# SPECCTRA® Design Language Reference

## Product Version 10.0
## May 2000



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

# Contents

# About This Manual

This manual describes the design language that defines printed circuit board (PCB) designs for SPECCTRA®.

## Audience

This manual is written for software programmers who develop software to translate design data between SPECCTRA and a PCB layout system.

## Where to Find Information in This Manual

The SPECCTRA Design Language Reference contains the following chapters:

Chapter 1, "Design Language Syntax," provides an alphabetical design language reference.

Chapter 2, "Sample Files," provides samples of SPECCTRA design files.

## Conventions Used in This Manual

The following fonts, characters, and styles have specific meanings throughout this manual.

- Boldface type identifies text that you type exactly as shown, such as SPECCTRA command names, keywords, and other syntax elements. In the following example, connect, on, and off are keywords.

  ```
  (average_pair_length [on | off])
  ```

  Syntax or command examples that appear on a separate line are not bold.

  ```
  (boundary (rect pcb 0 0 9000 4000))
  ```

  The Backus-Naur Form (BNF) metalanguage conventions used to represent the design language syntax is explained at the beginning of Chapter 1.

- Italic type identifies titles of books and emphasizes portions of text.

  For installation information, see the *SPECCTRA Installation and Configuration Guide*.

Italicized words enclosed in angle brackets (<>) are placeholders for keywords, values, filenames, or other information that you must supply.

<*directory_path_name*>::= <*id*>

- References to keys on your keyboard and mouse buttons are enclosed in brackets. [Shift] refers to the shift key. The carriage return key is labeled "Enter" on some keyboards and "Return" on others. This manual uses [Enter].

  Mouse buttons are identified by two uppercase letters enclosed in brackets.
  [LB] left button
  [MB] middle button
  [RB] right button

  If you have a 2-button mouse, press [ALT] and [RB] simultaneously when you see [MB].

## Special Terms

The following special terms are used in this manual.

- The word *enter* used with commands means type the command and press [Enter].

  “Enter the command **grid wire 1**” means

  1. Type **grid wire 1**.
  2. Press [Enter].

- *Click* means press and release the left mouse button.
- *Click-middle* means press and release the middle mouse button.
- *Click-right* means press and release the right mouse button.
- *Drag* means press and hold the left mouse button while you move the pointer.
- *Drag* [MB] means press and hold the middle mouse button while you move the pointer.
- *Double-click* means press and release the left mouse button twice in rapid succession.
- *Click twice* means click twice at the same location in the SPECCTRA work area.
- *Select* means to identity objects (such as wires, nets, or components) for exclusive processing by routing or placement commands. When you *select* objects before using a command, SPECCTRA works only on the objects that are selected.
- *Switch* refers to one or more characters you can use with an operating system command, such as the command you use to start SPECCTRA. A hyphen (-) precedes each command line switch.

## Where to Find Additional Information

For information about installing and configuring SPECCTRA software and licenses, see the *SPECCTRA Installation and Configuration Guide* (instcnfg.pdf in the help directory of the installation tree).

For an overview of new features and enhancements, known problems and solutions, and documentation addenda in SPECCTRA, see the “SPECCTRA 9.0 Release Notes” (specctra.html in the README directory of the installation tree).

For information about starting SPECCTRA, command syntax, menus and dialog menus, and design methodology, see the SPECCTRA online help (specctra.hlp in the help directory of the installation tree). Use WinHelp for Windows 95/98 and Windows NT, or HyperHelp™ for UNIX. You can also access the online help system from the SPECCTRA Help menu.

## Your Comments About This Manual

We are interested in your comments and opinions about this manual. If you comment, please consider the following questions:

- Is the information organized logically? If your answer is no, how could we better organize the information?
- Did you find any inaccuracies or omissions? If your answer is yes, what are the inacurracies or omissions?
- What suggestions do you have for improving this manual?

Please send your comments by fax to (408) 342-5647, or via the Internet by email to **cct_pubs@cadence.com**. Remember to include the document title with your comments.

## How to Contact Technical Support

If you have questions about installing or using SPECCTRA, contact the Cadence Customer Response Center at

http://sourcelink.cadence.com/supportcontacts.html

# Design Language Syntax

## Chapter content



## Overview

This chapter defines the syntax and semantics to represent a printed circuit board (PCB) in a design file or session file. Design prototypes at the beginning of the chapter show how a design is represented at the highest level. The remainder of the chapter lists descriptors in alphabetical order, fully expands them, and describes their functions.

A design file contains all the data, or a portion of the data with pointers to other files that contain additional data, to define a PCB. The design file is a text file.

## Syntax Conventions

Design language syntax consists of keywords and descriptors. Keywords are usually followed by one or more descriptors. Keywords and descriptors are sometimes enclosed within parentheses.

Keywords and parentheses must appear in a design or session file exactly as shown. Descriptors are alphabetic, numeric, or alphanumeric character strings, such as identifiers, values, filenames, or additional syntax. Angle brackets <  > enclose all descriptors.

The Backus-Naur Form (BNF) metalanguage conventions are used to expand descriptors, and to show whether they are optional or exclusive and whether they can be repeated.

The ::= symbol indicates that an expanded definition follows. This symbol can be interpreted to mean: *is defined as*.

Square brackets [ ] enclose a set. When a set contains only one keyword or descriptor, the set is optional. For example:

[a]—Can include a.

When a set contains alternatives, the keywords or descriptors are separated by a vertical bar ( | ). For example:

[a **|** b **|** c]—Must include either a or b or c.

[a **|** b **|** c **|** null]—Can include either a or b or c.

If the word *null* appears within the brackets, the set is optional. Any member of the set other than null can be used. Null is only a symbol to indicate that all the enclosed keywords or descriptors are optional.

Braces { } indicate that the enclosed set can occur one or more times.

# Top Level Design File Prototype

The following design file prototype lists the syntax at the highest level in the order each construct must appear in a design file. Five sections that must be included in every design file are pcb, structure, placement, library, and network. All other sections are optional.

File descriptors can substitute for all or part of the structure, placement, library, floor_plan, or network section descriptors. Each file descriptor must point to a file that only contains descriptors appropriate to that section.

### *<design_descriptor>::=*

(**pcb** <*pcb_id*>
[<*parser_descriptor*>]
[<*capacitance_resolution_descriptor*>]
[<*conductance_resolution_descriptor*>]
[<*current_resolution_descriptor*>]
[<*inductance_resolution_descriptor*>]
[<*resistance_resolution_descriptor*>]
[<*resolution_descriptor*>]
[<*time_resolution_descriptor*>]
[<*voltage_resolution_descriptor*>]
[<*unit_descriptor*>]

[<*structure_descriptor*> **|** <*file_descriptor*>]
[<*placement_descriptor*> **|** <*file_descriptor*>]
[<*library_descriptor*> **|** <*file_descriptor*>]
[<*floor_plan_descriptor*> **|** <*file_descriptor*>]
[<*part_library_descriptor*> **|** <*file_descriptor*>]
[<*network_descriptor*> **|** <*file_descriptor*>]
[<*wiring_descriptor*>]
[<*color_descriptor*>]

)

# Second Level Design File Prototype

The next design file prototype expands the highest level keywords to include descriptors and keywords at the next level below.

(**pcb** <*pcb_id*>
(**parser**
[(**string_quote** <*quote_char*>)]
(**space_in_quoted_tokens** [**on | off**]
[(**host_cad** <*id*>)]
[(**host_version** <*id*>)]
[{(**constant** <*id*> <*id*>)}]
[(**write_resolution** {<*character*> <*positive_integer*})]
[(**routes_include** {[**testpoint | guides | image_conductor**]})]
[(**wires_include testpoint**)]
[(**case_sensitive** [**on | off**])]
[(**rotate_first** [**on | off**])]
)
(**resolution** <*dimension_unit*> <*positive_integer*>
)
(**unit** <*dimension_unit*>
)
(**structure**
[<*unit_descriptor*> **|** <*resolution_descriptor*> **|** null]
{<*layer_descriptor*>}
[<*layer_noise_weight_descriptor*>]
{<*boundary_descriptor*>}
{<*place_boundary_descriptor*>}
[{<*plane_descriptor*>}]
[{<*region_descriptor*>}]
[{<*keepout_descriptor*>}]
<*via_descriptor*>

```
    [<control_descriptor>]
    <rule_descriptor>
    [<structure_place_rule_descriptor>]
    {<grid_descriptor>}
)
(placement
    [<unit_descriptor> | <resolution_descriptor> | null]
    [<place_control_descriptor>]
    {<component_instance>}
)
(library
    [<unit_descriptor>]
    {<image_descriptor>}
    [{<jumper_descriptor>}]
    {<padstack_descriptor>}
    [<directory_descriptor>]
    [<extra_image_directory_descriptor>]
    [{<family_family_descriptor>}]
    [{<image_image_descriptor>}]
)
(floor_plan
    [<unit_descriptor>]
    [<resolution_descriptor>]
    [{<cluster_descriptor>}]
    [{<room_descriptor>}]
)
(part_library
    [{<physical_part_mapping_descriptor>}]
    {<logical_part_mapping_descriptor>}
    [{<logical_part_descriptor>}]
    [<directory_descriptor>]
)
(network
    {<net_descriptor>}
    [{<class_descriptor>}]
    [{<class_class_descriptor>}]
    [{<group_descriptor>}]
    [{<group_set_descriptor>}]
    [{<pair_descriptor>}]
    [{<bundle_descriptor>}]
)
(wiring
    [<unit_descriptor> | <resolution_descriptor> | null]
```

{<*wire_descriptor*>}
<*test_points_descriptor*>]
)
(**colors**
{(**color** <*color_number*> <*color_name*> <*R*> <*G*> <*B*>)}
{(**set_color** <*color_object*> <*color_name*>)}
{(**set_pattern** <*pattern_object*> <*pattern_name*>)}
)
)

# Routing and Placement Rule Hierarchies

Routing and placement rules can be defined at multiple levels of design specification. When a routing or placement rule is defined for an object at multiple levels, a predefined routing or placement precedence order automatically determines which rule to apply to the object.

The routing rule precedence order is

pcb < layer < class < class layer < group_set < group_set layer < net < net layer < group < group layer < fromto < fromto layer < class_class < class_class layer < padstack < region < class region < net region < class_class region

A pcb rule (global rule for the PCB design) has the lowest precedence in the hierarchy. A class-to-class region rule has the highest precedence. Rules set at one level of the hierarchy override conflicting rules set at lower levels.

The placement rule precedence order is

pcb < image_set < image < component < super cluster < room < room_image_set < family_family < image_image

A pcb rule (global rule for the PCB design) has the lowest precedence in the hierarchy. An image-to-image rule has the highest precedence. Rules set at one level of the hierarchy override conflicting rules set at lower levels.

# The Syntax

This section lists the complete design language syntax in alphabetical order.

## *<ancestor_file_descriptor>*

<*ancestor_file_descriptor*>::=
(**ancestor** *<file_path_name>* (**created_time** *<time_stamp>*)
[(**comment** *<comment_string>*)])

The <*ancestor_file_descriptor*> is included in a session file to identify previous session files on which the current session might depend. The <*file_path_name*> is the directory path and filename for a previous session file. The **comment** block is optional.

## *<aperture_width>*

<*aperture_width*>::= <*positive_dimension*>

## *<attach_descriptor>*

<*attach_descriptor*>::=
(**attach** [**off**| **on** [(**use_via** <*via_id*>)]])

The **attach** control determines whether a via padstack can be positioned under an SMD pad. The default is **on**, which allows vias under SMD pads. The **via_at_smd** rule must also be turned **on** to place vias under SMD pads (the default **via_at_smd** rule is **off**).

The **use_via** control specifies which via padstack is used when a via is located under an SMD pad. The **use_via** control applies only when the **via_at_smd** rule is turned on. The <*via_id*> must be a padstack that is defined in the design file.

## *<begin_index>*

<*begin_index*>::= <*positive_integer*>

## *<bond_shape_descriptor>*

<*bond_shape_descriptor*>::=
(**bond**
*<pin_reference>*
<*padstack_id*>

*<vertex>*
[**front | back**]
*<bond_shape_rotation>*)

## *<bond_shape_rotation>*

*<bond_shape_rotation>*::= *<positive_integer>*

## *<boundary_descriptor>*

*<boundary_descriptor>*::=
(**boundary**
[{*<path_descriptor>*} |
*<rectangle_descriptor>*]
[*<rule_descriptor>*])

Use *<boundary_descriptor>* to define the PCB boundary for all features of the printed circuit board and the signal boundary for routing. A boundary is considered as an area object type.

The *<boundary_descriptor>* must form a closed loop. The boundary automatically closes, if the last *<vertex>* in a *<path_descriptor>* is not the same as the first *<vertex>*.

Every design must have a PCB boundary that defines the absolute bounding box for storing shapes. For example:

(boundary (rect pcb -18900.00 9800.00 -3351.00 17496.00))

The PCB boundary must be equal to or larger than the signal boundary. Only the *<rectangle_descriptor>* should be used when the **pcb** reserved layer name is used as the *<layer_id>* in the *<path_descriptor>*.

The area inside the signal boundary is available for routing. A signal keepin boundary can be defined as follows:

(boundary (path signal 0.00 -18400.00 10300.00 -3851.00 10300.00 -3851.00 16996.00 -7851.00 16996.00 -7851.00 16317.00 -18400.00 16317.00 -18400.00 10300.00))

When *<rectangle_descriptor>* is used to describe a boundary, it is not considered a filled shape.

A signal type of boundary cannot contain arcs.

## *<bundle_descriptor>*

<*bundle_descriptor*>::=
(**bundle** <*bundle_id*>
(**nets** {<*net_id*>})
{[(**gap** [<*positive_dimension*> **| -1**] {[(**layer** {<*layer_id*>})]})]})

The <*bundle_descriptor*> defines named bundled nets. Bundled nets are two or more nets that you want to route side by side with the same topology for each connection.

Use **nets** to identify the nets you want included in a bundle.

Use **gap** to control the minimum distance (<*positive_dimension*>) allowed between each routed wire in the bundle. If **gap** is not included in a <*bundle_descriptor*>, the wire-to-wire clearance rule is used. To reset a specified **gap** to the default wire-to-wire clearance, use **-1** for the **gap** value. The **gap** can apply to one or more layers, and multiple gaps can be specified.

## *<bundle_id>*

<*bundle_id*>::= <*id*>

## *<capacitance_resolution_descriptor>*

<*capacitance_resolution_descriptor*>::=
(**capacitance_resolution** [**farad | mfarad | ufarad | nfarad | pfarad | ffarad**
<*positive_integer*> )

| Symbol | Capacitance Unit |
|---|---|
| farad | farad |
| mfarad | millifarad |
| ufarad | microfarad |
| nfarad | nanofarad |
| pfarad | picofarad |
| ffarad | femtofarad |

The default capacitance unit is **nfarad** with a <*positive_integer*> equal to 1000.

## *<character>*

*<character>*::= [*<letter>* | *<digit>* | *<special_character>*]

## *<checking_trim_descriptor>*

*<checking_trim_descriptor>*::=
(**checking_trim_by_pin** [**on** | **off**])

The **checking_trim_by_pin** control defaults to on. When a shape terminates in a pin (or SMD), the checker automatically trims the end point to the edge of the pin. If **checking_trim_by_pin** is **off**, the automatic trimming of shapes is not performed.

## *<circle_descriptor>*

*<circle_descriptor>*::=
(**circle** *<layer_id>* *<diameter>* [*<vertex>*])

The default *<vertex>* is the PCB origin.

## *<circuit_descriptor>*

*<circuit_descriptor>*::= (**circuit** {*<circuit_descriptors>*})

## *<circuit_descriptors>*

*<circuit_descriptors>*::=
[*<delay_descriptor>* |
*<total_delay_descriptor>* |
*<length_descriptor>* |
*<total_length_descriptor>* |
*<match_fromto_length_descriptor>* |
*<match_fromto_delay_descriptor>* |
*<match_group_length_descriptor>* |
*<match_group_delay_descriptor>* |
*<match_net_length_descriptor>* |
*<match_net_delay_descriptor>* |
*<relative_delay_descriptor>* |
*<relative_group_delay_descriptor>* |
*<relative_group_length_descriptor>* |
*<relative_length_descriptor>* |
*<sample_window_descriptor>* |

*<switch_window_descriptor>* |
*<shield_descriptor>* |
*<max_restricted_layer_length_descriptor>* |
(**priority** <*positive_integer*>) |
(**use_layer** {*<layer_name>*}) |
(**use_via** {[<*padstack_id*> **|** (**use_array**
<*via_array_template_id*> [*<row>* *<column>*])]})]

The **priority** values range from 1 to 255. If you specify **-1**, priority is set to the default value of 10. A value of 0 is invalid. The highest priority is 255. Higher priority nets route first. Automatic placement also uses net priorities to determine the order in which components are placed and the proximity among components that share a priority net. Components with higher priority nets tend to be placed earlier and closer together during automatic placement.

If a **use_layer** rule applies to a net or class of nets, the net or class of nets are routed on the assigned layers even if the assigned layers are unselected for routing.

The **use_via** rule can include a via array specification, where <*row*> and <*column*> define the size of the array. (This requires **microvia on** in the *<control_descriptor>*.) The dimensions can be interchanged based on routing topology. In the following example, the via array identified as via2a is used instead of the default via array in nets A and B. In net B, the array size is specified as 1x4.

```
(net A
   (use_via (use_array via2a))
)
(net B
   (use_via (use_array via2a 1 4))
)
```

The <*circuit_descriptor*> is closely related to rule setting. Design rules, which can be set at multiple levels of a design, have a precedence hierarchy. For the order of routing rule precedence, see the Routing and Placement Rule Hierarchy section at the beginning of this manual.

Example of circuit syntax:

```
(network
   (class c1 sig1 sig2 sig3 (circuit (match_net_length on (tolerance 500))))
)
```

Manhattan lengths:

sig1 - 1500 mils
sig2 - 1750 mils
sig3 - 1600 mils

All nets in class c1 are matched to the routed length of sig2 within a tolerance of plus or minus 500 mils.

## *<class_descriptor>*

*<class_descriptor>*::=
(**class**
   *<class_id>* {[{*<net_id>*} **|** {*<composite_name_list>*}]}
   [*<circuit_descriptor>*]
   [*<rule_descriptor>*]
   [{*<layer_rule_descriptor>*}]
   [*<topology_descriptor>*]
)

For example:

```
(network
   (class C3 sig10 sig11 sig12 (layer_rule S1 S4
      (rule (width 0.020))) (layer_rule S2 S3 (rule (width 0.015))))
)
```

Nets sig10, sig11 and sig12 have a class_layer width rule of 0.020 on layers S1 and S4, and a class_layer width rule of 0.015 on layers S2 and S3.

## *<class_class_descriptor>*

*<class_class_descriptor>*::=
(**class_class**
   (**classes** *<class_id>* {*<class_id>*})
   {[*<rule_descriptor>* **|** *<layer_rule_descriptor>*]}
)

A class pair is formed for each possible combination of two class id's. The *<class_class_descriptor>* defines clearance rules, parallel noise and segment rules, and tandem noise and segment rules between wires in different classes and between wires in the same class. For example, a design file could include the following:

```
(network
   (class TTL tnet1 tnet2)
   (class ECL enet1 enet2)
   (class CLKS clk1 clk2 clk3)
   (class INTS in0 in1 in2 in3)
   (class SENSE sx sy sz)
   (class_class (classes TTL ECL) (rule
   (tandem_segment gap .01) (limit .1))))
   (class_class (classes CLKS INTS TTL SENSE)
      (rule (parallel_segment (gap .02)(limit .2))))
   (class_class (classes CLKS CLKS) (rule
      (parallel_segment (gap .015) (limit .15))))
```

In this sample design file:

- Five classes are defined: TTL, ECL, CLKS, INTS, and SENSE. A class_class tandem rule is defined between the TTL wires and the ECL wires.
- A class_class parallel rule is applied between the wires of six paired classes CLKS-to-INTS, CLKS-to-TTL, CLKS-to-SENSE, INTS-to-TTL, INTS-to-SENSE, and TTL-to-SENSE.
- A class_class parallel rule is applied between wires that belong to class CLKS. The rules applied with this construct apply only between wires of the CLKS class.

A class_class rule has higher precedence than a fromto rule. For routing rule precedence order, see the Routing and Placement Rule Hierarchies section at the beginning of this manual.

All via-to-object and wire-to-object clearance rules can be specified in <*class_class_descriptor*>.

## *<classes_descriptor>*

<*classes_descriptor*>::=
(**classes** {<*class_id*>})

When two or more classes are included in a <*classes_descriptor*>, each class is paired with every other class but not paired with itself. For example: (classes A B C) pairs AB, AC, and BC.

## *<class_id>*

<*class_id*>::= <*id*>

## *<clearance_descriptor>*

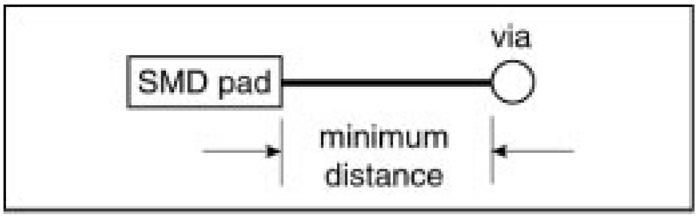
*<clearance_descriptor>*::=
(**clearance**
*<positive_dimension>*
[(**type** {*<clearance_type>*})]
)

## *<clearance_type>*

*<clearance_type>*::=
[*<object_type>*_*<object_type>* |
**smd_via_same_net** |
**via_via_same_net** |
**buried_via_gap** [(**layer_depth** *<positive_integer>*)] |
**antipad_gap** |
**pad_to_turn_gap** |
**smd_to_turn_gap**]

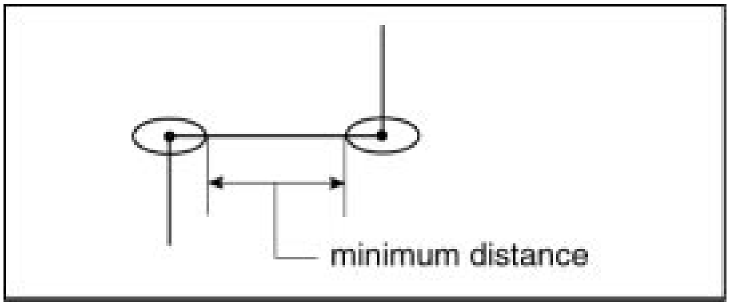
The **smd_via_same_net** clearance is the minimum clearance from the SMD pad to the first via, as shown in the following figure.

**Smd_via_same_net Clearance**




The **via_via_same_net** clearance is the minimum clearance between any two vias on the same net and the same layer.

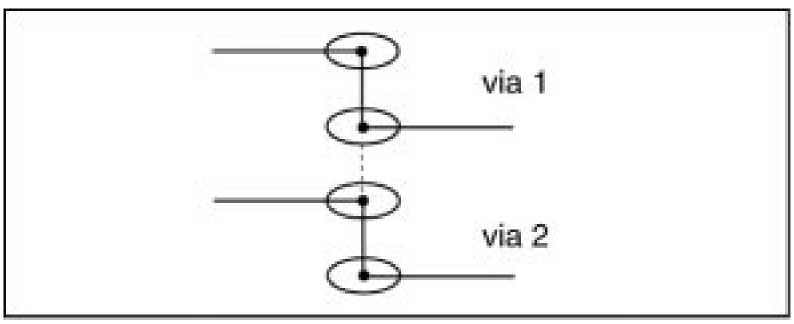
#### Via_via_same_net Clearance




The **buried_via_gap** is the gap between coincident vias for hybrid circuits. The following rules apply:

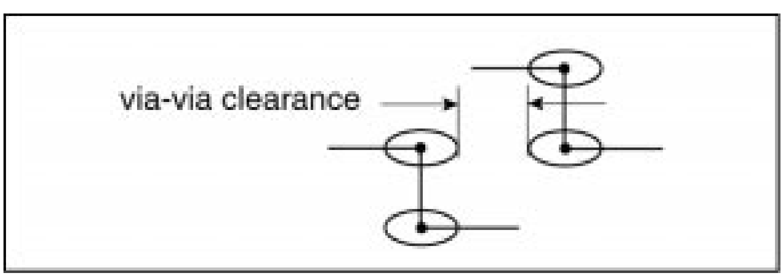
- A **buried_via_gap** can be defined in the design file by using the clearance rule. You can also define **buried_via_gap** in a do file, from the command line, and by using the GUI.
- A **buried_via_gap** can prevent coincident vias. If **buried_via_gap** is not specified (or is a negative number), coincident vias can occur as shown in the following figure.

#### Specifying a Buried_via_gap Prevents Coincident Vias



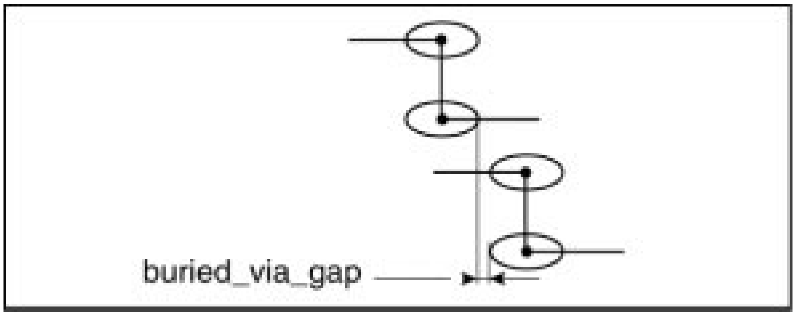

- A **buried_via_gap** has no effect on vias that have shapes on the same layer. The **via_via** clearance rule is used instead.

#### Via_via clearance Applies to Via Shapes on the Same Layer



If a **buried_via_gap** is used, this value defines the clearance between buried vias that do not share a common layer, as shown in the following figure. The **layer_depth** option specifies the number of layers over which the clearance rule applies. See also the **bbv_ctr2ctr** setting in the *<control_descriptor>*.

**Buried_via_gap Defines Clearance**




The **antipad_gap** clearance defines the gap between antipads for power nets to power nets and power nets to signal nets. Signal nets to signal nets are not checked for antipad gap clearance. The following apply:

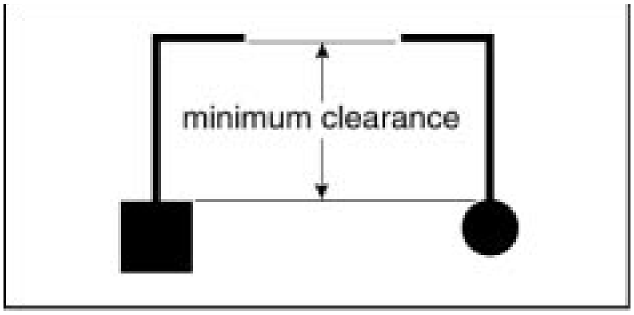
- An **antipad_gap** can be defined in the *<rule_descriptor>* within the *<structure_descriptor>* of the design file. The user can also define **antipad_gap** in a do file, from the command line, and by using the GUI.
- If the **antipad_gap** is not explicitly defined, there is no restriction on power layers and the **via_via** (or **pin_via**) rule checking does not involve the power layer and antipad sizes.
- If the **antipad_gap** is greater than or equal to 0, the value defines the clearance between antipads. The router considers the antipad shapes as circles or squares and symmetrical around the padstack origin.
- If the **antipad_gap** is not specified or is less than 0, no restriction is assumed.

The following example shows how to specify **antipad_gap**:

```
(pcb...
   (structure...
      (rule (clearance 0.2 (type antipad_gap)))
...))
```

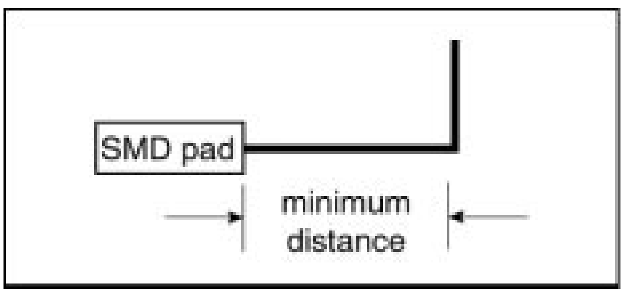
The **pad_to_turn_gap** clearance is the minimum clearance from any through-pin to the first 90 degree turn in the wire.

### Pad_to_turn_gap Clearance




The **smd_to_turn_gap** clearance is defined as the minimum clearance from any SMD pad to the first 90 degree turn in the wire.

### Smd_to_turn_gap Clearance




## *<cluster_descriptor>*

<*cluster_descriptor*>::=
(**cluster** <*cluster_id*>)
  (**comp** {<*component_id*>})
  [(**type**
    [**plan** | **super** [**piggyback** |
    (**super_placement** {*<super_place_reference>*}) | null] | **piggyback**])]
  [*<place_rule_descriptor>*]
)

The <*place_rule_descriptor*> applies only to **type super** or **type super piggyback** clusters. The default cluster type is **plan**.

## *<cluster_id>*

<*cluster_id*>::= <*id*>

## *<color_descriptor>*

*<color_descriptor>*::=
(**colors**
{(**color** *<color_number>* *<color_name>* *<R>* *<G>* *<B>*)}
{(**set_color** *<color_object>* *<color_name>*)}
{(**set_pattern** *<pattern_object>* *<pattern_name>*)}
)

*<R>*, *<G>*, *<B>*::= *<positive_integer>* in the range from 0 to 255.

Color definitions can be overwritten if more than one color has the same ID.

## *<color_name>*

*<color_name>*::= *<id>*

## *<color_number>*

*<color_number>*::= *<positive_integer>*

The range for *<color_number>* is from 0 to 99. The application default range is from 0 to 15.

## *<color_object>*

*<color_object>*::=
[**antipad | background | component back |**
**component front | error balance | error clearance |**
**error crosstalk | error length | error placement |**
**fix component | grid major | grid major place |**
**grid major route | grid place | grid via |**
**grid wiring | guide | highlight | histogram grid |**
**histogram peak | histogram segment | iroute target |**
**power** *<layer_number>* **| pin | protect | routability max | routability min |**
**ruler | select | signal** *<layer_number>* **| site | testpoint | viakeepouts | vias**]

The default colors are

```
(colors
   (color 0  background   170 210 255)
   (color 1  blue           0   0 255)
   (color 2  green          0 255   0)
   (color 3  violet       175   0 175)
```

```
(color 4  red      255   0   0)
(color 5  magenta  125   0 125)
(color 6  white    255 255 255)
(color 7  lgtgray  180 180 180)
(color 8  drkgray  120 120 120)
(color 9  drkblue    0   0 170)
(color 10 drkgreen   0 170   0)
(color 11 coral    255 127   0)
(color 12 drkred   170   0   0)
(color 13 lgtmgnta 255   0 255)
(color 14 yellow   255 255   0)
(color 15 black      0   0   0)
(set_color background black)
(set_color vias drkgreen)
(set_color viakeepouts drkgreen
(set_color pin darkgray)
(set_color select yellow)
(set_color antipad white)
(set_color guide drkgray)
(set_color highlight white)
(set_color histogram grid blue)
(set_color histogram segment red)
(set_color histogram peak red)(set_color grid via blue)
(set_color grid wiring white)
(set_color grid place drkblue)
(set_color grid major lightmagenta)
(set_color grid major place lightmagenta)
(set_color grid major route drkgreen)
(set_color error clearance yellow)
(set_color error length yellow)
(set_color error crosstalk yellow)
(set_color error balance white
(set_color error placement white)
(set_color routability min green)
(set_color routability max coral)
(set_color iroute target white)
(set_color testpoint yellow)
(set_color ruler drkgray)
(set_color site darkblue)
(set_color protect white)
(set_color component front red)
(set_color fix component lgtmgnta
(set_color signal 1 red)
```

```
    (set_color signal 2 drkblue)
    (set_color signal 3 drkred)
    (set_color signal 4 coral)
    (set_color signal 5 violet)
    (set_color signal 6 drkgreen)
    (set_color signal 7 lgtmgnta)
    (set_color signal 8 magenta)
    (set_color signal 9 red)
    (set_color signal 10 drkblue)
    (set_color signal 11 drkred)
    (set_color signal 12 coral)
    (set_color signal 13 violet)
    (set_color signal 14 drkgreen)
    (set_color signal 15 blue)
    (set_color component back blue)
    (set_color power 1 drkred )
    (set_color power 2 violet )
    (set_color power 3 coral )
    (set_color power 4 drkgreen )
    (set_color power 5 drkblue )
    (set_color power 6 red )
    (set_color power 7 magenta )
    (set_color power 8 lgtmgnta )
    (set_color power 9 drkred )
    (set_color power 10 violet )
    (set_color power 11 coral )
    (set_color power 12 drkgreen )
    (set_color power 13 drkblue )
    (set_color power 14 red )
    (set_color power 15 magenta)
)
```

Signal and power layer numbers are determined by the order in which they are defined in the design file. For example:

| Layer | Color | Layer Number |
|---|---|---|
| s1 | red | 1 (signal) |
| s2 | drkblue | 2 (signal) |
| p1 | drkgreen | 1 (power) |
| s3 | drkred | 3 (signal) |
| s4 | coral | 4 (signal) |
| p2 | violet | 2 (power) |
| s5 | violet | 5 (signal) |
| s6 | blue | 6 (signal) |

If the number of power or signal layers exceeds 15, the color is determined by the formula:

<*color_number*> = <*layer_number*> mod 15 + 1

SMD pins use the color of the layer (top or bottom) on which they are mounted.

