cadence®

Virtuoso Space-based Router User Guide

Product Version IC23.1
August 2023

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

1

Introduction

This chapter provides a general overview of Virtuoso Space-based Router and discusses the following topics:

What is Virtuoso Space-based Router?

The Virtuoso® Space-based Router enables high-speed shape-based routing, allowing gridded or gridless, and track-based routing of regular and power signals, for physical designs. By using Virtuoso Space-based Router, r, you can quickly and efficiently edit, check, and manipulate interconnects. The hierarchical connectivity extraction and shape model ensures that the edits are design-rule and connectivity correct by default. It comprises a constraint driven routing environment that supports the following features.

- Automatic routing/Signal routing
- Specialty routing

  - Bus
  - Shielding
  - Differential Pair
  - Matched Length
- Power routing
- Pin to Trunk routing

Why Use Virtuoso Space-based Router?

- Space-based Routing Technology

  The physical implementation process including routing is what can make or break your yield and manufacturing objectives. Shapes on the layout not equal to the shapes on the silicon means costly silicon re-spins. To compensate, designers rely on resolution enhancement techniques and density management techniques. They also employ overly conservative rules which result in a die area penalty.

  Virtuoso Space-based router has the capacity and flexibility to help you manage these complex issues simultaneously, greatly reducing mask re-spin costs. Its unique routing architecture is based on a patented space-based approach that meets the manufacturing, lithography, materials, and performance requirements of high-end digital and analog/mixed-signal designs.

  The router models advanced processes and design constraints, providing maximum control and exceptional results up front in the design process for high performance blocks and full chips. It also features specialty mixed-signal routing and design-for-manufacturing and design-for yield optimization.

  Virtuoso Space-based router improves the designs manufacturability and reduces time-consuming post-processing for OPC and copper planarity issues, ensuring that you meet your manufacturing and electrical objectives the first time around. You can specify advanced constraints for high-performance routing such as sophisticated wire tapering, layer control, and noise avoidance. Complex pre-routes, previously done manually, can now be performed automatically.

Licensing Requirements

You can launch the Virtuoso Space-based Router from either the Layout Suite XL or a higher tier. The Virtuoso Space-based routing features use the token-based license scheme under

the Layout Suite licenses. The following table lists the features and the Layout Suite levels at which each is available.

Feature	VLS XL	VLS EXL
DRD editing	yes	yes
Wire Assistant	yes	yes
Speciality routing (shielding, symmetry, differential pairs)	yes	yes
Automatic signal routing	NA	yes
Automatic specialty routing (shielding, symmetry, differential pairs)	NA	yes
Congestion Analysis	NA	yes

VSR further checks out a minimum of 12 GXL tokens in addition to the licenses already checked out.

Note: When the Pin to Trunk routing features of VSR are accessed from VLS EAD or VLS EXL, no additional licenses or tokens are required.

For information about licensing in the Virtuoso Studio design environment, see *Virtuoso Studio Licensing and Configuration User Guide*.

Key Benefits

- High capacity without difficulty handles flat and hierarchical data for 250K net designs
- Innovative hierarchical, 3-D, space-based architecture enables accurate modeling, manipulation, and checking of sophisticated geometries and constraints for advanced node designs
- Preserves the art of precision handcrafting while offering the benefits of automation
- Offers an intuitive and simple-to-use interactive and automatic interconnect environment

Key Features

Virtuoso Space-based Router provides the following features for routing your placed design:

- Gridded or Gridless

  - In gridless mode, the edges of all shapes must be on the manufacturing grid.
  - In gridded mode, the centerlines and endpoints of route segments must be on a routing grid and the origins of vias must coincide with the X and Y of a routing grid.

- WSPs, Tracks, and Snap Patterns

  Supports Patterns (WSP) to create tracks in the layout. WSPs are an advanced form of snap pattern definitions (SPDef) that define the tracks on which shapes can be placed. It overall includes the use of orthogonal grids and local regions.

- Tapering

  Virtuoso Space-based Router will taper routing when necessary to connect pins. Tapering can be controlled independently by each constraint group.

- Tie-up and Tie-down Routing for Power Connections

Platforms/Operating Systems

- Sun™ Solaris™ (32-bit, 64-bit)
- Linux® (32-bit, 64-bit)

Interoperability

Virtuoso® Layout Suite XL and Higher Tiers Layout Editor

- Speeds physical layout of custom digital, mixed-signal and analog designs at the device, cell, and block levels
- Supports both constraint- and schematic-driven physical implementation
- Native Cadence Space-based Router technology delivers differential, bus, and shielding routing in a single, common cockpit
- Provides constraint-driven enforcement of process and design rules

Design Data

Connectivity

Virtuoso Space-based Router functionality is driven by design connectivity. There are different ways to create routable views with connectivity.

- from LEF/DEF

  For information, see Design Translation Using LEF/DEF Translator in the *Design Data Translator's Reference*.

- from a schematic-driven Virtuoso flow

  The `Connectivity Extractor` allows you to extract the top level of your design, the entire design, or extract to the level of the hierarchy you specify using the `extractStopLevel`.

  For more information see Preparing Your Design for Routing in the Virtuoso Layout Suite documentation.

- Mixed signal flow

  The analog nets are either routed manually or by using the Virtuoso Space-based Router (VSR) and the signal nets are routed by Cadence® NanoRoute® Advanced Digital Router supported by the Innovus system.

Virtuoso Space-based Router uses rectangles, polygons, paths, segments, and vias to determine connectivity. Shapes not recognized as interconnect include circles, ellipses, and shapes with non-orthogonal angles.

Hierarchy Depth Control

As part of Virtuoso Layout Suite XL and higher tiers, VSR supports hierarchical designs. You can control hierarchy depth for the following.

- Display - set *Start* and *Stop Display Levels* in the *Display Options* form. (*Options > Display...*)

  This setting controls the range of hierarchy levels displayed in the canvas window.

  Note: The setting also controls hierarchy depth for interactive and automatic routing commands including point-to-point, finish routing, and signal routing.

- Routing - set *Start* and *Stop Display Levels* in the *Display Options* form.

- Extraction - set Extract Connectivity to Level in the *Connectivity* tab of the Connectivity form. (*Options – Layout XL...*).

  This setting controls the number of hierarchy levels to check when extracting connectivity.

- Interactive rule checking - set Hierarchy Depth in the DRD Options form (*Options – DRD Edit...*).

  This settings tells the DRD rule checker the level to which to check for violations.

Note: It is important to be aware of all three depth settings to ensure intended results.

Data Size and Performance

Routing performance depends on several factors such as the number of nets and devices, whether the design is hierarchical or contains abstracted blocks. The three design types supported by Virtuoso Space-based Router are: Microprocessor, Mixed Signal, and Digital SoC (Std Cell & Memory).

Data Object Support

Wiring Objects

- Segments

  All Space-based router functionality supports segments. If you have preroutes that are paths, they remain as paths in the router. However, the router generates only segments.

- Vias

  Interactive and automatic routing support both custom and standard vias and via variants. You can have all standard vias, all custom vias, or a mixture of standard and custom vias.

Pins

Polygon pins are supported as long as the edges are orthogonal or diagonal. Polygons with diagonal edges are broken into octagons.

Blockages

Any non-routing object that does not have connectivity is seen as a blockage. The router supports rectangles and polygons as long as the edges are orthogonal or diagonal. Any angle edges are not supported.

Multipart paths (MPPs)

The Automatic routing does not route MPPs or change them during routing. However, the autorouter can route to MPPs. You can route to MPPs where only one layer is a routing layer, for example a guardring. You cannot route correctly to MPPs containing multiple routing layers because the autorouter does not trace connectivity between layers through via shapes in MPPs.

2

Technology Requirements

This section discusses the following:

- Technology File Requirements
- Constraint Groups
- Supported Constraints and Parameters
  - Supported Constraints and Parameters (Virtuoso Advanced Node for Layout Standard)
  - Supported Constraints and Parameters (Virtuoso Advanced Node for Layout Only)

Technology File Requirements

This section specifies minimum requirements for the technology file.

Specifying Layer Order

The `layerRules` section of the technology file specifies attributes for user-defined layers. You must specify the mask number for each interconnect layer and the preferred routing direction for each routing layer.

The mask number assigned to each layer in the `functions` section of the technology file is the layer sequence number, which helps determine if layers are adjacent to each other.

The mask number is used to:

- Order the layers in the *Select Via* form.
- Determine the next layer in sequence when using the *Via Up* or *Via Down* command.
- Determine automatically the “next” or “previous” layer when the *Create Wire* command is run with a non-routing layer as the current layer.

If mask numbers are not specified, layer-purpose priority is used. However, this is not as reliable as defining the mask numbers because mask numbers are independent of purposes.

Note: The router may not work if the mask numbers are not specified correctly.

Valid Values: `cut`, `li`, `metal`, `ndiff`, `pdiff`, `nplus`, `pplus`, `nwell`, `pwell`, `poly`, `diff`, `recognition`, `other`, `unknown`

Note that the following example is specific to space-based routing. The `layerRules` section in your techfile can have more layers and layer definitions.

```
layerRules(
    functions(
        ;( layer function [maskNumber])
        ;( ----- -------- ------------)
         ( POLY   "poly"     0 )
         ( CO     "cut"      1 )
         ( Metal1 "metal"    2 )
         ( VIA    "cut"      3 )
         ( Metal2 "metal"    4 )
    ...
```

Note:

- Include only one poly layer in the technology file because Virtuoso Space-based Router can route only one poly layer at a time.
- Define multiple cuts between each pair of metal layers. However, you can use only a single cut layer between the same pair of metal layers.
- If the functions section is not defined or is defined without the mask numbers, vias are still available when the *Create – Wire* command is run. However, the following warning message is displayed.

  ```
  \w *WARNING* geViaSet : Incomplete layer maskNumber, the order of the vias may
  not be correct.
  ```

For more information about functions, see Technology File Layer Attributes in *Virtuoso Technology Data ASCII Files Reference*.

Specifying Routing Layer Directions

Set a preferred routing direction for each routing layer.

```
layerRules(
    routingDirections(
    ;( layer direction )
    ;( ----- --------- )
        ( POLY       "none" )
        ( Metal1     "horizontal" )
        ( Metal2     "vertical" )
```

```
    ;routingDirections
) ;layerRules
```

For more information, see Technology File Layer Attributes in *Virtuoso Technology Data ASCII Files Reference*.

Specifying Via Definitions

At a minimum, define all standard vias for interconnect using routing layers in the `viaDefs` section of the technology file. Typically, this is a set of standard vias between the poly layer and the highest metal layer.

For example.

```
viaDefs(
standardViaDefs(
;(
viaDefName  layer1  layer2  (cutLayer  cutWidth  cutHeight  [resistancePerCut])
; (cutRows  cutCol  (cutSpace))
; (layer1Enc)   (layer2Enc)   (layer1Offset)   (layer2Offset)   (origOffset)
; [implant1     (implant1Enc) [implant2        (implant2Enc)  [well/substrate]]])
;( ------------------------------------------------------------------------ )
  ( mpoly    Metal1  POLY    ("CO"      0.09      0.09)
    (1        1      (0.11 0.11))
    (0.04 0.0)    (0.04 0.01)   (0.0 0.0)      (0.0 0.0)        (0.0 0.0)
  )
  ( m1m2     Metal1  Metal2  ("VIA"     0.1       0.1)
    (1        1      (0.1 0.1))
    (0.04 0.0)    (0.04 0.0)    (0.0 0.0)      (0.0 0.0)        (0.0 0.0)
  )
...
```

By default, standard vias are automatically created before assisted routing. To disable the creation of standard vias and use only the existing vias in the `validRoutingVias` constraint, specify the following:

```
rdeSetVar "db.enable_create_derived_vias" "false"
```

In addition, define the custom vias required for routing.

```
customViaDefs(
;(
  viaDefName      libName   cellName  viewName  layer1  layer2  resistancePerCut)
;(------------    --------- --------- --------- ------- ------  --------------)
 (VIAm1m2_2CUT_N  tsmc65lp  VIAm1m2   via       Metal1  Metal2  0.95)
 (VIAm1m2_2CUT_S  tsmc65lp  VIAm1m2   via       Metal1  Metal2  0.95)
 (VIAm1m2_2CUT_E  tsmc65lp  VIAm1m2   via       Metal1  Metal2  0.95)
 (VIAm1m2_2CUT_W  tsmc65lp  VIAm1m2   via       Metal1  Metal2  0.95)
...
```

Note: Via variants used by other tools are currently not recognized by Virtuoso.

For more information, see *Technology File Via Definitions and Via Specifications* in *Virtuoso Technology Data ASCII Files Reference*.

Specifying Valid Layers and Valid Vias

Define a list of valid layers and vias in the `virtuosoDefaultSetup` constraint group. Use the `foundry` constraint group to define only process rules. Do not use the `foundry` constraint group to define valid layers and valid vias.

```
;( group [override] )
;( ----- ---------- )
 ( "virtuosoDefaultSetup" nil
    interconnect(
    ( validLayers     (POLY    Metal1    Metal2 ...) )
    ( validVias       (mpoly   m1m2    ... ) )
    ) ;interconnect
 ) ;virtuosoDefaultSetup
```

Note: If no `validVias` statement exists, a full set of standard via is defined for the session based on the layers defined in the `validLayers` statement.

