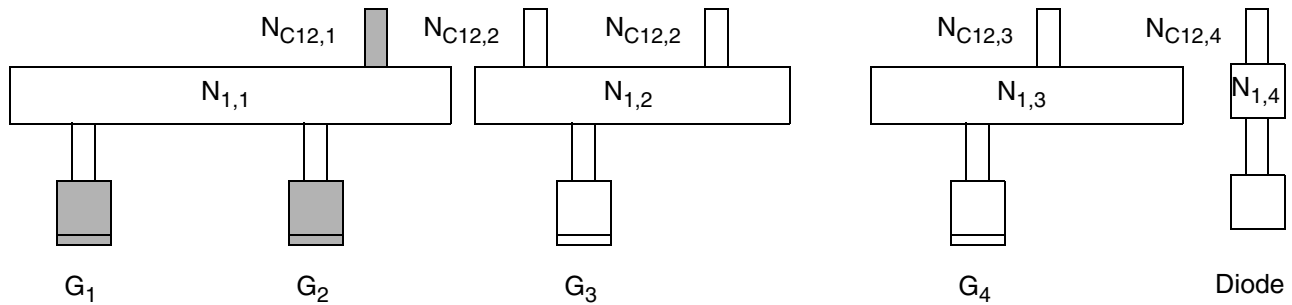
*minusDiffFactor*, and the `ANTENNAGATEPLUSDIFF` statement is shown as *plusDiffFactor*. For cut layer, the ratio equation illustrates the effect of an `ANTENNAAREAFAC` *cutFactor* statement as *metalFactor*. If there is no preceding `ANTENNAAREAFAC` statement, the *metalFactor* value defaults to 1.0.

In the figures and text that follow,

- $C_{ij}$  is the cut layer between  $metal_i$  and  $metal_j$ .
- $N_{Cij,k}$  specifies an electrically connected node on  $C_{ij}$ .
- The nodes are numbered sequentially, from left to right.

Figure C-17 on page 410 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](#)”:

$$CAR(c_i) = PAR(c_i) + CAR(c_{i-1})$$

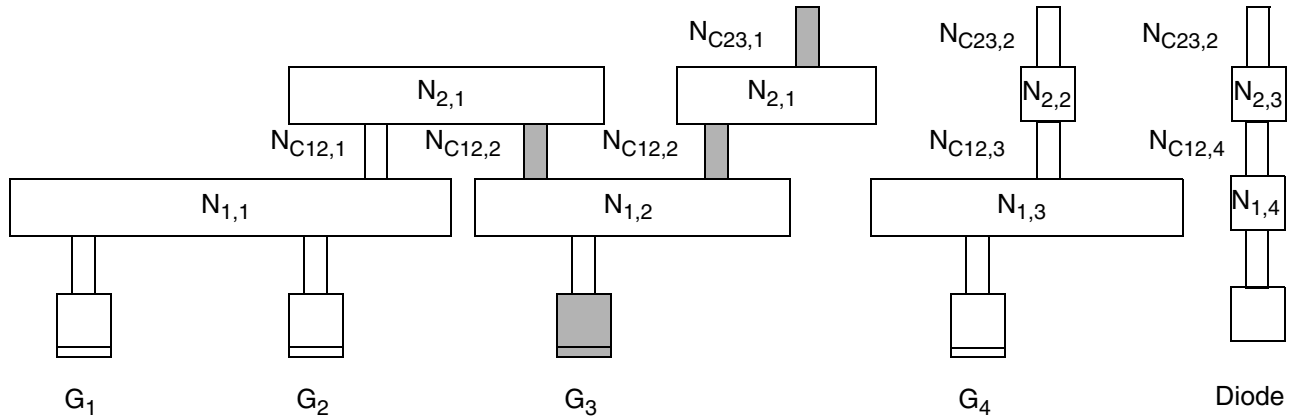
To create a single accumulative model that combines both metal and cut damage into one model, specify the `ANTENNACUMROUTINGPLUSCUT` statement for the layer, so that:

$$CAR(c_i) = PAR(c_i) + CAR(m_{i-1})$$

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 411 shows the chip after the C23 process step.