The bottom signal layer is always the signal 15 color.

## *<column>*

<*column*>::= <*positive_integer*>

## *<command>*

<*command*>::=

Any command.

## *<command_group>*

<*command_group*>::=
*<command>* [{*<continuation_character>* *<command>*}]

## *<comment_string>*

<*comment_string*>::= *<string>*

## *<component_id>*

<*component_id*>::= <*id*>

The <*component_id*> is the same as the component reference designator.

## *<component_instance>*

<*component_instance*>::=
(**component** <*image_id*> {*<placement_reference>*})

## *<component_order_descriptor>*

<*component_order_descriptor*>::=
(**comp_order** {*<placement_id>*})

The **comp_order** keyword orders nets or classes of nets, by using only component reference designators, when exact pin numbers are not known. For example

(net sig1 (comp_order U1 U2 U3))

The placer finds the shortest connection from U1 to U2 and from U2 to U3. If more than one pin from a component is used in the net, they are joined by using the shortest path. For example, the result could be U1-12 to U1-15, to U2-3 to U2-5 to U2-7, and U2-7 to U3-8.

The <*placement_id*> can contain wildcards to specify one or more component reference designators. The placer finds all components that match the wildcard pattern, but ignores the components that do not have pins on the specified nets.

See also *<topology_descriptor>*.

## *<component_property_descriptor>*

<*component_property_descriptor*>::=
(**property**
{[*<physical_property_descriptor>* |
*<electrical_value_descriptor>* |
*<property_value_descriptor>*]}
)

If you add or change the <*physical_property_descriptor*> or <*electrical_value_descriptor*> during a session, the session file or the placement file reflects the change. If you add or change the <*property_value_descriptor*>, the session or placement file does not reflect the change.

## *<component_status_descriptor>*

<*component_status_descriptor*>::= (**status** [**added** | **deleted** | **substituted**])

This option appears in session file placement references and in the *<placement_descriptor>* created by the **write placement** command. The **added** status means the component is not specified in the design file and was added during the session. The **deleted** status means the component is specified in the design file, but was deleted during the session. The **substituted** status means the component is specified in the design file and the component's image was substituted during the session.

## *<composite_name_list>*

<*composite_name_list*>::=
(**composite**
[*<prefix>*]
*<begin_index>*
*<end_index>*
*<step>*
[*<suffix>*]
)

## *<conductance_resolution_descriptor>*

<*conductance_resolution_descriptor*>::=
(**conductance_resolution** [**kg** | **g** | **mg**] <*positive_integer*>)

| Symbol | Conductance Unit |
|---|---|
| kg | kilomho |
| g | mho |
| mg | millimho |

The default conductance unit symbol is **kg** with a positive integer of 1000. (mho is reciprocal ohm)

## *<conductor_shape_descriptor>*

*<conductor_shape_descriptor>*::=
(**conductor**
  *<shape_descriptor>*
  [(**attr fanout**)]
  [(**type route**)]
)

The **conductor** keyword defines wires embedded within an image. A **route** type conductor cannot be altered, although the router can complete a connection to this conductor type. The default **type** is **route**.

## *<conductor_via_descriptor>*

*<conductor_via_descriptor>*::=
(**via**
  *<padstack_id>* {*<vertex>*}
  [(**attr fanout**)]
  [(**type route**)]
)

The **via** keyword defines vias embedded within an image. A **route** type conductor cannot be altered, although the router can complete a connection to this conductor type.

The default **type** is **route**. Conductors and vias are written to a routes or wires file only if **routes_include** is set to **image_conductor** (see *<parser_descriptor>* for details).

For example:

```
(image IU1
   (pin p25x75 layer_1 0.0000 0.1500)
   (pin P25x75 layer_2 0.0000 -0.1500)
      (conductor (path layer_1 0.005 0.0000 0.1500 0.0000 0.3500) (attr fanout))
      (conductor (path layer_2 0.005 0.0000 -0.1500 0.0000 -0.3500) (attr fanout))
   (via VIA 0.0000 0.3500 (attr fanout))
   (via VIA 0.0000 -0.3500 (attr fanout))
)
```

## *<conflict_descriptor>*

*<conflict_descriptor>*::=

([**cross** | **near**] *<layer_id>* {*<vertex>*})

## *<conflict_file_descriptor>*

*<conflict_file_descriptor>*::=
(**conflict** *<resolution_descriptor>* {*<conflict_descriptor>*})

## *<continuation_character>*

*<continuation_character>*::= [**;** | **\n**]

## *<control_descriptor>*

*<control_descriptor>*::=
(**control**
[*<via_at_smd_descriptor>*]
[*<off_grid_descriptor>*]
[*<route_to_fanout_only_descriptor>*]
[*<force_to_terminal_point_descriptor>*]
[*<same_net_checking_descriptor>*]
[*<checking_trim_descriptor>*]
[*<noise_calculation_descriptor>*]
[*<noise_accumulation_descriptor>*]
[(**include_pins_in_crosstalk** [**on** | **off**])]
[(**bbv_ctr2ctr** [**on** | **off**])]
[(**average_pair_length** [**on** | **off**])]
[(**crosstalk_model** [**cct1** | **cct1a**])]
[(**roundoff_rotation** [**on** | **off**])]
[(**microvia** [**on** | **off**])]
[(**reroute_order_viols** [**on** | **off**])]
)

When **include_pins_in_crosstalk** is **on**, pin shapes are included in the measurements for calculations that use parallel and tandem noise rules. The default is **off**.

When **bbv_ctr2ctr** is **on**, the minimum distance (gap) allowed between buried vias is measured from via center to via center. The default is **off**, meaning gap is measured from via edge to via edge. See also **buried_via_gap** in *<clearance_type>*.

When **average_pair_length** is **on**, rule checking that involves wire length uses the average length of the wires in a pair. When **average_pair_length** is **off**, the rule checker uses the actual length of each wire. The default is **on**.

The **cct1** crosstalk model uses parallel and tandem noise rules to control routing and report cumulative noise violations on routed nets. The **cct1a** crosstalk model takes into account saturation noise rules. The default is **cct1**.

The **roundoff_rotation** control prevents design rule checking discrepancies with Allegro or other layout systems that roundoff rotation calculations. When **on**, pad rotation values are rounded off. When **off**, pad rotation values are truncated. The default is **off**.

When **microvia** is **on**, features licensed under the RouteMVIA license are enabled. See the **set microvia** command in the online help for a list of features.

When **reroute_order_viols** is **on**, the autorouter reroutes connections with order violations. When **off** (the default), connections with order violations can exist at the end of routing passes. You can report and highlight order violations.

## *<corner_descriptor>*

*<corner_descriptor>*::=
(**corner** <*layer_id*> {*<vertex>*})

## *<corner_file_descriptor>*

*<corner_file_descriptor>*::=
(**corners** *<resolution_descriptor>*{*<corner_descriptor>*})

## *<cost_descriptor>*

*<cost_descriptor>*::=
[**forbidden | high | medium | low | free |**
<*positive_integer*> **| -1**]

Applying a value of **-1** resets the cost to its default (system-assigned) value.

## *<cost_type>*

*<cost_type>*::=
[**way | cross | via | off_grid | off_center | side_exit | squeeze**]

## *<current_resolution_descriptor>*

<*current_resolution_descriptor*>::=
(**current_resolution** [**amp | mamp**] <*positive_integer*>)

The symbol mamp means milliampere. The default unit of current is **mamp** with a <*positive_integer*> equal to 1000.

## *<daisy_type>*

<*daisy_type*>::=
(**type** [**mid_driven | balanced**])

Use **type** to specify **mid_driven** or **balanced** daisy chain routing, rather than simple daisy chain routing. See *<order_type>* for a description of simple daisy chain routing.

The <*daisy_type*> keywords are described in the following table.

| Keyword | Description |
|---|---|
| mid_driven | Applies to any net that has exactly two terminator pins and one or more source pins. The tool first chains all source pins together and then attaches each terminator pin to its nearest load pin. The load pins are connected to the next nearest load or source pin. This progression continues until all loads are chained together back to a source pin. If the net does not contain exactly two terminator pins and one or more source pins, the net is ordered as a simple daisy chain. |
| balanced | Applies to nets that have one or more source pins and two or more terminator pins. Loads are evenly distributed between the source and terminator pins. If the net does not contain two or more terminator pins and one or more source pins, the net is ordered as a simple daisy chain. |

## *<degree>*

*<degree>*::= *<positive_integer>*

The range of values for *<degree>* is 0 - 360.

## *<delay_descriptor>*

*<delay_descriptor>*::=
([**max_delay | min_delay**] *<delay_value>*)

## *<delay_value>*

*<delay_value>*::= *<real>*

## *<design_descriptor>*

*<design_descriptor>*::=
(**pcb** *<pcb_id>*
[*<parser_descriptor>*]
[*<capacitance_resolution_descriptor>*]
[*<conductance_resolution_descriptor>*]
[*<current_resolution_descriptor>*]
[*<inductance_resolution_descriptor>*]
[*<resistance_resolution_descriptor>*]
[*<resolution_descriptor>*]
[*<time_resolution_descriptor>*]
[*<voltage_resolution_descriptor>*]
[*<unit_descriptor>*]
[*<structure_descriptor>* | *<file_descriptor>*]
[*<placement_descriptor>* | *<file_descriptor>*]
[*<library_descriptor>* | *<file_descriptor>*]
[*<floor_plan_descriptor>* | *<file_descriptor>*]
[*<part_library_descriptor>* | *<file_descriptor>*]
[*<network_descriptor>* | *<file_descriptor>*]
[*<wiring_descriptor>*]
[*<color_descriptor>*]
)

## *<diameter>*

<*diameter*>::=
<*positive_dimension*>

## *<digit>*

<*digit*>::=
[**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**]

## *<dimension>*

<*dimension*>::= *<number>*

Dimension implies a length value associated with a *<dimension_unit>* such as **mm** or **inch**.

## *<dimension_unit>*

<*dimension_unit*>::=
[**inch** | **mil** | **cm** | **mm** | **um**]

## *<direction_type>*

<*direction_type*>::=
[**horizontal** | **vertical** | **orthogonal** | **positive_diagonal** |
**negative_diagonal** | **diagonal** | **off**]

The <*direction_type*> keywords are described in the following table.

| Keyword | Description |
|---|---|
| horizontal | Routing is free (cost = 0) in the horizontal direction. |
| vertical | Routing is free (cost = 0) in the vertical direction. |
| orthogonal | Routing is free (cost = 0) in both horizontal and vertical directions. |

| Keyword | Description |
|---|---|
| positive_diagonal | Routing is free (cost = 0) in the positive diagonal direction, which is from bottom left-to-top right and top right-to-bottom left. |
| negative_diagonal | Routing is free (cost = 0) in the negative diagonal direction, which is from bottom right-to-top left and top left-to-bottom right. |
| diagonal | Routing is free (cost = 0) in both positive and negative diagonal directions. |
| off | The layer is not available for routing. |

## *<directory_descriptor>*

<*directory_descriptor*>::=
(**directory** {*<directory_path_name>*})

## *<directory_path_name>*

<*directory_path_name*>::= <*id*>

## *<effective_via_length_descriptor>*

<*effective_via_length_descriptor*>::=
(**effective_via_length** [<*positive_dimension*> **|** **-1**])

For each via in a connection, the **effective_via_length** value adds to the wire length to determine the total effective length of the connection.

If <*positive_dimension*> is 0, vias have no effective length. Only wire length is used to perform length calculations. (See also **length_factor** rule.)

When **effective_via_length** value is **-1**, the rule is unspecified.

## *<electrical_value_descriptor>*

<*electrical_value_descriptor*>::= (**value** *<string>*)

## *<end_index>*

*<end_index>*::= *<positive_integer>*

## *<exclude_descriptor>*

*<exclude_descriptor>*::=
(**exclude**
[{[<*component_id*> | <*cluster_id*>]} | **remain**] [(**type** [**hard** | **soft**])]
)

The <*exclude_descriptor*> excludes components from a room. The **remain** keyword can be used to specify all remaining components. The **type hard** keywords specify that no part of the excluded components can occupy the room. The **type soft** keywords specify that a portion of the excluded components can occupy the room.

See also *<room_rule_descriptor>*.

## *<expression>*

*<expression>*::=
[*<numeric_expression>* | *<string_expression>*]

## *<extra_image_directory_descriptor>*

*<extra_image_directory_descriptor>*::=
(**extra_image_directory** *<directory_path_name>*)

This directory lets you add image files for new images during a session. The files must be named <*image_id*>.i. Specifying the <*image_id*> in a command that defines a new component or assigns a component to a different image reads the image file from this directory.

## *<family_family_descriptor>*

*<family_family_descriptor>*::=
(**family_family**
(**family** <*family_id*> {<*family_id*>})
(**place_rule** {*<family_family_spacing_descriptor>*})
)

The *<family_family_spacing_descriptor>* rule applies between images in the family identified by the first <*family_id*> and images in one or more of the other families. You can repeat the first <*family_id*> to apply the spacing rule between images in the same family. If an image belongs to more than one family, the rule applied is the largest **family_family** spacing rule specified for any of the families.

## *<family_family_spacing_descriptor>*

<*family_family_spacing_descriptor*>::=
(**family_family_spacing**
[<*positive_dimension*> **| -1**]
[(**type** [**pad_pad | pad_body | body_body**])]
[(**side** [**front | back | both**])])

The **family_family_spacing** rule applies minimum edge-to-edge spacing values between image pad edges and body edges (pad-to-pad, pad-to-body, or body-to-body). When **type** is not specified, the spacing rule is applied to all types. The default **side** is **both**.

An **image_image_spacing** rule has higher precedence than a **family_family_spacing** rule. For the order of placement rule precedence, see the Routing and Placement Rule Hierarchies section at the beginning of this manual.

The default rule is **-1**, which means the **family_family_spacing** rule is undefined.

## *<family_id>*

<*family_id*>::= <*id*>

## *<family_property>*

<*family_property*>::=
(**family** {<*family_id*>})

## *<file_descriptor>*

<*file_descriptor*>::=
(**file** *<file_path_name>*)

The <*file_descriptor*> construct can be substituted in *<design_descriptor>* for any of the constructs listed in this chapter. The <*file_descriptor*> points to a file whose contents are immediately processed. This file must contain the entire syntax for the construct replaced.

For example, a sample PCB could be defined as

```
(pcb sample
   (file gpcb/sample/structure)
   (file gpcb/sample/placement)
   (file gpcb/sample/library)
   (file gpcb/sample/network)
)
```

The gpcb/sample/structure file must contain all the rule, via, grid, boundary, and layer definitions. This technique allows you to create reusable libraries for printed circuit board technologies and for component definitions.

## *<filename>*

*<filename>*::= *<id>*

## *<file_path_name>*

*<file_path_name>*::= *<id>*

## *<file_prefix>*

*<file_prefix>*::= *<id>*

## *<flip_style_descriptor>*

*<flip_style_descriptor>*::=
(**flip_style**
   [**mirror_first | rotate_first**]
)

The **mirror_first** flip style flips and mirrors the component, and then rotates it. The **rotate_first** flip style applies rotation before the component is flipped to the opposite surface.

The default **flip_style** is **mirror_first**, but the preferred **flip_style** is **rotate_first**. Although **rotate_first** is used for new design files, **mirror_first** is maintained as the default for compatibility with design files from previous versions.

See also *<place_control_descriptor>*.

## *<floor_plan_descriptor>*

*<floor_plan_descriptor>*::=
(**floor_plan**
[*<unit_descriptor>*]
[*<resolution_descriptor>*]
[{*<cluster_descriptor>*}]
[{*<room_descriptor>*}]
)

See also *<design_descriptor>*.

## *<float>*

*<float>*::=

A C-language floating-point number.

## *<force_to_terminal_point_descriptor>*

*<force_to_terminal_point_descriptor>*::=
(**force_to_terminal_point** [**on**| **off**])

The default **force_to_terminal_point** is **off**, which means the router can connect to a point on any side of a polygonal pad shape. If **force_to_terminal_point** is **on**, the router must wire to the origin of the pad.

## *<fraction>*

*<fraction>*::=
*<positive_integer>* / *<positive_integer>*

## *<fromto_descriptor>*

<*fromto_descriptor*>::=
{(**fromto**
[*<pin_reference>* | *<virtual_pin_descriptor>*] | <*component_id*>]
[*<pin_reference>* | *<virtual_pin_descriptor>* | <*component_id*>]
[(**type** [**fix | normal | soft**])]
[(**net** <*net_id*>)]
[*<rule_descriptor>*]
[*<circuit_descriptor>*]
[{*<layer_rule_descriptor>*}]
)}

Fromto **type** options are described as follows:

| Option | Description |
|---|---|
| fix | A fromto that cannot be routed. |
| normal | A fromto that is unrestricted. |
| soft | A fromto that is defined for length or delay measurements only. It does not define the net topology. |

The default **type** is **normal**. The **type soft** fromto does not define the pin order on a net. To explicitly order the fromtos on a net, include that definition as well as the soft definition.

Use the **net** keyword for a fromto that is part of a group and contains two virtual pins.

## *<gate_id>*

<*gate_id*>::= <*id*>

See also *<part_pin_descriptor>*.

## *<gate_pin_id>*

<*gate_pin_id*>::= <*id*>

The <*gate_pin_id*> is the logical name of a pin in a gate.

See also *<part_pin_descriptor>*.

## *<gate_pin_swap_code>*

<*gate_pin_swap_code*>::= <*integer*>

Pins that belong to the same subgate, or to the same gate if the gate contains no subgates, and that have the same <*gate_pin_swap_code*> are swappable. A <*gate_pin_swap_code*> of 0 means the pin is not swappable.

See also *<part_pin_descriptor>*.

## *<gate_swap_code>*

<*gate_swap_code*>::= <*integer*>

Gates that have the same <*gate_swap_code*> are swappable. A <*gate_swap_code*> of 0 means that the gate is not swappable.

See also *<part_pin_descriptor>*.

## *<grid_descriptor>*

<*grid_descriptor*>::=
[(**grid via** <*positive_dimension*> [<*via_id*>]
[(**direction** [**x** | **y**])] [(**offset** <*positive_dimension*>)]) |
(**grid wire** <*positive_dimension*> [*<layer_name>*]
[(**direction** [**x** | **y**])] [(**offset** <*positive_dimension*>)]) |
(**grid via_keepout** <*positive_dimension*>
[(**direction** [**x** | **y**])] [(**offset** <*positive_dimension*>)]) |
(**grid place** <*positive_dimension*>
[(**image_type** [**smd** | **pin**])]) |
(**grid snap** <*positive_dimension*>
[(**direction** [**x** | **y**])] [(**offset** <*positive_dimension*>)])
]

The statement (grid via 0.020) defines a via grid of 0.020 for all vias. Grids can also be assigned to specific vias. For example, (grid via 0.025 V25) defines a 0.025 grid for via V25.

The statement (grid wire 0.007) defines a wire grid of 0.007 for all layers. Wire grids can also be assigned to specific layers. For example, the statement (grid wire 0.005 L1) defines a 0.005 routing grid on layer L1.

The **grid via_keepout** keyword identifies grid positions that the router can't use. The following example specifies that the router can place vias on 0.025, 0.050, and 0.075 centers but cannot place vias on .100 centers.

(grid via 0.025)

(grid via_keepout .100)

The **grid place** keyword defines a single placement grid for all components, or separate placement grids for SMD components and through-pin components.

The **grid snap** keyword defines grid points that the pointer snaps to during interactive modes.

The via, wire, via keepout, and placement grids have precedence over snap grids.

The **direction** and **offset** options apply to via, wire, via keepout, and snap grids. If a **direction** option is not specified, the grid spacing value and offset value (if given) apply equally in the x and y directions. If a **direction** option is specified, the grid spacing value and offset value (if given) only apply to the specified direction. To specify nonuniform grids or offsets in the x and y directions, you must use two **grid via**, **grid wire**, **grid via_keepout**, or **grid snap** option expressions.

## *<group_descriptor>*

*<group_descriptor>*::=
(**group** *<group_id>*
{*<fromto_descriptor>*}
[*<circuit_descriptor>*]
[*<rule_descriptor>*]
[{*<layer_rule_descriptor>*}]
)

## *<group_id>*

*<group_id>*::= *<id>*

## *<group_set_descriptor>*

*<group_set_descriptor>*::=
(**group_set** *<group_set_id>* {[*<group_id>* |
*<composite_name_list>*]}
[*<circuit_descriptor>*]
[*<rule_descriptor>*]
[{*<layer_rule_descriptor>*}]
)

## *<group_set_id>*

*<group_set_id>*::= *<id>*

## *<history_descriptor>*

*<history_descriptor>*::=
(**history** [{*<ancestor_file_descriptor>*}] *<self_descriptor>*)

The <*history descriptor*> is included in a session file to provide a list of all session files created for a design. The **history** block always includes a *<self_descriptor>* that identifies the current session file. Each <*ancestor_file_descriptor*> identifies a previous session file.

## *<id>*

*<id>*::= [*<character>* | *<character>* *<id>*]

## *<image_descriptor>*

```
<image_descriptor>::=
(image <image_id>
   [(side [front | back | both])]
   [<unit_descriptor>]
   [<outline_descriptor>]
   {(pin <padstack_id> [(rotate <rotation>)]
      [<reference_descriptor> | <pin_array_descriptor>]
      [<user_property_descriptor>])}
   [{<conductor_shape_descriptor>}]
   [{<conductor_via_descriptor>}]
   [<rule_descriptor>]
   [<place_rule_descriptor>]
   [{<keepout_descriptor>}]
[<image_property_descriptor>]
)
```

The <*outline_descriptor*> is the true outline of a component. The accuracy of the image outline specification in the design file is important to assure correct intercomponent spacing. The outline must be defined from the top view of the design.

If the image outline does not encompass all pins on the image, the outline expands to cover the pins. If an image outline is not defined in the design file, the tool generates a bounding box that includes all pins or pads of the component.

One library part can have two images linked, where one image is used when the component is placed on the front (primary side) of the design, and the other image is used when the component is placed on the back (secondary side) of the PCB. If **side** is **both** or not specified, front and back instances use the same image definition. Each image can have different padstacks associated with it, but this is not necessary. The images must be defined from the top view.

See also *<library_descriptor>* and *<outline_descriptor>*.

Example:

```
(placement
    :
    :
  (component IU1
    (place U1 0 0 front 0)
    (place U1 0 0 back 0)
  )
    :
    :
)
(library
    :
    :
  (image IU1
    (side front)
    (pin p25x25 layer_1 0.0000 0.1500)
    (pin p25x75 layer_1 0.0000 -0.1500)
  )
  (image IU1
    (side back)
    (pin p25x75 layer_1 0.0000 0.2000)
    (pin p25x75 layer_1 0.0000 -0.2000)
  )
    :
    :
)
```

The geometric features of front and back instances of U1 are different because the geometric definition of image IU1 is different for the front and back sides.

## *<image_image_descriptor>*

*<image_image_descriptor>::=*
(**image_image**
{(**image** *<image_id>* {*<image_id>*})}
{*<image_image_place_rule_descriptor>*})

## *<image_image_place_rule_descriptor>*

*<image_image_place_rule_descriptor>*::=
(**place_rule** {*<image_image_spacing_descriptor>*})

## *<image_image_spacing_descriptor>*

*<image_image_spacing_descriptor>::=*
(**image_image_spacing**
[*<positive_dimension>* **| -1**]
[(**type** [**pad_pad | pad_body | body_body**])]
[(**side** [**front | back | both**])])

The **image_image_spacing** rule applies minimum edge-to-edge spacing values between image pad edges and body edges (pad-to-pad**,** pad-to-body**,** body-to-body). When **type** is not specified, the spacing rule applies to all types. The default **side** is **both**.

An **image_image_spacing** rule has higher precedence than any other spacing rule. For the order of placement rule precedence, see the Routing and Placement Rule Hierarchies section at the beginning of this manual.

The default rule is **-1**, which means the **image_image_spacing** rule is undefined.

## *<image_id>*

*<image_id>*::= *<id>*

## *<image_property_descriptor>*

*<image_property_descriptor>*::=
(**property**
{[*<physical_property_descriptor>* **|**
*<family_property>* **|**
*<property_value_descriptor>*]}
)

If you add or change the <*physical_property_descriptor*> or <*family_property*> during a session, the session or placement file includes these changes. If you add or change the <*property_value_descriptor*>, the session or placement file does not include these changes.

## *<image_type_descriptor>*

<*image_type_descriptor*>::=
(**image_type** [**smd** | **pin**])

## *<include_descriptor>*

<*include_descriptor*>::=
(**include**
[{[<*component_id*> | <*cluster_id*>]} | **remain**]
[(**type** [**hard** | **soft**])]
)

The <*include_descriptor*> includes components from a room. The **remain** keyword can be used to specify all remaining components. The **type hard** keywords specify that no part of the included components can occupy the room. The **type soft** keywords specify that a portion of the included components can occupy the room.

See also *<room_rule_descriptor>*.

## *<inductance_resolution_descriptor>*

<*inductance_resolution_descriptor*>::=
(**inductance_resolution** [**mhenry** | **uhenry** | **nhenry**]
<*positive_integer*>)

| Symbol | Inductance Unit |
|---|---|
| mhenry | millihenry |
| uhenry | microhenry |
| nhenry | nanohenry |

The default inductance unit is **nhenry** with a positive integer of 1000.

## *<index_step>*

*<index_step>*::= *<positive_integer>*

## *<integer>*

*<integer>*::=
[*<sign>*] *<positive_integer>*

## *<interlayer_clearance_descriptor>*

*<interlayer_clearance_descriptor>*::=
(**inter_layer_clearance** *<positive_dimension>*
[(**type** {*<object_type>*_*<object_type>*})]
[(**layer_pair** *<layer_id>* *<layer_id>*)]
[(**layer_depth** *<integer>*)]
)

The **layer_pair** option specifies two layers between which the clearance rule applies. Clearance rules between layer pairs are followed even if the pair is separated by a power layer.

The **layer_depth** option specifies the number of layers above and below the current layer over which the clearance rule applies. An adjacent layer separated by a power layer is not considered even if it falls in the layer range controlled by the **layer_depth** value.

The **layer_pair** option is applicable only at the pcb (global) level of the rule hierarchy. The **layer_depth** option is applicable only at the class_class level of the rule hierarchy.

## *<jumper_descriptor>*

*<jumper_descriptor>*::=
(**jumper** (**length** *<positive_dimension>*)
[(**use_via** *<padstack_id>*)]
[(**height** *<max_height>*)]
)

The **length** value defines the fixed jumper length used when jumper vias are added on the jumper layer. The jumper attribute attaches to jumper vias and wires on the jumper layer. The jumper vias, wires, and attributes are included in the wires and routes files.

The autorouter chooses vias for jumpers from the via padstack list unless a particular via padstack name is specified in the **use_via** option.

The autorouter can rotate nonsymmetrical jumper vias when the **set rotate_jumper_via** command is **on**. See the online help for more information.

The **height** option allows jumpers under components whose height is greater than <*max_height*>. Jumpers are not allowed under components whose height is equal to or less than <*max_height*>. When **height** is not specified, jumpers are also not allowed under components. The autorouter uses the actual component outline as a jumper keepout on the jumper layer. These are visible when the jumper layer is defined.

For more information about jumper layers, see also *<layer_type>*.

## *<junction_type_descriptor>*

<*junction_type_descriptor*>::=
(**junction_type** [**term_only | all | supply_only**])

The **junction_type** determines where tjunctions can occur.

The **term_only** option permits tjunctions only at pins, SMD pads, and vias. The **all** option permits tjunctions at pins, SMD pads, vias, and wires. The **supply_only** option does not permit tjunctions. This means that power nets and signal nets are routed directly from pin to source terminal. If **junction_type** is not specified, it defaults to **all**.

The **junction_type** also controls the type of connection that can be used for virtual pins. If **term_only** is set, virtual pins use only vias. If **all** is set, virtual pins use vias or wire tjunctions. See also *<virtual_pin_descriptor>*.

## *<keepout_descriptor>*

<*keepout_descriptor*>::=
([**keepout | place_keepout | via_keepout | wire_keepout |**
  **bend_keepout | elongate_keepout**]
[<*id*>]
[(**sequence_number** *<keepout_sequence_number>*)]
*<shape_descriptor>*
[(**rule** *<clearance_descriptor>*)]
[(**place_rule** *<spacing_descriptor>*)]
[{*<window_descriptor>*}]
)

All shapes defined by <*keepout_descriptor*> are treated as area object types.

The <*id*> is used only in the session file to record a defined keepout area. If you do not assign an <*id*> when you define the keepout, the tool assigns one. Keepout areas defined in an *<image_descriptor>* can't be changed.

The **sequence_number** option provides the sequence number of a modified keepout that was defined in the structure section and is written in the structure section of the session file.

The rules that you can specify depend on the keepout type.

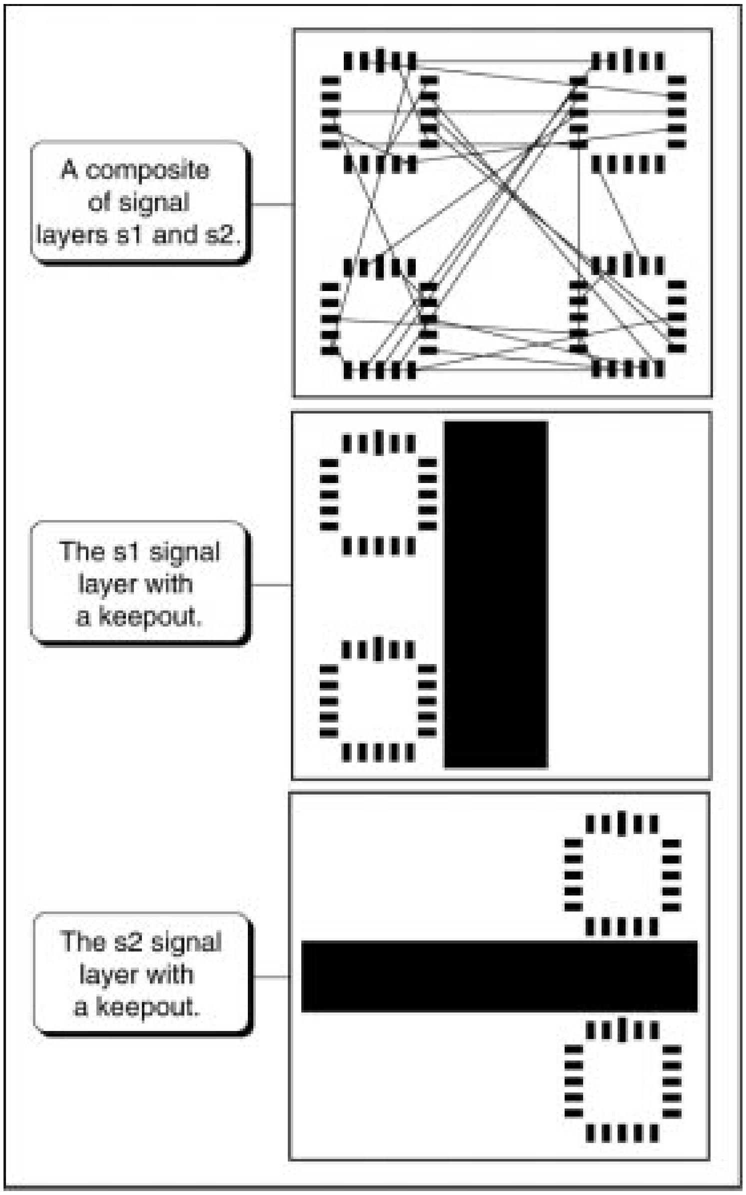
- A *<clearance_descriptor>* can be specified with **keepout**, **via_keepout, wire_keepout**, **bend_keepout**, or **elongate_keepout**. The clearance type must be **area_pin**, **area_smd, area_wire**, or **area_via**.
- A *<spacing_descriptor>* can be specified with **keepout** or **place_keepout**. The spacing type must be **area**.