The automatic router uses vias that have both layers defined as valid routing layers. If a via has a metal layer that is not a valid routing layer, the router can still use the via for pin access provided it is in the valid via list. In particular, vias for which the bottom layer is oxide, well, or implant are not used during automatic routing. Virtuoso Space-based Router displays a warning message when initializing a design if the constraint group contains such vias.

Note: Pcell vias are not supported by Virtuoso Space-based Router.

For more information, see `validLayers` and `validVias` in the *Routing Constraints* chapter of the Virtuoso Technology Data Constraints Reference.

Limiting Routing on a layer

The current method for restricting the amount of wiring on any given layer is to apply a *taper* constraint group. The intent is to enable the router to connect to the pins on the restricted layer, but limit the amount of routing on the layer using a taper constraint group.

To restrict the amount of routing, define a taper constraint group similar to the one below. These examples illustrate how to restrict routing on the Poly layer, but this technique can be applied to any layer(s).

Note: The validLayers include only Poly and Metal1. This allows the router to connect to Poly and traverse to Metal1, but limits the amount of Poly routing to the `taperHalo` value. Routing on the restricted layer will occur only within the halo of the pins.

Example:

```
constraintGroups(
;( group [override] )
;( ----- ---------- )
( "virtuosoDefaultTaper" nil "taper"
  interconnect(
    ( validLayers (Poly Metal1 ) )
    ( validVias (M1_PO ) )
  ) ;interconnect
  spacings(
    ( taperHalo 1.4 )
  ) ;spacings
 ) ;virtuosoDefaultTaper
);constraintGroups
```

Note: All other applicable wiring constraint groups should *exclude* the layer(s) in the `validLayers` statement. Otherwise routing may occur beyond the taper value limit.

Example:

```
;( group [override] )
;( ----- ---------- )
 ( "virtuosoDefaultSetup" nil
   interconnect(
     ( validLayers (Metal1 Metal2 ...) )
     ( validVias (M1_Poly m1m2 ... ) )
   ) ;interconnect
 ) ;virtuosoDefaultSetup
```

Via Usage Model

For detail routing, Virtuoso Space-based Router considers a limited number of vias and via variations for each layer pair. The standard via variations are chosen based on the constraints and the type of route (for example, taper).

The detail router chooses a limited number of custom vias and derived standard via variations based on the amount of routing resources required by each. The vias and the via variations chosen are those that require the fewest resources.

Specifying Track Patterns and Patterns

You can snap wires to track patterns and snap patterns. A track is a system reserved purpose and can be combined with any physical layer to create a layer-purpose pair. The tracks that make up the track pattern provide guidelines for laying interconnect routes.

For more information on how to create tracks and patterns, see Creating Track Patterns and Using Patterns.

Supporting Colored Trim Metal Layers

Virtuoso Space-based router supports colored trim metal layers with the correct color assignment when creating trim shapes on a routing layer. A colored trim layer can have color information specified using the color attribute, which lets you add more than one color mask to a single trim layer.

For more information about how colored trim layers are defined in the technology file, see `functions`.

Note: If only locked trim shapes cut the routing layer, then the created trim shape should be color locked.

For more information, seeInserting Colored TrimMetal Layers in Automatic Routing.

Using Layer-Purpose Pairs

In the Virtuoso Studio design environment, you can define technology rules for layer-purpose pairs (LPP). However, OpenAccess rules are applicable only to layers, and not layer-purpose pairs. In the Virtuoso Studio design environment, any technology rule that is defined for a layer-purpose pair is stored as a private extension and is available only to Virtuoso applications.

Layers based on purposes can be selected from the Palette, provided the layer-purpose pair is valid and visible. However, when tapping an existing shape, the layer-purpose pair is not used directly. You can create a prioritized list of purposes that allow you to use a layer based on a purpose. See Tapping Wires.

Poly Pins over an Implant Layer

If `minSpacing` is defined between poly and implant layers, then any poly pin shape that is fully overlapped by an implant layer is viewed as blocked and is not accessible to the automatic router, the Point-to-Point router, and the Finish Wire command and results in routing failures.

Support For 45 nm Rules

In order to use the 45 nm technology rules, these rules must coded in the technology file `techLayerProperties` section. The following example shows LEF57_AREA, LEF57_ENCLOSUREEDGE, and LEF57_SPACING for second EOL rule support.

```
techLayerProperties(
```

```
 ;( PropName               Layer1 [ Layer2 ]            PropValue )
 ;( --------               ------ ----------            --------- )
( LEF57_AREA   Metal1  "AREA 0.07 EXCEPTEDGELENGTH 0.41 EXCEPTMINSIZE 0.41 0.14 ;"
)
( LEF57_ENCLOSUREEDGE  Via2   "ENCLOSUREEDGE 0.08 WIDTH 0.26 PARALLEL 0.5 WITHIN
0.4 ;" )
( LEF57_SPACING  metal1   "SPACING 0.12 ENDOFLINE 0.10 WITHIN 0.035 MINLENGTH 0.07
PARALLELEDGE 0.12 WITHIN 0.10 ENCLOSECUT BELOW 0.05 CUTSPACING 0.15 ;")

) ;techLayerProperties
```

Constraint Groups

A constraint group contains a set of constraints or rules to be applied to a design under different circumstances, which allows the flexibility to experiment with less or more stringent process rules at different stages of the design process. Constraint groups are classified as follows.

1. Foundry constraint group

   Example: `foundry`

2. Application-specific constraint group(s)

   Examples: `virtuosoDefaultExtractorSetup, virtuosoDefaultSetup, LEFDefaultRouteSpec`

3. User-defined constraint group(s)

   Examples are unlimited: shieldTheseNets, wideWires

`Foundry`, Application-specific and User-defined constraint groups may be created automatically during a translation process such as lef2oa, and are based on the information in the LEF/DEF file. Other constraint groups may be created by your CAD group, and still others created by the individual layout designer. Some can be stored in a technology database and/or on the design itself.

Constraint groups are stored in the technology or design database and can be created by your CAD group or by the layout designer. Constraint groups can also be created automatically during the translation process, such as `lef2oa`.

In Virtuoso XL and higher tiers, you typically have the following three constraint groups as a required minimum: foundry, virtuosoDefaultExtractorSetup, and virtuosoDefaultSetup.

Foundry Constraint Group

The `foundry` constraint group represents the absolute minimum rules (i.e. `minWidth/minSpacing`, etc.,) which must be adhered to in order to manufacture the design. These are commonly referred to as the base process rules and are stored only in the technology database.

Define the complete set of process rules in the `foundry` constraint group. Do not depend on a technology file translated from elsewhere to contain all needed rules. For example, a 65nm tech file translated from CDB will not include all required rules because not all 65nm rules are code able in CDB.

Creation and usage

- Creating the Foundry Constraint Group

  The `foundry` constraint group is created and populated with information during the lef2oa translation. The translator get the information from the section within the LEF file.

  Alternatively, you can create the `foundry` constraint group by loading an ASCII technology file containing the foundry constraint group information through the Technology File Manager Toolbox, which you can invoke from the CIW.

- Information in the foundry constraint group

  The `foundry` constraint group contains all of the information necessary to manufacture the design. Everything from `minSpacing/minWidth` for layers, `minEnclosure`, `minStep`, to via rules, to electrical constraints such as antenna rules.

- Using the foundry constraint group in Virtuoso

  The `foundry` constraint group is the final stop along the constraint precedence lookup trail. If a constraint override is not defined elsewhere, the tools use the value(s) in the `foundry` constraint group.

  Most all functionality uses the `foundry` constraint group at some point. As the last stop in constraint precedence lookup, the `foundry` constraint group is the final place and many times the only place the value(s) for a constraint are defined.

  The `foundry` constraint group is not typically writable for most users, but can be viewed in the Process Rule Editor. You can launch the Process Rule Editor by clicking the RMB in the Constraint Manager assistant or by clicking the PRE icon from the toolbar in the Constraint Manger assistant. Within this dialog you can set the scope to Technology and view the foundry constraint group.

Application Constraint Groups

An application constraint group is used to target a basic set of rules within a given application. Although you can define numerous application specific constraint groups and switch between them at any time, you can reference *only* one as the default by setting the *Wire* cyclic field in the *Layout Editor Options* form (*Options – Editor*...) at any given time. It is important to note, however, that at any given time during routing or wire editing, the rules to apply to specific objects may be derived from a number of different constraint groups, based on specific constraint group lookup precedence.

Application-specific constraint groups can be stored within the technology database or directly on the design itself (i.e. lib/cell/cellview).

Examples: virtuosoDefaultExtractorSetup, virtuosoDefaultSetup, LEFDefaultRouteSpec

Specifying the Default Constraint Group for Routing

You can define numerous application-specific constraint groups and switch between them, but you can reference only one constraint group at any given time.

Note: You must set the required constraint group at the beginning of a session and use the Process Rule Editor to vary the constraints applied on different nets while working on the design. If you update the constraint group while working on a design, delete all routing that would be affected by the changes made to the constraint group and re-route.

virtuosoDefaultExtractorSetup

The `virtuosoDefaultExtractorSetup` constraint group is used by default by the extractor for connectivity extraction. This constraint group must contain a list of valid interconnect layers and can optionally include a list of valid vias. The layers can be physical, such as metal and poly, or derived, such as those used to define stop layers and bulk layers. In the technology file, physical layers are defined in the `layerRules` section and derived layer are defined in the `techDerivedLayers` section.

The `setupConstraintGroup` environment variable in your `.cdsenv` file is used to specify `virtuosoDefaultExtractorSetup` as the default constraint group for connectivity extraction. If required, you can change this default value by specifying the name of another constraint group. If the `setupConstraintGroup` environment variable is not specified or is set to an empty string, the extractor prompts you for a constraint group to use for extraction.

The `virtuosoDefaultExtractorSetup` constraint group can be stored in the technology database or in the design database along with the library/cell/view information.

Note: Though it is possible to specify `virtuosoDefaultExtractorSetup` as the default constraint group for routing, it is not recommended for use for routing. This is because the constraint group is designated for connectivity extraction and the layers and vias that it defines as valid may not be suitable for routing.

virtuosoDefaultSetup

The virtuosoDefaultSetup constraint group is by default used for interactive and automatic routing. For example, if a signal selected for routing does not have a constraint group assigned to it, the constraints in the virtuosoDefaultSetup constraint group apply. Moreover, the set of valid layers and vias defined in the virtuosoDefaultSetup constraint group is used exclusively for wiring and is usually a subset of that defined in the virtuosoDefaultExtractorSetup constraint group. You can also specify in this constraint group any process rules that should override those defined in the foundry constraint group.

The contents of validLayers in virtuosoDefaultSetup are usually a subset of those defined in the virtuosoDefaultExtractorSetup constraint group because, while you may want to extract connectivity for non-metal layers, you do not want interactive and automatic routing to use the non-metal layers.

The `wireConstraintGroup` environment variable in your `.cdsenv` file is used to specify `virtuosoDefaultSetup` as the default constraint group for interactive and automatic routing. If the `wireConstraintGroup` environment variable is not set, interactive and automatic routing use the value of the `setupConstraintGroup` environment variable as the default constraint group. Because the constraint group specified with `setupConstraintGroup` is for extraction, you do not want it to be used for routing. Therefore, ensure that you set a default constraint group for routing.

You can also specify `virtuosoDefaultSetup` as the default constraint group for interactive and automatic routing by using the *Default Wire Constraint Group* list in the Layout Editor Options form.

LEFDefaultRouteSpec

The `LEFDefaultRouteSpec` constraint group contains rules typically used in digital standard cell applications or custom and mixed signal applications that require only metal layers.

The `LEFDefaultRouteSpec` constraint group is specific to a LEF/DEF design flow and is created during lef2oa translation. As a result, the valid layers, the valid via definitions, and the valid rules become part of the `LEFDefaultRouteSpec` constraint group.

Note: The `lef2oa` translator translates any `NONDEFAULTRULES` in LEF as constraint groups in the OpenAccess technology database.

The `LEFDefaultRouteSpec` constraint group contains valid layer and via definitions. The via definitions typically include standard vias and several custom vias used by other routers such as NanoRoute in the Innovus environment. The `LEFDefaultRouteSpec` constraint group also contains process rules for wide metal shapes.

You can specify `LEFDefaultRouteSpec` as the default constraint group for interactive or automatic routing by selecting it from the *Default Wire Constraint Group* list in the *Wire Editing* section of the *Layout Editor Options* form.

virtuosoDefaultTaper

If routing requires tapering, for best results define a `virtuosoDefaultTaper` constraint group with type `taper`. Explicitly defining a taper constraint group ensures that the correct layers and widths are used for tapered wires.

The primary purpose of a constraint group of this type is to taper the width of a wire to match the width of the pin to which it connects. You can also adjust the widths of taper routing. Many different constraints can be included in the taper constraint group.

You can also use a constraint group of type `taper` to restrict routing on a layer to control the taper length. For more information, see Limiting routing on a layer.

Note: After you customize a constraint group of type `taper`, such as to restrict routing on poly, you cannot use the constraint group to also taper the widths of wires on other layers.

The `wireTaperConstraintGroup` environment variable in your `.cdsenv` file is used to specify `virtuosoDefaultTaper` as the default constraint group for tapering wires. The `virtuosoDefaultTaper` constraint group is only used when the router cannot connect to a target pin by using the regular net, design, or default constraints.

How the Router Uses the Taper Constraint Group

When you invoke the design, there are three constraint groups the application searches for by default. These constraint groups, whether they are defined in your technology library or not, are specified in the default `.cdsenv` with the following environment variables.