**Figure C-18**

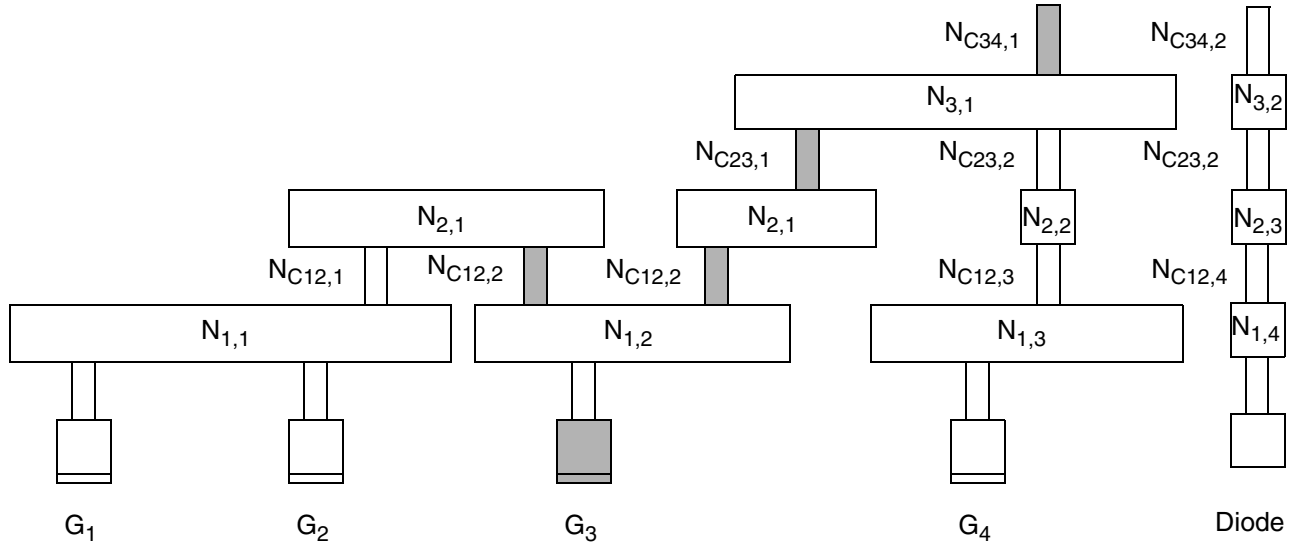


The router calculates the CAR with respect to gate  $G_3$  after the cut C23 process step as follows:

$$CAR(N_{C23,1}, G_3) = \frac{\text{Area}(N_{C12,2})}{\text{Area}(G_3)} + \frac{\text{Area}(N_{C23,1})}{\text{Area}(G_1) + \text{Area}(G_2) + \text{Area}(G_3)}$$

Figure C-19 on page 412 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 `ANTENNAAREAFactor` (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:

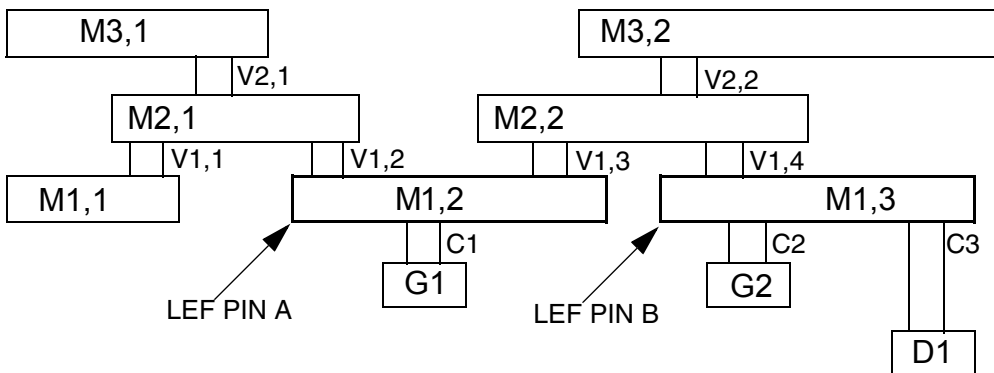
```
ANTENNAAREADIFFREDUCEPWL ( ( 0.0 1.0 ) ( 0.0999 1.0 ) ( 0.1 0.2 )
    ( 1000.0 0.2 ) ) ;
ANTENNACUMDIFFAREARATIO 10 ;
```

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 416. 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 ;  
ANTENNACUMDIFFFAREARATIO 1000 ;
```

Every cut layer should include the following information:

```
ANTENNAAREAFACOR 10 ; #10.0 x cut area  
ANTENNACUMROUTINGPLUSCUT ;  
ANTENNAAREAMINUDIFF 100.0 ;  
ANTENNACUMDIFFFAREARATIO 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  $PAR(M3, G1) = 5 \times 0.1555 = 0.775$
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 ;
    ANTENNADIFFAREA .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 antenna1
    CLASS CORE ANTENNACELL ;
    ...
    PIN ANT1
        AntennaDiffArea 1.0 ;
        PORT
            LAYER metall ;
            RECT 0.190 2.380 0.470 2.660 ;
        END
    END ANT1
END antenna1
```

## 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

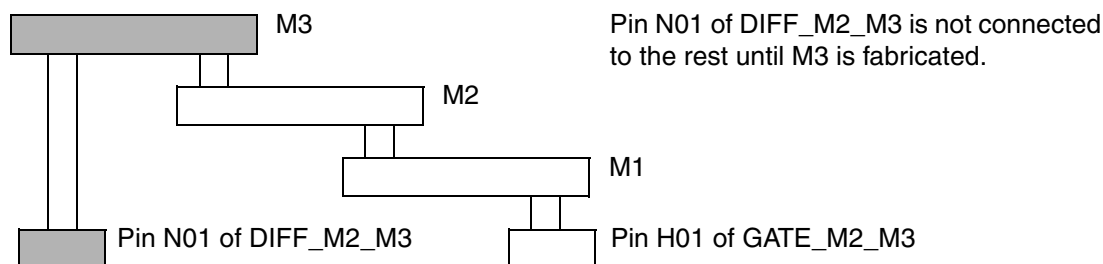
### Calculating and Fixing Process Antenna Violations

---

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

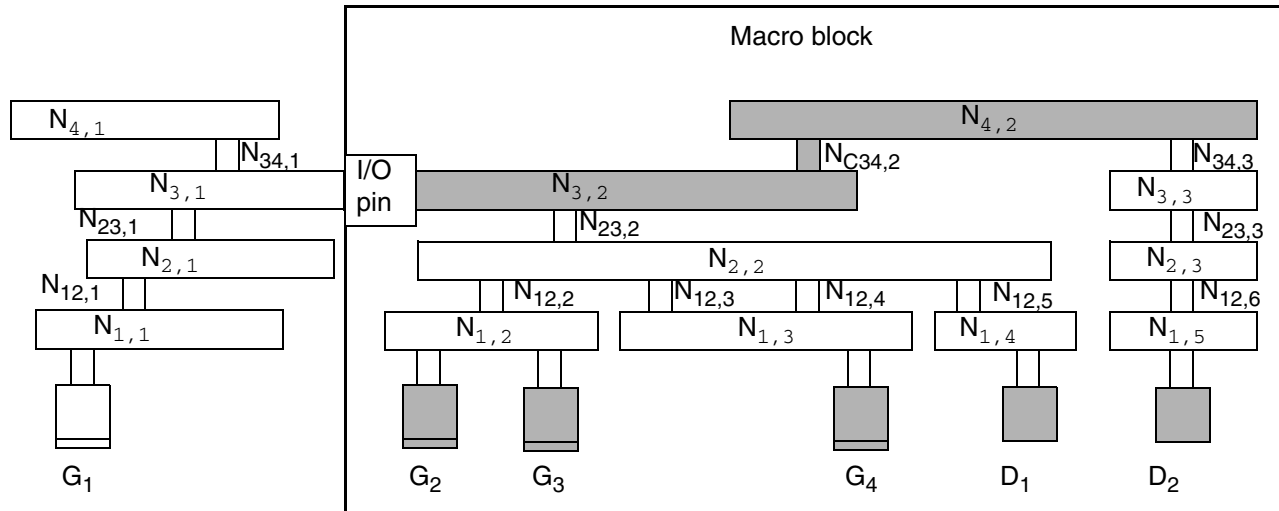
If the keyword ends with ...	It refers to ...	Examples
area sideArea	Drawn area or side area of the metal wires. Measured in square microns.	ANTENNAPARTIALCUTAREA ANTENNAPARTIALMETALAREA ANTENNAPARTIALMETALSIDEAREA ANTENNAPINDIFFAREA ANTENNAPINGATEAREA ANTENNAPINPARTIALCUTAREA
CAR	Relationship the router is calculating  CAR is used in keywords for cumulative antenna ratio.	ANTENNAMAXAREACAR ANTENNAMAXCUTCAR ANTENNAMAXSIDEAREACAR ANTENNAPINMAXAREACAR ANTENNAPINMAXCUTCAR ANTENNAPINMAXSIDEAREACAR

## Design Example

Figure C-22 on page 427 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 427, the keywords and calculations for *metal3* and *via34* would be:

```
ANTENNAMAXAREACAR  $\left( \frac{N_{1,3}}{G_4} + \frac{N_{2,2}}{G_2 + G_3 + G_4} + \frac{N_{3,2}}{G_2 + G_3 + G_4} \right)$  LAYER Metal3;
ANTENNAMAXSIDEAREACAR  $\left( \frac{N_{1,3}}{G_4} + \frac{N_{2,2}}{G_2 + G_3 + G_4} + \frac{N_{3,1}}{G_2 + G_3 + G_4} \right)$  LAYER Metal3;
ANTENNAMAXCUTCAR  $\left( \frac{N_{12,3} + N_{12,4}}{G_4} + \frac{N_{23,2}}{G_2 + G_3 + G_4} + \frac{N_{34,2}}{G_2 + G_3 + G_4} \right)$  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 427, 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 427 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 [147](#)  
 alias  
   expansion in DEF and LEF [217](#)  
   names in DEF and LEF [216](#)  
   statements in LEF and DEF [215](#)

## B

blockages, simplified [138](#)  
 braces in syntax [8](#)  
 brackets in syntax [8](#)  
 BUSBITCHARS statement  
   description, DEF [236](#)

## C

capacitance  
   peripheral [62](#)  
   wire-to-ground [59](#)  
 cell modeling  
   combining blockages [138](#)  
 characters  
   escape [228](#)  
   information [227](#)  
 COMPONENTS statement  
   description, DEF [237](#)  
 conventions  
   user-defined arguments [7](#), [8](#)  
   user-entered text [7](#)  
 COVER model, definition in LEF [114](#)

## D

database  
   converting LEF values to integer  
     values [159](#)  
 debugging  
   DEF files [220](#)  
   LEF libraries [220](#)  
   parametric macros [220](#)  
 DEF  
   example [364](#)  
   syntax overview [226](#)  
 DEF syntax  
   PINS [279](#)  
 DEF syntax and description  
   BUSBITCHARS [236](#)  
   COMPONENTS [237](#)  
   DESIGN [244](#)  
   DIEAREA [245](#)  
   DIVIDERCHAR [13](#), [245](#)  
   GCELLGRID [250](#)  
   GROUPS [252](#)  
   HISTORY [253](#)  
   NETS [253](#)  
   PINPROPERTIES [296](#)  
   PROPERTYDEFINITIONS [296](#)  
   REGIONS [298](#)  
   ROW [299](#)  
   SPECIALNETS [307](#)  
   TECHNOLOGY [336](#)  
   TRACKS [337](#)  
   UNITS DISTANCE MICRONS [338](#)  
   VERSION [339](#)  
   VIAS [339](#)  
 DESIGN statement  
   description, DEF [244](#)  
 diagonal vias, recommendation for  
   RGrid [111](#)  
 DIEAREA statement  
   description, DEF [245](#)  
 DIVIDERCHAR statement  
   description, DEF [13](#), [245](#)

## E

edge capacitance [62](#)  
 EEQ statement, LEF syntax [121](#)  
 electrically equivalent models, LEF  
     syntax [121](#)  
 endcap models, definition in LEF [116](#)  
 equivalent models  
     electrically equivalent (EEQ) [121](#)  
 error checking, utilities [220](#)  
 escape character [228](#)

## F

feedthrough pins  
     LEF [147](#)  
 FOREIGN references  
     LEF syntax [121](#)  
     offset between LEF and GDSII [121](#)

## G

GCell grid  
     restrictions [251](#)  
     uniform, in DEF [251](#)  
 GCELLGRID statement  
     description, DEF [250](#)  
 GROUPS statement  
     description, DEF [252](#)

## H

HISTORY statement  
     description, DEF [253](#)

## I

INOUT pins, netlist [140](#)  
 INPUT DEF command  
     error checking [220](#)  
 INPUT GDSII command  
     with incremental LEF [219](#)  
 INPUT LEF command  
     error checking [220](#)  
     incremental capability [219](#)  
 INPUT pins, netlist [140](#)

italics in syntax [7](#)