The following figure illustrates an unrouted design without keepouts, and the separate layers after keepouts are defined. A keepout is positioned on each signal layer. The keepout definitions are specified as

(keepout (rect s2 0.560  0.909  1.739  0.589))

(keepout (rect s1 0.992  1.477  1.319  0.170))

### A Sample Design With and Without Keepouts




## *<keepout_sequence_number>*

*<keepout_sequence_number>*::= *<positive_integer>*

This integer indicates the sequence number of a keepout added, modified, or deleted during the session.

## *<layer_descriptor>*

*<layer_descriptor>*::=
(**layer** *<layer_name>*
(**type** *<layer_type>*)
[*<user_property_descriptor>*]
[(**direction** *<direction_type>*)]
[*<rule_descriptor>*]
[(**cost** *<cost_descriptor>* [(**type** [**length** | **way**])])]
[(**use_net** {<*net_id*>})]
)

Normally, layer cost is free.

Layers are ordered by their relative positions in the structure data. The first layer is the top physical layer and the last layer is the bottom physical layer.

The maximum number of signal and power layers is 255.

## *<layer_id>*

*<layer_id>*::=
[*<reserved_layer_name>* | *<layer_name>*]

## *<layer_name>*

*<layer_name>*::= *<id>*

## *<layer_noise_weight_descriptor>*

*<layer_noise_weight_descriptor>*::=
(**layer_noise_weight** {*<layer_pair descriptor>*})

The following example shows how layer noise weight is specified in a design file:

```
(layer_noise_weight
   (layer_pair L1 L1 1.000)
   (layer_pair L1 L2 1.000)
   (layer_pair L2 L2 .900)
   (layer_pair L4 L4 .870)
   (layer_pair L4 L5 .880)
   (layer_pair L5 L5 .870)
   .
   .
   .
)
```

This syntax example can be illustrated by the layer noise weight matrix in the following table. Shaded boxes represent power layers.

| | **L1** | **L2** | **L3** | **L4** | **L5** | **L6** | **L7** | **L8** |
|---|---|---|---|---|---|---|---|---|
| L1 | 1.000 | 1.000 | | 0.000 | 0.000 | | 0.000 | 0.000 |
| L2 | 1.000 | .900 | | 0.000 | 0.000 | | 0.000 | 0.000 |
| L3 | | | | | | | | |
| L4 | 0.000 | 0.000 | | .870 | .880 | | 0.000 | 0.000 |
| L5 | 0.000 | 0.000 | | .880 | .870 | | 0.000 | 0.000 |
| L6 | | | | | | | | |
| L7 | 0.000 | 0.000 | | 0.000 | 0.000 | | 1.000 | 1.000 |
| L8 | 0.000 | 0.000 | | 0.000 | 0.000 | | 1.000 | 1.000 |

When **layer_noise_weight** is supplied in a design file, it must appear in the structure section.

A **layer_noise_weight** matrix can also be specified in a do file by using the **define** command. Layer pairs not assigned a **layer_noise_weight** value have a default value of 1.0. If layers are separated by a power layer, the default **layer_noise_weight** value is 0.

## *<layer_number>*

<*layer_number*>::= <*positive_integer*>

The range for <*layer_number*> is 0 - 15.

## *<layer_pair descriptor>*

<*layer_pair_descriptor*>::=
(**layer_pair** <*layer_id*> <*layer_id*> )

## *<layer_rule_descriptor>*

<*layer_rule_descriptor*>::=
(**layer_rule** {<*layer_id*>} *<rule_descriptor>*)

The <*layer_rule_descriptor*> is meaningful only for rules specified for classes, groups, nets, and fromtos. For example:

```
(net sig9 (pins U1-1 U2-2)
   (layer_rule S1 S4 (rule (width 0.010)))
   (layer_rule S2 S3 (rule (width 0.015)))
)
```

Net sig9 has a net_layer width rule of 0.01 on layers S1 and S4, and a net layer width rule of 0.015 on layers S2 and S3. For the order of routing rule precedence, see the Routing and Placement Rule Hierarchies section at the beginning of this manual.

## *<layer_type>*

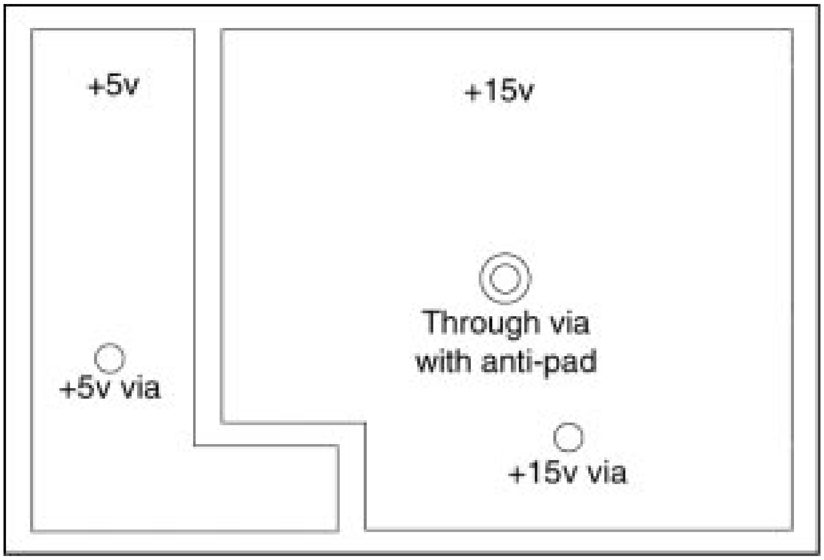
<*layer_type*>::=
[**signal** | **power** | **mixed** | **jumper**]

The following table defines the layer types.

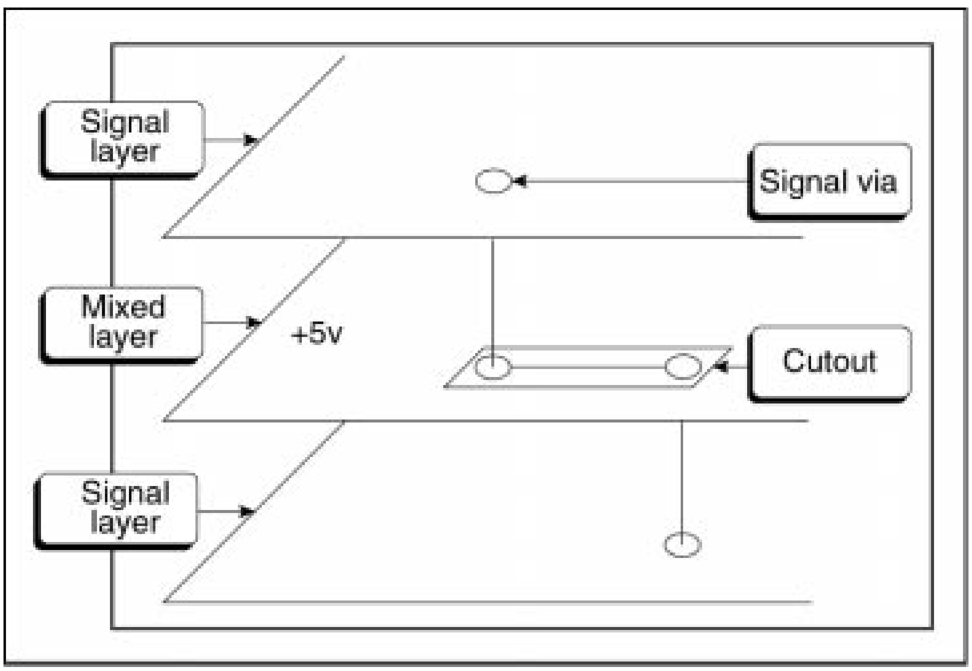
| Types | Description |
|---|---|
| signal | Layers used to route wires. Wires are made up of sets of overlapping filled shapes. The shapes can consist of pins, vias, wire segments, and wiring polygons. Wire segments can connect to polygons, which act as blockages for the routing of other nets. |
| power | A layer that supplies voltage or ground. Connections to power layers can be made by through-pins or vias. The router observes clearance rules for all shapes on power layers. Power planes can be either continuous or split. |
| mixed | Mixed layers provide a combination of signal and power layer features. Routing of short signal wires at a very high cost is allowed on mixed layers. |
| jumper | Jumper layers are used to define jumper connections that are installed during the PCB assembly process. Jumper layers must be defined as the top-most or bottom-most layer of a design. The cost to use the jumper layer is greater than the cost to use a wire on a signal layer. Wrong-way connections are forbidden on the jumper layer unless orthogonal is specified in the <*direction_type*> descriptor. See also <*jumper_descriptor*>. |

Continuous power layers are the most common. The entire layer is dedicated to a single voltage or ground net. Split power layers are defined by non-overlapping polygons. Each polygon represents a different power net, as shown in the following figures.

### Polygons Represent Different Power Nets




### Illustration of Mixed Layer with a Wired Connection



## *<layer_weight>*

<*layer_weight*>::= <*real*>

The <*layer_weight*> value is a factor that adjusts parallel and tandem noise calculations by layer. For example, noise coupling between wires on outer layers can be different from the coupling that occurs when the same nets are routed on an inner layer.

## *<length_amplitude_descriptor>*

<*length_amplitude_descriptor*>::=
(**length_amplitude** *<max_amp>* [*<min_amp>*])

The length amplitude rule controls the maximum and optional minimum peak-to-peak excursion from a straight wire path when the autorouter uses an accordion pattern to lengthen a wire.

## *<length_descriptor>*

<*length_descriptor*>::=
(**length**
*<max_length>* [*<min_length>*]
[(**type** [**ratio** | **actual**])]
)

The <*max_length*> value must be specified before the <*min_length*> value. A value of **-1** means the maximum length is undefined. If you don't want to control minimum length, either specify **-1** or omit the value. If only <*min_length*> is specified, set <*max_length*> to **-1**. For example:

(net sig1 QR2 (circuit (length -1 23)))...

If the <*max_length*> value is less than the <*min_length*> value, the <*max_length*> value is ignored.

You can specify maximum and minimum lengths as dimensional values or as ratios of routed length to Manhattan length. The **type** rule controls whether the length values are actual dimensions or ratios. Length values are actual if **type** is not specified. For example

(net XYZ (circuit (length 1500 500)))

is the same as

(net XYZ (circuit (length 1500 500 (type actual))))

The following example sets length rules as a ratio of maximum and minimum Manhattan length:

(net XYZ (circuit (length 1.5 1.1 (type ratio))))

A maximum length rule of 1.5 times the Manhattan distance is set, and a minimum length rule of 1.1 times the Manhattan distance is set.

See also the **circuit** command in the online help.

## *<length_factor_descriptor>*

<*length_factor_descriptor*>::=
(**length_factor** )

The **length_factor** adjusts calculated wire lengths to account for the distance between layers or for layer characteristics. A **length_factor** is usually used in controlled-impedance applications that impose different length constraints on different layers.

## *<length_gap_descriptor>*

<*length_gap_descriptor*>::=
(**length_gap** <*positive_dimension*>)

The **length_gap** rule controls the minimum gap between wire segments when accordion or trombone patterns are used to increase wire length. A value equal to three times the wire width is used, if **length_gap** is set to a value less than three times the wire width or is not specified.

## *<letter>*

<*letter*>::=

Any letter in the English alphabet.

## *<library_descriptor>*

*<library_descriptor>*::=
(**library**
[*<unit_descriptor>*]
{*<image_descriptor>*}
[{*<jumper_descriptor>*}]
{*<padstack_descriptor>*}
{*<via_array_template_descriptor>*}
[*<directory_descriptor>*]
[*<extra_image_directory_descriptor>*]
[{*<family_family_descriptor>*}]
[{*<image_image_descriptor>*}]
)

When a directory descriptor or extra image directory descriptor is used in place of image descriptors or padstack descriptors, the system expects one or more files in that directory with <*image_id*>.i or <*padstack_id*>.i filenames.

## *<library_out_descriptor>*

*<library_out_descriptor>*::=
(**library_out** {[*<padstack_descriptor>* **|** *<virtual_pin_descriptor>*]})

## *<limit_bends_descriptor>*

*<limit_bends_descriptor>*::=
(**limit_bends** [<*positive_integer*> **| -1**])

The bend limit applies to fromtos of nets or classes. When you apply a limit value of **-1**, the rule is set to the unspecified state.

## *<limit_crossing_descriptor>*

*<limit_crossing_descriptor>*::=
(**limit_crossing** [<*positive_integer*> **| -1**])

The crossing limit applies to fromtos of nets or classes. When you apply a limit value of **-1**, the rule is set to the unspecified state.

## *<limit_vias_descriptor>*

*<limit_vias_descriptor>*::=
(**limit_vias** [*<positive_integer>* **| -1**])

The via limit applies to fromtos of nets or classes. When a limit value of **-1** is applied, the rule is set to the unspecified state.

See also *<max_total_vias_descriptor>*.

## *<limit_way_descriptor>*

*<limit_way_descriptor>*::=
(**limit_way** [*<positive_dimension>* **| -1**])

The way limit applies to fromtos of nets or classes. When a limit value of **-1** is applied, the rule is set to the unspecified state.

## *<logical_part_descriptor>*

*<logical_part_descriptor>*::=
(**logical_part** *<logical_part_id>* {*<part_pin_descriptor>*})

See also *<part_library_descriptor>*.

## *<logical_part_id>*

*<logical_part_id>*::= *<id>*

See also *<logical_part_descriptor>* and *<logical_part_mapping_descriptor>*.

## *<logical_part_mapping_descriptor>*

*<logical_part_mapping_descriptor>*::=
(**logical_part_mapping** *<logical_part_id>* {[(**component** {*<component_id>*}) **|**
(**image** {*<image_id>*}) **|** (**physical** {*<physical_part_id>*})]})

See also *<part_library_descriptor>*.

## *<match_fromto_delay_descriptor>*

<*match_fromto_delay_descriptor*>::=
(**match_fromto_delay** [**off | on**]
[(**tolerance** *<delay_value>*)]
)

A **match_fromto_delay** rule applies to only nets, classes of nets, and groups of fromtos.

## *<match_fromto_length_descriptor>*

<*match_fromto_length_descriptor*>::=
(**match_fromto_length** [**off | on**]
[(**tolerance** <*positive_dimension*>) **|**
(**ratio_tolerance** <*real*>) **|** null]
)

The **match_fromto_length** rule applies to only nets, classes of nets, and groups of fromtos. It forces the autorouter to match the length of all fromtos of each net or group within the specified **tolerance** value. If the actual routed fromto lengths in each net or group differ by more than the **tolerance** value, the condition is a violation.

The **ratio_tolerance** value is a percentage value that can contain up to two digits after the decimal point. The autorouter calculates a dimensional ratio based on the longest total Manhattan length. For example, if the **ratio_tolerance** is .20 and the longest total Manhattan length is 1.5 inches, the autorouter calculates a tolerance of 0.3 inches, which is 20% of 1.5 inches.

The default setting for **match_fromto_length** is **off**.

## *<match_group_delay_descriptor>*

<*match_group_delay_descriptor*>::=
(**match_group_delay** [**off | on**]
[(**tolerance** *<delay_value>*)]
)

A **match_group_delay** rule can be applied only to a set of groups. The total routed delay of all groups in the set must match within the specified **tolerance** value. If the total routed delay of a group in the group set differs by more than the **tolerance** value, the condition is a violation. The default **tolerance** value is one inch.

## *<match_group_length_descriptor>*

<*match_group_length_descriptor*>::=
(**match_group_length** [**off | on**]
[(**tolerance** <*positive_dimension*>) **|**
(**ratio_tolerance** <*real*>) **|** null]
)

A **match_group_length** rule can be applied only to a set of groups. The total routed length of all groups in the set must match within the specified **tolerance** value. If the total routed length of a group in the group set differs by more than the **tolerance** value, the condition is a violation. The default **tolerance** value is one inch.

The **ratio_tolerance** value is a percentage value that can contain up to two digits after the decimal point. The autorouter calculates a dimensional ratio based on the longest total Manhattan length. For example, if the **ratio_tolerance** value is .15 and the group's longest total Manhattan length is 1.8 inches, The autorouter calculates a tolerance of 0.27 inches, which is 15% of 1.8 inches.

## *<match_net_delay_descriptor>*

<*match_net_delay_descriptor*>::=
(**match_net_delay** [**off | on**]
[(**tolerance** *<delay_value>*)]
)

A **match_net_delay** rule can be applied only to a class of nets. The routed delay of all nets in the class must match within the specified **tolerance** value. If the routed delays differ by more than the **tolerance** value, the condition is a violation. The default **tolerance** value is one inch.

## *<match_net_length_descriptor>*

<*match_net_length_descriptor*>::=
(**match_net_length** [**off | on**]
[(**tolerance** <*positive_dimension*>) **|**
(**ratio_tolerance** <*real*>) **|** null]
)

The **match_net_length** rule can be applied only to a class of nets. The routed length of all nets in the class must match within the specified **tolerance** value. If the routed lengths differ by more than the **tolerance** value, the condition is a violation. The default **tolerance** value is one inch.

The **ratio_tolerance** value is a percentage value that can contain up to two digits after the decimal point. The autorouter calculates a dimensional ratio based on the longest total Manhattan length. For example, if the **ratio_tolerance** value is .20 and the net's longest total Manhattan length is 1.5 inches, The autorouter calculates a tolerance of 0.3 inches, which is 20% of 1.5 inches.

## *<max_amp>*

<*max_amp*>::= [<*positive_dimension*> **| 0 | -1**]

If you set maximum amplitude to **0**, the autorouter cannot use an accordion pattern, which can result in more length violations. To reset amplitude to unspecified, use a value of **-1**.

See also *<length_amplitude_descriptor>*.

## *<max_height>*

<*max_height*>::=
[<*positive_dimension*> **| -1**]

A <*max_height*> value of **-1** sets the maximum height value to unspecified.

See also *<room_rule_descriptor>*, *<physical_property_descriptor>*, and *<jumper_descriptor>*.

## *<max_length>*

<*max_length*>::= <*positive_dimension*>

## *<max_noise_descriptor>*

<*max_noise_descriptor*>::=
(**max_noise** [<*real*> **| -1**])

The **max_noise** rule controls the maximum noise that can accumulate on a net before a coupled noise violation occurs. This rule can be applied at the pcb, net, and class levels of the rule hierarchy, and is typically expressed in units of millivolts. When the **max_noise** value for a net is **-1**, the net is not checked for parallel and tandem noise violations.

## *<max_restricted_layer_length_descriptor>*

<*max_restricted_layer_length_descriptor*>::=
(**max_restricted_layer_length** <*real*> [(**total**)])

A **max_restricted_layer_length** rule applies to nets, classes of nets, groups of fromtos, and group sets. It specifies a maximum length limit on certain layers for each fromto in a net, each net in a class, each fromto in a group, and each group in a group set.

The **total** syntax applies only to groups. When used, the sum of the routed fromtos in the group must be within the **max_restricted_layer_length** limit.

If the actual routes are greater in length than the **max_restricted_layer_length** value, the condition is a violation. The **max_restricted_layer_length** must be a positive real value.

When using the **max_restricted_layer_length** rule, you must assign a length factor (see also *<restricted_layer_length_factor_descriptor>*). For example, to limit routing on the external layers of a PCB for EMI control, you could do the following. Assign a **restricted_layer_length_factor** value of 1.0 to the external layers and a value of 0.0 to the internal signal layers, and then limit the routing on the outer layers to the **max_restricted_layer_length** value.

## *<max_stagger_descriptor>*

<*max_stagger_descriptor*>::=
(**max_stagger** [<*positive_dimension*> **| -1**])

The **max_stagger** rule controls the maximum wire length allowed on a mixed layer. The tolerance for **max_stagger** is one times the specified length value. For example, if you use a value of 100, the resulting routing length could be 200.

An example that allows routing for short distances on the GND layer is

```
(network
   (net #162
      (rule layer GND (max_stagger 500))
   )
)
```

The **max_stagger** rule can also control the maximum wire length allowed on certain signal layers. An example that specifies routing all clock signals on int2 and int3, and allows only short distances on int1 and int4 is

```
(network
   (class CLK clk*
      (circuit (use_layer int2 int3 in1 int4))
      (layer_rule int1 int4 (rule (max_stagger 100)))
   )
)
```

## *<max_stub_descriptor>*

<*max_stub_descriptor*>::=
(**max_stub** [<*positive_dimension*> **| 0**])

The **max_stub** rule controls tjunction routing at pads and pins on daisy-chain nets.

If the **max_stub** value is greater than zero, tjunctions are allowed up to <*positive_dimension*> distance from the terminal point, based on which **junction_type** option is set. If the **max_stub** value equals zero, no tjunctions are allowed on the nets; pad and pin entry or exit must be unique for each wire. The maximum stub condition is defined from the center of the pad to the center of the tjunction.

## *<max_total_vias_descriptor>*

<*max_total_vias_descriptor*>::=
(**max_total_vias** [<*positive_integer*> **| -1**]

The **max_total_vias** rule limits the total number of vias in a group of fromtos or on a net. The **max_total_vias** rule applies to the entire net or group. A value of **-1** means there is no limit to the number of vias that can be used.

## *<microvia_descriptor>*

<*microvia_descriptor*>::=
(**via**
   (**via_size** *<via_width>* [*<via_height>*])
   (**clear** *<x_clearance>* [*<y_clearance>*])
   {(**overlap** <*layer_id*> *<x_overlap>* [*<y_overlap>*])}
)

The <*microvia_descriptor*> is used with the *<via_array_template_descriptor>* to generate via arrays during routing. (This requires **microvia on** in the <*control_descriptor*>.)

The via size dimensions of a single via are <*via_width*> and <*via_height*>. If only <*via_width*> is defined, <*via_height*> is set to the same value.

The horizontal and vertical edge-to-edge distances between two vias are <*x_clearance*> and <*y_clearance*>, respectively. If only <*x_clearance*> is defined, <*y_clearance*> is set to the same value.

Each routing layer spanned by a via array must have an overlap area that covers the via array area. For each layer identified by <*layer_id*>, the overlap in the X-dimension and Y-dimension is indicated by <*x_overlap*> and <*y_overlap*>. If only <*x_overlap*> is defined, <*y_overlap*> is set to the same value.

Via arrays are generated during routing, based on a particular <*via_array_template_id*> and <*microvia_descriptor*>. The number of rows and columns in an array is determined by the available overlap area. The via array must fit in the overlap area defined by the wire widths on the adjacent levels.

Example:

```
(via_array_template VIA2
   (via (via_size 20) (clear 6)
      (overlap L2 20) (overlap L3 18)
   )
)
```

## *<min_amp>*

<*min_amp*>::= [<*positive_dimension*> **| 0 | -1**]

When minimum amplitude is set to **-1**, the autorouter uses the default value, which is the larger of the following

- Three times the wire width
- One wire width plus one wire-to-wire clearance value

See also *<length_amplitude_descriptor>*.

## *<min_height>*

<*min_height*>::=
[<*positive_dimension*> **| -1**]

A <*min_height*> value of **-1** sets the minimum height value to unspecified.

See also *<room_rule_descriptor>*.

## *<min_length>*

<*min_length*>::= <*positive_dimension*>

## *<mirror_descriptor>*

<*mirror_descriptor*>::=
(**mirror** [**X** | **Y** | **XY** | **off**])

Use <*mirror_descriptor*> to control the X and Y mirroring of a component when you translate. You can mirror a component with respect to its X axis, its Y axis, or both. A mirrored component cannot be changed by the user in a PCB design.

A mirror image is generated with respect to the origin of the component's image. For example, suppose a component appears in your layout system with pin 1 at the top left corner. If you specify **mirror X** to mirror the component across its X axis, it appears with pin 1 at the bottom left corner after you translate.

If you specify **mirror off**, the component is always displayed as it appears in your layout system. If mirror is not specified, mirroring is not performed, and components are displayed according to the side of the PCB on which they are placed.

## *<name_descriptor>*

<*name_descriptor*>::= *<string>*

The <*string*> is the name of a user-defined property.

## *<net_descriptor>*

*<net_descriptor>*::=
(**net** <*net_id*>
[(**unassigned**)]
[(**net_number** <*integer*>)]
[(**pins** {*<pin_reference>*}) | (**order** {*<pin_reference>*})]
[*<component_order_descriptor>*]
[(**type** [**fix** | **normal**])]
[*<user_property_descriptor>*]
[*<circuit_descriptor>*]
[*<rule_descriptor>*]
[{*<layer_rule_descriptor>*}]
[*<fromto_descriptor>*]
[(**expose** {*<pin_reference>*})]
[(**noexpose** {*<pin_reference>*})]
[(**source** {*<pin_reference>*})]
[(**load** {*<pin_reference>*})]
[(**terminator** {*<pin_reference>*})]
[(**supply** [**power** | **ground**])]
)

The **unassigned** keyword, if used, must follow the <*net_id*>. This keyword designates wiring polygons as unassigned (no net assignment). Unassigned wiring polygons are saved in the network_out section of the routes or session file.

The **net_number** option is for use only with translators. The router does not use them.

All the pins in a net must be listed by using either the **pins** list or the **order** list. If the **order** list and the *<fromto_descriptor>* are both used, the pin ordering in the **fromto** list must match the ordering in the **order** list.

The **expose** pin list treats the referenced through-pins as SMD pins. The autorouter routes from the exposed pin to an escape via on an external layer. Routing from the escape via can continue on any signal layer. The following example forces routing from pins U1-1 and U4-5 to escape vias on an external layer.

(network (net net1 (pins U1-1 U2-3 U4-5 U5-7)

(expose U1-1 U4-5)))

The **fanout (pintype signal)** and **fanout (pintype power)** commands generate vias for **expose** type through-hole pins. Pins specified in the **noexpose** list in the design file are not affected by the **fanout** command.

If a <*net_descriptor*> includes **source**, **load**, or **terminator** pin lists, it must also include the **reorder daisy** rule. The autorouter routes the net in daisy-chain fashion, combining the source, load, and terminator pins into a single daisy chain with the source pins at one end, the load pins in the middle, and the terminator pins at the other end.

The following example shows a design file entry for a <*net_descriptor*> with **source**, **load**, and **terminator** pin lists:

```
(net net1
   (pins U1-1 U2-1 U3-1 U4-1)
   (source U2-1)
   (load U3-1 U4-1)
   (rule (reorder daisy))
)
```

See *<component_order_descriptor>* for details about ordering nets using component reference designators.

## *<net_id>*

<*net_id*>:: = <*id*>

## *<net_out_descriptor>*

<*net_out_descriptor*>::=
(**net** <*net_id*>
 [(**net_number** <*integer*>)]
 [*<rule_descriptor>*]
 {[*<wire_shape_descriptor>* **|** *<wire_guide_descriptor>* **|**
  *<wire_via_descriptor>* **|** *<bond_shape_descriptor>*]}
 {[*<supply_pin_descriptor>*]}
)

The <*supply_pin_descriptor*> identifies wire shapes in routes and session files that are used as source terminals. Other shapes assigned to the same net are routed directly to the source terminal. See the *<supply_pin_descriptor>* for more details.

## *<net_pair_descriptor>*

<*net_pair_descriptor*>::=
(**nets** <*net_id*> <*net_id*>
 {[(**gap** [<*positive_dimension*> **|** **-1**] {[(**layer** <*layer_id*>)]})]}
)

Use **nets** to identify the two nets you want included in a pair. Use question marks (?) as wildcards to specify multiple pairs in which the nets have similar names. The wildcards must appear in the same position in each <*net_id*>. For example, to pair net sig1A with net sig1B and net sig2A with net sig2B, specify

(nets sig?A sig?B)

Use **gap** to control the minimum distance (<*positive_dimension*>) allowed between the two routed wires in the pair. If **gap** is not included in a <*net_pair_descriptor*>, the wire-to-wire clearance rule is used. To reset a specified **gap** to the default wire-to-wire clearance, use **-1** for the **gap** value.

You can use the **layer** keyword to apply the **gap** value to only the layer identified in <*layer_id*>.

## *<net_pin_changes_descriptor>*

<*net_pin_changes_descriptor*>::=
(**net_pin_changes**
{(**net** <*net_id*>
[(**add_pins** {*<pin_reference>*})]
[(**delete_pins** {*<pin_reference>*})])}
)

This descriptor only appears in a session file and indicates pin changes that were made to the design during the session. The **add_pins** option lists the pin references for the added pins, and the **delete_pins** option lists the pin references for the deleted (forgotten) pins.

The <*net_id*> can be the name of a net not specified in the *<network_descriptor>* of the design file if you defined the net in the session.

## *<network_descriptor>*

<*network_descriptor*>::=
(**network**
{<net_descriptor>}
[{*<class_descriptor>*}]
[{*<class_class_descriptor>*}]
[{*<group_descriptor>*}]
[{*<group_set_descriptor>*}]
[{*<pair_descriptor>*}]
[{*<bundle_descriptor>*}]
)

## *<network_out_descriptor>*

<*network_out_descriptor*>::=
(**network_out** {*<net_out_descriptor>*})

## *<no_of_large_components>*

<*no_of_large_components*>::= <*positive_integer*>

## *<noise_accumulation_descriptor>*

<*noise_accumulation_descriptor*>::=
(**noise_accumulation** [**RSS** | **linear**])

Use the <*noise_accumulation_descriptor*> to control whether total accumulated noise is calculated as the sum of the noise at each interaction between a victim net and the aggressor nets (**linear**) or as the root of the sum of the squares (**RSS**). The default is **linear**.

Either method looks up lengths for each gap specified in the noise table provided by **parallel_noise** and **tandem_noise** rules.

## *<noise_calculation_descriptor>*

<*noise_calculation_descriptor*>::=
(**noise_calculation**
[**linear_interpolation** | **stairstep**])

Accumulated noise calculation uses either stairstep or linear interpolation when looking up noise values from the noise table provided by **parallel_noise** and **tandem_noise** rules. The default is **stairstep**.

## *<number>*

<*number*>::=
[*<sign>*] [*<positive_integer>* | *<real>* | *<fraction>*]

Exponential numbers are not supported.

## *<numeric_binary_operator>*

<*numeric_binary_operator*>::=
[**==** | **!=** | **<** | **>** | **<=** | **>=** | **+** | **-** | ***** | **/** | **%** | **&&** | **||**]

When more than one operator is used in an expression, rules of precedence determine the order of evaluation. Evaluation of operators at the same precedence level in a single expression is from left to right.

The following table lists the numeric binary operators in descending order of precedence, with the highest precedence operator at the top. Operators that have the same precedence level are grouped together. For example, multiply, divide, and modulo have the same precedence level.

| Operator | Function |
|---|---|
| ( ) | grouping |
| - | negation |
| ! | logical NOT |
| *<br>/<br>% | multiply<br>divide<br>modulo |
| +<br>- | add or concatenate<br>subtract |
| <<br>><br><=<br>>= | less than<br>greater than<br>less than or equal to<br>greater than or equal to |
| ==<br>!= | equal to<br>not equal to |
| && | logical AND |
| \|\| | logical OR |

## *<numeric_expression>*

*<numeric_expression>*::=
[*<numeric_expression>* *<numeric_binary_operator>*
*<numeric_expression>* |
*<numeric_unary_operator>* *<numeric_expression>* |
*<string_expression>* *<string_compare_operator>*
*<string_expression>* |
(*<numeric_expression>*) | *<integer>* | *<float>* |
*<variable_name>*]

## *<numeric_unary_operator>*

<*numeric_unary_operator*>::= [**-** | **!**]

For an explanation of operator precedence, see the *<numeric_binary_operator>* syntax note.