- layout setupConstraintGroup string `"virtuosoDefaultExtractorSetup"`
- layout wireConstraintGroup string `"virtuosoDefaultSetup"`
- layout wireTaperConstraintGroup string `"virtuosoDefaultTaper"`

It is important to note, however, that at any given time during routing or wire editing, the rules to apply to specific objects may be derived from a number of different constraint groups, based on specific constraint group lookup precedence.

First, the router attempts to route a net using constraints from the net constraint group, design constraint group, and application default constraint group. The precedence order is

a. net constraint group

b. application default constraint group

c. design constraint group

d. foundry constraint group

If the router cannot connect to a pin for some reason, it attempts to connect using constraints from the input/output taper constraint group, application default taper constraint group. The precedence order is

a. input/output taper constraint groups

b. application taper constraint group

Defining the Taper Constraint Group

When routing requires tapering, for best results define a `virtuosoDefaultTaper` constraint group. Explicitly defining a taper constraint group ensures that the correct layers and widths are being used for the tapered wires. You can control the global taper routing mode using the environment variable setTaperMode.

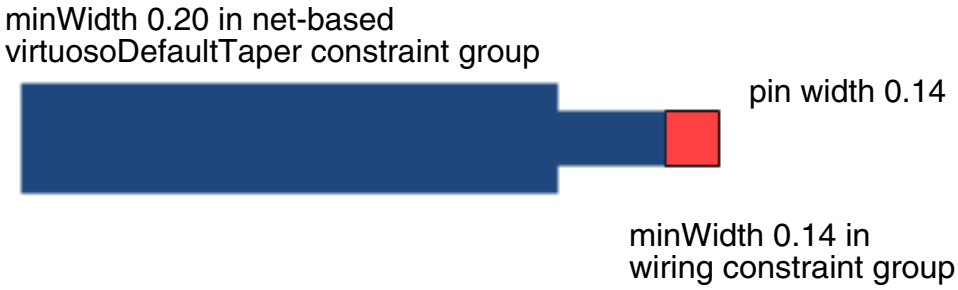
Also, in order to load the taper constraint, the constraint must be specifically marked with `taper` tag. The following is an example of a taper constraint group.

```
constraintGroups(
;( group [override] )
;( ----- ---------- )
( "virtuosoDefaultTaper" nil "taper"
interconnect(
( validLayers (Poly Metal1 ) )
( validVias (M1_PO ) )
) ;interconnect
spacings(
( taperHalo 1.4 )
) ;spacings
) ;virtuosoDefaultTaper
;constraintGroups
```

You can then use the Technology File Manager to load the constraint group.

Note: If you include all the layers in the `validLayers` constraint of the taper constraint group, but some of your nets require a more restricted layer set, use the input and output taper constraint groups to reduce the `validLayers` list for those specific nets. You can use the Process Rule Editor to assign the `validLayers` constraint to the input/output taper constraint groups.

Note: If you have a taper constraint group attached to the net and the `minWidth` defined in that constraint group is greater than the width of the pins, the router may taper down using the `minWidth` defined in the wire constraint group when connecting to those pins. This optimizes routing for pin connections. The router does not consider the behavior as a violation of the net based constraint group. See the illustration below.



User-Defined Constraint Groups

Non-default constraint groups defined in the technology LEF will be translated as user-defined constraint groups and stored in the technology database.

A common example of a user-defined constraint group would be for wide wire rules for a particular set of signals. User-defined constraint groups can be used whenever particular routing considerations need to be applied to signal(s).

Note: If you attach a constraint group to a net and the `minWidth` defined in that constraint group is greater than the width of the pins, the router may taper down using the `minWidth` defined in the wire constraint group when connecting to those pins. This optimizes routing for pin connections. The router does not consider the behavior as a violation of the net based constraint group.

customRoutingConstraintGroup Use Model

The `customRouting` limits the use of constraints that can be used. This limitation is based on the override limitation. The Virtuoso Space-based router can override only the following constraints: `minWidth`, `minSpacing (One Layer)`, `minNumCuts`, `validLayers`, `validVias`, `minSpacing (Two Layers)`, and `minOppExtension`.

Note: Virtuoso Space-based router does not recognize derived layers. For more information, see Derived Layer Support in the Virtuoso Design Rule Driven Editing User Guide.

Storing Constraint Groups

The foundry constraint group is stored in the OpenAccess technology database. Application-specific and user-defined constraint groups are also stored in the OpenAccess technology database. Constraint groups stored in the technology database apply to all designs associated with the design specific constraints.

Design-specific constraints that apply to specific objects in a design, instead of the entire design, are stored in the OpenAccess design database.

You can also store any user-defined constraint groups within the technology database, provided you have write permission to the technology database. This has the advantage of being a centrally accessible location and provides added control over the definitions in the constraint groups.

Note: Constraint groups stored in a technology database apply to any design derived from it.

Constraint Group Lookup Precedence

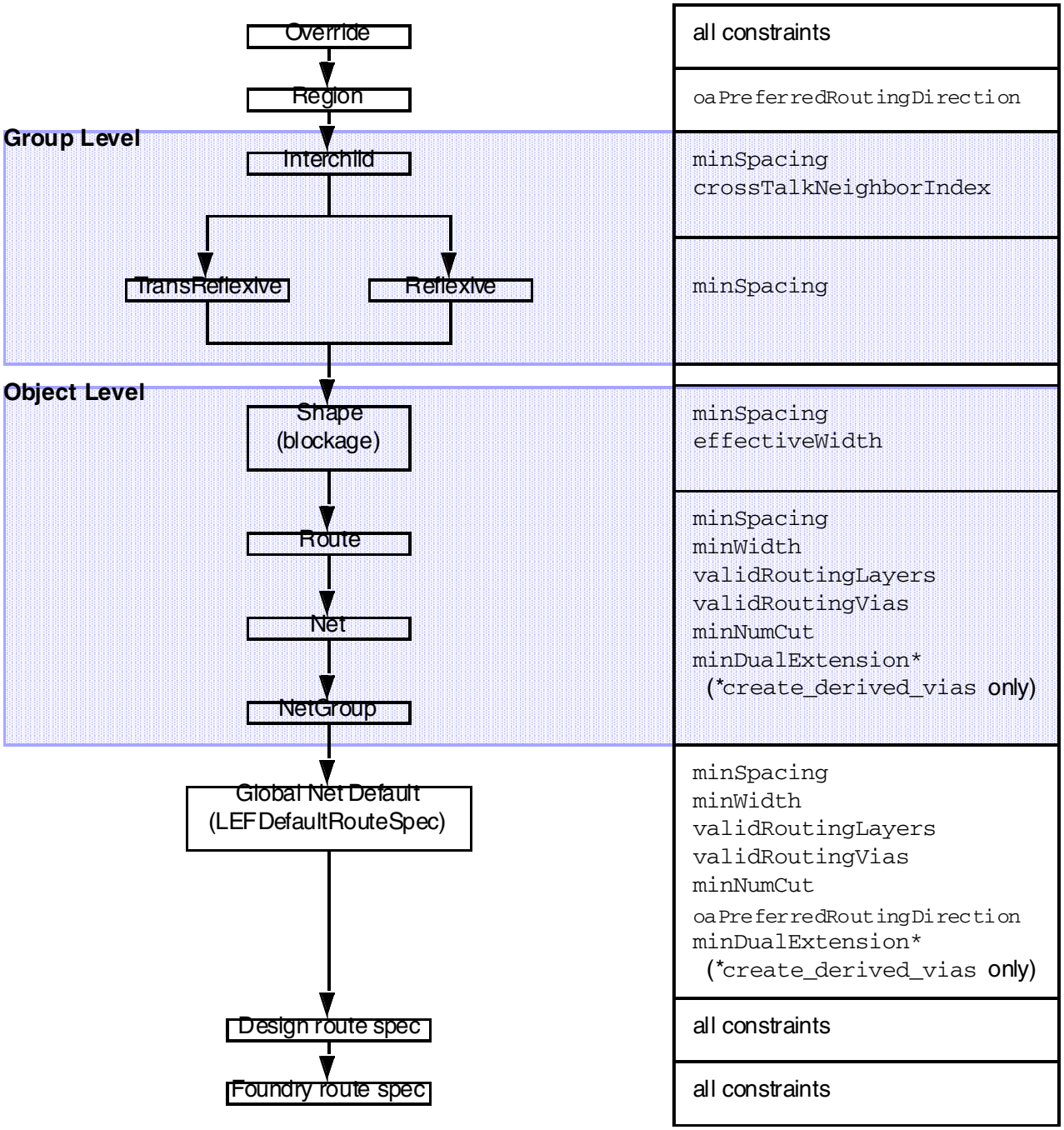
In Virtuoso, the `foundry` constraint group provides process rules for all objects. However, you can override the rules in the `foundry` constraint group for specific objects or for the design.

A technology database can contain multiple constraint groups that define the same constraints. When this is the case, the constraint groups are applied according to the following precedence:

1. The constraint group, being referenced by a command that is currently active in the design window.
2. A setup or a constraint group recognized as the constraint group of first precedence by Virtuoso.
3. The `foundry` constraint group.

Virtuoso searches for the constraint groups it recognizes in the order of precedence. It applies first the hard constraint it finds. If a hard constraint is not found, it applies the first soft constraint that it finds.

The following diagram shows the simplified constraint search hierarchy.



The following table shows constraint group lookup precedence in the Virtuoso Space-based Router. Precedence starts with constraints attached to objects and ends with constraints in the Foundry constraint group.

Entities	Examples
`Net`	
`Net group`	reflexive, transreflexive, default
`Application-specific constraint group`	`virtuosoDefaultSetup` constraint group
`Design`	constraints specific to this design
`Foundry`	process technology rules

For more information, see *Getting Started with Virtuoso Unified Custom Constraints* and *Virtuoso Technology Data Constraints Reference*.

Important

When routing a design, it is important to maintain the constraint group definition used for specific nets throughout the routing process. Changing constraint groups on a net can result in unpredictable layers and constraints being applied during routing.

Scalar Versus Table Spacing Rule Precedence

When both scalar and table spacing rules exist in the technology database, the rules are applied as follows.

- When both are present in a single constraint group, the first spacing rule found is the one applied. The second one is ignored. This precedence holds true regardless of whether the value is a scalar spacing rule or a table spacing rule.
- When each exists alone in its own constraint group, the scalar value is used as a default regardless of the precedence order. The table values are also used, but only for the larger widths, not the default.

Supported Constraints and Parameters

The following table shows the constraints supported by the Virtuoso Space-based Router and Wire Editing commands.

Note: For information about constraints supported by DRD, see "Supported Constraints" in the *Virtuoso Design Rule Driven Editing* User Guide. DRD includes options to check for errors during interactive editing or to turn off checking during editing and check for errors after editing the layout. To know about the potential rules violations during interactive editing, DRD needs to be enabled.

Technology Process Rule Constraints	Supported by Automatic Router
allowedWidthRanges	yes
cutClasses	yes
horizontalOffset	yes
horizontalPitch	yes
largeRectViaArrayAllowed	yes
maxDiffDensity	yes
maxFloatingArea	no
maxLength	no
maxNumMinEdges	yes
maxWidth	no
minArea	yes
minAreaEdgeLength	yes
minCutClassSpacing(Two layers)	yes
minDiagonalSpacing	no
minDiagonalWidth	no
minEdgeAdjacentDistance	yes
minEdgeAdjacentLength	yes
minEndOfLinePerpSpacing	yes
minEndOfLineSpacing	yes
minEndOfLineExtensionSpacing	yes

Technology Process Rule Constraints	Supported by Automatic Router
minExtensionEdge	yes
minFillToFillSpacing	yes
minHoleArea	yes
minLargeViaArrayCutSpacing	yes
minLargeViaArraySpacing	yes
minLargeViaArrayWidth	yes
minLength	no
minNeighborViaSpacing	yes
minNumCut	yes
minOppExtension	yes
minOppSpanSpacing	yes
minOutsideCornerEdgeLength	yes
minParallelSpanSpacing	yes
minParallelViaSpacing	yes
minParallelWithinViaSpacing	yes
minProtrusionNumCut	yes
minRectArea	yes
minSameMetalSharedEdgeViaSpacing	no
minSameNetSpacing	yes
minSpacing	yes
minWidth	yes
minViaExtension	yes
minViaSpacing	yes
minVoltageSpacing(two layer)	yes
redundantViaSetback	yes
stackable	yes
taperHalo	yes

Technology Process Rule Constraints	Supported by Automatic Router
verticalOffset	yes
verticalPitch	yes
viaSpacing	yes
viaStackingLimits	yes
allowedLengthRanges	no
minSideExtension	yes
minExtensionOnLongSide	yes
minWireOverlap	yes
rectShapeDir	yes
forbiddenCutClassSpacingRange	no
minQuadrupleExtension	yes
minEndOfLineEdgeExtension	yes

Constraint Manager Constraints	Supported by Automatic Router
symmetry	yes
diffPair	yes
shielding	yes
net priority	yes
process rule overrides	yes

Supported Constraints and Parameters (Virtuoso Advanced Node for Layout Standard)

The following table lists the constraints that are supported by Virtuoso Space-based Router and Wire Editor.

Technology Process Rule Constraints	Supported by Automatic Router
allowedLengthRanges	no

Technology Process Rule Constraints	Supported by Automatic Router
minSideExtension	yes
minExtensionOnLongSide	yes
minWireOverlap	yes
minDirectionalOverlap	yes

Note: The constraints listed in the above table and their corresponding ASCII techfile syntax are fully supported by OpenAccess. However, only the *Create Wire* command in Wire Editor is fully compliant with these Local Interconnect (Li) constraints.
Also, the constraints mentioned in the table are supported by Stretch and Reshape commands.