## L

LAYER (nonrouting) statement  
     description, LEF [15](#), [45](#)  
 layers  
     for LEF via descriptions [163](#)  
     routing order in LEF [52](#)  
 LEF  
     example [353](#)  
     files  
         distance precision [10](#)  
         line length [10](#)  
         overview [10](#)  
         routing layer order [52](#)  
 LEF syntax  
     overview [12](#)  
 LEF syntax and description  
     LAYER, nonrouting [15](#), [45](#)  
     MACRO [114](#)  
     NONDEFAULT rule [149](#)  
     OBS, macro obstruction [136](#)  
     PIN macro [139](#)  
     PROPERTYDEFINITIONS [154](#)  
     SITE [155](#)  
     UNITS [158](#)  
     VERSION [161](#)  
     VIA [162](#)  
     VIARULE [173](#)  
     VIARULE viaRuleName  
         GENERATE [175](#)  
 LEF values converted to integer  
     values [159](#)  
 legal characters [227](#)  
 library design, simplifying blockages [138](#)  
 literal characters [7](#)

## M

macro obstruction, OBS statement  
     description, LEF [130](#), [136](#)  
 macro PIN statement  
     description, LEF [139](#)  
 MACRO statement  
     description, LEF [114](#)  
 models, site orientation [125](#)  
 mustjoin pins [143](#)



### N

netlist pins  
     INOUT [140](#)  
     INPUT [140](#)  
     OUTPUT [140](#)  
nets  
     mustJoin nets [255](#)  
NETS statement  
     description, DEF [253](#)  
NONDEFAULT rule statement  
     description, LEF [149](#)  
     syntax, DEF [279](#)  
PITCH parameter, ratio in three-layer design [111](#)  
placement site function, SITE statement in LEF [155](#)  
ports  
     in LEF [145](#)  
     multiple pins [145](#)  
power pin  
     geometries in LEF [147](#)  
PROPERTYDEFINITIONS statement  
     description, DEF [296](#)  
     description, LEF [154](#)

### O

OBS (macro obstruction) statement  
     description, LEF [130](#), [136](#)  
obstructions, simplified [138](#)  
Or-bars in syntax [8](#)  
orientation  
     models [125](#)  
     pin [244](#), [275](#), [294](#)  
OUTPUT pins, netlist [140](#)  
overlaps, specifying in LEF [139](#)

### P

peripheral capacitance [62](#)  
PIN (macro) statement  
     description, LEF [139](#)  
PINPROPERTIES statement  
     description, DEF [296](#)  
pins  
     abutment [147](#)  
     direction in LEF [143](#)  
     external, DEF [280](#)  
     feedthrough pins in LEF [147](#)  
     INOUT [140](#)  
     INPUT [140](#)  
     modeling in LEF [140](#)  
     mustjoin [143](#)  
     netlist [140](#)  
     orientation [244](#), [275](#), [294](#)  
     OUTPUT [140](#)  
     power geometries [147](#)  
     ring [147](#)  
     using in LEF [148](#)  
PINS statement

### R

REGIONS statement  
     description, DEF [298](#)  
regular wiring  
     orthogonal paths [269](#), [322](#)  
RGrid, description [111](#)  
ring pins [147](#)  
routing time, diagonal vias [111](#)  
routing width, LEF syntax [100](#)  
ROW statement  
     description, DEF [299](#)

### S

scan chains  
     example [369](#)  
     rules [305](#)  
SI units in LEF [159](#)  
SITE statement  
     description, LEF [155](#)  
sites  
     symmetry [157](#)  
special wiring  
     description [312](#)  
     pins and wiring, DEF [323](#)  
SPECIALNETS statement  
     description, DEF [307](#)  
syntax conventions [7](#)

### T

TECHNOLOGY statement  
     description, DEF [336](#)

three-layer design, pitch ratio [111](#)  
TRACKS statement  
    description, DEF [337](#)

## U

UNITS DISTANCE MICRONS statement  
    description, DEF [338](#)  
UNITS statement  
    description, LEF [158](#)

## V

values in library database [159](#)  
VERSION statement  
    description, DEF [339](#)  
vertical bars in syntax [8](#)  
VIA statement  
    description, LEF [162](#)  
VIARULE statement  
    description, LEF [173](#)  
VIARULE viaRuleName GENERATE  
    statement  
    description, LEF [175](#)  
vias  
    default vias in LEF [163](#)  
    layers for vias in LEF [163](#)  
VIAS statement  
    description, DEF [339](#)

## W

wide wire signal wire, specifying [149](#)  
wiring, regular  
    orthogonal paths [269](#), [322](#)  
wiring, special  
    description [312](#)  
    pins and wiring [323](#)