## *<object_type>*

<*object_type*>::=
[**pin** | **smd** | **via** | **wire** | **area** | **testpoint**]

Object types are defined in the following table.

| Types | Description |
|---|---|
| pin | Through-pin shapes or oval pin shapes using a <*path_descriptor*> |
| smd | Surface mount pad shapes |
| via | Via shapes |
| wire | Wire shapes using a <*path_descriptor*> |
| area | Keepout, boundary, or wire shapes using a <*polygon_descriptor*> |
| testpoint | Through-pins or vias marked as test points because a **testpoint** rule is in effect for the net containing the pin or via |

## *<off_grid_descriptor>*

<*off_grid_descriptor*>::=
(**off_grid** [**on** | **off**])

The **off_grid** option controls whether off grid routing is permitted or prohibited. The default is **on**. When **off_grid off** is set in the design file, off grid routing is prohibited. You can also use the **cost off_grid forbidden** command to prohibit off grid routing.

## *<opposite_side_descriptor>*

*<opposite_side_descriptor>*::=
(**opposite_side** [**on**| **off**]
[(**type** {[**large_large** | **large_small** | **small_small**]})]
)

This rule controls the back-to-back placement of one type of component (large or small) with respect to the same or other type of component. The default is **opposite_side on**, which means opposite placement is permitted for all types of components. You can also use the **place_rule** command to control opposite side placement.

## *<order_type>*

*<order_type>*::=
[**starburst** | **daisy** [*<daisy_type>*]]

Use **starburst** when multiple entries and exits on pins are permitted in your design. Use **daisy** to order a net as a simple daisy chain, which permits a single entry and a single exit on each pin in the net and does not allow tjunctions. Use <*daisy_type*> to specify mid-driven or balanced daisy chain routing.

## *<orientation>*

*<orientation>*::=
[**0** | **90** | **180** | **270**]

See also *<permit_orient_descriptor>*.

## *<outline_descriptor>*

*<outline_descriptor>*::=
(**outline** *<shape_descriptor>*)

The **outline** shape can be a path, polygon, rectangle, or circle. The <*shape_descriptor*> must be defined from the top view of the image. For example:

```
(outline (rect signal_1 0.0000 0.0000 1.2500 0.3250))
(outline (polygon signal_1 0.0 0.0550 0.0000 0.4100
      0.0000 0.4650 0.0550 0.4650 0.4250 0.4250 0.4650
      0.0550 0.4650 0.0000 0.4100 0.0000 0.0550 0.0550
      0.0000))
```

The <*outline_descriptor*> of an image is used to assure optimum component-to-component spacing during placement. The layer of the outline shape is ignored.

See also *<image_descriptor>*.

## *<padstack_descriptor>*

<*padstack_descriptor*>::=
(**padstack** <*padstack_id*>
  [*<unit_descriptor>*]
  {(**shape** *<shape_descriptor>*
    [*<reduced_shape_descriptor>*]
    [(**connect** [**on** | **off**])]
    [{*<window_descriptor>*}]
  )}
  [*<attach_descriptor>*]
  [{*<pad_via_site_descriptor>*}]
  [(**rotate** [**on** | **off**])]
  [(**absolute** [**on** | **off**])]
  [(**rule** *<clearance_descriptor>*)]
)

The following table explains the main keyword parameters used in the <*padstack_descriptor*>.

| **Keyword** | **Description** |
|---|---|
| shape <*shape_descriptor*> | Controls the geometry of the padstack. |
| connect | Controls connection of a wire to the padstack. The default is **on**. This control applies only to vias and component through-pins. |
| rotate | Controls whether a pin padstack rotates when a component is rotated. The default is **on**. If **rotate** is turned **off**, any pin rotation, flipping, or mirroring that results from component placement or rotation that you specify is ignored. |

| Keyword | Description |
|---|---|
| absolute | Controls whether the Z-direction stackup of a pin padstack is flipped 180 degrees (Z rotation) when a component is placed on the back side of the PCB. If **absolute** is **on**, the padstack is not flipped. By default, **absolute** is **off** and pin padstacks are flipped when components are flipped. |
| rule *<clearance_descriptor>* | Assigns clearance rules for the padstack. |

The padstack origin must be inside at least one shape of the padstack.

The *<attach_descriptor>* controls whether a via padstack can be positioned under an SMD pad. The default is **on**, which allows vias under SMD pads. The **via_at_smd** rule must also be turned **on** to place vias under SMD pads (the default **via_at_smd** rule is **off**).

The *<pad_via_site_descriptor>* controls the location of a via placed under an SMD pad relative to the padstack's origin.

The *<reduced_shape_descriptor>* places an optional, smaller shape in a pin padstack. If the autorouter has difficulty in the converge routing phase, this smaller shape can be substituted to increase wiring space. Substitution is permitted on any layer where the padstack is not connected. The presence of a reduced shape in a design file indicates that the layout system supports reduced shapes.

If a padstack includes shapes on signal layers that have one or more intervening power layers, the router can connect through the padstack to the power layers even though padstack shapes are not included on the power layers. By contrast, a padstack must have a shape on a power layer to form a connection on that layer when the power layer is not bounded by two signal layers.

In the following example, a single via is defined by padstack V1_2 for a four-signal-layer, two-power-layer PCB. Both power layers can be accessed from layers L1 and L2 with this single via.

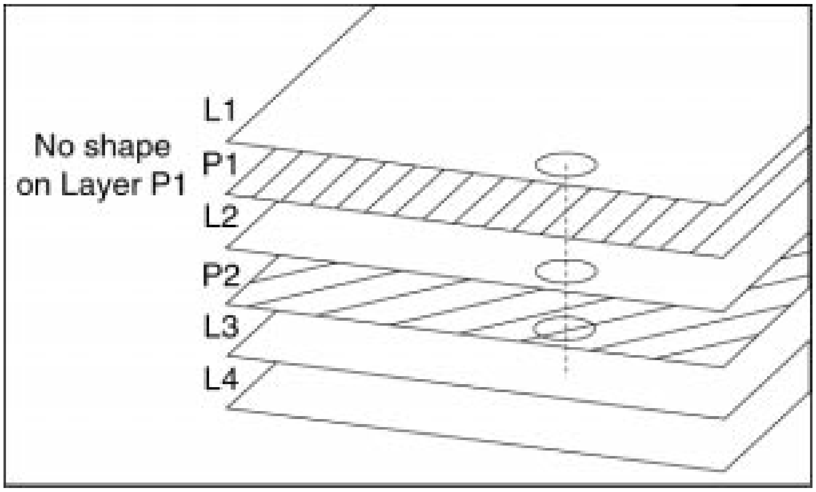
```
(padstack V1_2
   (shape (circle L1 0.2360))
   (shape (circle L2 0.2360))
   (shape (circle P2 0.3100))
)
```

Padstack shape versus layer connectivity is shown in the table below.

| Layer | Type | Shape | Connected |
|---|---|---|---|
| L1 | signal | yes | yes |
| P1 | power | no | yes |
| L2 | signal | yes | yes |
| P2 | power | yes | yes |
| L3 | signal | no | no |
| L4 | signal | no | no |

This relationship can also be illustrated by the following figure.

**Single Via Defined for PCB with Four Signal Layers and Two Power Layers**




## *<padstack_id>*

*<padstack_id>*::= *<id>*

## *<pad_via_site_descriptor>*

<*pad_via_site_descriptor*>::=
(**via_site** *<vertex>* **| off**)

Use the pad via site rule to place a via under an SMD pad at a specific location such as the edge of an SMD padstack. The <*vertex*> value is a coordinate relative to the padstack origin. To remove the via site data, specify **off**.

## *<pair_descriptor>*

<*pair_descriptor*>::=
(**pair** {[*<wire_pair_descriptor>* **|** *<net_pair_descriptor>*]})

The <*pair_descriptor*> defines differential pairs. Differential pairs are two nets or wires that you want to route side by side with the same topology for each connection.

Use the <*net_pair_descriptor*> to define a pair as two nets and the <*wire_pair_descriptor*> to define a pair as two fromtos (pin-to-pin connections).

## *<parallel_noise_descriptor>*

<*parallel_noise_descriptor*>::=
(**parallel_noise**
    [**off |**
    (**gap** <*positive_dimension*>)
    [(**threshold** <*positive_dimension*>)]
    (**weight** <*real*>)]
)

Noise coupling between nets is controlled by computing the total noise that impinges on receiving nets from surrounding transmitting nets. Each net in a design can have a different noise weight or transmitting characteristic. A net's noise weight determines how much noise it transmits. Each net can also have a different maximum noise specification or receiving characteristic. The maximum noise specification determines how much noise a net can accumulate or pick up from other nets before a noise violation occurs. See also *<max_noise_descriptor>*.

The following table describes <*parallel_noise_descriptor*> keywords.

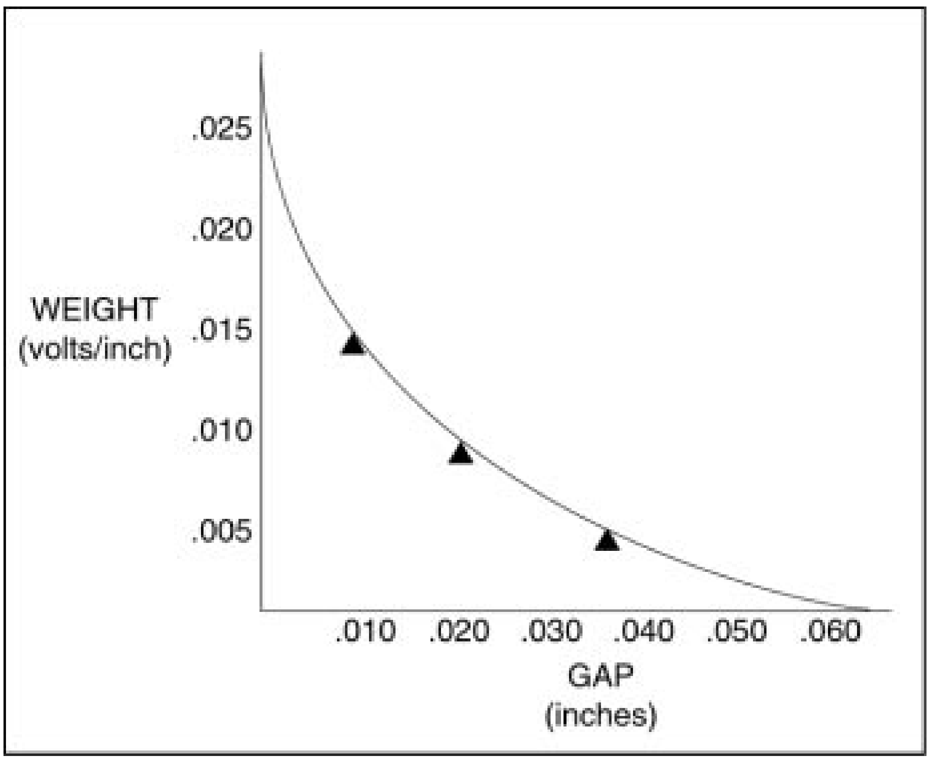
| Keyword | Description |
|---|---|
| off | Resets a rule to the unspecified state. To change existing parallel noise rules, always use **parallel_noise off** before specifying new rules. |
| gap | Is measured edge-to-edge between parallel wires on the same layer. Coupled noise is calculated for parallel wires when the edge-to-edge distance is equal to or less than the specified **gap** value and the wires are parallel for a distance that exceeds the **threshold** value. If a wire does not have a **max_noise** value, no noise is computed for that wire. |
| threshold | Is the minimum parallel length that is considered when parallel noise violations are computed. When **threshold** is unspecified, its value defaults to the **gap** value. |
| weight | Represents units of noise per unit of length, where the unit of noise is typically volts or millivolts and the unit of length is the current dimensional unit. The **weight** value corresponds to the noise transmitted by the net over a unit length of wire to surrounding wires. the tool computes the noise coupled from a parallel transmitting wire by multiplying the transmitting wire's parallel length by its **weight** value. All coupled noise sources are accumulated for each receiving net and the sum is compared against that net's maximum noise specification to determine if a violation exists. |

A coupled noise weight and gap curve can be approximated for a net by entering two or more weight and gap rules. For example:

```
unit inch
rule net clk1 (parallel_noise (gap .010) (threshold .050)
   (weight .015))
(parallel_noise (gap .020) (threshold .100)
   (weight .010))
(parallel_noise (gap .036) (threshold .100)
   (weight .005))
```

The following illustration shows the approximation.

**Coupled Noise Weight Versus Gap Approximation**




If multiple **parallel_noise** rules apply to the same net, at different precedence levels, violations are checked only for the highest level rule. For the order of routing rule precedence, see the Routing and Placement Rule Hierarchies section at the beginning of this manual.

## *<parallel_segment_descriptor>*

<*parallel_segment_descriptor*>::=
(**parallel_segment**
  [**off** |
  (**gap** <*positive_dimension*>)
  (**limit** <*positive_dimension*>)]
)

Parallelism between wire segments on the same layer is controlled by setting a parallel segment length limit and a minimum wire-to-wire gap.

The following table describes <*parallel_segment_descriptor*> keywords.

| Keyword | Description |
|---|---|
| off | Resets the rule to the unspecified state. To change existing parallel segment rules, always use **parallel_segment off** before specifying new rules. |
| gap | Is measured edge-to-edge between parallel wire segments on the same layer. Parallel segment violations do not occur when **gap** is greater than the specified value. |
| limit | Is the maximum parallel length that is allowed before a parallel segment violation occurs. When **limit** is unspecified or is less than the **gap** value, the **limit** value defaults to the **gap** value. |

Power nets are not included in parallel segment rule checking. The following example illustrates how a table of rules can be created by supplying multiple parallel segment rules.

```
(Net clk1
   (pins...)
(rule (parallel_segment (gap 11) (limit 500))
   (parallel_segment (gap 14) (limit 1200))
   (parallel_segment (gap 16) (limit 1800))
)
)
```

For the two parallel wire segments, violations occur when:

- **gap** is less than or equal to 11 and the parallel length is greater than 500
- **gap** is less than or equal to 14 and the parallel length is greater than 1200
- **gap** is less than 16 and the parallel length is greater than 1800

If multiple **parallel_segment** rules apply to a wire, at different precedence levels, violations are checked only for the highest level rule. For the order of routing rule precedence, see the Routing and Placement Rule Hierarchies section at the beginning of this manual.

## *<parser_descriptor>*

*<parser_descriptor>*::=
(**parser**
[(**string_quote** *<quote_char>*)]
(**space_in_quoted_tokens** [**on | off**])
[(**host_cad** *<id>*)]
[(**host_version** *<id>*)]
[{(**constant** *<id>* *<id>*)}]
[(**write_resolution**] {*<character>* *<positive_integer>*})]
[(**routes_include** {[**testpoint | guides |**
**image_conductor**]})]
[(**wires_include testpoint**)]
[(**case_sensitive** [**on | off**])]
[(**via_rotate_first** [**on | off**])]
)

The **parser** keyword embeds information about the PCB layout in your design file.

The following table describes the types of data that can be included in the parser section of the design file.

| Keyword | Description |
|---|---|
| string_quote | Temporarily disables parentheses as delimiters for text strings. A blank space is an absolute delimiter in a design file unless you set<br><br>space_in_quoted_tokens on<br><br>Once you define the <*quote_char*>, you can use it to include parentheses in <*id*> strings such as net names, component names, and layer names. You must include a blank space after the closing string quote character.<br><br>Within a command, if a name or other <*id*> string includes a parentheses, enclose the string within string quote characters. For example, if the string quote character is the single quotation mark, you can enter the command<br><br>select net 'DATA_BUS(0)'<br><br>Valid string quote characters are single quotation mark ('), double quotation mark ("), and dollar sign ($). There is no default string quote character. |
| space_in_quoted_tokens | Controls the use of blank spaces within quoted strings. By default (**off**), blank spaces are an absolute delimiter. **For example, a blank space indicates the end of the string if its used within a quoted string.** When **on**, blank spaces are permitted within quoted strings. You must use the closing quote to end a string. |
| host_version | Identifies layout system version. |
| host_cad | Identifies the layout system. |

| Keyword | Description |
|---|---|
| constant | Generates two constant <*id*> strings. These character strings pass from the design file to the routes file. The translator uses them to generate constant information in the layout system interface file. |
| write_resolution | Defines the dimensional units and resolution of the data in the translated design file. |
| routes_include testpoint | Forces the **write routes** command to include testpoint records in routes files. |
| routes_include guides | Forces the **write routes** command to include guides information automatically in routes files. |
| routes_include image_conductor | Forces the **write routes** and **write wires** commands to include wires and vias embedded within an image in the routes or wires file. |
| wires_include testpoint | Forces the **write wire** command to include testpoint records in wires files. |
| case_sensitive | Controls case-sensitivity for object names. For example, by default (**off**), the tool recognizes two nets called clk and CLK as a single net. When **on**, the tool recognizes clk and CLK as separate nets. |
| via_rotate_first | Controls whether vias rotate or mirror first. For example, by default (**on**), rotation is done before mirroring. When **off**, mirroring is done before rotation. The **via_rotate_first** control prevents data translation discrepancies between Allegro and other layout systems that do mirroring first. |

## *<part_library_descriptor>*

*<part_library_descriptor>*::=
(**part_library**
[{*<physical_part_mapping_descriptor>*}]
{*<logical_part_mapping_descriptor>*}
[{*<logical_part_descriptor>*}]
[*<directory_descriptor>*]
)

The *<part_library_descriptor>* describes the equivalency of gates, subgates, and pins.

- A gate is a set of pins for which net connections can be swapped between components or within a component. A gate consists of all the input and output pins of a functional block.
- A subgate is a set of pins for which net connections can be swapped only within a gate. A subgate usually consists of only a subset of the input pins in a functional block.

You can replace all logical part descriptors with a directory descriptor that identifies a common user library directory. When a directory descriptor is used, the tool expects to find one or more files that contain logical part information.

You can combine one or more logical part descriptors with a directory descriptor in the same part library descriptor. For example:

```
(part_library
(physical_part_mapping MC54HC688 (component U1 U2))
(logical_part_mapping SN54HC688 (physical MC54HC688) (component U3 U4))
(logical_part_mapping SN54HC804 (comp U5 U6 U7))
(logical_part_mapping SN54HC139 (comp U9 U10))
(directory /usr/designer/library)
)
```

The following information explains the example:

- The logic definition for components U1, U2, U3, and U4 is contained in part SN54HC688.
- The logic definition for components U5, U6, and U7 is defined by part SN54HC804.
- MC54HC688 and SN54HC688 are logically equivalent.
- The logic definition for components U9 and U10 is contained in SN54HC139.

A logical part descriptor looks like a table in the design file. Note that pin IDs in the logical part table must be the same as the pin IDs used with the *<reference_descriptor>* in the library image definition.

The following pages show examples of logical part descriptors for the parts SN54HC688, SN54HC804, and SN54HC139. The <*logical_part_id*> is a filename consisting of the name of the part followed by .part. For example, the logical part SN54HC688 has a filename of SN54HC688.part. The SN54HC688.part file contains:

(logical_part SN54HC688

| **#Physical #Pin ID** | **Pin Type** | **Gate** | **Gate Swap** | **Gate Pins** | **Gate Pin Swap** | **Subgate** | **Subgate Swap** | **Subgate Pins** |
|---|---|---|---|---|---|---|---|---|
| (pin 1 | 3 | gate1 | 1 | 1 | 0 | sub1 | 0 | 1 ) |
| (pin 2 | 3 | gate1 | 1 | 2 | 1 | sub2 | 1 | 1 ) |
| (pin 3 | 3 | gate1 | 1 | 3 | 1 | sub2 | 1 | 2 ) |
| (pin 4 | 3 | gate1 | 1 | 4 | 2 | sub2 | 1 | 3 ) |
| (pin 5 | 3 | gate1 | 1 | 5 | 2 | sub2 | 1 | 4 ) |
| (pin 6 | 3 | gate1 | 1 | 6 | 3 | sub2 | 1 | 5 ) |
| (pin 7 | 3 | gate1 | 1 | 7 | 3 | sub2 | 1 | 6 ) |
| (pin 8 | 3 | gate1 | 1 | 8 | 1 | sub3 | 1 | 1 ) |
| (pin 9 | 3 | gate1 | 1 | 9 | 1 | sub3 | 1 | 2 ) |
| (pin 10 | 2 | gate1 | 1 | 10 | 0 | sub5 | 0 | 1 ) |
| (pin 11 | 3 | gate1 | 1 | 11 | 2 | sub3 | 1 | 3 ) |
| (pin 12 | 3 | gate1 | 1 | 12 | 2 | sub3 | 1 | 4 ) |
| (pin 13 | 3 | gate1 | 1 | 13 | 3 | sub3 | 1 | 5 ) |
| (pin 14 | 3 | gate1 | 1 | 14 | 3 | sub3 | 1 | 6 ) |
| (pin 15 | 3 | gate1 | 1 | 15 | 1 | sub4 | 0 | 1 ) |
| (pin 16 | 3 | gate1 | 1 | 16 | 1 | sub4 | 0 | 2 ) |
| (pin 17 | 3 | gate1 | 1 | 17 | 2 | sub4 | 0 | 3 ) |
| (pin 18 | 3 | gate1 | 1 | 18 | 2 | sub4 | 0 | 4 ) |
| (pin 19 | 4 | gate1 | 1 | 19 | 0 | sub6 | 0 | 1 ) |
| (pin 20 | 2 | gate1 | 1 | 20 | 0 | sub7 | 0 | 1 ) |
| ) | | | | | | | | |

The file SN54HC804.part contains:

(logical_part SN54HC804

| #Physical #Pin ID | Pin Type | Gate | Gate Swap | Gate Pins | Gate Pin Swap | Subgate | Subgate Swap | Subgate Pins |
|---|---|---|---|---|---|---|---|---|
| (pin 1 | 3 | gate1 | 2 | 1 | 1 ) | | | |
| (pin 2 | 3 | gate1 | 2 | 2 | 1 ) | | | |
| (pin 3 | 4 | gate1 | 2 | 3 | 0 ) | | | |
| (pin 4 | 3 | gate2 | 2 | 1 | 1 ) | | | |
| (pin 5 | 3 | gate2 | 2 | 2 | 1 ) | | | |
| (pin 6 | 4 | gate2 | 2 | 3 | 0 ) | | | |
| (pin 7 | 3 | gate3 | 2 | 1 | 1 ) | | | |
| (pin 8 | 3 | gate3 | 2 | 2 | 1 ) | | | |
| (pin 9 | 4 | gate3 | 2 | 3 | 0 ) | | | |
| (pin 10 | 2 | gate7 | 0 | 1 | 0 ) | | | |
| (pin 11 | 4 | gate4 | 2 | 3 | 0 ) | | | |
| (pin 12 | 3 | gate4 | 2 | 1 | 1 ) | | | |
| (pin 13 | 3 | gate4 | 2 | 2 | 1 ) | | | |
| (pin 14 | 4 | gate5 | 2 | 3 | 0 ) | | | |
| (pin 15 | 3 | gate5 | 2 | 1 | 1 ) | | | |
| (pin 16 | 3 | gate5 | 2 | 2 | 1 ) | | | |
| (pin 17 | 4 | gate6 | 2 | 3 | 0 ) | | | |
| (pin 18 | 3 | gate6 | 2 | 1 | 1 ) | | | |
| (pin 19 | 3 | gate6 | 2 | 2 | 1 ) | | | |
| (pin 20 | 2 | gate8 | 0 | 1 | 0 ) | | | |
| ) | | | | | | | | |

The file SN54HC139.part contains:

(logical_part SN54HC139

| #Physical #Pin ID | Pin Type | Gate | Gate Swap | Gate Pins | Gate Pin Swap | Subgate | Subgate Swap | Subgate Pins |
|---|---|---|---|---|---|---|---|---|
| (pin 1 | 3 | gate1 | 3 | 1 | 0 | subg1 | 0 | 1 ) |
| (pin 2 | 3 | gate1 | 3 | 2 | 0 | subg1 | 0 | 2 ) |
| (pin 3 | 3 | gate1 | 3 | 3 | 0 | subg1 | 0 | 3 ) |
| (pin 4 | 4 | gate1 | 3 | 4 | 0 | subg1 | 0 | 4 ) |
| (pin 1 | 3 | gate1 | 3 | 1 | 0 | subg2 | 0 | 1 ) |
| (pin 2 | 3 | gate1 | 3 | 2 | 0 | subg2 | 0 | 2 ) |
| (pin 3 | 3 | gate1 | 3 | 3 | 0 | subg2 | 0 | 3 ) |
| (pin 5 | 4 | gate1 | 3 | 5 | 0 | subg2 | 0 | 4 ) |
| (pin 1 | 3 | gate1 | 3 | 1 | 0 | subg3 | 0 | 1 ) |
| (pin 2 | 3 | gate1 | 3 | 2 | 0 | subg3 | 0 | 2 ) |
| (pin 3 | 3 | gate1 | 3 | 3 | 0 | subg3 | 0 | 3 ) |
| (pin 6 | 4 | gate1 | 3 | 6 | 0 | subg3 | 0 | 4 ) |
| (pin 1 | 3 | gate1 | 3 | 1 | 0 | subg4 | 0 | 1 ) |
| (pin 2 | 3 | gate1 | 3 | 2 | 0 | subg4 | 0 | 2 ) |
| (pin 3 | 3 | gate1 | 3 | 3 | 0 | subg4 | 0 | 3 ) |
| (pin 7 | 4 | gate1 | 3 | 7 | 0 | subg4 | 0 | 4 ) |
| (pin 14 | 3 | gate2 | 3 | 1 | 0 | subg1 | 0 | 1 ) |
| (pin 13 | 3 | gate2 | 3 | 2 | 0 | subg1 | 0 | 2 ) |
| (pin 15 | 3 | gate2 | 3 | 3 | 0 | subg1 | 0 | 3 ) |
| (pin 12 | 4 | gate2 | 3 | 4 | 0 | subg1 | 0 | 4 ) |
| (pin 14 | 3 | gate2 | 3 | 1 | 0 | subg1 | 0 | 1 ) |
| (pin 13 | 3 | gate2 | 3 | 2 | 0 | subg2 | 0 | 2 ) |
| (pin 15 | 3 | gate2 | 3 | 3 | 0 | subg2 | 0 | 3 ) |

| #Physical #Pin ID | Pin Type | Gate | Gate Swap | Gate Pins | Gate Pin Swap | Subgate | Subgate Swap | Subgate Pins |
|---|---|---|---|---|---|---|---|---|
| (pin 11 | 4 | gate2 | 3 | 5 | 0 | subg2 | 0 | 4 ) |
| (pin 14 | 3 | gate2 | 3 | 1 | 0 | subg3 | 0 | 1 ) |
| (pin 13 | 3 | gate2 | 3 | 2 | 0 | subg3 | 0 | 2 ) |
| (pin 15 | 3 | gate2 | 3 | 3 | 0 | subg3 | 0 | 3 ) |
| (pin 10 | 4 | gate2 | 3 | 6 | 0 | subg3 | 0 | 4 ) |
| (pin 14 | 3 | gate2 | 3 | 1 | 0 | subg4 | 0 | 1 ) |
| (pin 13 | 3 | gate2 | 3 | 2 | 0 | subg4 | 0 | 2 ) |
| (pin 15 | 3 | gate2 | 3 | 3 | 0 | subg4 | 0 | 3 ) |
| (pin 9 | 4 | gate2 | 3 | 7 | 0 | subg4 | 0 | 4 ) |
| (pin 8 | 2 | gate3 | 0 | 1 | 0 ) | | | |
| (pin 16 | 2 | gate4 | 0 | 1 | 0 ) | | | |
| ) | | | | | | | | |

Note the following equivalency of gates, subgates, and pins shown in the preceding files.

- The gates identified by gate1, gate2, gate3, gate4, gate5, and gate6 of part SN54HC804 are swappable, and the subgates identified by sub2 and sub3 of part SN54HC688 are swappable.
- The gates identified by gate1 on part SN54HC688 and gate1 on part SN54HC804 are not swappable because the *<gate_swap_code>* for each is different.
- The pins identified by pin 2 and pin 3 on part SN54HC688 are swappable since they have the same *<gate_pin_swap_code>*, but pin 3 and pin 4 of part SN54HC688 are not swappable because their swap codes are different. Pin 2 and pin 8 of part SN54HC688 are not swappable because they are in different subgates.
- Part SN54HC139, a dual 2-line to 4-line decoder, is an example of a package with common pins. Physical pins 1, 2, 3, 13, 14, and 15 are common pins. Subgates subg1, subg2, subg3, and subg4 of gate1 and gate2 are not swappable because their outputs must be in order. Gates gate1 and gate2 are swappable.

## *<part_number>*

*<part_number>*::= *<id>*

## *<part_pin_descriptor>*

*<part_pin_descriptor>*::=
(**pin** *<pin_id>* *<pin_type>* *<gate_id>*
*<gate_swap_code>* *<gate_pin_id>* *<gate_pin_swap_code>*
[*<subgate_id>* *<subgate_swap_code>* *<subgate_pin_id>*]
)

When a component has a pin that is common to two or more gates, the same *<pin_id>* is used with different *<gate_id>*s to construct the *<part_pin_descriptor>*.

See also *<logical_part_descriptor>*.

## *<passes>*

*<passes>*::= *<positive_integer>*

## *<path_descriptor>*

*<path_descriptor>*::=
(**path**
*<layer_id>*
*<aperture_width>* {*<vertex>*}
[(**aperture_type** [**round** | **square**])]
)

A path is drawn by moving the aperture through all vertexes in straight lines. The **path** keyword is used to define wires, oval pins, and the PCB boundary. The default **aperture_type** is **round**.

## *<pattern_name>*

*<pattern_name>*::=
[**brickpat | cctpat | checkpat | diaghatchpat | dotpat | empty | gridpat | horizdashpat | horizpat | horizwavepat | orthohatchpat | peakpat | plaidpat | pluspat | slantleftpat | slantrightpat | tilepat | vertdashpat | vertpat | vertwavepat |**
*<bit_map_filename>*]

<*bit_map_filename*>::= user-defined bit map filename with a *.bit* extension. The filename without the *.bit* extension becomes the user-defined pattern name.

## *<pattern_object>*

<*pattern_object* >::=
[**component front** | **component back** | **keepouts** | **pin** | **poly_wire** |
**power** *<layer_number>* |
**signal** *<layer_number>* | **viakeepouts** | **vias**]

## *<pcb_id>*

<*pcb_id*>::= <*id*>

## *<permit_orient_descriptor>*

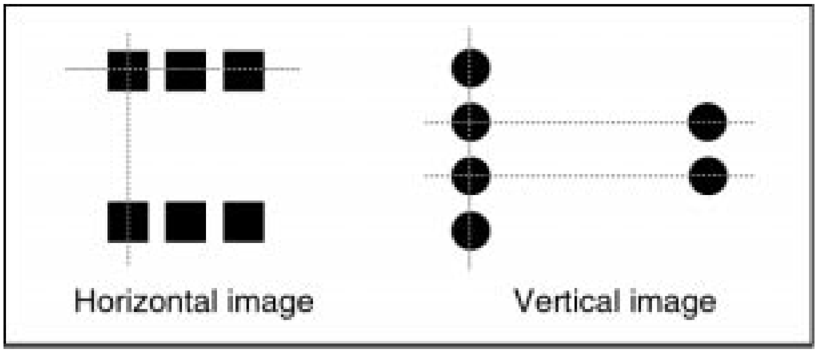
<*permit_orient_descriptor*>::=
(**permit_orient**
[**-1** |
{*<orientation>*} |
**horizontal** |
**vertical**]
[(**side** *<place_side>*)]
)

A **permit_orient** value of **-1** sets the rule to unspecified. When permit_orient is not specified, components can be interactively placed at any angle in increments of one degree.

An image is horizontal or vertical based on its footprint. Image footprints are analyzed by examining the rows and columns of pins. A row is a horizontal array of pins that have the identical Y-coordinate. A column is a vertical array of pins that have the identical X-coordinate.

When the number of pins in a row is greater than the number of pins in any column, the image is horizontal. When the number of pins in a column is greater than the number of pins in any row, the image is vertical. If the largest row and the largest column of pins are equal in number, the lengths of the rows and columns of pins are considered to determine whether the image is horizontal or vertical. If the row and column lengths are also equal, the image is neither horizontally nor vertically oriented. The following figure shows examples of horizontal and vertical images.

**Horizontal and Vertical Images**




See also *<place_rule_descriptor>*.

## *<permit_side_descriptor>*

*<permit_side_descriptor>*::=
(**permit_side** *<place_side>*)

See also *<place_rule_descriptor>*.