Supported Constraints and Parameters (Virtuoso Advanced Node for Layout Only)

In ICADVM20.1, the Virtuoso Spaced-based Routing (VSR) and Wire editing has been enhanced to provide support for new rules. The Virtuoso Space-based Router has been enhanced to support fully interactive and automatic multi-patterned routing with a unique approach to avoid complex coloring conflicts. This approach simplifies layout creation and minimizes coloring errors that can be prevalent when designing at 10nm. The Virtuoso Space-based Router has added support for pre and post-optimization of routes to account for the complex EM rules at 10nm and below.

Important

Requires Virtuoso Advanced Node for Layout (95511) license.

The following table lists the constraints that are supported by Virtuoso Space-based Router and Wire Editor.

Technology Process Rule Constraints	Supported by Automatic Router
allowedWidthRanges Parameters: measureHorizontal, measureVertical	yes

Technology Process Rule Constraints	Supported by Automatic Router
minViaSpacing	yes
Parameters: centerToCenter, exactLength, manhattan, sameNet, sameMetal, exceptExactAligned, horizontal, vertical	
minSpacing	yes
Parameter: any	
orthogonalSnappingLayer	yes
Note: This constraint is supported for the Stretch command as well.	
minEndOfLineSpacing	yes
Parameters: width, distance, negativePRL, maxLength | minLength | twoSides, sameMask | diffMask, endToEndSpace | otherEndWidth | extension.	
Note: Skip otherEndWidth parameter when endToEndSpace does not apply.	
minParallelSpanSpacing	yes
minEndOfLineExtensionSpacing	yes
orthogonalWSPGrid	yes
Parameters: 'mask1 | 'mask2 | 'mask3, 'upperLineEndWSSPDefName	
Note: The orthogonalWSPGrid constraint is used in the interoperability flow with Innovus. It is used to initiate the trim creation at specific locations.	
viaSpacing	yes
Parameters: edgeExtension, layer, extensionDirection	
viaKeepoutZone	yes
forbiddenEdgePitchRange	yes

3

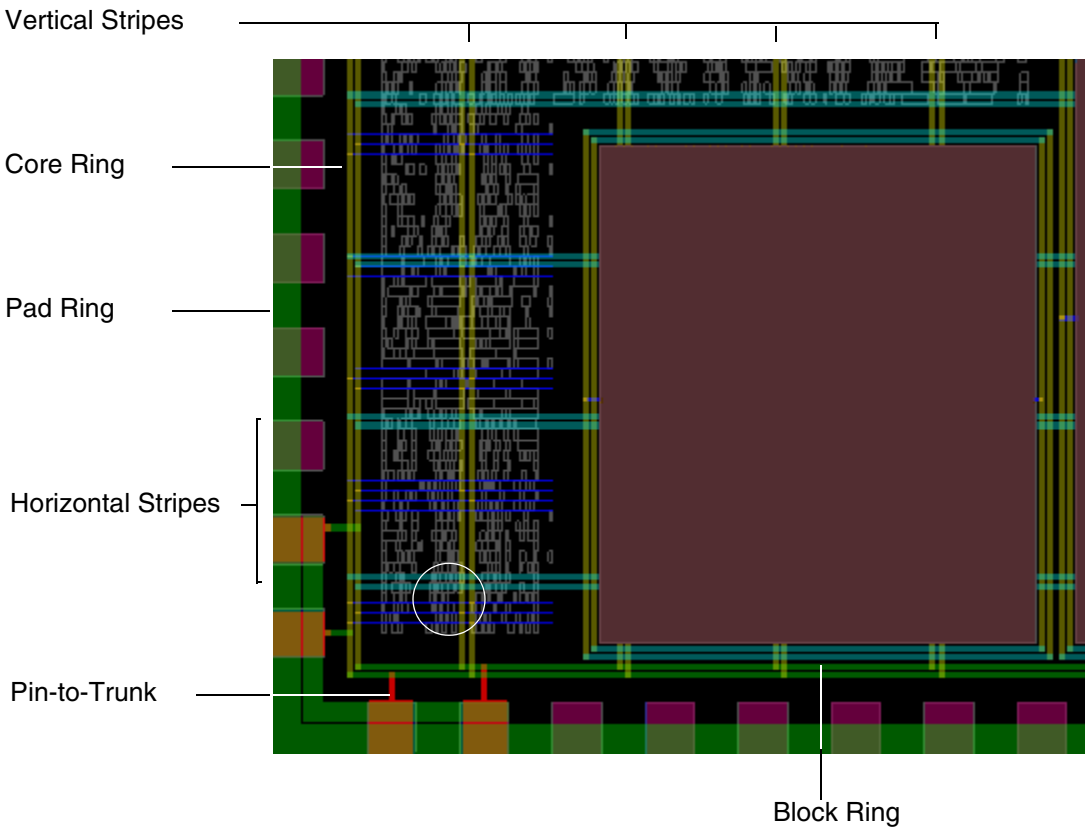
Power Routing

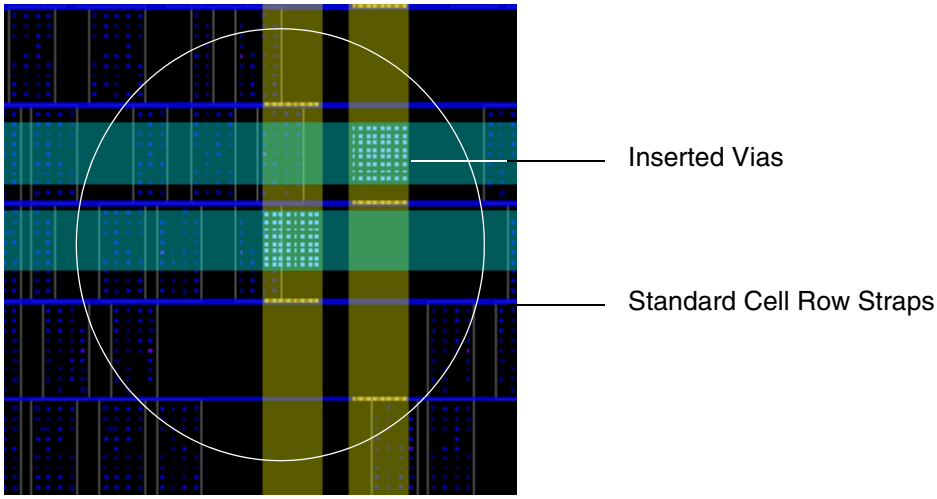
This chapter describes the features and functionality of the Virtuoso Space-based Router power router.

The following sections are included:

Power Router Features

Virtuoso Space-based Router power routing constructs the common power components shown in the following figure:





Requirements

To use the power router, you must load a design that includes technology information and pin-to-net assignments.

Power routing functions are dependent on signal type being set to power or ground. Use the Property Editor Assistant to set the signal type.

Recommended Power Router Flow

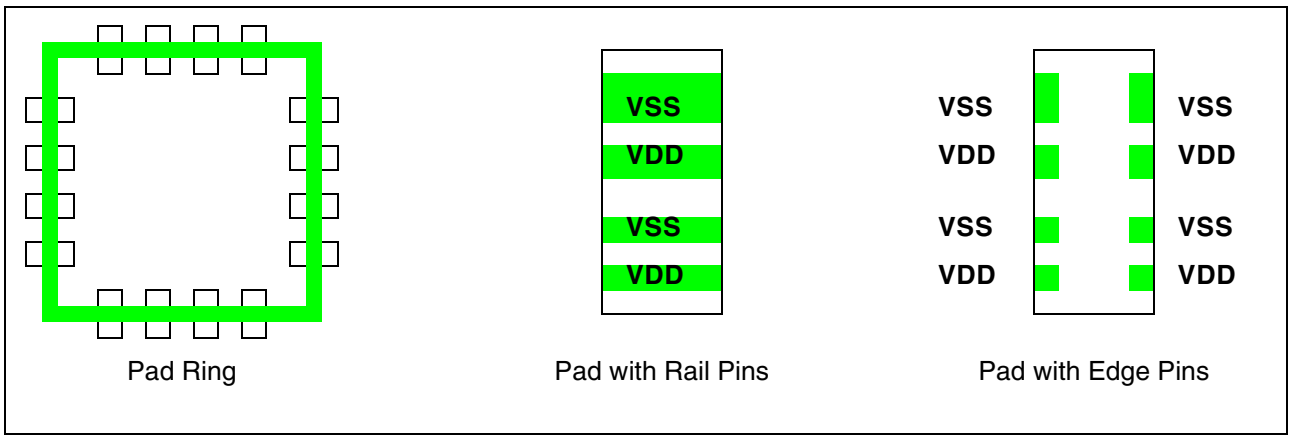
For a simple power supply network, you perform the following tasks in sequence.

- Adding a Pad Ring
- Adding Core Rings
- Adding Block Rings
- Connecting Pin-to-Trunk
- Adding Stripes
- Adding Standard Cell Row Straps
- Trimming Stripes
- Inserting Vias
- Tie Shielding

The rings and stripes provide the foundation for hooking up each cell to a power source. The order of the steps is design-dependent. Not all designs will require all of the steps to be performed. For example, if the design does not include a ring of pads, the pad ring step is not needed. You might choose to connect pin-to-trunk following the addition of the core ring and after adding each block ring.

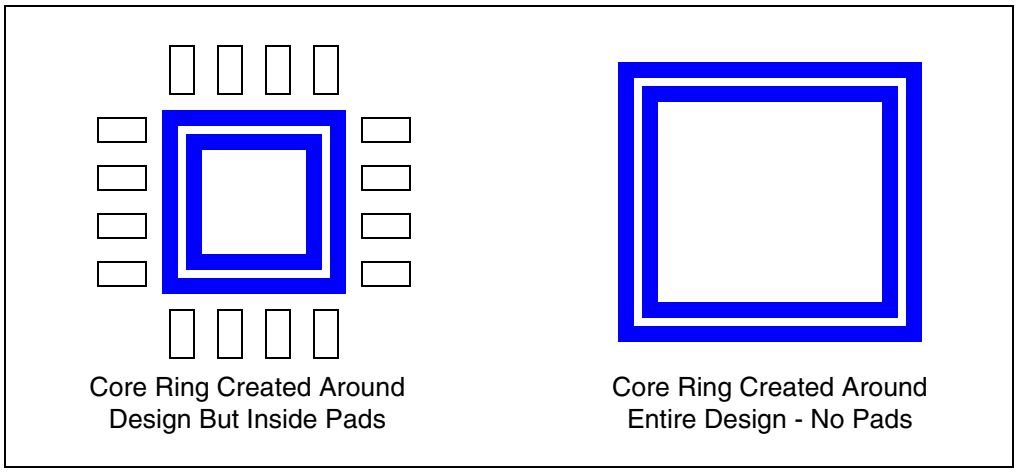
Adding a Pad Ring

Pad rings are routed between pads on the periphery of the design. You can route to rail pins and/or edge pins on pads.



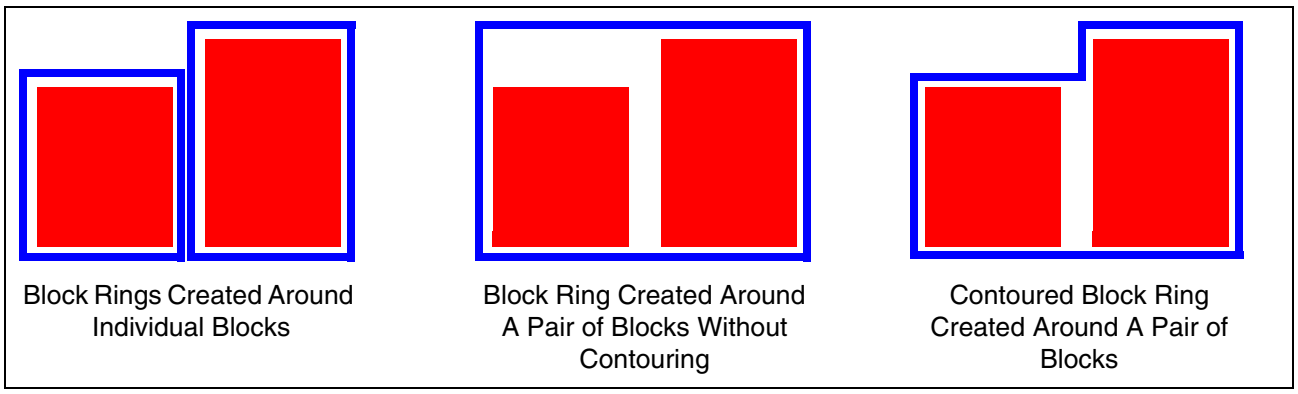
Adding Core Rings

If the design has pads, rings are added around the core of the design. If there are no pads, rings are added around the entire design.



Adding Block Rings

Block rings are wires around one or more block instances, on a set of specified layers with a specific offset between the boundaries of the blocks and the power rings. Rings can follow the contour of the blocks or form a rectangle around the blocks.



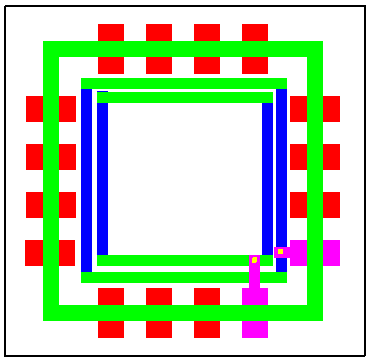
Block Rings Created Around Individual Blocks

Block Ring Created Around A Pair of Blocks Without Contouring

Contoured Block Ring Created Around A Pair of Blocks

Connecting Pin-to-Trunk

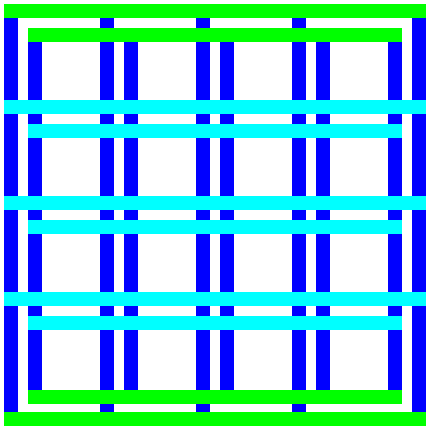
Power pins on blocks and power pads are connected to existing power rings or rails. The following example shows pin-to-trunk connections between pad pins and the core rings in the lower right corner.



Adding Stripes

Stripes, which can be used to form a power mesh, are added to the core area in the design. You specify parameters such as which layers to use for the given nets, the clearance required

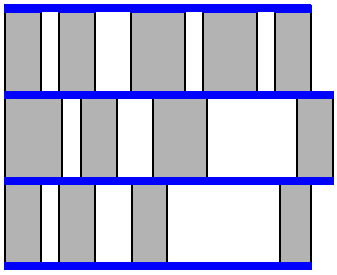
between nets, pin clearance, the location and interval for the stripes. In a later step, vias are inserted to connect the stripes within the mesh to the rings.



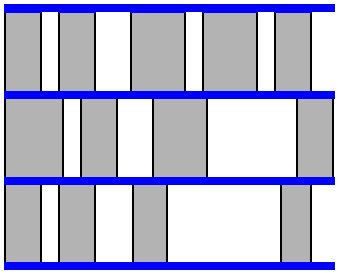
Horizontal and Vertical Stripes in this example form a power mesh inside the Core Ring.

Adding Standard Cell Row Straps

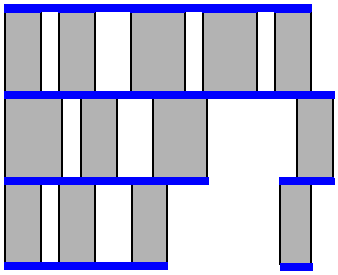
Straps are added along aligned power pins of standard cells. The straps can be extended to the end of rows or to the last cell in a physical row.



Standard Cell Row Straps to the Last Cell in Each Row (Default)



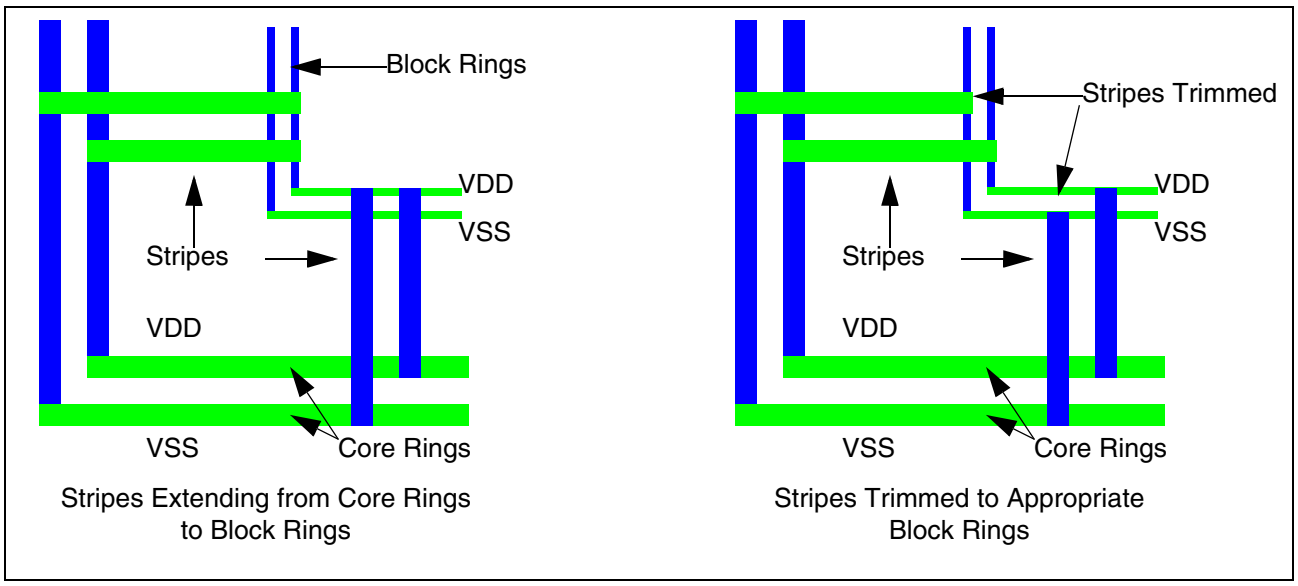
Standard Cell Row Straps to the End of Rows



Standard Cell Row Straps With Maximum Row Gap Size

Trimming Stripes

Stripes are trimmed back to an intersecting ring on the same net, if any. You can run this command after all stripes and rings have been created.

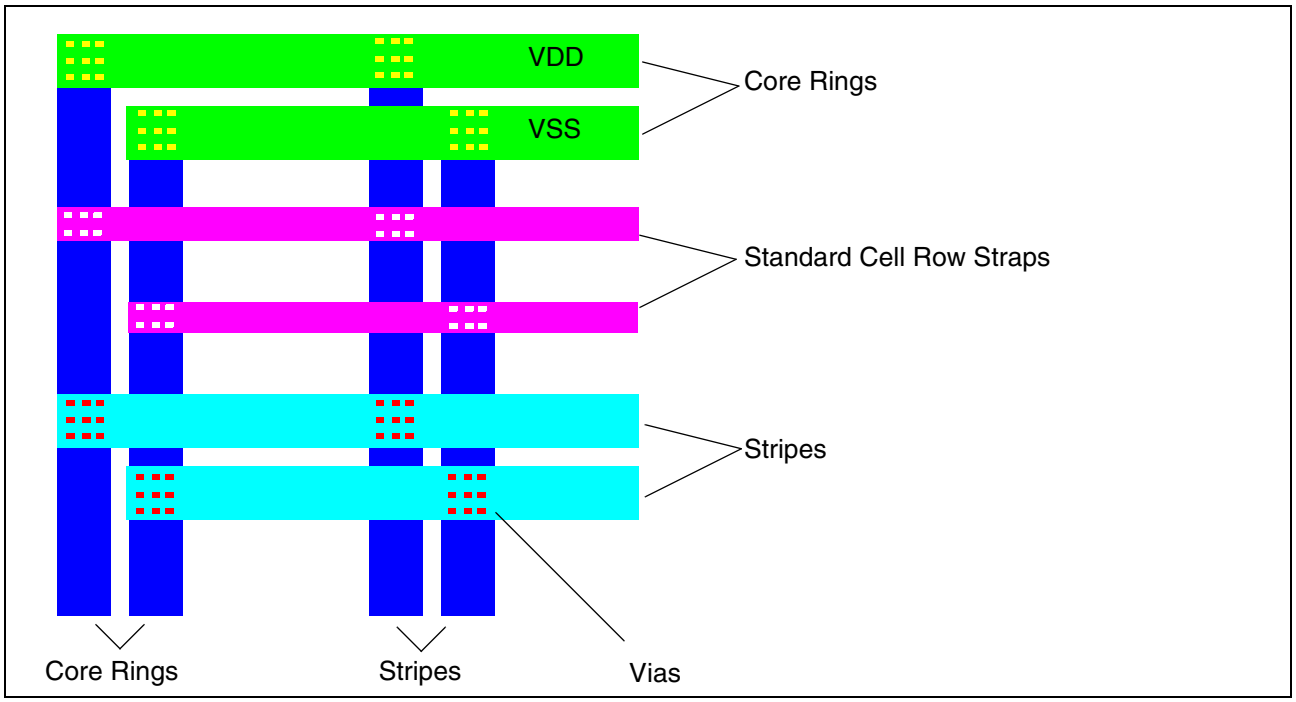


See rtePowerRouteTrimStripes in the Virtuoso Layout Suite SKILL Reference.

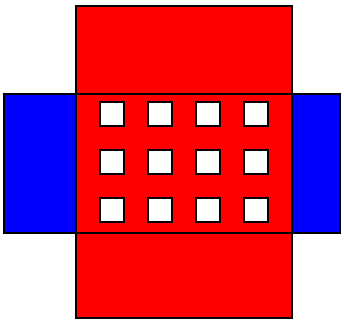
Inserting Vias

The current power routing topology does not support inserting vias instantaneously except connecting pin-to-trunk. Therefore, vias are added manually to provide interlayer connections

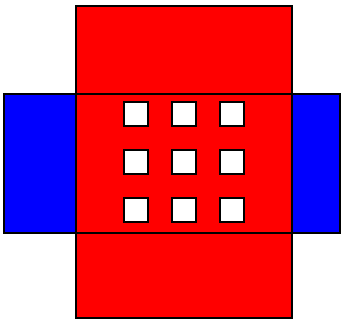
at intersections of rings, stripes and straps. The following example shows vias inserted to connect stripes and straps to the core rings in a corner of a simple design.



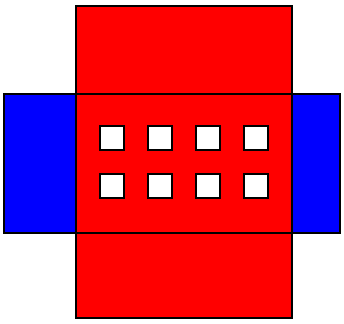
By default, cut arrays are inserted with the maximum number of rows and columns for the size of the via area. The following examples show different cut arrays that can be created for the same via by changing command arguments.



Maximum Number of Rows and Columns (Rectangle, default)



Equal Number of Rows and Columns (Square)



Specific Number of Rows and Columns

Tie Shielding

You can add ties to tie shield wires to the shield nets in the design either using the R*oute – Tie Shield* command or using the *Tie Shield* tab in the *Power Routing* form. You can also set the tie frequency. For more information, refer to Using the Tie Shield Form and Tie Shield.

Using the Power Routing Form

Using the Power Routing form, you can create power components in your design. To display the Power Routing form, choose one of the following methods.

- The Route menu is available by default in layout window. Choose *Route – Power Routing*.
- Alternatively, you can add the Space-based Router toolbar by choosing *Window - Toolbars - Space-based Router* and click the Power Routing icon in the toolbar

- With the *Route* menu displayed, you can press the *P* key on your keyboard to display the *Power Routing* form.

  The *Power Routing* form appears with tabs that represent a power component.

- Pad Ring
- Core Ring
- Block Ring
- Stripes
- Cell Rows
- Pin To Trunk
- Vias
- Tie Shield

In each tab you choose settings for the power component and can specify or edit the *scheme* to use. The settings on the main form for the component typically specify the objects to operate on. For example, these settings might specify the nets, pads, or blocks to use.

Schemes provide a fast, simple, and reproducible method to set up the power routing environment variables. Each power routing option has its unique scheme *package* comprised of a group of environment variables associated with it by default. To modify environment variables for a specific routing option, you can either alter the variable values in the default scheme or save changes into a customized scheme.

For more information on creating, deleting, comparing and other scheme functions, refer to Managing Schemes.

To route power components in your design, do the following.

1. Select at least one power or ground net before starting any power routing operation.

   If a power routing package is not defined in the currently selected scheme, the tab corresponding to that power routing package is disabled. To enable the tab, define a scheme for the power routing package. You can then route that particular power routing component.

   Note: Power routing does not support hierarchical routing. In other words, all routing is done at the top level, not through the hierarchy.

2. In the Power Routing form, click the tab of the component you want to route.
3. Set all the options in the scheme subform for that power routing component.
4. Click the *Route* button at the bottom of the Power Routing form to perform that particular power routing operation.

The following sections describe how to use the tabs in the Power Routing forms:

- Using the Pad Ring Form
- Using the Core Ring Form
- Using the Block Ring Form
- Using the Stripes Form
- Using the Cell Rows Form
- Using the Pin To Trunk Form
- Using the Vias Form
- Using the Tie Shield Form

The *Power Routing Options* form, which lets you view and edit the scheme package for each power routing option, can be accessed by choosing the *Edit* button from each of the tabs on the power routing form.

Using the Pad Ring Form

Pad Ring routing adds a ring between pad instances on the periphery of the design.

Video

For a short demonstration on how to create pad rings using the power routing GUI and the `rtePowerRoutePadRing` SKILL command, see Creating Pad Ring Using Power Routing.

To add pad rings to your design, do the following:

1. Click the *Pad Ring* tab.

   The *Pad Ring* tabbed page displays.

2. Use the Navigator or the Search Assistant to select the power or ground nets to route.
3. Use the Navigator or Search Assistant to select the pads to connect.

   If you do not select a pad, pad ring routing connects all pads.
4. To change or modify scheme options, do one of the following:
   - Select a different scheme from the *Name* drop-down list box.
   - Click *Edit...* to view or change the pad ring scheme definition.

The Power Routing Options form displays.

Follow the procedure in Changing the Pad Ring Scheme Definition.