## *<physical_part_id>*

*<physical_part_id>*::= *<id>*

See also *<physical_part_mapping_descriptor>* and *<logical_part_mapping_descriptor>*.

## *<physical_part_mapping_descriptor>*

*<physical_part_mapping_descriptor>*::=
(**physical_part_mapping**
  *<physical_part_id>* [(**component** {*<component_id>*}) **|**
  (**image** {*<image_id>*})]
)

See also *<part_library_descriptor>*.

## *<physical_property_descriptor>*

<*physical_property_descriptor*>::=
{[(**type** {[**capacitor** | **resistor** | **discrete** | **small** | **large**]}) |
(**height** *<max_height>*) |
(**power_dissipation** <*real*>)
]}

By default, the placer treats components with three or fewer pins as **small**, and components with more than three pins as **large**. The **large** and **small** types are mutually exclusive. You can assign **type large** to a component or image with three or fewer pins. You cannot assign **type small** to a component or image with more than three pins.

The **capacitor**, **resistor**, and **discrete** types are mutually exclusive. Assigning one type removes a previously assigned mutually exclusive type. You can assign **type capacitor**, **type resistor**, or **type discrete** to any large or small component or image, but only small capacitors, resistors, or discretes can be specified for processing in automatic placement.

A capacitor is defined as a decoupling (bypass) capacitor. If a component with three pins or fewer that are all connected to power nets has not been assigned **type large**, **type resistor**, or **type discrete**, the placer automatically treats it as **type capacitor**.

The **height** property assigns a maximum height value. The **power_dissipation** property assigns a maximum power dissipation value. Each value must be either a positive number or a **-1**, which means the property is undefined. Note that each value must be expressed in units consistent with your design (usually milliwatts for power dissipation).

See also *<room_rule_descriptor>*.

## *<pin_array_descriptor>*

<*pin_array_descriptor*>::=
(**array**
*<begin_index>*
*<end_index>*
*<index_step>*
*<x0>*
*<y0>*
*<xstep>*
*<ystep>*
[*<pin_prefix_id>*]
[*<pin_suffix_id>*]
)

## *<pin_id>*

*<pin_id>*::= *<id>*

Pin IDs cannot include a hyphen.

## *<pin_prefix_id>*

*<pin_prefix_id>*::=
(**prefix** *<id>*)

## *<pin_reference>*

*<pin_reference>*::=
*<component_id>***-***<pin_id>*

## *<pin_suffix_id>*

*<pin_suffix_id>*::=
(**suffix** *<id>*)

## *<pin_type>*

*<pin_type>*::= *<integer>*

The following table shows pin type definitions.

| Pin Type | Definition |
|---|---|
| 0 | not specified |
| 1 | no internal connection |
| 2 | power pin |
| 3 | logical input |
| 4 | logical output |
| 5 | input or output |

See also *<part_pin_descriptor>*.

## *<pin_width_taper_descriptor>*

<*pin_width_taper_descriptor*>::=
(**pin_width_taper** [**up | down | up_down | off**]
[(**max_length** <*positive_dimension*>)]
)

The <*pin_width_taper_descriptor*> controls the width of the wire segment entering or exiting a pin. This rule is used to match the connecting segment width to the pin width, when wire and pin widths differ. (All other segments of the wire obey the width rule that applies to the wire as a whole.) No width tapering occurs if it leads to any rule violation.

If you want to permit only enlarged widths, choose the **up** option. Widths are not reduced for narrow pins. Similarly, if you want to permit only reduced widths, choose the **down** option. Widths are not enlarged for wide pins. To enable both widening and narrowing of segment widths as needed, choose the **up_down** option. Choose **off** to disable the **pin_width_taper** capability. The default is **down**.

If a pin width is smaller than the minimum wire width, as defined by a pcb or layer width rule, tapering **down** does not occur.

If you want to control the maximum length of a tapered wire segment, use **max_length**. When the wire segment entering or exiting a pin is greater than the **max_length** value, only the portion of the segment that matches the specified length is tapered. When the wire segment is less than or equal to the **max_length** value, or **max_length** is not specified, the entire segment is tapered.

Length is measured from the edge of the pin to the end point of the wire segment.

## *<place_boundary_descriptor>*

<*place_boundary_descriptor*>::=
(**place_boundary** [{*<path_descriptor>*} |
*<rectangle_descriptor>*])

The <*place_boundary_descriptor*> defines the area of the PCB that permits component placement. This boundary must be smaller than the signal boundary defined in *<boundary_descriptor>*. For example

(boundary (rect pcb -2000 -2000 2000 2000))

(boundary (rect signal -1900 -2000 1900 2000))

(place_boundary (rect signal -1800 -1800 1800 1800))

If <*place_boundary_descriptor*> is not defined, the placer uses the signal boundary as the boundary for component placement.

The <*place_boundary_descriptor*> must describe a closed boundary. The first vertex of a <*path_descriptor*> must match the last vertex of the preceding <*path_descriptor*>. If the last vertex of the last <*path_descriptor*> does not match the first vertex of the first <*path_descriptor*>, the boundary automatically closes.

If you use <*rectangle_descriptor*> to define <*place_boundary_descriptor*>, the placer does not consider the boundary to be a filled shape.

The <*layer_id*> in <*path_descriptor*> or <*rectangle_descriptor*> must be the **signal** keyword.

## *<place_control_descriptor>*

<*place_control_descriptor*>::=
(**place_control** [*<flip_style_descriptor>*])

See also *<placement_descriptor>*.

## *<place_object>*

<*place_object*>::=
[**pin** | **smd** | **area**]

The **pin** place object represents through-pin components, **smd** represents surface mount components, and **area** represents general keepouts and placement keepouts.

See also *<spacing_type>*.

## *<place_rule_descriptor>*

<*place_rule_descriptor*>::=
(**place_rule**
{[*<spacing_descriptor>* |
*<permit_orient_descriptor>* |
*<permit_side_descriptor>* |
*<opposite_side_descriptor>*]}
)

## *<place_side>*

*<place_side>*::=
[**front** | **back** | **both**]

## *<placement_descriptor>*

*<placement_descriptor>*::=
(**placement**
[*<unit_descriptor>* | *<resolution_descriptor>* | null]
[*<place_control_descriptor>*]
{*<component_instance>*}
)

## *<placement_id>*

*<placement_id>*::= *<id>*

A *<placement_id>* identifies a component reference designator.

## *<placement_reference>*

*<placement_reference>*::=
(**place**
*<component_id>*
[*<vertex>* *<side>* *<rotation>*]
[*<mirror_descriptor>*]
[*<component_status_descriptor>*]
[(**logical_part** *<logical_part_id>*]
[*<place_rule_descriptor>*]
[*<component_property_descriptor>*]
[(**lock_type** {[**position** | **gate** | **subgate** | **pin**]})]
[*<rule_descriptor>*> | *<region_descriptor>* | null]
[(**PN** *<part_number>*)]
)

If *<vertex>* is not specified, the component is placed outside the PCB boundary.

The component status descriptor only appears in the placement descriptor of the session file or placement file.

The logical part descriptor only appears in the placement descriptor of the session file or in the component instance descriptor of the placement file.

## *<plane_descriptor>*

<*plane_descriptor*>::=
(**plane**
<*net_id*>
*<shape_descriptor>*
[{*<window_descriptor>*}]
)

The <*plane_descriptor*> is used to describe split power planes. The <*plane_descriptor*> must occur after the *<layer_descriptor>* in the structure section of the design file. For example:

```
(pcb split_plane
  (structure
    (layer s1 (type signal) (direction horizontal))
    (layer p1 (type power) (use_net +5V GND))
    (layer s2 (type signal) (direction horizontal))
      (plane +5V (polygon p1 0.010  0.560  0.160
      0.560  1.480  1.00  1.480  1.00  0.700  1.280
      0.700  0.560  0.160))
      (plane GND (polygon p1 0.010  1.740  1.480
      1.740  0.160  1.300  0.160  1.300  0.720  1.740
      1.480))
    . . .
```

Two planes are defined as polygons on the power layer named p1. Note that the nets assigned to the planes are identical to those specified in the layer statement for layer p1.

## *<polygon_descriptor>*

<*polygon_descriptor*>::=
(**polygon**
<*layer_id*>
*<aperture_width>*
{*<vertex>*}
[(**aperture_type** [**round** | **square**])]
)

A polygon is a closed, filled shape. The polygon outline is drawn by moving the aperture's center point through all vertexes in straight lines. If **aperture_type** is not specified, it defaults to **round**.

For regions, the <*aperture_width*> value must be zero.

## *<positive_dimension>*

<*positive_dimension*>::=
[*<positive_integer>* | *<real>* | *<fraction>*]

## *<positive_integer>*

<*positive_integer*>::=
[*<digit>* | *<digit>*<*positive_integer*>]

## *<power_fanout_descriptor>*

<*power_fanout_descriptor*>::=
(**power_fanout** (**order**
([**pin_cap_via | pin_via_cap | none**]))
)

The **power_fanout** rule controls the routing order from pins assigned power nets to nearby decoupling capacitors during the fanout operation. The escape wire can connect first to the capacitor (**pin_cap_via**) or the escape via (**pin_via_cap**).

This rule is applicable at the pcb, class, and net levels of the rule hierarchy. For the order of routing rule precedence, see the Routing Rule Hierarchy section at the beginning of this manual.

## *<prefer_place_side>*

<*prefer_place_side*>::=
[**front_only | back_only | prefer_front | prefer_back | both**]

## *<prefix>*

<*prefix*>::= <*id*>

## *<property_value_descriptor>*

*<property_value_descriptor>*::=
(*<name_descriptor>* *<value_descriptor>*)

The <*name_descriptor*> is either a known property or a user-defined property and <*value_descriptor*> is the integer, real, or string value of the property. If you add or remove a property, or change a property value during a session, these changes do not record in the session or placement file.

The following table lists known properties for recognized pins. See also *<component_property_descriptor>* and *<image_property_descriptor>*.

| Known Property Name | Value | Description |
|---|---|---|
| exit_direction | <*string*> | The pin exit-direction property controls wire exit directions from individual pins. The exit directions are<br>left and right for the x-direction<br>top and bottom for the y-direction<br>up and down for the z-direction<br>The up direction is toward the first or top layer, and the down direction is toward the last or bottom layer. |
| force_to_terminal_point | on \| off | This is a pin property used in a pin statement to control whether a route connects to the terminal point of the pin. The terminal point is usually the center of the shape. |

## *<qarc_descriptor>*

<*qarc_descriptor*>::=
(**qarc**
<*layer_id*>
*<aperture_width>*
*<vertex> <vertex> <vertex>*
)

The <*qarc_descriptor*> is the construct used to define an arc. It cannot be used in the **pcb** boundary descriptor within the structure section of the design file.

The first <*vertex*> is the starting point of the arc. The second <*vertex*> is the endpoint of the arc. The third <*vertex*> is the center of the arc. The qarc (quarter arc) is drawn between the first and second vertexes. The four types of qarcs are

0 - 90 degrees
90 - 180 degrees
180 - 270 degrees
270 - 360 degrees

## *<real>*

<*real*>::=
[*<positive_integer>*. |
*<positive_integer>*.*<positive_integer>* |
*<positive_integer>*]

## *<rectangle_descriptor>*

<*rectangle_descriptor*>::=
(**rect** <*layer_id*> *<vertex>* *<vertex>*)

The two vertexes define the opposite corners of a rectangle. They can represent either the lower left and upper right corners or the upper left and lower right corners. If you specify upper left and lower right vertexes, the tool calculates and reports the lower left and upper right corner coordinates.

## *<reduced_shape_descriptor>*

<*reduced_shape_descriptor*>::=
(**reduced** *<shape_descriptor>*)

The <*reduced_shape_descriptor*> is added to a padstack to indicate an alternate shape, which can substitute for the normal shape. The reduced shape is smaller than the normal shape and is used if there is difficulty in the converge routing phase. The substitution can be made only if a wire does not connect to the normal shape on a particular layer.

The autorouter uses the <*reduced_shape_descriptor*> in the **reduce_padstack** command with the **on** or **auto** option, and displays an information message in the output window. An example padstack is

```
(padstack pin_60
   (shape (circle signal 60))
   (reduced (circle signal 40))
)
```

The autorouter generally uses the first shape (60). The second shape (40) is used for converge routing phase problems.

## *<redundant_wiring_descriptor>*

<*redundant_wiring_descriptor*>::=
(**allow_redundant_wiring** [**off** | **on**])

The **allow_redundant_wiring** rule is applicable only at the pcb, class, and net levels of the rule hierarchy. When this rule is **on** the checker allows redundant wiring on any specified net or nets (not just power nets) during interactive routing. The rule is ignored for nets with daisy ordering. The default is **off**.

The **allow_redundant_wiring** rule is used only when Allow Redundant Wiring On Enabled Nets is turned on in the Interactive Routing Setup dialog box.

## *<reference_descriptor>*

<*reference_descriptor*>::=
<*pin_id*> *<vertex>*

## *<region_descriptor>*

*<region_descriptor>*::=
(**region** [<*region_id*>]
[*<rectangle_descriptor>* | *<polygon_descriptor>*]
[(**region_net** <*net_id*>) |
(**region_class** <*class_id*>) |
(**region_class_class** (**classes** <*class_id*> <*class_id*>)) |
null]
(**rule** {[*<width_descriptor>* | *<clearance_descriptor>*]})
)

The <*region_descriptor*> defines a rectangular or polygon-shaped region and assigns wire width and object-to-object clearance rules to the area within the region. The tool encloses any diagonal side of the region with a rectangular corner. If you do not specify a <*region_id*>, the tool assigns one.

If regions overlap, rules assigned to each region apply to the overlap area. Rules at the higher level of the rule hierarchy take precedence over other region rules. If rules at the same level of the hierarchy conflict, the rules assigned to the most recently defined region apply. Only clearance rules can be assigned to the region class-to-class level. For the order of rule precedence, see the Routing and Placement Rule Hierarchies section at the beginning of this manual.

Layer definitions must precede any region rules in the design file, as shown in the following example.

```
(pcb area
  (structure
    (layer s1 (type signal) (direction horizontal))
    (layer p1 (type power) (use_net +5V GND))
    (layer s2 (type signal) (direction vertical))
    (region (rect signal 4.35  2.75  6.35  4.75)
      (rule (clearance 0.008 (type wire_wire))))
    (region (rect s2  2.0  2.0  6.0  6.0) (rule (width 0.003)))
  )
)
```

## *<region_id>*

<*region_id*>::= <*id*>

## *<relative_delay_descriptor>*

*<relative_delay_descriptor>*::=
(**relative_delay** [**off | on**]
{(**fromto** [*<pin_reference>* **|** (**virtual_pin** *<virtual_pin_name>*)]
[*<pin_reference>* **|** (**virtual_pin** *<virtual_pin_name>*)])
[[(**delta** *<dimension>*)] **|**
[(**tolerance** *<positive_dimension>*) **|** (**ratio_tolerance** *<real>*)] **|** null]}
)

The *<relative_delay_descriptor>* specifies delay for a fromto (pin-to-pin connection) relative to a reference fromto in the same group. Other fromtos within the group use **delta** and **tolerance** values to route relative to the reference fromto.

A fromto without a **delta** or **tolerance** is the reference fromto for the group. If **delta** and **tolerance** values are specified for every fromto in a group, the fromto with the longest manhattan length is considered the reference fromto for the group.

If you do not specify a **delta** value, the default value is 0.

If you do not specify a **tolerance** or **ratio_tolerance** value, the default is **ratio_tolerance** with a value of 5. **Ratio_tolerance** is a percentage of the reference fromto plus the **delta** and uses the same units as the **delta**.

Examples:

The first example defines a group, then applies the <relative_delay_descriptor> to the fromtos in the group.

```
(group group1
   (fromto U6-3 U12-6)
   (fromto (virtual_pin VP5) ( virtual_pin VP9))
   (fromto U7-9 U8-10)
   (circuit (relative_delay on)
      (fromto U6-3 U12-6)
      (fromto (virtual_pin VP5) ( virtual_pin VP9)) (delta -0.5) (tolerance .01)
      (fromto U7-9 U8-10) (delta 0.5) (ratio_tolerance .05)
   )
)
```

The second example applies the <relative_delay_descriptor> to the fromto as part of the group definition.

```
(group group1
   (fromto U6-3 U12-6
      (circuit (relative_delay on)))
   (fromto U6-5 U10-2
      (circuit (relative_delay on (delta -0.5) (tolerance .01))))
   (fromto U7-9 U8-10
      (circuit (relative_delay on (delta 0.5) (ratio_tolerance .05))))
)
```

## *<relative_group_delay_descriptor>*

*<relative_group_delay_descriptor>*::=
(**relative_group_delay** [**off | on**]
{(**group** *<group_id>*)
[(**delta** *<dimension>*) **|**
[(**tolerance** *<positive_dimension>*) | (**ratio_tolerance** *<real>*)] **|** null]}
)

The *<relative_group_delay_descriptor>* specifies delay for a group relative to a reference group within the same groupset. Fromtos in other groups within the groupset use **delta** and **tolerance** values to route relative to the reference group.

A group without a **delta** or **tolerance** is the reference group for the groupset. If **delta** and **tolerance** values are specified for every group in a groupset, the group with the fromto having the longest manhattan length is considered the reference group for the groupset.

If you do not specify a **delta** value, the default value is 0.

If you do not specify a **tolerance** or **ratio_tolerance** value, the default is **ratio_tolerance** with a value of 5. **Ratio_tolerance** is a percentage of the reference group plus the **delta** and uses the same units as the **delta**.

Examples:

The first example defines a groupset, then applies the <relative_group_delay_descriptor> to the groups in the groupset.

```
(group_set gpset1group1 group2 group3)
(circuit (relative_group_delay on)
   (group group1)
   (group group2 (delta -0.5) (tolerance .05))
   (group group3 (delta 0.5) (ratio_tolerance .01))
)
```

Group1 is the reference group.

The second example applies the <relative_group_delay_descriptor> to each group as part of the groupset definition.

```
(group_set gpset1 (add_group group1)
   (circuit (relative_group_delay on))
   (add_group group2)
   (circuit (relative_group_delay on (delta -0.5) (tolerance .05)))
   (add_group group3)
   (circuit (relative_group_delay on (delta 0.5) (ratio_tolerance .01)))
)
```

## *<relative_group_length_descriptor>*

*<relative_group_length_descriptor>*::=
(**relative_group_length** [**off** | **on**]
   {(**group** *<group_id>*)
   [(**delta** *<dimension>*) |
   [(**tolerance** *<positive_dimension>*) | (**ratio_tolerance** *<real>*)] | null]}
)

The *<relative_group_length_descriptor>* specifies length for a group relative to a reference group within the same groupset. Fromtos in other groups within the groupset use **delta** and **tolerance** values to route relative to the reference group.

A group without a **delta** or **tolerance** is the reference group for the groupset. If **delta** and **tolerance** values are specified for every group in a groupset, the group with the fromto having the longest manhattan length is considered the reference group for the groupset.

If you do not specify a **delta** value, the default value is 0.

If you do not specify a **tolerance** or **ratio_tolerance** value, the default is **ratio_tolerance** with a value of 5. **Ratio_tolerance** is a percentage of the reference group plus the **delta** and uses the same units as the **delta**.

Examples:

The first example defines a groupset, then applies the <relative_group_length_descriptor> to the groups in the groupset.

```
(group_set gpset1group1 group2 group3)
(circuit (relative_group_length on)
   (group group1)
   (group group2 (delta 100) (tolerance 50))
   (group group3 (ratio_tolerance .05))
)
```

The second example applies the <relative_group_length_descriptor> to each group as part of the groupset definition.

```
(group_set gpset1 (add_group group1)
   (circuit (relative_group_length on))
   (add_group group2)
   (circuit (relative_group_length on (delta 100) (tolerance 50)))
   (add_group group3)
   (circuit (relative_group_length on (ratio_tolerance .05)))
)
```

## *<relative_length_descriptor>*

*<relative_length_descriptor>*::=
(**relative_length** [**off** | **on**]
{(**fromto** [*<pin_reference>* | (**virtual_pin** *<virtual_pin_name>*)]
[*<pin_reference>* | (**virtual_pin** *<virtual_pin_name>*)])
[[(**delta** *<dimension>*)] |
[(**tolerance** *<positive_dimension>*) | (**ratio_tolerance** *<real>*)] | null]}
)

The *<relative_length_descriptor>* specifies length for a fromto (pin-to-pin connection) relative to a reference fromto in the same group. Other fromtos within the group use **delta** and **tolerance** values to route relative to the reference fromto.

A fromto without a **delta** or **tolerance** is the reference fromto for the group. If **delta** and **tolerance** values are specified for every fromto in a group, the fromto with the smallest **delta** and **tolerance** values is considered the reference fromto for the group.

If you do not specify a **delta** value, the default value is 0.

If you do not specify a **tolerance** or **ratio_tolerance** value, the default is **ratio_tolerance** with a value of 5. **Ratio_tolerance** is a percentage of the reference fromto plus the **delta** and uses the same units as the **delta**.

Examples:

The first example defines a group, then applies the <relative_length_descriptor> to the fromtos in the group.

```
(group group1
   (fromto U6-3 U12-6)
   (fromto (virtual_pin VP5) ( virtual_pin VP9))
   (fromto U7-9 U8-10)
   (circuit (relative_length on)
      (fromto U6-3 U12-6)
      (fromto (virtual_pin VP5) ( virtual_pin VP9)) (delta 100) (tolerance 50)
      (fromto U7-9 U8-10) (ratio_tolerance .05)
   )
)
```

The second example applies the <relative_length_descriptor> as part of the group definition.

```
(group group1
   (fromto U6-3 U12-6)
   (circuit (relative_length on))
   (fromto U6-5 U10-2)
   (circuit (relative_length on (delta 100) (tolerance 50)))
   (fromto U7-9 U8-10)
   (circuit (relative_length on (ratio_tolerance .05)))
)
```

## *<reorder_descriptor>*

<*reorder_descriptor*>::=
(**reorder** *<order_type>*)

The **reorder** rule controls what method of ordering fromtos in nets is used. The <*order_type*> specifies either starburst routing or simple, mid-driven, or balanced daisy chain routing.

## *<reserved_layer_name>*

<*reserved_layer_name*>::=
[**pcb** | **signal** | **power**]

The **pcb** reserved layer name means the pcb layer can be used only to define the PCB boundary, **signal** implies all signal layers, and **power** implies all power layers.

When **pcb** is the layer name in a *<boundary_descriptor>*, the bounding box of this boundary is the absolute bounding box of the design. No shapes outside this bounding box are recognized. Do not use these reserved names for layer names.

## *<resistance_resolution_descriptor>*

<*resistance_resolution_descriptor*>::=
(**resistance_resolution** [**kohm** | **ohm** | **mohm**]
<*positive_integer*>)

The symbols kohm and mohm mean kilo-ohm and milli-ohm, respectively. The default resistance unit is **mohm** with a positive integer of 1000.

## *<resolution_descriptor>*

<*resolution_descriptor*>::=
(**resolution** *<dimension_unit>* <*positive_integer*>)

The <*resolution_descriptor*> is used to map units for translation between the layout system and the tool, and has different meanings in the design file and the session file.

When a <*resolution_descriptor*> is not included in the design file, the default dimension unit is inch and the default resolution value is 2540000.

A <*resolution_descriptor*> should be included before the structure section in a design file. The <*dimension_unit*> value defines the dimensional units of the design and determines the internal representation of all dimensional numbers.

The implications of the <*resolution_descriptor*> are listed in the following example.

```
(pcb
   (resolution mil 10)
   (structure
      (boundary (rect pcb 0  0  9000  4000))
      . . .
   )
)
```

The keyword **mil** within the resolution descriptor specifies the units for all dimensions in the design. It is the default unit unless overridden by a *<unit_descriptor>* elsewhere in the design file, such as within a library section.

In the previous example, the PCB boundary is 9000 mil by 4000 mil. The tool stores this as 90000 database units by 40000 database units. Notice that value 10 in the <*resolution_descriptor*> defines the internal resolution as 0.1 mil. This value (10) does not affect the unit of the dimensions supplied in the design.

If the smallest dimension in a design has four digits to the right of the decimal point (such as 0.0001), the <*resolution_descriptor*> must have at least five zeros as the least significant digits (such as 100000). Otherwise, a roundoff error can occur in representing the smallest dimension. For example, a circle with a diameter of 4.2221 has a radius of 2.11105 and requires a fifth decimal place to avoid roundoff.

The combination of resolution and maximum PCB size must not exceed the value for an integer ($2^{31}$ or 2,147,483,648). For example, a resolution specification of **resolution mm 100000** limits the maximum dimension to 21474 mm, or 21 meters. If the resolution used is **resolution mm 10000000**, the maximum design size is only 214 mm, which is not large enough for most printed circuit boards.

When a **<***resolution_descriptor*> is included in the session file, it describes how physical dimensions in real numbers are mapped to database units. If the previous example is in a session file rather than a design file, the <*resolution_descriptor*> maps database units to the units used in the host layout system. In this example, the translator writes a boundary of 900 mil by 400 mil into the layout system database.

## *<restricted_layer_length_factor_descriptor>*

<*restricted_layer_length_factor_descriptor*>::=
(**restricted_layer_length_factor** [**1** | **0** | **-1**])

The **restricted_layer_length_factor** adjusts calculated wire lengths to account for restricted layer characteristics. A **restricted_layer_length_factor,** in conjunction with the <*max_restricted_layer_length_descriptor*>, is usually used to control EMI by imposing length constraints on external layers. The default **restricted_layer_length_factor** value is **0**. See also *<max_restricted_layer_length_descriptor>*.

## *<room_descriptor>*

<*room_descriptor*>::=
(**room** <*room_id*>
[*<shape_descriptor>*]
[{*<room_rule_descriptor>*}]
[*<room_place_rule_descriptor>*]
)

The <*room_id*> must be unique. Component placement rules specified by <*room_place_rule_descriptor*> apply to components within the room.

Only polygonal and rectangular shapes are valid for rooms.

See also *<floor_plan_descriptor>*.

## *<room_id>*

<*room_id*>::= <*id*>

## *<room_place_rule_descriptor>*

<*room_place_rule_descriptor*>::=
(**place_rule**
[*<room_place_rule_object>*]
[*<spacing_descriptor>* |
*<permit_orient_descriptor>* |
*<permit_side_descriptor>* |
*<opposite_side_descriptor>*]
)

## *<room_place_rule_object>*

*<room_place_rule_object>*::=
(**object_type**
[**room** |
**room_image_set** [**large** | **small** | **discrete** | **capacitor**]
[(**image_type** [**smd** | **pin**])]]
)

The default **object_type** is **room**.

## *<room_rule_descriptor>*

*<room_rule_descriptor>*::=
[(**height** *<max_height>* [*<min_height>*]) |
(**power** {*<net_id>*}) |
(**power_dissipation** [**-1** | *<real>*]) |
{*<include_descriptor>*} | {*<exclude_descriptor>*}]

The *<max_height>* value must be specified before the *<min_height>* value. A value of **-1** means the height is undefined. If you do not want to control maximum height, specify **-1**. If you do not want to control minimum height, either specify **-1** or omit the value. If the *<max_height>* value is less than the *<min_height>* value, the *<max_height>* value is ignored. If the **height** option is not used, the default heights are **-1**.

The unit of power you use to set a room's power dissipation rule must be consistent with the unit used to set component or image power dissipation properties. A value of **-1** means the power dissipation property is undefined. The default power dissipation is **-1**.

## *<rotation>*

*<rotation>*::= *<real>*

Rotation is expressed in degrees. The rotation value can contain up to two digits after the decimal point. Rotation direction is counterclockwise from the positive X axis.

## *<route_descriptor>*

*<route_descriptor>*::=
(**routes**
*<resolution_descriptor>*
*<parser_descriptor>*
*<structure_out_descriptor>*
*<library_out_descriptor>*
*<network_out_descriptor>*
*<test_points_descriptor>*
)

## *<route_file_descriptor>*

*<route_file_descriptor>*::=
*<route_descriptor>*

## *<route_to_fanout_only_descriptor>*

*<route_to_fanout_only_descriptor>*::=
(**route_to_fanout_only** [**on** | **off**])

If **route_to_fanout_only** is **on**, the autorouter routes to the fanout via, or if a fanout via is not present, to the SMD pad. If **route_to_fanout_only** is **off**, the autorouter can connect to either the SMD pad or its fanout via. The **route_to_fanout_only** control is **on** by default.

## *<row>*

*<row>*::= *<positive_integer>*

## *<rule_descriptor>*

*<rule_descriptor>*::=
(**rule** {*<rule_descriptors>*})

## *<rule_descriptors>*

<*rule_descriptors*>::=
[*<clearance_descriptor>* |
*<effective_via_length_descriptor>* |
*<interlayer_clearance_descriptor>* |
*<junction_type_descriptor>* |
*<length_amplitude_descriptor>* |
*<length_factor_descriptor>* |
*<length_gap_descriptor>* |
*<limit_bends_descriptor>* |
*<limit_crossing_descriptor>* |
*<limit_vias_descriptor>* |
*<limit_way_descriptor>* |
*<max_noise_descriptor>* |
*<max_stagger_descriptor>* |
*<max_stub_descriptor>* |
*<max_total_vias_descriptor>* |
{*<parallel_noise_descriptor>*} |
{*<parallel_segment_descriptor>*} |
*<pin_width_taper_descriptor>* |
*<power_fanout_descriptor>* |
*<redundant_wiring_descriptor>* |
<reorder_descriptor> |
*<restricted_layer_length_factor_descriptor>* |
*<saturation_length_descriptor>* |
*<shield_gap_descriptor>* |
*<shield_loop_descriptor>* |
*<shield_tie_down_interval_descriptor>* |
*<shield_width_descriptor>* |
{*<stack_via_descriptor>*} |
{*<stack_via_depth_descriptor>*} |
{*<tandem_noise_descriptor>*} |
{*<tandem_segment_descriptor>*} |
*<tandem_shield_overhang_descriptor>* |
*<testpoint_rule_descriptor>* |
*<time_length_factor_descriptor>* |
*<tjunction_descriptor>* |
*<track_id_descriptor>* |
*<via_at_smd_descriptor>* |
*<via_pattern_descriptor>* |
*<width_descriptor>*]

## *<same_net_checking_descriptor>*

<*same_net_checking_descriptor*>::=
(**same_net_checking** [**on** | **off**])

When **same_net_checking** is not set, it defaults to **off**.

## *<sample_window_descriptor>*

<*sample_window_descriptor*>::=
(**sample_window** [**-1** [<*integer*>] **|**
{<*positive_integer*> <*positive_integer*>}])

Use this descriptor in the crosstalk (standard parallel and tandem noise routing rules) model. The descriptor is part of a circuit descriptor used with nets or classes. A value of **-1** means the sample window is undefined. One or more sample windows can be specified.

Noise transmission and reception in nets occurs during switching and sampling time intervals, respectively. These time intervals are described by switch and sample windows, which are based on the master clock cycle. The switch window represents a time interval during which a net can broadcast noise, and the sample window represents a time interval during which a net can receive noise. A receiving net picks up transmitted noise only during the overlap time of the two windows. If overlap occurs, the transmitting net is known as an unfriendly net, and the receiving net is known as a victim net.

Each sample (and switch) window is specified by a pair of non-negative integers that represent beginning and ending times and define the interval. Noise coupling occurs only if sample and switch integer intervals overlap, either partially or completely.

See also the following descriptors:

*<switch_window_descriptor>*
*<parallel_noise_descriptor>*
*<tandem_noise_descriptor>*

## *<saturation_length_descriptor>*

<*saturation_length_descriptor*>::=
(**saturation_length** <*positive_dimension*>)

The **saturation_length** rule is applicable at the pcb, class, and net levels of the rule hierarchy. Use this rule to include the effect of noise saturation in parallel and tandem noise rules.

When the total length that a victim and aggressor net are parallel to each other is greater than the **saturation_length** value, the tool scales the total accumulated noise by the ratio of saturation length to total length.

See also **noise_calculation**, **noise_accumulation**, and **crosstalk_model** parameters in the *<control_descriptor>*.

## *<self_descriptor>*

*<self_descriptor>*::=
(**self** (**created_time** *<time_stamp>*)
{(**comment** *<comment_string>*)})

The self descriptor is included in a session file to document the time and date that the session file was created.