5. Click *Route* to add pad rings with the options in the scheme.

Changing the Pad Ring Scheme Definition

The *Pad Ring* tab of the Power Routing Options form lets you specify pin layers and pin types to which to connect pad rings for the scheme.

To change the Pad Ring scheme definition:

1. Select the scheme name you want to use from the *Name* drop-down list box.
2. In the *Scheme Definition* group box, choose the Pin Layers that you want to connect.
   - By default, pad ring routing uses all the available routing layers. To specify pin layer to route on pad, select *Specified Pin Layers*. Once the *Specified Pin Layers* is selected, you can select the required routing layers in the list box.

     Click a layer to highlight it. `Ctrl+`click to add an additional layer. `Shift+`click adds layers inclusively.

     To remove one or more layers from the highlighted set, click the layer name.
3. Choose the *Pin Types* that you want the pad rings to connect to.
   - *Rail Pins* are pins that traverse a pad.
   - *Edge Pins* are pins that are on the edge of the pads.

   Note: If the pad pin type is edge pin, pad ring routing will fail with default pin type and displays a following warning message, No pad-rail POWER pins found on the given layers/nets, in the CIW.
4. To apply the settings without closing the form, click *Apply*. To apply the settings and close the form click *OK*.

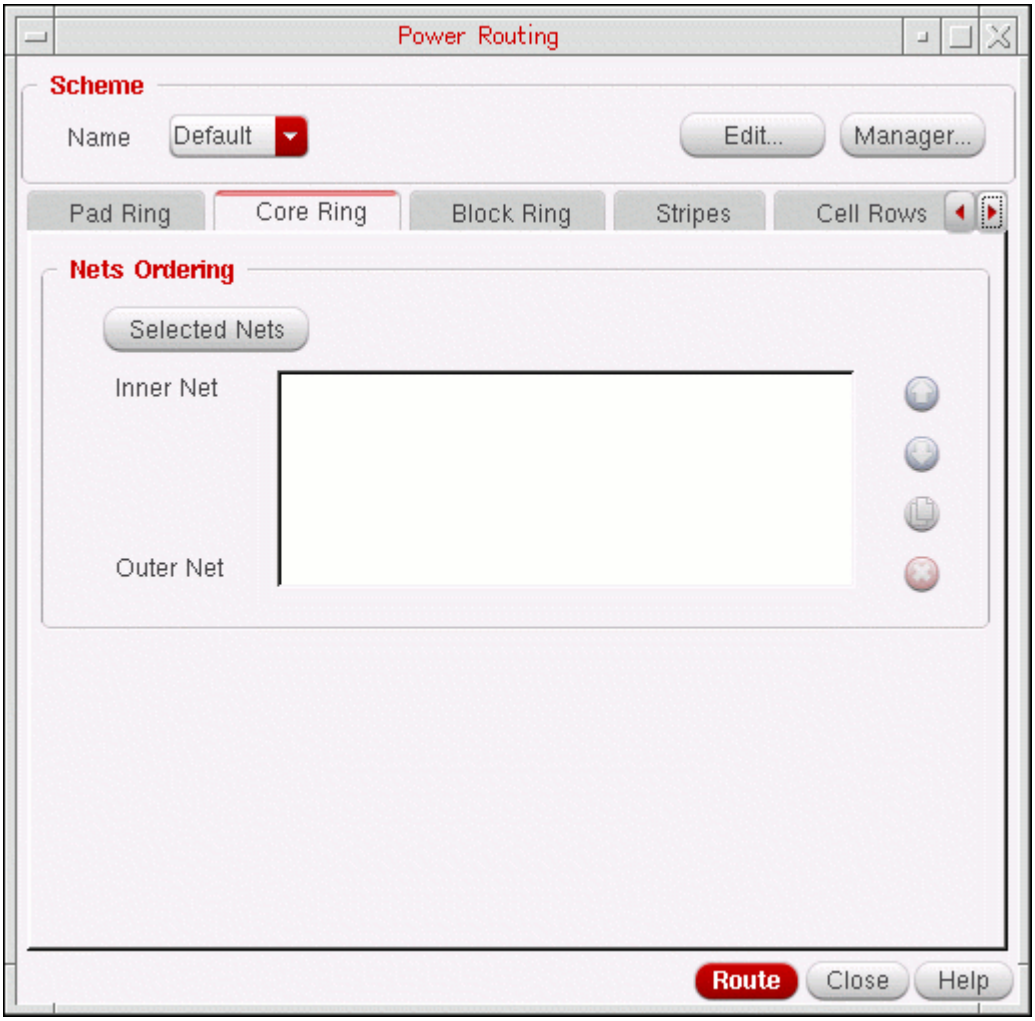
Using the Core Ring Form

Core Ring routing adds a ring around the core of a design that is surrounded by pad instances or I/O pins.

To add core rings to your design, do the following:

1. Click the *Core Ring* tab.

   The *Core Ring* tabbed page displays.



2. In the *Nets Ordering* group box, order the nets for the core rings.
   - Use the Navigator or Search Assistant to select power or ground nets.
   - Click *Selected Nets* to import the selected nets to the form.

Nets are routed with the first net in the list as the innermost ring and the last net in the list as the outermost ring. A net can appear in the list more than once.

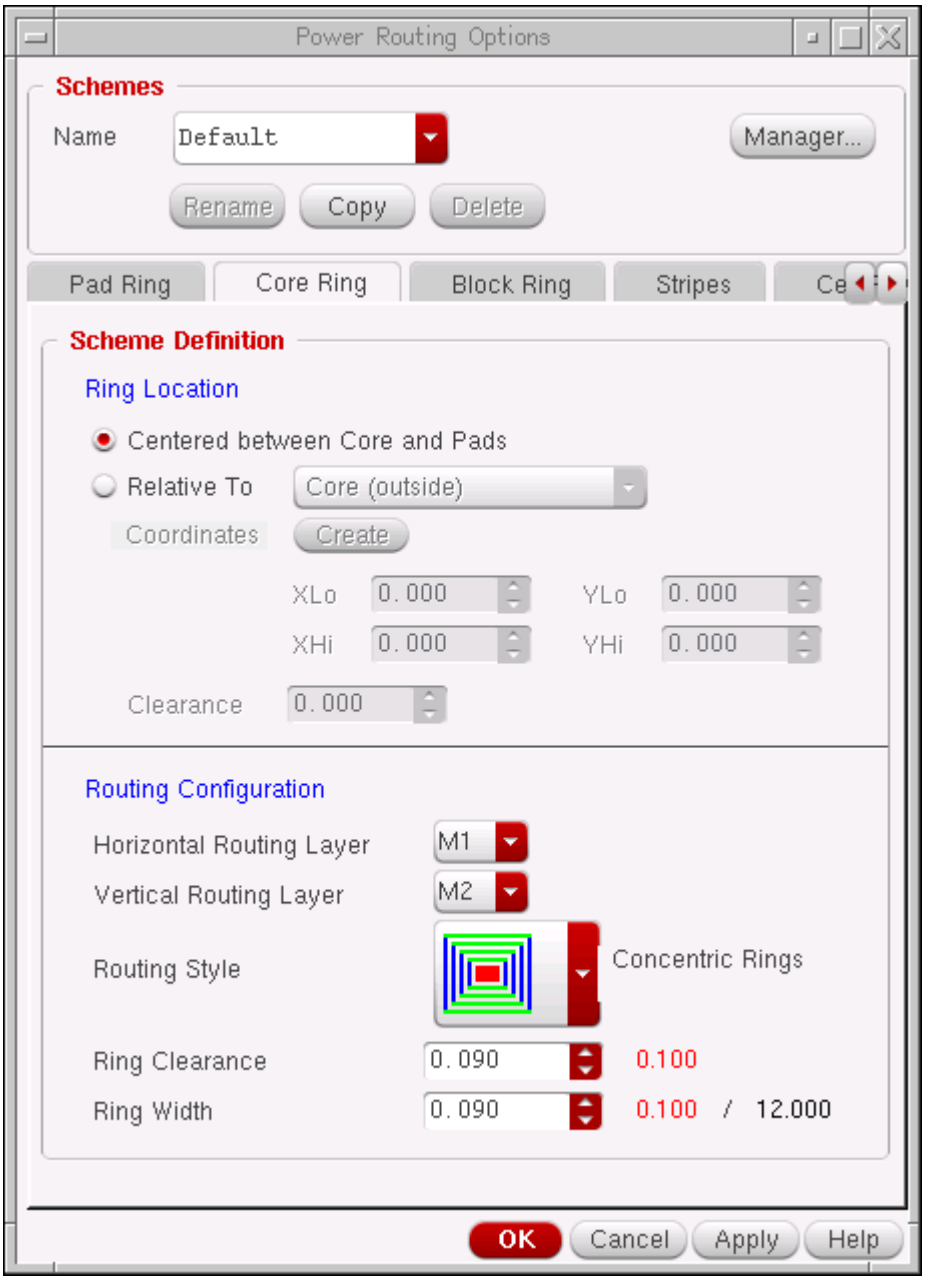
- Reorder nets by highlighting a net, then click the up or down arrow to change its position in the list.
- Duplicate one or more nets by highlighting the net names, then click the duplicate icon.

  By default, the duplicate nets are appended at the bottom of the list.
- Delete one or more nets by highlighting the net names, then click the delete icon.

3. To change or modify scheme options, do one of the following:

- Select a different scheme from the *Name* drop-down list box.
- Click *Edit...* to view or change the core ring scheme definition.

The Power Routing Options form displays.



Follow the procedure in Changing the Core Ring Scheme Definition.

4. Click *Route* to add core rings with the options in the scheme.

Changing the Core Ring Scheme Definition

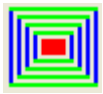
The *Core Ring* tab of the Power Routing Options form lets you specify the core ring location and configuration for the scheme.

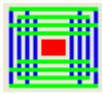
To change the Core Ring scheme definition:

1. Select the scheme name you want to use from *Name* drop-down list box.
2. In the *Ring Location* section of the *Scheme Definition* group box, choose between the following:
   - *Centered between Core and Pads* creates a ring between the core area of the design and the surrounding pad instances.
   - Route the core ring *Relative To* one of the following:
     - *Core (outside)* places the core ring relative to the outside the core of the design.
     - *I/O Pads (inside)* places the core ring relative to the inside of the pads.
     - *Specific Region (inside)* places the core ring inside the region given in step 3.
     - *Specific Region (outside)* places the core ring outside the region given in step 3.
3. If you chose a *Specific Region* in step 2, the *Coordinates* section allows you to specify the region by doing one of the following:
   - Enter the values using the spin boxes for the lower-left (*XLo*, *YLo*) and the upper-right (*XHi*, *YHi*) coordinates for the region.
   - Click *Create*. Then, draw the region (LMB) in the workspace by dragging a rectangle from a lower-left coordinate to an upper-right coordinate. The coordinate positions are automatically entered in the form.
4. In the *Ring Location* section, enter the *Clearance* between the core ring and the boundary you chose in step 2.
5. In the *Routing Configuration* section of the *Scheme Definition* group box, select a *Horizontal Routing Layer* from the drop-down list box.
6. In the *Routing Configuration* section, select a *Vertical Routing Layer* from the drop-down list box.

   Note: *Horizontal Routing Layer* and *Vertical Routing Layer* on the *Core Ring* tab follow the *routingDirections* defined in the *layerRules* section of the technology file.

7. In the *Routing Configuration* section, choose the *Routing Style* for multiple core rings from the following:

   
   Adds concentric core rings.

   
   Adds latticed core rings. Duplicate wire segments extend to form a lattice.

8. In the *Routing Configuration* section, set a *Ring Clearance* to specify the minimum spacing between the power/ground nets in the core ring. The *Ring Clearance* label next to the field indicates the minimum spacing value of the layer. If the *Ring Clearance* value is greater than the mentioned value, the label color is black; otherwise, the color is red.
9. In the *Routing Configuration* section, set a *Ring Width* to specify the total width for each power/ground net. The *Ring Width* label next to the field indicates the minimum and maximum width of the layer. If the *Ring Width* value is within the minimum and maximum values, the label color is black; otherwise, the color is red.
10. To apply the settings without closing the form, click *Apply*. To apply the settings and close the form click *OK.*

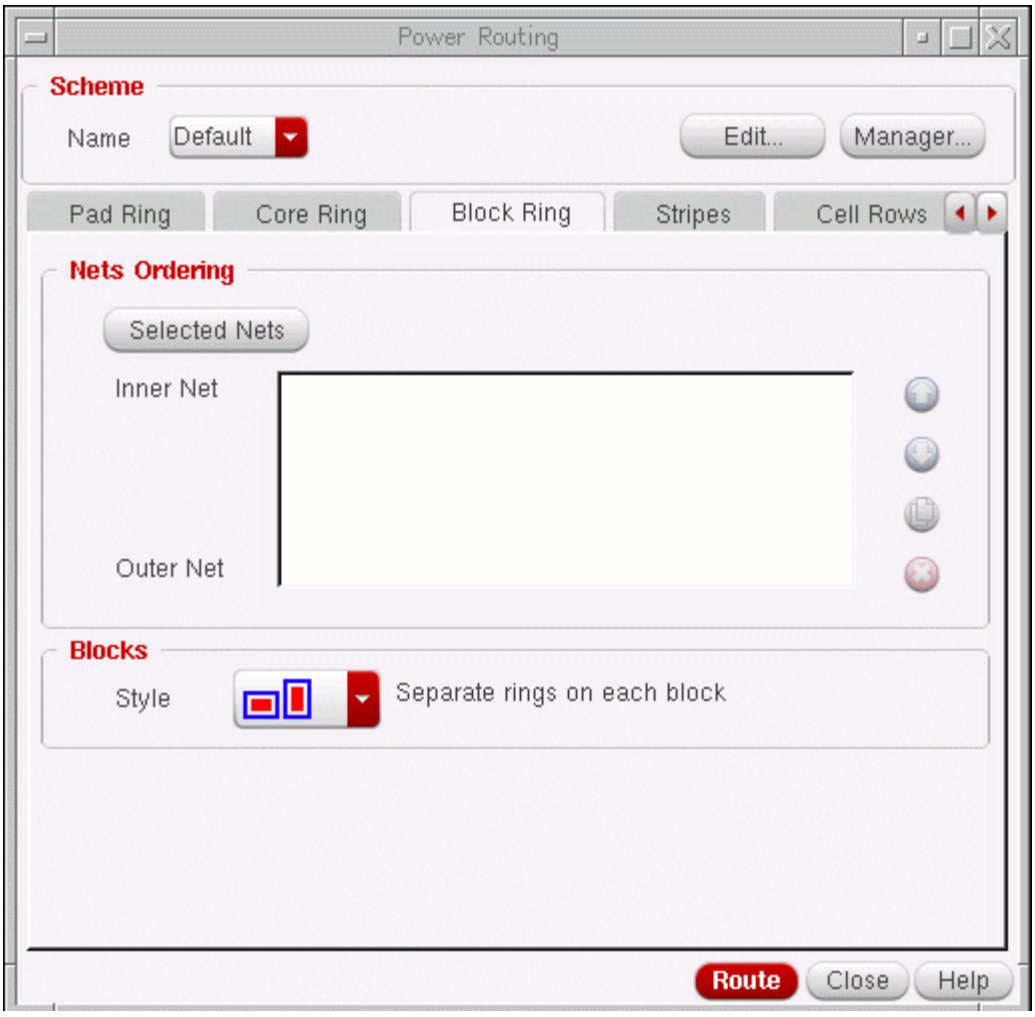
Using the Block Ring Form

Block Ring routing adds rings around block instances.

To add block rings to your design, do the following:

1. Click the *Block Ring* tab.

   The *Block Ring* tabbed page displays.



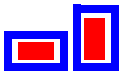
2. Use the Navigator or Search Assistant to select the blocks around which to add rings.
3. In the *Nets Ordering* group box, order the nets for the block rings.
   - ❑ Use the Navigator or Search Assistant to select the power or ground nets.
   - ❑ Click *Selected Nets* to add the selected nets to the form.

Nets are routed with the first net in the list as the innermost ring and the last net in the list as the outermost ring. A net can appear in the list more than once.

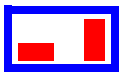
- Reorder nets by highlighting a net, then click the up or down arrow to change its position in the list.
- Duplicate one or more nets by highlighting the net names, then click the duplicate icon.

  The net name duplicates at the bottom of the list.
- Delete one or more nets by highlighting the net names, then click the delete icon.

4. In the *Blocks* group box, choose the *Style* in which to add block rings.



Adds separate rings on each block.

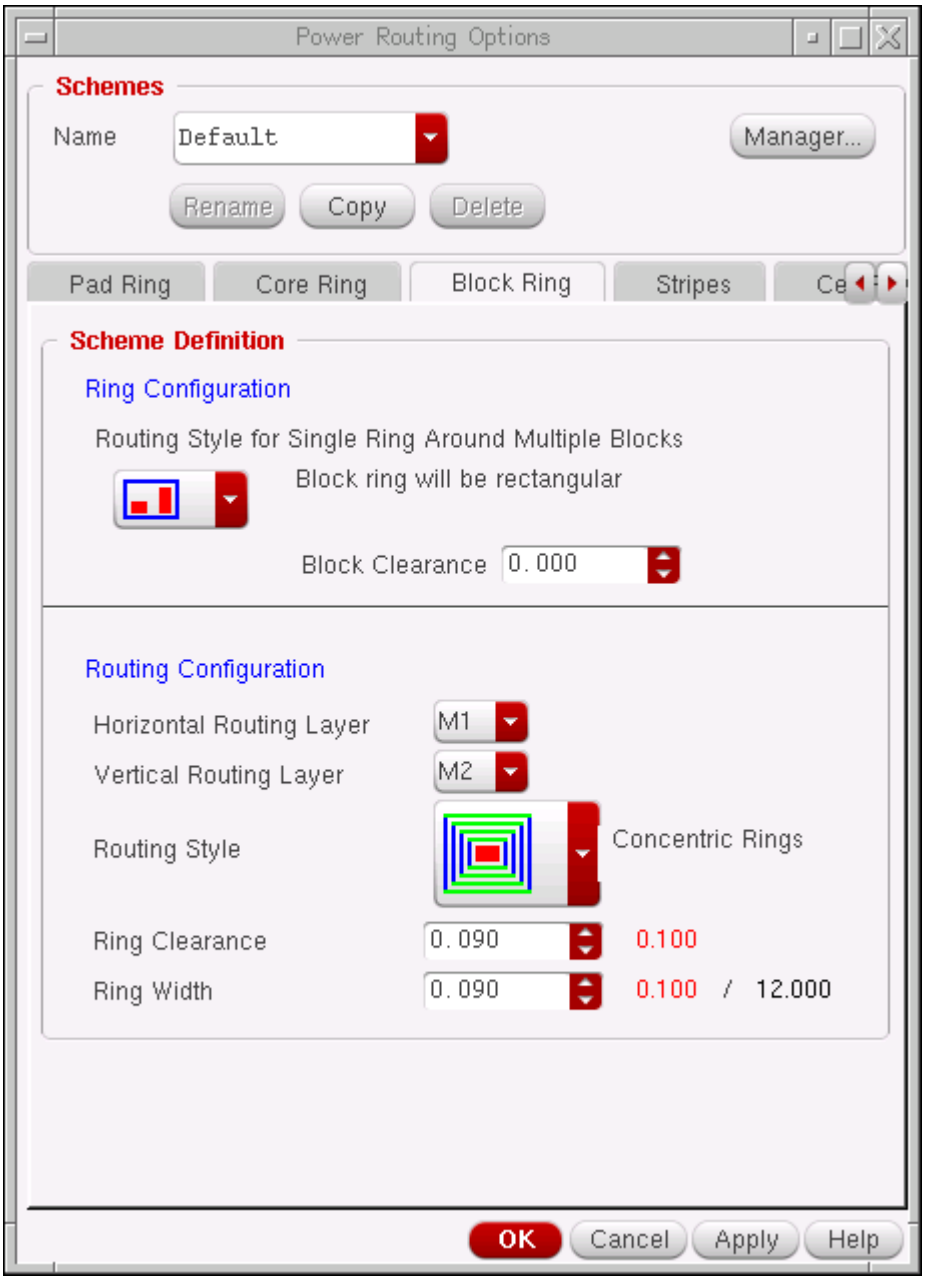


Adds a shared ring around all blocks.

Note: If Separate rings on each block option is selected, the block rings are executed individually on each block. In this case, the channel setting in ring configuration has no effect on individual block ring creation.

5. To change or modify scheme options, do one of the following:
   - Select a different scheme from the *Name* drop-down list box.
   - Click *Edit...* to view or change the block ring scheme definition.

The Power Routing Options form displays.



Follow the procedure in Changing the Block Ring Scheme Definition.

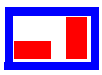
6. Click *Route* to add block rings with the options in the scheme.

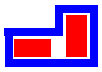
Changing the Block Ring Scheme Definition

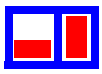
The *Block Ring* tab of the Power Routing Options form lets you specify the ring and routing configuration for block rings.

To change the Block Ring scheme definition:

1. Select a scheme name from the *Name* drop-down list box.
2. In the *Ring Configuration* section, if you chose to create shared rings around all of the given blocks as a group in step 4 of the Block Ring tab, choose the *Routing Style* for the block rings.

   
   Adds block rings in a rectangular shape around the group of blocks.

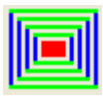
   
   Adds block rings contoured around the group of blocks.

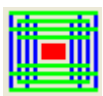
   
   Adds block rings in a rectangular shape around the group of blocks with rails in the channels between blocks.

   
   Adds block rings contoured around the group of blocks with rails in the channels between blocks.

3. In the *Ring Configuration* section, choose the block clearance to specify the clearance of the block rings to the prBoundary of the block.
4. In the *Routing Configuration* section, select a *Horizontal Routing Layer* from the drop-down list box.
5. In the *Routing Configuration* section, select a *Vertical Routing Layer* from the drop-down list box.

   Note: *Horizontal Routing Layer* and *Vertical Routing Layer* on the *Block Ring* tab follow the *routingDirections* defined in the *layerRules* section of the technology file.
6. In the *Routing Configuration* section, choose the *Routing Style* for multiple block rings from the following:

   
   Adds concentric block rings.

   
   Adds latticed block rings. Duplicate wire segments extend to form a lattice.

7. In the *Routing Configuration* section, choose the *Ring Clearance* to specify the minimum spacing between the power/ground nets in the ring. The *Ring Clearance* label next to the field indicates the minimum spacing value of the layer. If the *Ring Clearance* value is greater than the mentioned value, the label color is black; otherwise, the color is red.
8. In the *Routing Configuration* section, choose the *Ring Width* to specify the total width for each power/ground net. The *Ring Width* label next to the field indicates the minimum and maximum width of the layer. If the *Ring Width* value is within the minimum and maximum values, the label color is black; otherwise, the color is red.
9. To apply the settings without closing the form, click *Apply*. To apply the settings and close the form click *OK*.

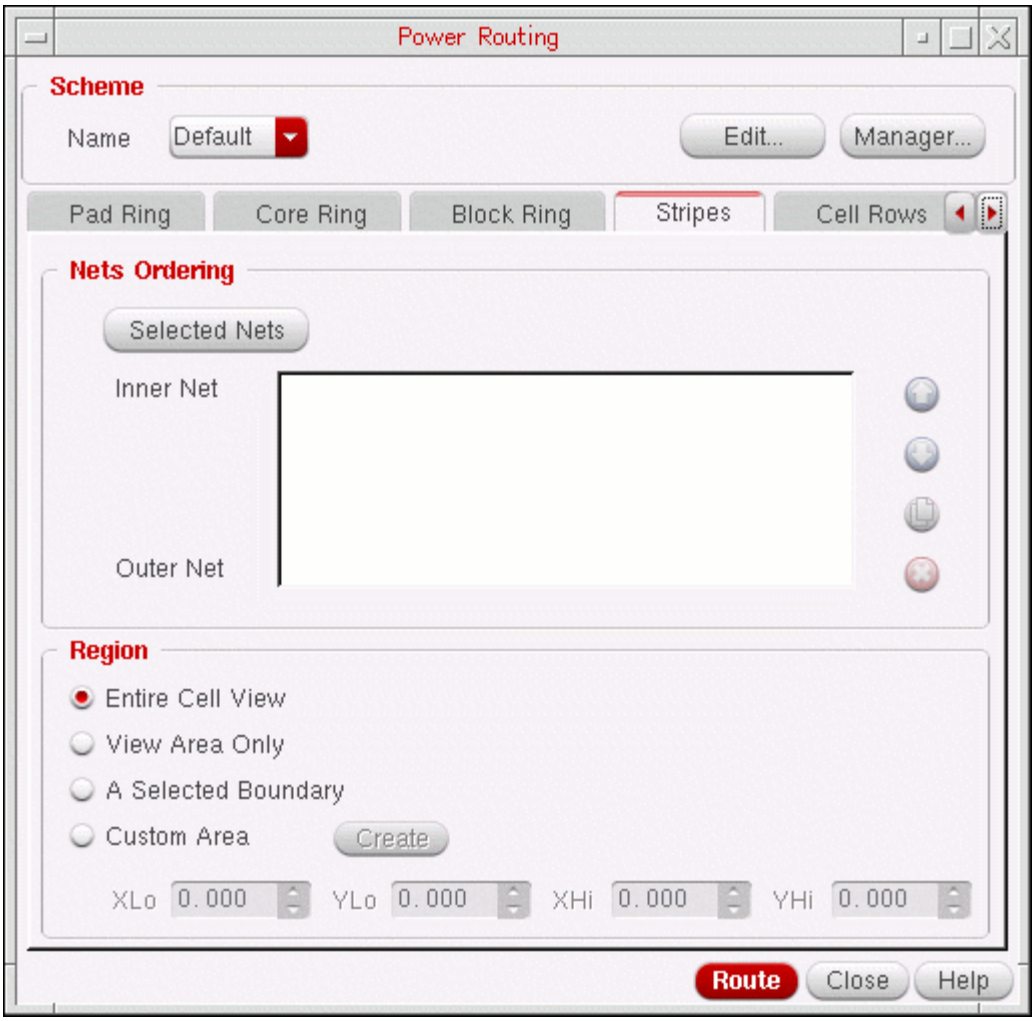
Using the Stripes Form

Stripes routing adds net stripes at regular intervals.

To add stripes to your design, do the following:

1. Click the *Stripes* tab.

   The *Stripes* tabbed page displays.



2. In the *Nets Ordering* section, order the nets for the stripes.
   - ❑ Use the Navigator or Search Assistant to select the power or ground nets.
   - ❑ Click *Selected Nets* to add the selected nets to the form.