## *<session_file_descriptor>*

*<session_file_descriptor>*::=
(**session** <*session_id*>
(**base_design** <*path/filename*>)
[*<history_descriptor>*]
[*<session_structure_descriptor>*]
[*<placement_descriptor>*]
[*<floor_plan_descriptor>*]
[*<net_pin_changes_descriptor>*]
[*<was_is_descriptor>*]
*<swap_history_descriptor>*]
[*<route_descriptor>*]
)

A session file is created by issuing the **write session** command (see the online help for information about the **write** command). The following is an example of a session file.

```
(session ed_session3
   (base_design gpcb/am13/ed.dsn)
   (history
      (ancestor gpcb/am13/ed_session1.ses
         (created_time Jul 10 11:36:48 1993)
      (comment initial placement)
   )
      (ancestor gpcb/am13/ed_session2.ses
         (created_time Jul 19 5:36:48 1993)
         (comment pin/gate swapping)
      )
      (self
         (created_time Jul 21 15:36:48 1993)
         (comment routed 25 passes by Ed)
      )
   )
   (placement
      (component PART1
         (place IC22 142.2400 83.8200 FRONT 0)
         (place IC23 142.2400 63.5000 FRONT 0)
      )
      . . . .
   )
   (was_is
      (pins U1-1 U2-1)
      . . . .
   )
   (routes
   . . .
   )
)
```

## *<session_id>*

<*session_id*>::= <*id*>

## *<session_structure_descriptor>*

<*session_structure_descriptor*>::=
(**structure**
*<place_boundary_descriptor>*
{*<keepout_descriptor>*}
[(**deleted_keepout** {*<keepout_sequence_number>*})]
)

The tool adds a <*keepout_descriptor*> to the session file for each keepout you define during the session. You can change the definitions of keepouts defined in the structure section of the design file, but not the definitions of keepouts defined in an *<image_descriptor>*.

The tool adds a <*place_boundary_descriptor*> to the session file if you define or change the placement boundary during the session.

Use **deleted_keepout** to delete a keepout from the session file if you delete the keepout during the session. The <*keepout_sequence_number*> identifies the keepout to be deleted.

## *<setback>*

<*setback*>::=
<*positive_dimension*>

## *<shape_descriptor>*

<*shape_descriptor*>::=
[*<rectangle_descriptor>* |
*<circle_descriptor>* |
*<polygon_descriptor>* |
*<path_descriptor>* |
*<qarc_descriptor>*]

Shapes are the only objects the tool recognizes. The tool also generates shapes. Polygons and circles are closed, filled shapes. Rectangles are also closed, filled shapes except boundaries, which are closed, unfilled shapes.

## *<shield_descriptor>*

<*shield_descriptor*>::=
(**shield** [**off** | **on** [*<shield_type_descriptor>*]
(**use_net** <*net_id*>)])

When **shield** is **on**, the **use_net** keyword identifies the net that is used as the shield. If you don't specify a shield type, the default is parallel shielding. The **shield** default is **off**.

## *<shield_gap_descriptor>*

<*shield_gap_descriptor*>::=
(**shield_gap** <*positive_dimension*>)

The **shield_gap** value defines the edge-to-edge distance between the wire being shielded and the shield wire. If **shield_gap** is not supplied, the value defaults to the wire_wire clearance for the connection being shielded.

## *<shield_loop_descriptor>*

<*shield_loop_descriptor*>::=
(**shield_loop** [**open | closed**])

The **shield_loop** value defines whether open or closed end loops are generated around pins, pads, or vias. The default is **closed**. With the **open** option, no attempt is made to close the shield loop, and two stub wires, as well as two vias, might be added to connect the shield wires to the shield net.

## *<shield_tie_down_interval_descriptor>*

<*shield_tie_down_interval_descriptor*>::=
(**shield_tie_down_interval** <*positive_dimension*>)

This rule controls the distance between vias, or vias with stub wires, when multiple connections from the shield wire to the power layer are needed. The **shield_tie_down_interval** value is determined by the frequency of the signal on the shielded net.

## *<shield_type_descriptor>*

<*shield_type_descriptor*>::=
(**type** [**parallel | tandem | coax**])

You can specify parallel or tandem shielding rules. For both parallel and tandem shielding, use the **coax** keyword. The default **type** is **parallel**.

## *<shield_width_descriptor>*

<*shield_width_descriptor*>::=
(**shield_width** <*positive_dimension*>)

The **shield_width** value defines the width of the shield, including the wire segment that connects to a pin or via. If **shield_width** is not used, the value defaults to the same width as the connection being shielded.

## *<side>*

<*side*>::= [**front** | **back**]

The **front** side is the first layer defined in the layer stackup in the structure data and **back** is the last layer defined in the stackup.

## *<sign>*

<*sign*>::= [**+** | **-**]

## *<site_array_descriptor>*

<*site_array_descriptor*>::=
(**site** <*positive_integer*> *<x0>* *<y0>* *<xstep>* *<ystep>*)

The <*site_array_descriptor*> defines an array of bond sites for wirebond applications. The <*positive_integer*> value defines the total number of bond sites in the array, <*x0*> and <*y0*> determine the coordinate location of the first site in the array, and <*xstep*> and <*ystep*> define the step increment for the array.

## *<spacing_descriptor>*

<*spacing_descriptor*>::=
(**spacing** [**-1** | <*positive_dimension*>]
[(**type** *<spacing_type>*)] [(**side** *<place_side>*)]
)

A value of **-1** sets the spacing rule to unspecified.

See also the *<place_rule_descriptor>*.

## *<spacing_type>*

<*spacing_type*>::=
*<place_object>*_*<place_object>*

## *<special_character>*

<*special_character*>::=

Any ASCII special character except a blank space, left or right parenthesis, semicolon, single quote ('), and newline.

## *<stack_via_descriptor>*

<stack_via_descriptor>::=
(**stack_via** [**on** | **off**])

When **on**, the **stack_via** rule allows vias to stack center on center. See also the *<stack_via_depth_descriptor>*.

## *<stack_via_depth_descriptor>*

<*stack_via_depth_descriptor*>::=
(**stack_via_depth** <*positive_dimension*>)

This rule is only applicable at the pcb level of the rule hierarchy. The **stack_via_depth** rule controls the number of vias over which the **stack_via** rule applies. Vias that fall outside the specified range are generally connected in a staggered via pattern. See also the *<stack_via_descriptor>*.

## *<start_pass>*

<*start_pass*>::= <*positive_integer*>

## *<step>*

<*step*>::= <*positive_integer*>

## *<string>*

*<string>*::=
[*<character>* | *<character>* *<string>*]

## *<string_compare_operator>*

*<string_compare_operator>*::=
[**==** | **!=** | **<** | **>** | **<=** | **>=**]

## *<string_expression>*

*<string_expression>*::=
[*<string_expression>* + *<string_expression>* | (*<string_expression>*) |
*<one_word_string>* | *<variable_name>*]

## *<structure_descriptor>*

*<structure_descriptor>*::=
(**structure**
[*<unit_descriptor>* | *<resolution_descriptor>* | null]
{*<layer_descriptor>*}
[*<layer_noise_weight_descriptor>*]
{*<boundary_descriptor>*}
[*<place_boundary_descriptor>*]
[{*<plane_descriptor>*}]
[{*<region_descriptor>*}]
[{*<keepout_descriptor>*}]
*<via_descriptor>*
[*<control_descriptor>*]
*<rule_descriptor>*
[*<structure_place_rule_descriptor>*]
{*<grid_descriptor>*}
)

## *<structure_out_descriptor>*

*<structure_out_descriptor>*::=
(**structure_out**
{*<layer_descriptor>*}
[*<rule_descriptor>*]
)

## *<structure_place_rule_descriptor>*

<*structure_place_rule_descriptor*>::=
(**place_rule** [*<structure_place_rule_object>*]
{[*<spacing_descriptor>* |
*<permit_orient_descriptor>* |
*<permit_side_descriptor>* |
*<opposite_side_descriptor>*]}
)

## *<structure_place_rule_object>*

<*structure_place_rule_object*>::=
(**object_type**
[**pcb** |
**image_set** [**large** | **small** | **discrete** | **capacitor** | **resistor**]
[(**image_type** [**smd** | **pin**])]]
)

The default **object_type** is **pcb**.

## *<subgate_id>*

<*subgate_id*>::= <*id*>

See also *<part_pin_descriptor>*.

## *<subgate_pin_id>*

<*subgate_pin_id*>::= <*id*>

The <*subgate_pin_id*> is the logical pin name of a subgate pin.

See also *<part_pin_descriptor>*.

## *<subgate_swap_code>*

<*subgate_swap_code*>::= <*integer*>

Subgates within the same gate that have the same subgate swap code can be swapped. A <*subgate_swap_code*> value of 0 identifies a subgate that cannot be swapped.

## *<suffix>*

<*suffix*>::= <*id*>

## *<super_place_reference>*

<*super_place_reference*>::=
(**place**
<*component_id*>
*<vertex>*
*<side>*
*<rotation>*
)

The <*super_place_reference*> descriptor is used with the *<cluster_descriptor>*. The <*vertex*> values are relative to the origin of the super component. The <*rotation*> values are relative to the origin of the image in the *<image_descriptor>*.

## *<supply_pin_descriptor>*

<*supply_pin_descriptor*>::=
(**supply_pin** {*<pin_reference>*} [(**net** <*net_id*>)])

Use the <*supply_pin_descriptor*> to define supply pins that you identify with <*pin_reference*>. You identify pins of a net as supply pins with <*net_id*>.

Nets with pins designated as **supply_pin** are ordered so that the pin is the source terminal for other pins on the net.

See also

*<net_out_descriptor>*
*<wiring_descriptor>*
*<wire_shape_descriptor>*
*<wire_via_descriptor>*

## *<swap_history_descriptor>*

<*swap_history_descriptor*>::=
(**swapping**
{[(**gates** <*component_id*> <*gate_id*> <*component_id*> <*gate_id*>) |
(**subgates** <*component_id*> <*gate_id*> <*subgate_id*> <*component_id*>
<*gate_id*> <*subgate_id*>) |
(**pins** *<pin_reference>* *<pin_reference>*)]}
)

## *<switch_window_descriptor>*

<*switch_window_descriptor*>::=
(**switch_window** [**-1** [<*integer*>] |
{<*positive_integer*> <*positive_integer*>}])

This descriptor is used in the crosstalk (standard parallel and tandem noise routing rules) model. The descriptor is part of a circuit descriptor used with nets or classes. A value of **-1** means the switch window is undefined. One or more switch windows can be specified.

Noise transmission and reception in nets occurs during switching and sampling time intervals, respectively. These time intervals are described by switch and sample windows, which are based on the full clock cycle. The switch window represents a time interval during which a net can broadcast noise, and the sample window represents a time interval during which a net can receive noise. A receiving net picks up transmitted noise only during the overlap time of the two windows. If overlap occurs, the transmitting net is known as an unfriendly net, and the receiving net is known as a victim net.

Each switch (and sample) window is specified by a pair of non-negative integers that represent beginning and ending times and define the interval. Noise coupling occurs only if switch and sample integer intervals overlap, either partially or completely.

See also

*<sample_window_descriptor>*
*<parallel_noise_descriptor>*
*<tandem_noise_descriptor>*

## <*system_variable*>

<*system_variable*>::=
[**bottom_layer_sel** | **complete_wire** | **conflict_clearance** | **conflict_crossing** | **conflict_length** | **conflict_wire** | **conflict_xtalk** | **connections** | **current_wire** | **locked_comp** | **partial_selection** | **power_layers** | **reduction_ratio** | **reroute_wire** | **route_pass** | **sel_signal_layers** | **selectedcomp** | **signal_layers** | **smd_pins** | **thru_pins** | **top_layer_sel** | **total_pass** | **total_pins** | **totalcomp** | **unconnect_wire** | **units** | **unplaced_comp** | **unplaced_large** | **unplaced_small**]

The following table shows system variable definitions.

| Variable Name | Definition |
|---|---|
| bottom_layer_sel | 1 if bottom layer is selected, 0 if not selected. |
| complete_wire | Completion ratio expressed as a percentage. |
| conflict_clearance | Number of clearance conflicts. |
| conflict_crossing | Number of crossing conflicts. |
| conflict_length | Number of length rule violations. |
| conflict_wire | Number of crossing and clearance conflicts. |
| conflict_xtalk | Number of crosstalk rule violations. |
| connections | Total number of connections to be routed. |
| current_wire | Current wire being routed or rerouted. |
| locked_comp | Number of locked components. |
| partial_selection | Value equals 0 if no nets or all nets are selected; value equals 1 when one or more nets but fewer than all nets are selected. |
| power_layers | Number of power layers. |
| reduction_ratio | Conflicts reduction ratio from last completed routing pass. |
| reroute_wire | Number of wires and wire segments to be rerouted in the current pass. |

| Variable Name | Definition |
|---|---|
| route_pass | Current routing pass or last pass. |
| sel_signal_layers | Number of selected signal layers. |
| selectedcomp | Number of selected components. |
| signal_layers | Number of signal layers. |
| smd_pins | Number of smd pads. |
| thru_pins | Number of through-hole pins. |
| top_layer_sel | 1 if top layer is selected, 0 if not selected. |
| total_pass | Total passes for the current command. |
| total_pins | Total number of pins and pads. |
| totalcomp | Total number of components on the PCB. |
| unconnect_wire | Unconnected wires (unconnects). |
| units | Unit of measure set by user. |
| unplaced_comp | Number of components outside the placement boundary. |
| unplaced_large | Number of large components outside the placement boundary. |
| unplaced_small | Number of small components outside the placement boundary. |

## *<tandem_noise_descriptor>*

<*tandem_noise_descriptor*>::=
(**tandem_noise**
  [**off |**
  (**gap** <*dimension*>)
  [(**threshold** <*positive_dimension*>)]
  (**weight** <*real*>)]
)

Noise coupling between nets is controlled by computing the total noise that impinges on receiving nets from transmitting nets on adjacent layers. Each net in a design can have a different noise weight or transmitting characteristic. A net's noise weight determines how much noise it transmits. Each net can also have a different maximum noise specification or receiving characteristic. The maximum noise specification determines how much noise a net can accumulate or pick up from other nets before a **tandem_noise** violation occurs. See also *<max_noise_descriptor>*.

The following table describes <*tandem_noise_descriptor*> keywords.

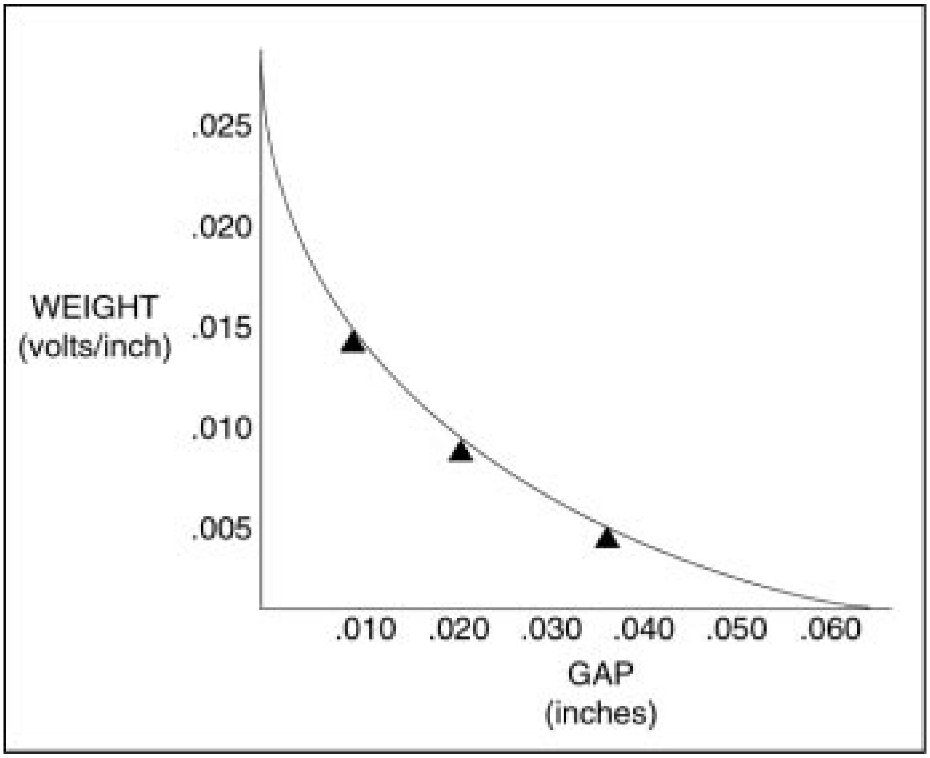
| Keyword | Description |
|---|---|
| off | Resets a rule to unspecified. To change an existing tandem noise rule, always use **tandem_noise off** before setting the new rule. |
| gap | Is measured edge-to-edge between tandem wires. Coupled noise is calculated for tandem wires when the edge-to-edge distance is equal to or less than the specified **gap** value and the wires are parallel for a distance that exceeds the **threshold** value. If a wire does not have a **max_noise** value, no noise is computed for that wire.<br><br>A negative **gap** value determines the amount of coupling if wires overlap. If wires of different width completely overlap, the negative **gap** value between those wires equals the width of the smaller wire. |
| threshold | Is the minimum tandem length that is considered when tandem noise violations are computed. When **threshold** is unspecified, its value defaults to the **gap** value. |

| Keyword | Description |
|---|---|
| weight | Represents units of noise per unit of length, where the unit of noise is typically volts or millivolts and the unit of length is the current dimensional unit. The **weight** value corresponds to the noise transmitted over a unit length of wire to surrounding wires. The tool computes the noise coupled from a tandem transmitting wire by multiplying the transmitting wire's tandem length by its **weight** value. All coupled noise sources are accumulated for each receiving net, and the sum is compared against that net's maximum noise specification to determine if a violation exists. |

A coupled noise weight versus gap curve can be approximated for a net by entering two or more weight versus gap pairs. For example:

```
unit inch
rule net clk1 (tandem_noise (gap .010) (threshold .050)
   (weight .015))
(tandem_noise (gap .020) (threshold .100)
   (weight .010))
(tandem_noise (gap .036) (threshold .100)
   (weight .005))
```

**Coupled Noise Weight Versus Gap**




If multiple **tandem_noise** rules are applied to a net at different precedence levels, violations are checked only at the highest level.

## *<tandem_segment_descriptor>*

<*tandem_segment_descriptor*>::=
(**tandem_segment**
[**off |**
(**gap** <*dimension*>)
(**limit** <*positive_dimension*>)]
)

Tandem is defined as parallelism of wires on adjacent layers. This form of parallelism is controlled by setting a tandem length limit and a minimum wire-to-wire gap.

The following table describes <*tandem_segment_descriptor*> keywords.

| **Keyword** | **Description** |
|---|---|
| off | Resets a rule to the unspecified state. To change an existing tandem segment rule, always use **tandem_segment off** before setting the new rule. |
| gap | Is measured edge-to-edge between tandem wires. A tandem segment violation does not occur when **gap** is greater than the specified value.<br><br>A negative **gap** value allows receiving and transmitting nets to overlap by the specified value. If overlap is such that an entire net width is within the other net width, specify a negative **gap** value which is the width of the smaller net. |
| limit | Is the maximum tandem length that is allowed before a **tandem_segment** violation occurs. When **limit** is unspecified or is less than the **gap** value, the **limit** value defaults to the **gap** value. |

Power nets are not included in tandem segment rule checking.

The following example illustrates how a table of rules can be created by supplying multiple **tandem_segment** rules.

```
(Net clk1
   (pins...)
   (rule (tandem_segment (gap 11) (limit 500))
   (tandem_segment (gap 14) (limit 1200))
   (tandem_segment (gap 16) (limit 1800))
)
```

Violations occur when

- **gap** is less than or equal to 11 and the tandem length is greater than 500.
- **gap** is less than or equal to 14 and the tandem length is greater than 1200.
- **gap** is less than or equal to 16 and the tandem length is greater than 1800.

If multiple **tandem_segment** rules are applied at different precedence levels, violations are checked only at the highest level. For the order of routing rule precedence, see the Routing and Placement Rule Hierarchies section at the beginning of this manual.

## *<tandem_shield_overhang_descriptor>*

<*tandem_shield_overhang_descriptor*>::=
(**tandem_shield_overhang** <*positive_dimension*>)

Use this descriptor to specify the extra amount added to each side of the tandem shield wire. The tandem shield width is two times the **tandem_shield_overhang** value plus the width of the wire being shielded. The **tandem_shield_overhang** value defaults to the width of the shield wire.

## *<test_net_descriptor>*

<*test_net_descriptor*>::=
(**net** <*net_id*>)

## *<test_point_descriptor>*

<*test_point_descriptor*>::=
(**point** *<vertex>* [**front | back**]
[*<test_net_descriptor>*]
[*<test_type_descriptor>*])

## *<testpoint_rule_descriptor>*

<*testpoint_rule_descriptor*>::=
(**testpoint**
{[(**allow_antenna** [**off** | **on**])] |
[(**max_len** <*positive_dimension*>)] |
[(**center_center** <*positive_dimension*>)] |
[(**comp_edge_center** <*positive_dimension*>)] |
[(**grid** <*positive_dimension*>) [(**direction** [**x** | **y**])]
[(**offset** <*positive_dimension*>)])] |
[(**image_outline_clearance** <*positive_dimension*>)] |
[(**insert** [**off** | **on**])] |
[(**pin_allow** [**off** | **on** [(**comp** {<*component_id*>})]])] |
[(**side** [**front**] | **back** | **both**])] |
[(**use_via** {<*via_id*>})]
}
)

The **allow_antenna** control defaults to **on**. The **max_len** option restricts the length of antennas created during test point routing.

The **center_center** and **comp_edge_center** controls are not checked if values are not specified for these fields.

The **grid** setting defaults to the pcb via grid in effect at the time of test point identification. If a **direction** option is not specified, the grid spacing value and **offset** value (if given) apply equally in the x and y directions. If a **direction** option is specified, the **grid** spacing value and **offset** value (if given) only apply to the specified direction. To specify nonuniform grids or offsets in the x and y directions, you must use two **grid** option expressions.

The **image_outline_clearance** control defaults to area test point clearance, **pin_allow** defaults to **off**, and **side** defaults to **back**. If the **use_via** value is not specified, the narrowest diameter via is used.

## *<test_points_descriptor>*

<*test_points_descriptor*>::=
(**test_points** {*<test_point_descriptor>*})

## *<test_type_descriptor>*

*<test_type_descriptor>*::=
(**type** [**route** | **protect** | **normal**])

A **route** type test point cannot be altered, although the router can complete a connection to this type. A **protect** type cannot be altered unless the user first unprotects the test point. A **normal** type test point can be deleted, ripped up, or moved to a different location.

## *<time_length_factor_descriptor>*

*<time_length_factor_descriptor>*::=
(**time_length_factor** <*real*>)

The **time_length_factor** sets a constant for time delay per unit of wire length, which is used to calculate wire length limits when a **circuit** delay rule applies. The constant converts internally to delay per database unit. See also *<circuit_descriptors>*.

## *<time_resolution_descriptor>*

*<time_resolution_descriptor>*::=
(**time_resolution** [**sec** | **msec** | **usec** | **nsec** | **psec**]
<*positive_integer*>)

| **Symbol** | **Time Unit** |
|---|---|
| sec | second |
| msec | millisecond |
| usec | microsecond |
| nsec | nanosecond |
| psec | picosecond |

The default time unit is **nsec** with a positive integer of 1000.

## *<time_stamp>*

<*time_stamp*>::=
<*month*> <*date*> <*hour*> : <*minute*> : <*second*> <*year*>

The <*month*> is a string that has three alpha characters; <*date*>, <*hour*>, <*minute*>, and <*second*> are strings that each have two numeric characters; <*year*> is a string that has four numeric characters.

## *<tjunction_descriptor>*

<*tjunction_descriptor*>::=
(**tjunction** [**on** | **off**])

The <*tjunction_descriptor*> controls whether tjunctions are permitted on starburst ordered nets. When **tjunction on** is set, tjunctions can occur at the locations controlled by **junction_type**. When **tjunction off** is set, tjunctions are not permitted.

The also *<junction_type_descriptor>*.

The **tjunction** default is **on** for starburst nets and **off** for daisy-chained nets.

## *<topology_descriptor>*

<*topology_descriptor*>::=
(**topology** {[*<fromto_descriptor>* |
*<component_order_descriptor>*]})

The <*topology_descriptor*> defines the preferred topology for each net in a class. The tool ignores components that are included in <*topology_descriptor*> but are not connected to any net in the class.

See <*component_order_descriptor*> for details about ordering nets using component reference designators.

## *<total_delay_descriptor>*

<*total_delay_descriptor*>::=
([**max_total_delay** | **min_total_delay**] *<delay_value>*)

The **max_total_delay** and **min_total_delay** rules apply only to groups. The rules are checked against the sum of all fromto delays in a group. The sum of the routed delays of the fromtos in the group must be in the **max_total_delay** and **min_total_delay** range.

## *<total_length_descriptor>*

<*total_length_descriptor*>::=
([**max_total_length** | **min_total_length**]
<*positive_dimension*>)

The **max_total_length** and **min_total_length** rules apply only to groups. The rules are checked against the sum of all fromto lengths in a group. The sum of the routed lengths of the fromtos in the group must be in the **max_total_length** and **min_total_length** range.

## *<track_id_descriptor>*

<*track_id_descriptor*>::=
(**track_id** <*integer*>)

The **track_id** establishes a numbering base in the routes file, which is used when the routing information in the file is translated back to the layout system.

## *<turret#>*

<*turret#*>::=
<*positive_integer*>

The *<turret#>* descriptor indicates translated wires. The layout system can use *<turret#>* to tag wires read by the tool. This number is not used internally, but passes through the system and into the wires file. The acceptable range of values for *<turret#>* is from 1 to 127.

## *<unit_descriptor>*

<*unit_descriptor*>::=
(**unit** *<dimension_unit>*)

The dimensional units for information in a design file are set by the *<resolution_descriptor>* in the structure section. You can override the resolution units within a particular section by using a <*unit_descriptor*>.

For example, suppose the <*resolution_descriptor*> sets the PCB dimensional units to millimeters, but all the component images in the library section are defined in inches. You can identify the image dimensions as inches by using a <*unit_descriptor*> at the beginning of the library section. This <*unit_descriptor*> tells the tool to interpret the library information in inches. If a <*unit_descriptor*> is not used at the beginning of the next section, the tool interprets the information in this section in millimeters.

The <*unit_descriptor*> affects only the section in which it resides. For example:

(unit inch)

## *<user_property_descriptor>*

<*user_property_descriptor*>::=
(**property** {*<property_value_descriptor>*})

## *<user_variable>*

<*user_variable*>::=
*<letter>* [{[*<letter>* | *<digit>* | <*underscore*>]}]

User variable names must start with an alphabetic character. The remaining characters can consist of any combination of upper and lower case letters, the digits 0 through 9, and the underscore character (_). System variable names are reserved and cannot be used.

## *<value_descriptor>*

<*value_descriptor*>::= [<*integer*> | <*real*> | *<string>*]

The <*integer*>*, <real*>, and <*string*> are the values of a user-defined property.

## *<variable_name>*

<*variable_name*>::=
[*<system_variable>* | *<user_variable>*]

## *<vertex>*

<*vertex*>::=
*<x_coordinate>* *<y_coordinate>*

## *<via#>*

<*via#*>::= <*positive_integer*>

The *<via#>* descriptor indicates translated vias. The layout system can use *<via#>* to tag vias read by the tool. This number is not used internally, but passes it through the system and into the wires file. The acceptable range of values for *<via#>* is from 1 to 127.

## *<via_array_template_descriptor>*

<*via_array_template_descriptor*>::=
(**via_array_template** <*via_array_template_id*>
{*<microvia_descriptor>*}
)

You must specify the minimum dimensions of a via in the <*microvia_descriptor*>. (This requires **microvia on** in the *<control_descriptor>*.)

You can use Change Via mode to change row, column, via width, and via height at the same time.

## *<via_array_template_id>*

<*via_array_template_id*>::= <*id*>

## *<via_at_smd_descriptor>*

<*via_at_smd_descriptor*>::=
(**via_at_smd** [**off** | **on** [(**grid** [**on** | **off**])]
[(**fit** [**on** | **off**])]]
)

The <*via_at_smd_descriptor*> controls whether vias are permitted under SMD pads. When **via_at_smd** is **on**, the router can place vias under SMD pads. The default is **off**. You can also allow vias under SMD pads by using the **via_at_smd** rule.

When **via_at_smd** is **on** and the **grid** setting is **off**, the via is permitted at the origin of an SMD pad. If the pad origin is off grid, you can turn the **grid** setting **on** to position the via at a grid point nearest the pad origin within the pad boundary. The **grid** default is **off**.

Turn the **fit** setting **on** to ensure that any via placed under an SMD pad fits entirely within the boundary of the pad. If the via shape on the pad layer extends beyond a pad's boundary, the via is not located under the pad. The **fit** default is **off**.

Three conditions must be met before vias can be placed under SMDs:

- There must be at least one via available with a shape on the SMD mounting layer.
- The **attach** parameter for the SMD padstack must be set to **on** in the design file. The **attach** rule is usually set to **on** in the layout system.
- The **via_at_smd on** rule must be applied.

## *<via_descriptor>*

<*via_descriptor*>::=
(**via**
{<*padstack_id*>}
[(**spare** {<*padstack_id*>})]
)

During routing, any via in the via {<*padstack_id*>} list is available for use. Vias listed as spares are used only if they are associated with a net by a **use_via** rule, specified with the **testpoint** or **testpoint rule** commands, or selected by the user with the **select** command. Otherwise, spare vias are not used.

## *<via_height>*

<*via_height*>::= <*positive_dimension*>

## *<via_id>*

<*via_id*>::= <*id*>

The <*via_id*> identifies a pad defined as a via in the *<via_descriptor>*.

## *<via_pattern_descriptor>*

<*via_pattern_descriptor*>::=
([[**spiral_via | staggered_via | staired_via**] [**on | off**]]
[(**min_gap** <*positive_dimension*>)]
)

The spiral, staggered, and staired via patterns default to **off**. When a via pattern is turned **on** without specifying **min_gap**, the minimum gap between vias in the pattern defaults to the largest via_via clearance rule in effect.

## *<via_width>*

<*via_width*>::= <*positive_dimension*>

## *<virtual_pin_descriptor>*

<*virtual_pin_descriptor*>::=
(**virtual_pin** <*virtual_pin_name*>
[(**position** *<vertex>* [(**radius** <*positive_dimension*>)])]
)

The **position** rule specifies the X and Y coordinates for a virtual pin. If using the <*vertex*> location would cause a rule violation, use the **radius** option to set the virtual pin location at a certain distance from the vertex. The autorouter can move the pin to avoid the violation. The default radius is 0.5 inches.

The <*virtual_pin_descriptor*> describes a pseudo pin or via that can be used to specify a tree or other wiring topology. Use virtual pins to control delays (for example to minimize clock skew) by matching wire lengths without adding excessive wiring on each branch of a net.

For example:

```
(net CLK1 (fromto U1-1 (virtual_pin FP1)
   (circuit (length 350 300)))
(fromto (virtual_pin FP1) U2-1)
(fromto (virtual_pin FP1) U3-1))
```

You can also use virtual pins to control impedance by creating a common path or trunk with a width rule that is different from the rule used for the branches. You can use multiple levels of virtual pins to construct big tree topologies that include tjunctions.

Virtual pins are seeded in a way that satisfies routing length constraints. Use **junction_type** to control whether both vias and wire tjunctions or only vias are allowed as virtual pins. See also *<junction_type_descriptor>*.

You can disband virtual pin assignments by using the **forget net** command.

## *<virtual_pin_name>*

<*virtual_pin_name*>::= <*id*>

A <*virtual_pin_name*> must be unique within a net; however, the same <*virtual_pin_name*> can be used in more than one net.

## *<voltage_resolution_descriptor>*

<*voltage_resolution_descriptor*>::=
(**voltage_resolution** [**volt** | **mvolt**] <*positive_integer*>)

The symbol **mvolt** means millivolt. The default voltage unit is **volt** with a <*positive_integer*> equal to 1000.

## *<was_is_descriptor>*

<*was_is_descriptor*>::=
(**was_is** {(**pins** *<pin_reference>* *<pin_reference>*)})

The <*was_is_descriptor*> is included in a session file when gate, subgate, or pin swaps occur during a placement session.

The <*was_is_descriptor*> contains only information about the original pins and the new pins, no matter how the swaps are executed. You cannot determine the swapping history from the <*was_is_descriptor*>.

For example, if pin U8-3 swaps with U9-3, the result is recorded as

```
(was_is
   (pins U8-3 U9-3)
   (pins U9-3 U8-3)
)
```

Only one swap operation is performed, but the session file includes two entries because two pins change as a result of the swap.

In the following example, pin U8-3 swaps with U9-3 and pin U9-3 swaps with U10-3. The result is recorded as

```
(was_is
   (pins U8-3 U10-3)
   (pins U9-3 U8-3)
   (pins U10-3 U9-3)
)
```

## *<width_descriptor>*

<*width_descriptor*>::=
(**width** <*positive_dimension*>)

## *<window_descriptor>*

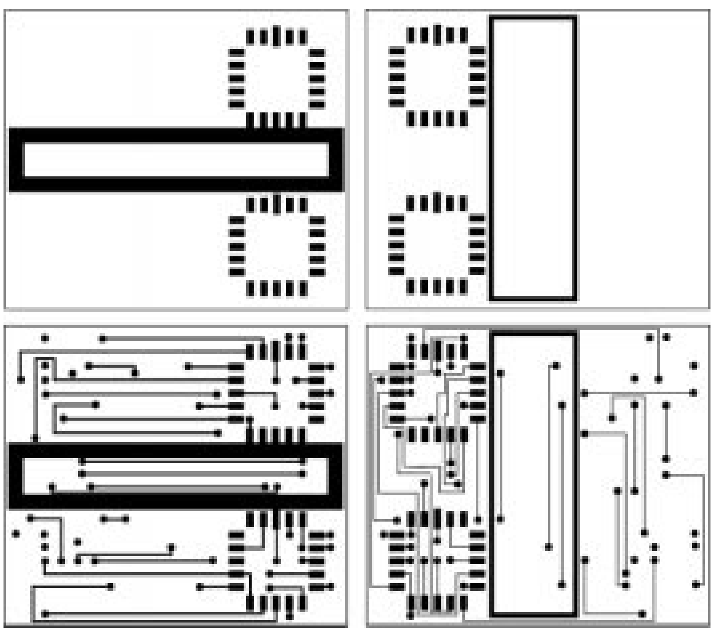
<*window_descriptor*>::=
(**window** *<shape_descriptor>*)

Windows cut out or subtract from the shapes they overlap. Therefore, windows are not physical shapes; they are used only to subtract from other shapes. Only rectangle and polygons shapes can be used for the <*window_descriptor*>. In the following examples, windows have been specified for each of the keepout areas shown in the illustration. The design file constructs are

```
(keepout (rect s2 0.060 0.494 1.240 0.818) (window
   (rect s2 0.110 0.580 1.165 0.750)))
(keepout (rect s1 0.490 0.090 0.770 1.210) (window
   (rect s1 0.505 0.105 0.755 1.195)))
```

The interior of each keepout is reached by routing on a different layer and by using a via to access the enclosed routing area.

**Keepout Areas Defined by Window_descriptors**



## *<wire_descriptor>*

<*wire_descriptor*>::=
[*<wire_shape_descriptor>* |
*<wire_via_descriptor>* |
*<bond_shape_descriptor>*]

## *<wire_guide_descriptor>*

<*wire_guide_descriptor*>::=
(**guide** (**connect**
(**terminal** *<object_type>* [*<pin_reference>*] *<vertex>*)
(**terminal** *<object_type>* [*<pin_reference>*] *<vertex>*)
))

The vertex indicates the terminal point of the guide. The vertex should match a pin, via, or wire tjunction coordinate in the layout system.

## *<wire_pair_descriptor>*

<*wire_pair_descriptor*>::=
(**wires** *<fromto_descriptor>* *<fromto_descriptor>*
{[(**gap** [<*positive_dimension*> **|** **-1**] {[(**layer** <*layer_id*>)]})]}
)

Use **wires** to identify the pin-to-pin connections you want included in a pair.

Use **gap** to control the minimum distance (<*positive_dimension*>) allowed between the two routed wires in a pair. If **gap** is not included in a <*wire_pair_descriptor*>, the wire-to-wire clearance rule is used. To reset a specified **gap** to the default wire-to-wire clearance, use **-1** for the **gap** value.

You can use the **layer** keyword to apply the **gap** value to only the layer identified in <*layer_id*>.

## *<wire_shape_descriptor>*

```
<wire_shape_descriptor>::=
(wire
  <shape_descriptor>
  [(net <net_id>)]
  [(turret <turret#>)]
  [(type [fix | route | normal | protect])]
  [(attr [test | fanout | bus | jumper])]
  [(shield <net_id>)]
  [{<window_descriptor>}]
  [(connect
    (terminal <object_type> [<pin_reference>])
    (terminal <object_type> [<pin_reference>])
  )]
  [(supply)]
)
```

Note that a wire shape can be any type of shape. See also *<shape_descriptor>*. When *<polygon_descriptor>* and *<rectangle_descriptor>* are applied in the wiring section of the design file, they become a wiring polygon.

A **fix** type wire cannot be altered in any way, and the router cannot route to this type. A **route** type wire cannot be altered, although the router can complete a connection to this wire type. A **normal** type wire can be deleted, ripped up, and rerouted. A **protect** type cannot be altered unless the user first unprotects the wire.

The **type** constructs also apply to wiring polygons. By default, wiring polygons are **route** type.

The **connect** constructs are used only in a routes file.

In the **shield** option, <*net_id*> is the name of the net being shielded.

The tool attaches the **jumper** attribute to wires that are added to the jumper layer.

The **supply** keyword designates wires as source terminals. For example, in a routes or session file, shapes assigned as supply are identified with the **supply** keyword. See also

*<net_out_descriptor>*
*<supply_pin_descriptor>*
*<wiring_descriptor>*
*<wire_via_descriptor>*

## *<wire_via_descriptor>*

<*wire_via_descriptor*>::=
(**via**
<*padstack_id*> {*<vertex>*}
[(**net** <*net_id*>)]
[(**via_number** *<via#>*)]
[(**type** [**fix** | **route** | **normal** | **protect**])]
[(**attr** [**test** | **fanout** | **jumper** |
**virtual_pin** *<virtual_pin_name>*])]
[(**contact** {<*layer_id*>})]
[(**supply**)]
)
(**virtual_pin**
*<virtual_pin_name>* *<vertex>* (**net** <*net_id*>)
)

A **fix** type via cannot be altered in any way, and the router cannot route to this type. A **route** type via cannot be altered, although the router can complete a connection to this via type. A **normal** type via can be deleted, ripped up, and rerouted. A **protect** type cannot be altered unless the user first unprotects the via.

The tool attaches the **jumper** attribute to vias that are used for jumpers on the jumper layer.

The **virtual_pin** attribute marks vias used as virtual pins and the **virtual_pin** parameter identifies virtual pins on a wire path. Virtual pin positions are saved in the routes or session file.

The **supply** keyword designates vias as source terminals. For example, in a routes or session file, shapes assigned as supply, are identified with the **supply** keyword. See also

*<net_out_descriptor>*
*<supply_pin_descriptor>*
*<wiring_descriptor>*
*<wire_shape_descriptor>*

## *<wires_file_descriptor>*

<*wires_file_descriptor*>::=
*<wiring_descriptor>*

## *<wiring_descriptor>*

<*wiring_descriptor*>::=
(**wiring**
  [*<unit_descriptor>* | *<resolution_descriptor>* | null]
  {*<wire_descriptor>*}
  [*<test_points_descriptor>*]
  {[*<supply_pin_descriptor>*]}
)

A wires file is created by an **autosave**, **bestsave** or **write wire** command.

The *<resolution_descriptor>* defines the unit and resolution used in the wires file.

The <*supply_pin_descriptor*> identifies wire shapes in routes and session files that are used as source terminals. Other shapes assigned to the same net are routed directly to the source terminal. See *<supply_pin_descriptor>* for more details.

## *<x0>*

<*x0*>::= <*dimension*>

## *<xstep>*

<*xstep*>::= <*dimension*>

## *<x_clearance>*

<*x_clearance*>::= <*positive_dimension*>

## *<x_coordinate>*

<*x_coordinate*>::= <*dimension*>

## *<x_overlap>*

<*x_overlap*>::= <*positive_dimension*>

## *<y0>*

<*y0*>::= <*dimension*>

## *<ystep>*

<*ystep*>::= <*dimension*>

## *<y_clearance>*

<*y_clearance*>::= <*positive_dimension*>

## *<y_coordinate>*

<*y_coordinate*>::= <*dimension*>

## *<y_overlap>*

<*y_overlap*>::= <*positive_dimension*>

# Sample Files

## Chapter content

- Sample Design File
- Sample Design File with High Speed Rules

This chapter provides two samples of SPECCTRA design files. The second design file includes high speed rules.

The SPECCTRA design file consists of four basic data types:

- Design data, which includes board boundaries, layer definitions, design rules, and keepout definitions
- Placement data, which includes X,Y locations of components and mounting holes on the PCB
- Library data, which includes images (footprint patterns) for all placed components, and pin and via padstack definitions
- Network data, which includes net names, component reference designators, and pin numbers

## Sample Design File

The following design file sample shows design, placement, library, network, and prerouted wiring types of data.

### Design Data

```
(PCB test_brd_20
```

#The PCB statement is used for documentation purposes.
#The name is used only to identify the listing. The design
#filename can be different.

```
 (resolution MIL 10)
```

```
 (structure
```

#The structure contains the PCB definition.

```
  (boundary
(rect pcb 5956.00000 345.90000 11202.00000
 3888.00000)
```

#The PCB outline is defined by a pcb boundary statement.
#This is the outermost perimeter that is displayed on the
#screen.

```
  )
  (boundary
(rect signal 6180 400 11000 3850)
```

#The signal boundary is identified by this statement. No
#routing is permitted outside this boundary.

```
 )
 (via VIA)
 (grid via 1)
 (grid wire 1)
 (rule
(width 8)
(clear 8)
(clear 16 (type wire_area))
(clear 12 (type via_smd via_pin))
 )
 (layer L1 (type signal) (direction vert))
 (layer L2 (type signal) (direction hori) (rule (width 6)))
 (layer L3 (type power) (use_net GND))
 (layer L4 (type power) (use_net VDD VCC))
 (layer L5 (type signal) (direction vert) (rule (width 6)))
 (layer L6 (type signal) (direction hori))
 (keepout (rect signal 6192 942 8011 402))
 (keepout (rect L1 7980 625 10991 402))
 (keepout (rect L6 6186 3847 6391 905))
 (via_keepout (rect signal 8129 2537 9277 2407))
```

```
 (plane VDD
  (polygon L4 0 6180 400 6180 3850 7100 3850 7100 400    6180 400)
 )
 (plane VCC
  (polygon L4 0 7150 400 7150 3850 11000 3850 11000 400    7150 400)
 )
 (plane GND
  (polygon L3 0 6180 400 6180 3850 11000 3850 11000 400   6180 400)
 )
)
```

## Placement Data

```
#
#The following are component instances for the PCB.
#
(placement
 (unit MIL)
 (component cap.01uf
  (place c1 9273.0000 1514.0000 front  90)
  (place c2 8334.0000 1508.0000 front  0)
  (place c3 8439.0000 729.0000 front  0)
  (place c4 10443.0000 720.0000 front  0)
  (place c5 10452.0000 2103.0000 front  0)
  (place c6 8334.0000 2077.0000 front  0)
  (place c7 7284.0000 1263.0000 front  0)
  (place c8 6794.0000 1893.0000 front  0)
  (place c9 10443.0000 2707.0000 front  0)
  (place c10 9805.0000 3468.0000 front  0)
  (place c11 7494.0000 2742.0000 front  0)
  (place c12 6978.0000 3442.0000 front  0)
 )
 (component plcc20
  (place U17 10500.0000 725.0000 front  0)
  (place U37 9100.0000 725.0000 front  0)
  (place U42 9800.0000 1325.0000 front  0)
  (place U89 9800.0000 725.0000 front  0)
  (place U94 9100.0000 1325.0000 front  0)
  (place U97 8400.0000 1325.0000 front  0)
  (place U100 10500.0000 1925.0000 front  0)
  (place U101 8400.0000 1925.0000 front  0)
```

```
  (place U102 9100.0000 1925.0000 front  0)
  (place U114 10500.0000 1325.0000 front  0)
  (place U115 9800.0000 1925.0000 front  0)
 )
 (component qfp68
  (place U74 8650.0000 2733.0000 front  0)
 )
 (component qfp84
  (place U75 10733.0000 3086.0000 front  0)
  (place U76 7817.0000 3100.0000 front  0)
 )
 (component qfp100
  (place U71 7638.0000 1197.0000 front  0)
 )
 (component so24
  (place U30 8600.0000 1075.0000 front  0)
 )
)
#
# End of placement data
#
```

## Library Data

#The following library statement defines an image named
#qfp100. The first pin statement defines a pin that uses
#padstack 868. The pin name is 1. The pin is located at
#coordinates X=0, Y=0.

#All pin locations defined in this section are offset from the
#location defined by the place statement found earlier in the
#file. The padstack definitions are included in the library
#section. The second pin statement also uses padstack 868,
#names the pin 2, and specifies a location offset.

```
 (library
  (image qfp100
   (pin 868 1 0 0)
   (pin 868 2 0 31)
   (pin 868 3 0 63)
   (pin 868 4 0 94)
   (pin 868 5 0 126)
```

```
(pin 868 6 0 157)
(pin 868 7 0 189)
(pin 868 8 0 220)
(pin 868 9 0 252)
(pin 868 10 0 283)
(pin 868 11 0 315)
(pin 868 12 0 346)
(pin 868 13 0 378)
(pin 868 14 0 409)
(pin 868 15 0 441)
(pin 868 16 0 472)
(pin 868 17 0 504)
(pin 868 18 0 535)
(pin 868 19 0 567)
(pin 868 20 0 598)
(pin 868 21 0 630)
(pin 868 22 0 661)
(pin 868 23 0 693)
(pin 868 24 0 724)
(pin 868 25 0 756)
(pin 847 26 -160 916)
(pin 847 27 -191 916)
(pin 847 28 -223 916)
(pin 847 29 -254 916)
(pin 847 30 -286 916)
(pin 847 31 -317 916)
(pin 847 32 -349 916)
(pin 847 33 -380 916)
(pin 847 34 -412 916)
(pin 847 35 -443 916)
(pin 847 36 -475 916)
(pin 847 37 -506 916)
(pin 847 38 -538 916)
(pin 847 39 -569 916)
(pin 847 40 -601 916)
(pin 847 41 -632 916)
(pin 847 42 -664 916)
(pin 847 43 -695 916)
(pin 847 44 -727 916)
(pin 847 45 -758 916)
(pin 847 46 -790 916)
```

```
(pin 847 47 -821 916)
(pin 847 48 -853 916)
(pin 847 49 -884 916)
(pin 847 50 -916 916)
(pin 868 51 -1076 756)
(pin 868 52 -1076 724)
(pin 868 53 -1076 693)
(pin 868 54 -1076 661)
(pin 868 55 -1076 630)
(pin 868 56 -1076 598)
(pin 868 57 -1076 567)
(pin 868 58 -1076 535)
(pin 868 59 -1076 504)
(pin 868 60 -1076 472)
(pin 868 61 -1076 441)
(pin 868 62 -1076 409)
(pin 868 63 -1076 378)
(pin 868 64 -1076 346)
(pin 868 65 -1076 315)
(pin 868 66 -1076 283)
(pin 868 67 -1076 252)
(pin 868 68 -1076 220)
(pin 868 69 -1076 189)
(pin 868 70 -1076 157)
(pin 868 71 -1076 126)
(pin 868 72 -1076 94)
(pin 868 73 -1076 63)
(pin 868 74 -1076 31)
(pin 868 75 -1076 0)
(pin 847 76 -916 -160)
(pin 847 77 -884 -160)
(pin 847 78 -853 -160)
(pin 847 79 -821 -160)
(pin 847 80 -790 -160)
(pin 847 81 -758 -160)
(pin 847 82 -727 -160)
(pin 847 83 -695 -160)
(pin 847 84 -664 -160)
(pin 847 85 -632 -160)
(pin 847 86 -601 -160)
(pin 847 87 -569 -160)
```

```
  (pin 847 88 -538 -160)
  (pin 847 89 -506 -160)
  (pin 847 90 -475 -160)
  (pin 847 91 -443 -160)
  (pin 847 92 -412 -160)
  (pin 847 93 -380 -160)
  (pin 847 94 -349 -160)
  (pin 847 95 -317 -160)
  (pin 847 96 -286 -160)
  (pin 847 97 -254 -160)
  (pin 847 98 -223 -160)
  (pin 847 99 -191 -160)
  (pin 847 100 -160 -160)
 )
 (image plcc20
  (pin 763 1 0 0)
  (pin 763 2 50 0)
  (pin 763 3 100 0)
  (pin 784 4 175 75)
  (pin 784 5 175 125)
  (pin 784 6 175 175)
  (pin 784 7 175 225)
  (pin 784 8 175 275)
  (pin 763 9 100 350)
  (pin 763 10 50 350)
  (pin 763 11 0 350)
  (pin 763 12 -50 350)
  (pin 763 13 -100 350)
  (pin 784 14 -175 275)
  (pin 784 15 -175 225)
  (pin 784 16 -175 175)
  (pin 784 17 -175 125)
  (pin 784 18 -175 75)
  (pin 763 19 -100 0)
  (pin 763 20 -50 0)
 )
 (image qfp84
  (pin 724 1 0 0)
  (pin 724 2 0 50)
  (pin 724 3 0 100)
  (pin 724 4 0 150)
```

```
(pin 724 5 0 200)
(pin 724 6 0 250)
(pin 724 7 0 300)
(pin 724 8 0 350)
(pin 724 9 0 400)
(pin 724 10 0 450)
(pin 724 11 0 500)
(pin 703 12 -67 567)
(pin 703 13 -117 567)
(pin 703 14 -167 567)
(pin 703 15 -217 567)
(pin 703 16 -267 567)
(pin 703 17 -317 567)
(pin 703 18 -367 567)
(pin 703 19 -417 567)
(pin 703 20 -467 567)
(pin 703 21 -517 567)
(pin 703 22 -567 567)
(pin 703 23 -617 567)
(pin 703 24 -667 567)
(pin 703 25 -717 567)
(pin 703 26 -767 567)
(pin 703 27 -817 567)
(pin 703 28 -867 567)
(pin 703 29 -917 567)
(pin 703 30 -967 567)
(pin 703 31 -1017 567)
(pin 703 32 -1067 567)
(pin 724 33 -1134 500)
(pin 724 34 -1134 450)
(pin 724 35 -1134 400)
(pin 724 36 -1134 350)
(pin 724 37 -1134 300)
(pin 724 38 -1134 250)
(pin 724 39 -1134 200)
(pin 724 40 -1134 150)
(pin 724 41 -1134 100)
(pin 724 42 -1134 50)
(pin 724 43 -1134 0)
(pin 724 44 -1134 -50)
(pin 724 45 -1134 -100)
```

```
  (pin 724 46 -1134 -150)
  (pin 724 47 -1134 -200)
  (pin 724 48 -1134 -250)
  (pin 724 49 -1134 -300)
  (pin 724 50 -1134 -350)
  (pin 724 51 -1134 -400)
  (pin 724 52 -1134 -450)
  (pin 724 53 -1134 -500)
  (pin 703 54 -1067 -567)
  (pin 703 55 -1017 -567)
  (pin 703 56 -967 -567)
  (pin 703 57 -917 -567)
  (pin 703 58 -867 -567)
  (pin 703 59 -817 -567)
  (pin 703 60 -767 -567)
  (pin 703 61 -717 -567)
  (pin 703 62 -667 -567)
  (pin 703 63 -617 -567)
  (pin 703 64 -567 -567)
  (pin 703 65 -517 -567)
  (pin 703 66 -467 -567)
  (pin 703 67 -417 -567)
  (pin 703 68 -367 -567)
  (pin 703 69 -317 -567)
  (pin 703 70 -267 -567)
  (pin 703 71 -217 -567)
  (pin 703 72 -167 -567)
  (pin 703 73 -117 -567)
  (pin 703 74 -67 -567)
  (pin 724 75 0 -500)
  (pin 724 76 0 -450)
  (pin 724 77 0 -400)
  (pin 724 78 0 -350)
  (pin 724 79 0 -300)
  (pin 724 80 0 -250)
  (pin 724 81 0 -200)
  (pin 724 82 0 -150)
  (pin 724 83 0 -100)
  (pin 724 84 0 -50)
 )
```

```
(image cap.01uf
 (pin 1030 1 0 0)
 (pin 1030 2 110 0)
)
(image qfp68
 (pin 703 1 0 0)
 (pin 703 2 50 0)
 (pin 703 3 100 0)
 (pin 703 4 150 0)
 (pin 703 5 200 0)
 (pin 703 6 250 0)
 (pin 703 7 300 0)
 (pin 703 8 350 0)
 (pin 703 9 400 0)
 (pin 724 10 467 67)
 (pin 724 11 467 117)
 (pin 724 12 467 167)
 (pin 724 13 467 217)
 (pin 724 14 467 267)
 (pin 724 15 467 317)
 (pin 724 16 467 367)
 (pin 724 17 467 417)
 (pin 724 18 467 467)
 (pin 724 19 467 517)
 (pin 724 20 467 567)
 (pin 724 21 467 617)
 (pin 724 22 467 667)
 (pin 724 23 467 717)
 (pin 724 24 467 767)
 (pin 724 25 467 817)
 (pin 724 26 467 867)
 (pin 703 27 400 934)
 (pin 703 28 350 934)
 (pin 703 29 300 934)
 (pin 703 30 250 934)
 (pin 703 31 200 934)
 (pin 703 32 150 934)
 (pin 703 33 100 934)
 (pin 703 34 50 934)
 (pin 703 35 0 934)
 (pin 703 36 -50 934)
```

```
  (pin 703 37 -100 934)
  (pin 703 38 -150 934)
  (pin 703 39 -200 934)
  (pin 703 40 -250 934)
  (pin 703 41 -300 934)
  (pin 703 42 -350 934)
  (pin 703 43 -400 934)
  (pin 724 44 -467 867)
  (pin 724 45 -467 817)
  (pin 724 46 -467 767)
  (pin 724 47 -467 717)
  (pin 724 48 -467 667)
  (pin 724 49 -467 617)
  (pin 724 50 -467 567)
  (pin 724 51 -467 517)
  (pin 724 52 -467 467)
  (pin 724 53 -467 417)
  (pin 724 54 -467 367)
  (pin 724 55 -467 317)
  (pin 724 56 -467 267)
  (pin 724 57 -467 217)
  (pin 724 58 -467 167)
  (pin 724 59 -467 117)
  (pin 724 60 -467 67)
  (pin 703 61 -400 0)
  (pin 703 62 -350 0)
  (pin 703 63 -300 0)
  (pin 703 64 -250 0)
  (pin 703 65 -200 0)
  (pin 703 66 -150 0)
  (pin 703 67 -100 0)
  (pin 703 68 -50 0)
  (via_keepout (rect signal -400 100 400 850))
 )
 (image so24
  (pin 1052 1 0 0)
  (pin 1052 2 -50 0)
  (pin 1052 3 -100 0)
  (pin 1052 4 -150 0)
  (pin 1052 5 -200 0)
  (pin 1052 6 -250 0)
```

```
  (pin 1052 7 -300 0)
  (pin 1052 8 -350 0)
  (pin 1052 9 -400 0)
  (pin 1052 10 -450 0)
  (pin 1052 11 -500 0)
  (pin 1052 12 -550 0)
  (pin 1052 13 -550 -350)
  (pin 1052 14 -500 -350)
  (pin 1052 15 -450 -350)
  (pin 1052 16 -400 -350)
  (pin 1052 17 -350 -350)
  (pin 1052 18 -300 -350)
  (pin 1052 19 -250 -350)
  (pin 1052 20 -200 -350)
  (pin 1052 21 -150 -350)
  (pin 1052 22 -100 -350)
  (pin 1052 23 -50 -350)
  (pin 1052 24 0 -350)
 )
 (padstack 402
  (shape (circ signal 30))
 )
 (padstack 868
  (shape (rect L1 -62 -8 62 8))
 )
 (padstack 847
  (shape (rect L1 -8 -62 8 62))
 )
 (padstack 763
  (shape (rect L1 -12 -40 12 40))
 )
 (padstack 784
  (shape (rect L1 -40 -12 40 12))
 )
 (padstack 703
  (shape (rect L1 -15 -35 15 35))
 )
 (padstack 724
  (shape (rect L1 -35 -15 35 15))
 )
```

```
  (padstack 1083
   (shape (rect L1 -30 -40 30 40))
  )
  (padstack 805
   (shape (rect L1 -40 -30 40 30))
  )
  (padstack 1030
   (shape (rect L6 -40 -30 40 30))
  )
  (padstack 1104
   (shape (rect L6 -40 -30 40 30))
  )
  (padstack 1052
   (shape (rect L1 -13 -40 13 40))
  )
  (padstack VIA
   (shape (circ signal 30))
  )
 )
# End of library data
```

## Network Data

```
# Network data for Sample PCB
#
 (network
  (net GND
   (pins U75-7 U75-6 U75-5 U75-4 U75-3 U75-2 U115-16
         U115-15 U115-14 U115-13 U115-12 U37-5 U30-24
         U30-23 U30-22 U76-71 U76-70 U76-68 U76-67 U76-66
         U76-63 U30-20 U89-10 U89-9 U89-8 U89-4 U76-84
         U76-83 U76-82 U76-80 U71-51 U71-50 U71-46 U71-44
         U71-43 U71-41 U71-40 U71-39 U71-38)
   (rule (width 16))
  )
  (net VDD
   (pins U101-11 U101-10 U101-8 U101-6 U101-3 U100-20
         U100-13 U71-95 U71-94 U71-93 U71-92 U71-91 U71-90
         U71-89 U71-88 U17-19 U17-16 U17-15 U17-14 U17-13
         U17-11 U17-10 U97-16 U97-14 U97-13 U97-11 U97-10
         U97-4 U97-3 U42-7 U42-4 U42-1 U37-20 U37-18 U37-15
         U37-14 U37-13 U42-20 U42-19 U42-18 U42-17 U42-16
         U42-15 U42-14)
   (rule (width 16))
  )
```

```
(net VCC
 (pins U71-16 U71-14 U71-13 U71-6 U71-4 U71-2 U71-1
       U42-13 U42-12 )
 (rule (width 16))
 )
(net CPU-D/C#
 (pins U71-11 U89-2 U102-8)
)
(net CPU-M/IO#
 (pins U71-15 U89-1 U102-7)
)
(net MC-BD2
 (pins U76-39 U74-18 U30-2 U114-9)
)
(net MC-BD3
 (pins U76-38 U74-19 U30-1 U97-5)
)
(net MC-BD5
 (pins U76-36 U74-22 U30-5 U17-7)
)
(net MC-BD7
 (pins U76-34 U74-24 U30-8 U75-71)
)
(net CPU-W/R#
 (pins U71-12 U89-3 U102-9)
)
(net CLK2B
 (pins U115-1 U100-1)
)
(net MC-BD0
 (pins U76-42 U74-16 U30-10 U37-19)
)
(net MC-BD1
 (pins U76-41 U74-17 U30-14 U75-35)
)
(net MC-BD4
 (pins U76-37 U74-20 U30-18 U75-38)
)
(net MC-BD6
 (pins U76-35 U74-23 U30-16 U75-41)
)
```

```
(net CPU-RESET
 (pins U89-6 U17-6)
)
(net CPU-HLDA
 (pins U89-5 U17-12 U75-22)
)
(net LCL-CMD#
 (pins U102-17 U100-2)
)
(net MC-CMD#
 (pins U74-6 U100-5)
)
(net DCD-INT-ACK#
 (pins U94-3 U89-19)
)
(net SA0
 (pins U76-52 U75-52)
)
(net SA1
 (pins U76-53 U75-53)
)
(net SA2
 (pins U71-3 U75-54)
)
(net MC-TO-MEMB#
 (pins U42-9 U100-3)
)
(net LBC-TO-MEM#
 (pins U42-5 U100-4)
)
(net SBUS3
 (pins U42-8  U100-15)
)
(net MEM-CMD#
 (pins U71-21 U100-12)
)
(net MEM-M/IO#
 (pins U71-9 U100-14)
)
(net MEM-ALE#
 (pins U71-20 U100-16)
```

```
  )
  (net MEM-S0#
   (pins U71-7 U100-17)
  )
  (net MEM-S1#
   (pins U71-8 U100-18)
  )
  (net Q9
   (pins U97-18 U102-10)
  )
  (net Q8
   (pins U94-18 U97-8 U100-8)
  )
 (net CLKA#
   (pins U17-18 U102-12 U101-9)
  )
  (net LEPB-ADS#
   (pins U102-6 U101-2)
  )
  (net SYS-RESET#
   (pins U76-29 U101-4)
  )
  (net CONVERT#
   (pins U102-5 U101-12)
  )
  (net Q2
   (pins U102-4 U97-12 U71-76 U114-20)
   (fromto U102-4 U71-76 (rule (width 5)))
   (fromto U71-76 U97-12 (rule (width 6)))
   (fromto U97-12 U114-20 (rule (width 7)))
  )
  (net Q1
   (pins U102-18 U101-14 U76-33 U17-9)
  )
  (net Q0
   (pins U102-2 U101-15 U76-40 U114-2)
  )
  (net LCL-CH-RDY#
   (pins U17-2 U101-18)
  )
```

```
(net CLKB
 (pins U89-7 U17-17 U94-5)
)
(net POS-CARD-EN
 (pins U74-43 U37-4)
)
(net GEN-CH-CHK#
 (pins U74-34 U75-14)
)
(net LCLL-S1#
 (pins U76-69 U42-3 U114-7 U75-58)
 (source U76-69)
 (load U75-58 U114-7)
 (terminator U42-3)
 (rule (reorder daisy))
)
(net SD7
 (pins U71-5 U76-51 U75-73)
)
(net SD6
 (pins U71-53 U76-50 U114-17 U75-61)
)
(net SD5
 (pins U71-62 U76-49 U75-79)
)
(net SD4
 (order U71-87 U76-48 U114-11 U75-1)
)
(net SD2
 (pins U71-98 U76-45 U75-9)
)
(net SD1
 (pins U71-85 U76-44 U75-18)
)
(net SD0
 (pins U71-45 U76-43 U75-43)
)
(net MC-BA0
 (pins U76-32 U74-26 U75-32)
)
```

```
  (net MCL-RD#
   (pins U76-27 U75-24)
  )
  (net SD3
   (pins U71-75 U76-47 U114-15 U75-47)
   (rule (reorder daisy))
  )
  (net WATCH
   (pins U76-13 U75-77)
  )
  (net XA15
   (pins U71-24 U74-56)
  )
 (net XA14
   (pins U71-73 U74-57)
  )
  (net XA13
   (pins U71-55 U74-58)
  )
  (net XA12
   (pins U71-42 U74-59)
  )
  (net MC-M/IO#
   (pins U74-3 U37-1)
  )
  (net MC-S0#
   (pins U74-1 U37-2)
   (layer_rule L1 (rule (width 10)))
   (layer_rule L6 (rule (width 10)))
  )
  (net MC-S1#
   (pins U74-2 U37-3)
  )
  (net BLITZ-RDY
   (pins U71-68 U17-4)
  )
  (net LCL-LEPB#
   (pins U97-17 U17-8)
   (layer_rule L1 (rule (width 10)))
   (layer_rule L6 (rule (width 10)))
  )
```

```
(net UPGD-PASS-A2
 (pins U75-70)
)
(net LCL-MCB#
 (pins U76-79 U42-6 U115-3)
)
(net MC-CMDA#
 (pins U76-28 U115-8 U75-27)
)
(net LCL-REFRESH#
 (pins U71-52 U76-11 U100-9)
)
(net POS-IO0
 (pins U76-20 U74-39)
)
(net POS-IO1
 (pins U76-21 U74-37)
)
(net POS-IO3
 (pins U76-23 U74-35)
)
(net CPUL-W/R#
 (pins U42-2 U115-4)
)
(net CLK2A
 (pins U71-17 U97-1 U17-1)
)
(net LCL-CMDB#
 (pins U76-65 U75-57)
)
(net SA3
 (pins U76-55 U75-55)
)
(net MC-BA1
 (pins U76-31 U74-27 U75-31)
)
(net MC-BA2
 (pins U76-30 U74-28 U75-30)
)
```

```
 (net MC-BA3
  (pins U76-22 U74-33 U75-29)
 )
 (net SYS-RESET
  (pins U71-71 U75-69)
 )
 (net DCD-P94#
  (pins U71-28 U76-74)
 )
 (net DCD-MEM#
  (pins U71-33 U94-8)
 )
 (net CAS/RAS#
  (pins U71-57 U89-11)
 )
 (net SBUS1
  (pins U94-2 U74-9 U75-63)
 )
      (net BUS-REQ#
  (pins U97-15 U74-10 U75-13)
 )
 (net BUS-GNT#
  (pins U97-19 U74-11)
)
 (net POS-CONF-SEC
  (pins U76-73 U74-40 U75-15)
 )
 (net MC-CH-RST
  (pins U74-44 U75-16)
)
 (net CLKC
  (pins U76-64 U75-64)
)
 (net SBUS2
  (pins U94-6 U75-37)
 )
 (net ASSIST-NEEDE
  (pins U76-81 U75-80)
)
```

```
(net DCD-HLT-SHUT
 (pins U94-4 U89-18)
)
(net CLK2C
 (pins U102-1 U101-1)
 (rule (width 15))
)
(net CPU-IO#
 (pins U94-1 U89-12)
)
(net DCD-CO-PROC#
 (pins U94-7 U89-17)
)
(net LCL-MC-DCD#
 (pins U97-9 U94-12)
)
(net LCL-SMP-DCD#
 (pins U97-7 U94-19)
)
(net LCL-MEM-DCD#
 (pins U97-6 U94-15)
)
(net DEL-LCL-MC-W
 (pins U42-11 U115-17)
)
(net LCL-SREG-DCD
 (pins U94-11 U75-59)
)
(net MC-SREG-DCD1
 (pins U76-24 U74-29 U37-16)
 (net_number 691)
)
(net MC-SREG-DCD#
 (pins U37-17 U75-23)
)
(net $20N98
 (pins U71-49 U71-48 U71-47)
)
(net TEMP154
 (pins U71-70 U71-66 U71-61 U71-59)
)
```

```
  (net MEM-ADS#
   (pins U71-22 U71-10)
  )
  (net SPEC/NORM#
   (pins U89-16 U76-78)
  )
  (net CTRLA-UPGD-P
   (pins U76-75 U74-8 U75-83)
  )
  (net DEL-MC-TO-ME
   (pins U94-14 U101-19 U100-19)
  )
  (net XDATA-0 (pins U101-7 U71-23))
  (net XDATA-3 (pins U71-18 U101-5))
  (class C1 SD6 XA13 CAS/RAS# TEMP154
          SD5 BLITZ-RDY SYS-RESET
          (rule (width 5) (reorder daisy))
  )
  (class C2 LCL-MC-DCD# LCL-SMP-DCD#
          LCL-MEM-DCD#
          (layer_rule L2 (rule (width 15)))
          (layer_rule L5 (rule (width 15)))
  )
 )
```

## Prerouted Wiring Data

```
(wiring
 (resolution MIL 10)
# Net SD2
 (wire (path L1 80  74150 10370 74150 15280 74460 15280
     74460 18550 73910 18550)
    (net SD2 )
    (type protect)
    (attr fanout))
 (wire (path L6 80  73910 18550 73910 19730 66510 19730)
    (net SD2 )
    (type protect))
 (wire (path L1 80  66510 19730 65910 19730 65910 30000)
    (net SD2 )
    (type protect))
```

```
 (wire (path L1 80  65910 30000 66830 30000)
    (net SD2 )
    (type protect)
    (attr fanout))
 (wire (path L6 80  65910 30000 106280 30000)
    (net SD2 )
    (type protect))
 (wire (path L1 80  106280 30000 106110 30000 106110
           34860 107330 34860)
    (net SD2)
    (type protect)
    (attr fanout))
 (via VIA 73910 18550 (net SD2)
    (type protect)
    (attr fanout) )
 (via VIA 66510 19730 (net SD2)
    (type protect) )
 (via VIA 65910 30000 (net SD2)
    (type protect)
    (attr fanout))
 (via VIA 106280 30000 (net SD2)
    (type protect)
    (attr fanout) )
)
```

# Sample Design File with High Speed Rules

The following design file sample includes high speed rules and shows design, placement, library, network, and prerouted wiring types of data.

## Design Data

```
(PCB test_brd_20
 (resolution MIL 10)
 (structure
  (boundary
   (rect pcb 5956.00000 345.90000 11202.00000
          3888.00000)
  )
```

```
  (boundary
   (rect signal 6180 400 11000 3850)
  )
  (via VIA)
  (grid via 1)
  (grid wire 1)
# Global, pcb rules
  (rule
    (width 8)
    (clear 8)
    (clear 16 (type wire_area ))
    (clear 12 (type via_smd via_pin))
  )
  (layer L1  (type signal) (direction vert))
  (layer L2  (type signal) (direction hori) (rule (width 6)))
  (layer L3  (type power) (use_net GND))
  (layer L4  (type power) (use_net VDD VCC))
  (layer L5  (type signal) (direction vert) (rule (width 6)))
  (layer L6  (type signal) (direction hori))
  (keepout (rect signal 6192 942 8011 402))
  (keepout (rect L1 7980 625 10991 402))
  (keepout (rect L6 6186 3847 6391 905))
  (via_keepout (rect signal 8129 2537 9277 2407))
  (plane VDD
   (polygon L4 0 6180 400 6180 3850 7100 3850 7100
          400 6180 400)
  )
  (plane VCC
   (polygon L4 0 7150 400 7150 3850 11000 3850 11000
          400 7150 400)
  )
  (plane GND
   (polygon L3 0 6180 400 6180 3850 11000 3850 11000
          400 6180 400)
  )
)
```

## Placement Data

```
#
# The following are component instances for the PCB.
#
(placement
 (unit MIL)
 (component cap.01uf
  (place c1 9273.0000 1514.0000 front  90)
  (place c2 8334.0000 1508.0000 front  0)
  (place c3 8439.0000 729.0000 front  0)
  (place c4 10443.0000 720.0000 front  0)
  (place c5 10452.0000 2103.0000 front  0)
  (place c6 8334.0000 2077.0000 front  0)
  (place c7 7284.0000 1263.0000 front  0)
  (place c8 6794.0000 1893.0000 front  0)
  (place c9 10443.0000 2707.0000 front  0)
  (place c10 9805.0000 3468.0000 front  0)
  (place c11 7494.0000 2742.0000 front  0)
  (place c12 6978.0000 3442.0000 front  0)
 )
 (component plcc20
  (place U17 10500.0000 725.0000 front  0)
  (place U37 9100.0000 725.0000 front  0)
  (place U42 9800.0000 1325.0000 front  0)
  (place U89 9800.0000 725.0000 front  0)
  (place U94 9100.0000 1325.0000 front  0)
  (place U97 8400.0000 1325.0000 front  0)
  (place U100 10500.0000 1925.0000 front  0)
  (place U101 8400.0000 1925.0000 front  0)
  (place U102 9100.0000 1925.0000 front  0)
  (place U114 10500.0000 1325.0000 front  0)
  (place U115 9800.0000 1925.0000 front  0)
 )
 (component qfp68
  (place U74 8650.0000 2733.0000 front  0)
 )
 (component qfp84
  (place U75 10733.0000 3086.0000 front  0)
  (place U76 7817.0000 3100.0000 front  0)
 )
```

```
 (component qfp100
  (place U71 7638.0000 1197.0000 front  0)
 )
 (component so24
  (place U30 8600.0000 1075.0000 front  0)
 )
)
#
# End of placement data
#
```

## Library Data

#The following library statement defines an image named
#qfp100. The first pin statement defines a pin that uses
#padstack 868. The pin name is 1. The pin is located at
#coordinates X=0, Y=0.

#All pin locations defined in this section are offset from the
#location defined by the place statement found earlier in the
#file. The padstack definitions are included in the library
#section. The second pin statement also uses padstack 868, #names the pin 2, and specifies a location offset.

```
 (library
  (image qfp100
   (pin 868 1 0 0)
   (pin 868 2 0 31)
   (pin 868 3 0 63)
   (pin 868 4 0 94)
   (pin 868 5 0 126)
   (pin 868 6 0 157)
   (pin 868 7 0 189)
   (pin 868 8 0 220)
   (pin 868 9 0 252)
   (pin 868 10 0 283)
   (pin 868 11 0 315)
   (pin 868 12 0 346)
   (pin 868 13 0 378)
   (pin 868 14 0 409)
   (pin 868 15 0 441)
   (pin 868 16 0 472)
```

```
(pin 868 17 0 504)
(pin 868 18 0 535)
(pin 868 19 0 567)
(pin 868 20 0 598)
(pin 868 21 0 630)
(pin 868 22 0 661)
(pin 868 23 0 693)
(pin 868 24 0 724)
(pin 868 25 0 756)
(pin 847 26 -160 916)
(pin 847 27 -191 916)
(pin 847 28 -223 916)
(pin 847 29 -254 916)
(pin 847 30 -286 916)
(pin 847 31 -317 916)
(pin 847 32 -349 916)
(pin 847 33 -380 916)
(pin 847 34 -412 916)
(pin 847 35 -443 916)
(pin 847 36 -475 916)
(pin 847 37 -506 916)
(pin 847 38 -538 916)
(pin 847 39 -569 916)
(pin 847 40 -601 916)
(pin 847 41 -632 916)
(pin 847 42 -664 916)
(pin 847 43 -695 916)
(pin 847 44 -727 916)
(pin 847 45 -758 916)
(pin 847 46 -790 916)
(pin 847 47 -821 916)
(pin 847 48 -853 916)
(pin 847 49 -884 916)
(pin 847 50 -916 916)
(pin 868 51 -1076 756)
(pin 868 52 -1076 724)
(pin 868 53 -1076 693)
(pin 868 54 -1076 661)
(pin 868 55 -1076 630)
(pin 868 56 -1076 598)
(pin 868 57 -1076 567)
```

```
(pin 868 58 -1076 535)
(pin 868 59 -1076 504)
(pin 868 60 -1076 472)
(pin 868 61 -1076 441)
(pin 868 62 -1076 409)
(pin 868 63 -1076 378)
(pin 868 64 -1076 346)
(pin 868 65 -1076 315)
(pin 868 66 -1076 283)
(pin 868 67 -1076 252)
(pin 868 68 -1076 220)
(pin 868 69 -1076 189)
(pin 868 70 -1076 157)
(pin 868 71 -1076 126)
(pin 868 72 -1076 94)
(pin 868 73 -1076 63)
(pin 868 74 -1076 31)
(pin 868 75 -1076 0)
(pin 847 76 -916 -160)
(pin 847 77 -884 -160)
(pin 847 78 -853 -160)
(pin 847 79 -821 -160)
(pin 847 80 -790 -160)
(pin 847 81 -758 -160)
(pin 847 82 -727 -160)
(pin 847 83 -695 -160)
(pin 847 84 -664 -160)
(pin 847 85 -632 -160)
(pin 847 86 -601 -160)
(pin 847 87 -569 -160)
(pin 847 88 -538 -160)
(pin 847 89 -506 -160)
(pin 847 90 -475 -160)
(pin 847 91 -443 -160)
(pin 847 92 -412 -160)
(pin 847 93 -380 -160)
(pin 847 94 -349 -160)
(pin 847 95 -317 -160)
(pin 847 96 -286 -160)
(pin 847 97 -254 -160)
(pin 847 98 -223 -160)
```

```
 (pin 847 99 -191 -160)
 (pin 847 100 -160 -160)
)
(image plcc20
 (pin 763 1 0 0)
 (pin 763 2 50 0)
 (pin 763 3 100 0)
 (pin 784 4 175 75)
 (pin 784 5 175 125)
 (pin 784 6 175 175)
 (pin 784 7 175 225)
 (pin 784 8 175 275)
 (pin 763 9 100 350)
 (pin 763 10 50 350)
 (pin 763 11 0 350)
 (pin 763 12 -50 350)
 (pin 763 13 -100 350)
 (pin 784 14 -175 275)
 (pin 784 15 -175 225)
 (pin 784 16 -175 175)
 (pin 784 17 -175 125)
 (pin 784 18 -175 75)
 (pin 763 19 -100 0)
 (pin 763 20 -50 0)
)
(image qfp84
 (pin 724 1 0 0)
 (pin 724 2 0 50)
 (pin 724 3 0 100)
 (pin 724 4 0 150)
 (pin 724 5 0 200)
 (pin 724 6 0 250)
 (pin 724 7 0 300)
 (pin 724 8 0 350)
 (pin 724 9 0 400)
 (pin 724 10 0 450)
 (pin 724 11 0 500)
 (pin 703 12 -67 567)
 (pin 703 13 -117 567)
 (pin 703 14 -167 567)
 (pin 703 15 -217 567)
```

```
(pin 703 16 -267 567)
(pin 703 17 -317 567)
(pin 703 18 -367 567)
(pin 703 19 -417 567)
(pin 703 20 -467 567)
(pin 703 21 -517 567)
(pin 703 22 -567 567)
(pin 703 23 -617 567)
(pin 703 24 -667 567)
(pin 703 25 -717 567)
(pin 703 26 -767 567)
(pin 703 27 -817 567)
(pin 703 28 -867 567)
(pin 703 29 -917 567)
(pin 703 30 -967 567)
(pin 703 31 -1017 567)
(pin 703 32 -1067 567)
(pin 724 33 -1134 500)
(pin 724 34 -1134 450)
(pin 724 35 -1134 400)
(pin 724 36 -1134 350)
(pin 724 37 -1134 300)
(pin 724 38 -1134 250)
(pin 724 39 -1134 200)
(pin 724 40 -1134 150)
(pin 724 41 -1134 100)
(pin 724 42 -1134 50)
(pin 724 43 -1134 0)
(pin 724 44 -1134 -50)
(pin 724 45 -1134 -100)
(pin 724 46 -1134 -150)
(pin 724 47 -1134 -200)
(pin 724 48 -1134 -250)
(pin 724 49 -1134 -300)
(pin 724 50 -1134 -350)
(pin 724 51 -1134 -400)
(pin 724 52 -1134 -450)
(pin 724 53 -1134 -500)
(pin 703 54 -1067 -567)
(pin 703 55 -1017 -567)
(pin 703 56 -967 -567)
```

```
  (pin 703 57 -917 -567)
  (pin 703 58 -867 -567)
  (pin 703 59 -817 -567)
  (pin 703 60 -767 -567)
  (pin 703 61 -717 -567)
  (pin 703 62 -667 -567)
  (pin 703 63 -617 -567)
  (pin 703 64 -567 -567)
  (pin 703 65 -517 -567)
  (pin 703 66 -467 -567)
  (pin 703 67 -417 -567)
  (pin 703 68 -367 -567)
  (pin 703 69 -317 -567)
  (pin 703 70 -267 -567)
  (pin 703 71 -217 -567)
  (pin 703 72 -167 -567)
  (pin 703 73 -117 -567)
  (pin 703 74 -67 -567)
  (pin 724 75 0 -500)
  (pin 724 76 0 -450)
  (pin 724 77 0 -400)
  (pin 724 78 0 -350)
  (pin 724 79 0 -300)
  (pin 724 80 0 -250)
  (pin 724 81 0 -200)
  (pin 724 82 0 -150)
  (pin 724 83 0 -100)
  (pin 724 84 0 -50)
 )
 (image cap.01uf
  (pin 1030 1 0 0)
  (pin 1030 2 110 0)
 )
 (image qfp68
  (pin 703 1 0 0)
  (pin 703 2 50 0)
  (pin 703 3 100 0)
  (pin 703 4 150 0)
  (pin 703 5 200 0)
  (pin 703 6 250 0)
  (pin 703 7 300 0)
```

```
(pin 703 8 350 0)
(pin 703 9 400 0)
(pin 724 10 467 67)
(pin 724 11 467 117)
(pin 724 12 467 167)
(pin 724 13 467 217)
(pin 724 14 467 267)
(pin 724 15 467 317)
(pin 724 16 467 367)
(pin 724 17 467 417)
(pin 724 18 467 467)
(pin 724 19 467 517)
(pin 724 20 467 567)
(pin 724 21 467 617)
(pin 724 22 467 667)
(pin 724 23 467 717)
(pin 724 24 467 767)
(pin 724 25 467 817)
(pin 724 26 467 867)
(pin 703 27 400 934)
(pin 703 28 350 934)
(pin 703 29 300 934)
(pin 703 30 250 934)
(pin 703 31 200 934)
(pin 703 32 150 934)
(pin 703 33 100 934)
(pin 703 34 50 934)
(pin 703 35 0 934)
(pin 703 36 -50 934)
(pin 703 37 -100 934)
(pin 703 38 -150 934)
(pin 703 39 -200 934)
(pin 703 40 -250 934)
(pin 703 41 -300 934)
(pin 703 42 -350 934)
(pin 703 43 -400 934)
(pin 724 44 -467 867)
(pin 724 45 -467 817)
(pin 724 46 -467 767)
(pin 724 47 -467 717)
(pin 724 48 -467 667)
```

```
  (pin 724 49 -467 617)
  (pin 724 50 -467 567)
  (pin 724 51 -467 517)
  (pin 724 52 -467 467)
  (pin 724 53 -467 417)
  (pin 724 54 -467 367)
  (pin 724 55 -467 317)
  (pin 724 56 -467 267)
  (pin 724 57 -467 217)
  (pin 724 58 -467 167)
  (pin 724 59 -467 117)
  (pin 724 60 -467 67)
  (pin 703 61 -400 0)
  (pin 703 62 -350 0)
  (pin 703 63 -300 0)
  (pin 703 64 -250 0)
  (pin 703 65 -200 0)
  (pin 703 66 -150 0)
  (pin 703 67 -100 0)
  (pin 703 68 -50 0)
  (via_keepout (rect signal -400 100 400 850))
 )
 (image so24
  (pin 1052 1 0 0)
  (pin 1052 2 -50 0)
  (pin 1052 3 -100 0)
  (pin 1052 4 -150 0)
  (pin 1052 5 -200 0)
  (pin 1052 6 -250 0)
  (pin 1052 7 -300 0)
  (pin 1052 8 -350 0)
  (pin 1052 9 -400 0)
  (pin 1052 10 -450 0)
  (pin 1052 11 -500 0)
  (pin 1052 12 -550 0)
  (pin 1052 13 -550 -350)
  (pin 1052 14 -500 -350)
  (pin 1052 15 -450 -350)
  (pin 1052 16 -400 -350)
  (pin 1052 17 -350 -350)
  (pin 1052 18 -300 -350)
```

```
 (pin 1052 19 -250 -350)
 (pin 1052 20 -200 -350)
 (pin 1052 21 -150 -350)
 (pin 1052 22 -100 -350)
 (pin 1052 23 -50 -350)
 (pin 1052 24 0 -350)
)
(padstack 402
 (shape (circ signal 30))
)
(padstack 868
 (shape (rect L1 -62 -8 62 8))
)
(padstack 847
 (shape (rect L1 -8 -62 8 62))
)
(padstack 763
 (shape (rect L1 -12 -40 12 40))
)
(padstack 784
 (shape (rect L1 -40 -12 40 12))
)
(padstack 703
 (shape (rect L1 -15 -35 15 35))
)
(padstack 724
 (shape (rect L1 -35 -15 35 15))
)
(padstack 1083
 (shape (rect L1 -30 -40 30 40))
)
(padstack 805
 (shape (rect L1 -40 -30 40 30))
)
(padstack 1030
 (shape (rect L6 -40 -30 40 30))
)
(padstack 1104
 (shape (rect L6 -40 -30 40 30))
)
```

```
  (padstack 1052
   (shape (rect L1 -13 -40 13 40))
  )
  (padstack VIA
   (shape (circ signal 30))
  )
 )
#
# End of library data
#
```

## Network data

```
 (network
  (net GND
   (pins U75-7 U75-6 U75-5 U75-4 U75-3 U75-2 U115-16
         U115-15 U115-14 U115-13 U115-12 U37-5 U30-24
         U30-23 U30-22 U76-71 U76-70 U76-68 U76-67 U76-66
         U76-63 U30-20 U89-10 U89-9 U89-8 U89-4 U76-84
         U76-83 U76-82 U76-80 U71-51 U71-50 U71-46 U71-44
         U71-43 U71-41 U71-40 U71-39 U71-38)
   (rule (width 16))
  )
  (net VDD
   (pins U101-11 U101-10 U101-8 U101-6 U101-3 U100-20
         U100-13 U71-95 U71-94 U71-93 U71-92 U71-91 U71-90
         U71-89 U71-88 U17-19 U17-16 U17-15 U17-14 U17-13
         U17-11 U17-10 U97-16 U97-14 U97-13 U97-11 U97-10
         U97-4 U97-3 U42-7 U42-4 U42-1 U37-20 U37-18 U37-15
         U37-14 U37-13 U42-20 U42-19 U42-18 U42-17 U42-16
         U42-15 U42-14)
   (rule (width 16))
   )
  (net VCC
   (pins U71-16 U71-14 U71-13 U71-6 U71-4 U71-2 U71-1
         U42-13 U42-12 )
   (rule (width 16))
   )
  (net CPU-D/C#
   (pins U71-11 U89-2 U102-8)
  )
  (net CPU-M/IO#
   (pins U71-15 U89-1 U102-7)
  )
```

```
(net MC-BD2
 (pins U76-39 U74-18 U30-2 U114-9)
)
(net MC-BD3
 (pins U76-38 U74-19 U30-1 U97-5)
)
(net MC-BD5
 (pins U76-36 U74-22 U30-5 U17-7)
)
(net MC-BD7
 (pins U76-34 U74-24 U30-8 U75-71)
)
(net CPU-W/R#
 (pins U71-12 U89-3 U102-9)
)
(net CLK2B
 (pins U115-1 U100-1)
)
(net MC-BD0
 (pins U76-42 U74-16 U30-10 U37-19)
)
(net MC-BD1
 (pins U76-41 U74-17 U30-14 U75-35)
)
(net MC-BD4
 (pins U76-37 U74-20 U30-18 U75-38)
)
(net MC-BD6
 (pins U76-35 U74-23 U30-16 U75-41)
)
(net CPU-RESET
 (pins U89-6 U17-6)
)
(net CPU-HLDA
 (pins U89-5 U17-12 U75-22)
)
(net LCL-CMD#
 (pins U102-17 U100-2)
)
```

```
(net MC-CMD#
 (pins U74-6 U100-5)
)
(net DCD-INT-ACK#
 (pins U94-3 U89-19)
)
(net SA0
 (pins U76-52 U75-52)
)
(net SA1
 (pins U76-53 U75-53)
)
(net SA2
 (pins U71-3 U75-54)
)
(net MC-TO-MEMB#
 (pins U42-9 U100-3) (circuit (shield on (use_net GND)))
)
(net LBC-TO-MEM#
 (pins U42-5 U100-4)
)
(net SBUS3
 (pins U42-8 U100-15)
)
(net MEM-CMD#
 (pins U71-21 U100-12)
)
(net MEM-M/IO#
 (pins U71-9 U100-14)
)
(net MEM-ALE#
 (pins U71-20 U100-16)
)
(net MEM-S0#
 (pins U71-7 U100-17)
)
(net MEM-S1#
 (pins U71-8 U100-18)
)
```

```
  (net Q9
   (pins U97-18 U102-10)
  )
  (net Q8
   (pins  U94-18 U97-8 U100-8)
  )
 (net CLKA#
   (pins U17-18 U102-12 U101-9)
  )
  (net LEPB-ADS#
   (pins U102-6 U101-2)
  )
  (net SYS-RESET#
   (pins U76-29 U101-4)
  )
  (net CONVERT#
   (pins U102-5 U101-12)
  )
  (net Q2
   (pins U102-4 U97-12 U71-76 U114-20)
   (fromto U102-4 U71-76 (rule (width 5)))
   (fromto U71-76 U97-12 (rule (width 6)))
   (fromto U97-12 U114-20 (rule (width 7)))
  )
  (net Q1
   (pins U102-18 U101-14 U76-33 U17-9)
  )
  (net Q0
   (pins U102-2 U101-15 U76-40 U114-2)
  )
  (net LCL-CH-RDY#
   (pins U17-2 U101-18)
  )
  (net CLKB
   (pins U89-7 U17-17 U94-5)
  )
  (net POS-CARD-EN
   (pins U74-43 U37-4)
  )
```

```
(net GEN-CH-CHK#
 (pins U74-34 U75-14)
)
(net LCLL-S1#
 (pins U76-69 U42-3 U114-7 U75-58)
 (source U76-69)
 (load U75-58 U114-7)
 (terminator U42-3)
 (rule (reorder daisy))
)
(net SD7
 (pins U71-5 U76-51 U75-73)
)
(net SD6
 (pins U71-53 U76-50 U114-17 U75-61)
)
(net SD5
 (pins U71-62 U76-49 U75-79)
)
(net SD4
 (order U71-87 U76-48 U114-11 U75-1)
)
(net SD2
 (pins U71-98 U76-45 U75-9)
)
(net SD1
 (pins U71-85 U76-44 U75-18)
)
(net SD0
 (pins U71-45 U76-43 U75-43)
)
(net MC-BA0
 (pins U76-32 U74-26 U75-32)
)
(net MCL-RD#
 (pins U76-27 U75-24)
)
(net SD3
 (pins U71-75 U76-47 U114-15 U75-47)
 (rule (reorder daisy))
)
```

```
(net WATCH
 (pins U76-13 U75-77)
)
(net XA15
 (pins U71-24 U74-56)
 (circuit (shield on (use_net GND)))
)
(net XA14
 (pins U71-73 U74-57)
)
(net XA13
 (pins U71-55 U74-58)
)
(net XA12
 (pins U71-42 U74-59)
)
(net MC-M/IO#
 (pins U74-3 U37-1)
)
(net MC-S0#
 (pins U74-1 U37-2)
 (layer_rule L1 (rule (width 10)))
 (layer_rule L6 (rule (width 10)))
 (circuit (use_layer L1 L6))
)
(net MC-S1#
 (pins U74-2 U37-3)
)
(net BLITZ-RDY
 (pins U71-68 U17-4)
)
(net LCL-LEPB#
 (pins U97-17 U17-8)
 (layer_rule L1 (rule (width 10)))
 (layer_rule L6 (rule (width 10)))
 (circuit (use_layer L1 L6))
)
(net UPGD-PASS-A2
 (pins U75-70)
)
```

```
(net LCL-MCB#
 (pins U76-79 U42-6 U115-3)
)
(net MC-CMDA#
 (pins U76-28 U115-8 U75-27)
)
(net LCL-REFRESH#
 (pins U71-52 U76-11 U100-9)
)
(net POS-IO0
 (pins U76-20 U74-39)
)
(net POS-IO1
 (pins U76-21 U74-37)
)
(net POS-IO3
 (pins U76-23 U74-35)
)
(net CPUL-W/R#
 (pins U42-2 U115-4)
)
(net CLK2A
 (pins U71-17 U97-1 U17-1)
)
(net LCL-CMDB#
 (pins U76-65 U75-57)
)
(net SA3
 (pins U76-55 U75-55)
)
(net MC-BA1
 (pins U76-31 U74-27 U75-31)
)
(net MC-BA2
 (pins U76-30 U74-28 U75-30)
)
(net MC-BA3
 (pins U76-22 U74-33 U75-29)
)
```

```
 (net SYS-RESET
  (pins U71-71 U75-69)
 )
 (net DCD-P94#
  (pins U71-28 U76-74)
 )
 (net DCD-MEM#
  (pins U71-33 U94-8)
 )
 (net CAS/RAS#
  (pins U71-57 U89-11)
 )
 (net SBUS1
  (pins U94-2 U74-9 U75-63)
 )
 (net BUS-REQ#
  (pins U97-15 U74-10 U75-13)
 )
 (net BUS-GNT#
  (pins U97-19 U74-11)
)
 (net POS-CONF-SEC
  (pins U76-73 U74-40 U75-15)
 )
 (net MC-CH-RST
  (pins U74-44 U75-16)
)
 (net CLKC
  (pins U76-64 U75-64)
)
 (net SBUS2
  (pins U94-6 U75-37)
 )
 (net ASSIST-NEEDE
  (pins U76-81 U75-80)
)
 (net DCD-HLT-SHUT
  (pins U94-4 U89-18)
 )
```

```
(net CLK2C
 (pins U102-1 U101-1)
 (rule (width 15))
)
(net CPU-IO#
 (pins U94-1 U89-12)
)
(net DCD-CO-PROC#
 (pins U94-7 U89-17)
)
(net LCL-MC-DCD#
 (pins U97-9 U94-12)
)
(net LCL-SMP-DCD#
 (pins U97-7 U94-19)
)
(net LCL-MEM-DCD#
 (pins U97-6 U94-15)
)
(net DEL-LCL-MC-W
 (pins U42-11 U115-17)
 (circuit (shield on (use_net GND)))
)
(net LCL-SREG-DCD
 (pins U94-11 U75-59)
)
(net MC-SREG-DCD1
 (pins U76-24 U74-29 U37-16)
 (net_number 691)
)
(net MC-SREG-DCD#
 (pins U37-17 U75-23)
)
(net $20N98
 (pins U71-49 U71-48 U71-47)
)
(net TEMP154
 (pins U71-70 U71-66 U71-61 U71-59)
)
```

```
  (net MEM-ADS#
   (pins U71-22 U71-10)
  )
  (net SPEC/NORM#
   (pins U89-16 U76-78)
  )
#
# The following nets have fast circuit (length, pair,
# parallel_segment, parallel_noise) rules.
#
  (net CTRLA-UPGD-P
   (pins U76-75 U74-8 U75-83)
   (circuit (length -1 5000))
  )
  (net DEL-MC-TO-ME
   (pins U94-14 U101-19 U100-19)
   (circuit (length -1 5000))
  )
  (net XDATA-0 (pins U101-7 U71-23))
  (net XDATA-3 (pins U71-18 U101-5))

  (pair (nets MEM-S1# MEM-S0#))
  (pair (nets CPU-M/IO# CPU-D/C#))
  (class C1 SD6 XA13 CAS/RAS# TEMP154
          SD5 BLITZ-RDY SYS-RESET
   (rule (width 5)(reorder daisy))
  )
  (class C2 LCL-MC-DCD# LCL-SMP-DCD#
          LCL-MEM-DCD#
   (layer_rule L2 (rule (width 15)))
   (layer_rule L5 (rule (width 15)))
   (circuit (use_layer L2 L5))
  )
  (class C3 LCL-MCB# MC-CMDA# LCL-REFRESH#)
  (class C4 SBUS1 BUS-REQ# BUS-GNT#
          POS-CONF-SEC)
  (class_class (classes C3 C4) (rule (parallel_segment (gap
          80)
   (limit 700)))
  )
```

```
  (class C5 MC-BD4 MC-BD6 MC-BD1 (rule (parallel_noise
          (threshold 500) (gap 50) (weight 0))))
  )
 )
```

## Prerouted Wiring Data

```
(wiring
 (resolution MIL 10)
# Net SD2
 (wire (path L1 80  74150 10370 74150 15280 74460 15280
          74460 18550 73910 18550)
    (net SD2 )
    (type protect)
    (attr fanout))
 (wire (path L6 80  73910 18550 73910 19730 66510 19730)
    (net SD2 )
    (type protect))
 (wire (path L1 80  66510 19730 65910 19730 65910 30000)
    (net SD2 )
    (type protect))
 (wire (path L1 80  65910 30000 66830 30000)
    (net SD2 )
    (type protect)
    (attr fanout))
 (wire (path L6 80  65910 30000 106280 30000)
    (net SD2 )
    (type protect))
 (wire (path L1 80  106280 30000 106110 30000 106110
          34860 107330 34860)
    (net SD2 )
    (type protect)
    (attr fanout))
 (via VIA 73910 18550 (net SD2 )
    (type protect)
    (attr fanout) )
 (via VIA 66510 19730 (net SD2 )
    (type protect) )
 (via VIA 65910 30000 (net SD2 )
    (type protect)
    (attr fanout) )
```

```
 (via VIA 106280 30000 (net SD2 )
    (type protect)
    (attr fanout) )
)
```

# Index

## A



## B

# C

# D



# E

# F

# G

# H



# I

## J

# M

# N

## O



## P

# S

# T

## U

# V

# W



# X

# Y