Nets are routed with the first net in the list as the innermost ring and the last net in the list as the outermost ring. A net can appear in the list more than once.

- Reorder nets by highlighting a net, then click the up or down arrow to change its position in the list.
- Duplicate one or more nets by highlighting the net names, then click the duplicate icon.

  The net name duplicates at the bottom of the list.
- Delete one or more nets by highlighting the net names, then click the delete icon.

3. In the *Region* section, specify the region in which to add the stripes.

- *Entire Cell View*

  Use the current place and route boundary (prBoundary) for the design.

  When E*ntire Cell View* option is selected, stripes extension is determined by the core ring and prBoundary of the current cell view. If core rings exist, stripes extends to the inner most core ring. Else, they are extended to the prBoundary and covers the entire design. More than one routing region can be given.
- *View Area Only*

  Use the current zoom area for routing. If the zoom area is outside the prBoundary, a warning is issued.
- *A Selected Boundary*

  Use a selected boundary for routing. The boundary can be either a cluster boundary or an area boundary. However, you can select only one boundary at a time for routing. If you select more than one boundary, a warning is issued.

  If the selected boundary is outside the prBoundary, a warning is issued.
- *Custom Area*

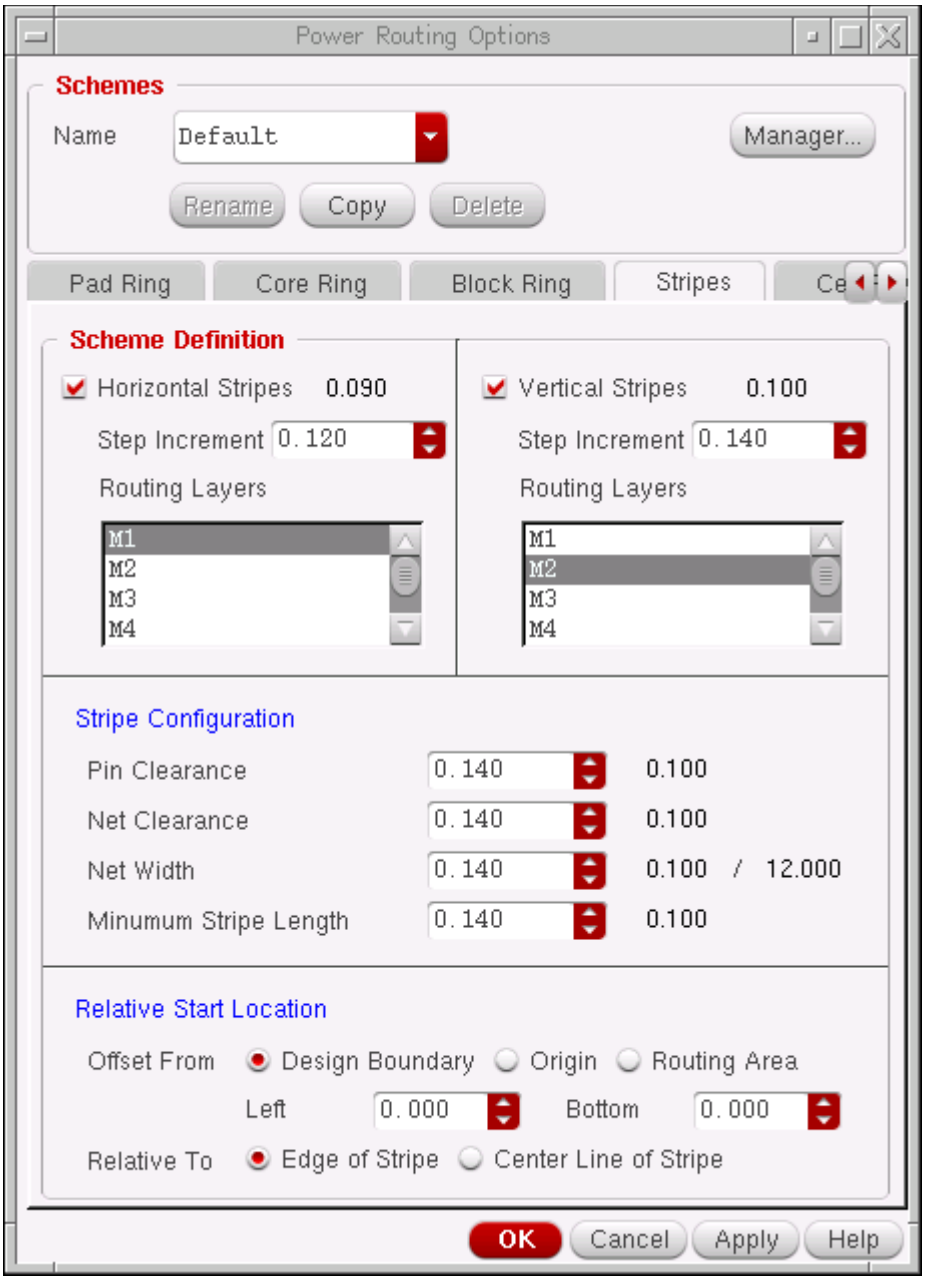
  Allows you to specify the region by doing one of the following:

  - Enter the values using the spin boxes for the lower-left (*XLo*, *YLo*) and the upper-right (*XHi*, *YHi*) coordinates for the region.
  - Click *Create*. Then draw the region in the workspace by dragging a rectangle from a lower-left coordinate to an upper-right coordinate. The coordinate positions are automatically entered in the form.

4. To change or modify scheme options, do one of the following:

- Select a different scheme from the *Name* drop-down list box.
- Click *Edit...* to view or change the stripes scheme definition.

The Power Routing Options form displays.



Follow the procedure in Changing the Stripes Scheme Definition.

5. Click *Route* to add stripes with the options in the scheme.

Changing the Stripes Scheme Definition

The *Stripes* tab of the Power Routing Options form lets you specify the placement and configuration of the horizontal and vertical stripes.

To change the stripes scheme definition:

1. Select the scheme name you want to use from the drop-down list box.
2. In the *Scheme Definition* group box, choose whether you want *Horizontal Stripes* to be created.
   - Choose the horizontal *Step Increment* from the spin box.

     The router uses this value to separate the horizontal stripes.
   - Specify the *Routing Layers* for the horizontal stripes.

     Highlight a layer in the list box to select it.

     `Ctrl+`click to highlight additional layers. `Shift+`click to highlight layers inclusively.

     To deselect a highlighted layer, click the layer in the list box.
3. In the *Scheme Definition* section, choose whether you want *Vertical Stripes* to be created.
   - Choose the vertical *Step Increment* from the spin box.

     The router uses this value to separate the vertical stripes.
   - Specify the *Routing Layers* for the vertical stripes**.**

     Highlight a layer in the list box to select it.

     `Ctrl+`click to highlight additional layers. `Shift+`click to highlight layers inclusively.

     To deselect a highlighted layer, click the layer in the list box.
4. In the *Stripe Configuration* section, set the following:
   - Choose the *Pin Clearance* from the spin box to specify the clearance required between signal pins and stripes. The *Pin Clearance* label next to the field indicates the minimum clearance required between signal pins and stripes. If the *Pin Clearance* value is greater than the mentioned value, the label color is black; otherwise, the color is red.
   - Choose the *Net Clearance* from the spin box to specify the minimum spacing required between the power/ground nets within a stripe. The *Net Clearance* label next to the field indicates the minimum spacing value between the power and ground

nets within a stripe. If the *Net Clearance* value is greater than the mentioned value, the label color is black; otherwise, the color is red.

- Choose the *Net Width* from the spin box to specify the total width for each routed net. The *Net Width* label next to the field indicates the minimum and maximum width of each routed net. If the *Net Width* value is within the minimum and maximum values, the label color is black; otherwise, the color is red
- Choose the *Minimum Stripe Length* from the spin box. The *Minimum Stripe Length* label next to the field indicates the minimum length of the stripes. If the *Minimum Stripe Length* value is greater than the mentioned value, the label color is black; otherwise, the color is red.

Note: The Minimum step increment = (Net Width + Net Clearance)* Number of selected Power nets. For instance, if you create power mesh for VSS, VDD, DVDD, and GND nets, then the minimum step increment is computed as follows:

```
min = (Wnet + Wnet_clearance)*N
N=len(VDD, DVDD, VSS, GND)
```

Else, the routing fails due to insufficient step size.

5. In the *Relative Start Location*, set the following:
   - Choose the stripes to be placed with the left and right *Offset From* either the *Design Boundary*, the design *Origin*, or the *Routing Area* (the routing regions given on the *Stripes* tab of the Power Routing form).
   - Choose the *Left* offset from the spin box to specify the X location of the first vertical stripe.
   - Choose the *Bottom* offset from the spin box to specify the Y location of the first horizontal stripe.
   - Choose whether the stripe offsets are measured *Relative To* the *Edge of Stripe* (left edge for vertical stripes and bottom edge for horizontal stripes), or the *Center Line of Stripe*.
6. To apply the settings without closing the form, click *Apply*. To apply the settings and close the form click *OK*.

Using the Cell Rows Form

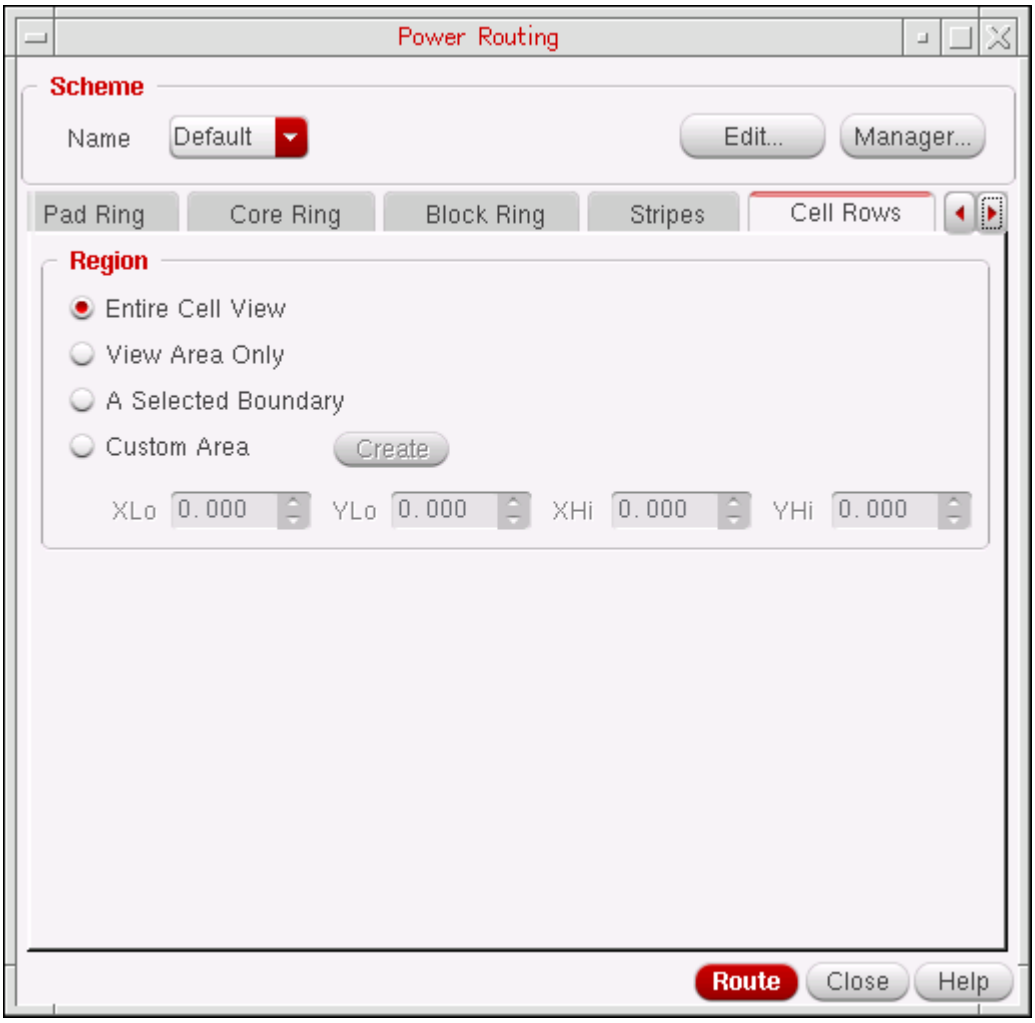
Cell Row routing adds straps along aligned pins of standard cells. The criteria for cell row power routing are the following:

- the net must be a power net
- the pin must be a power pin
- the cell must be a standard cell
- the pin shape should be rectangular

To add straps in your design, do the following:

1. Click the *Cell Rows* tab.

The *Cell Rows* tabbed page displays.



2. In the *Region* section, choose the regions in which to add the standard cell straps by doing one of the following:

   - *Entire Cell View*

     Use the current place and route boundary (prBoundary) for the design.

     If no routing regions are given, and core rings exist, the core rings become the bounds for the cell rows. If no routing regions are given, and core rings do not exist, cell rows will cover the entire design. More than one routing region can be given.

   - *View Area Only*

     Use the current zoom area for routing. If the zoom area is outside the prBoundary, a warning is issued.

- *A Selected Boundary*

  Use a selected boundary for routing. The boundary can be either a cluster boundary or an area boundary. However, you can select only one boundary at a time for routing. If you select more than one boundary, a warning is issued.

  If the selected boundary is outside the prBoundary, a warning is issued.

- *Custom Area*

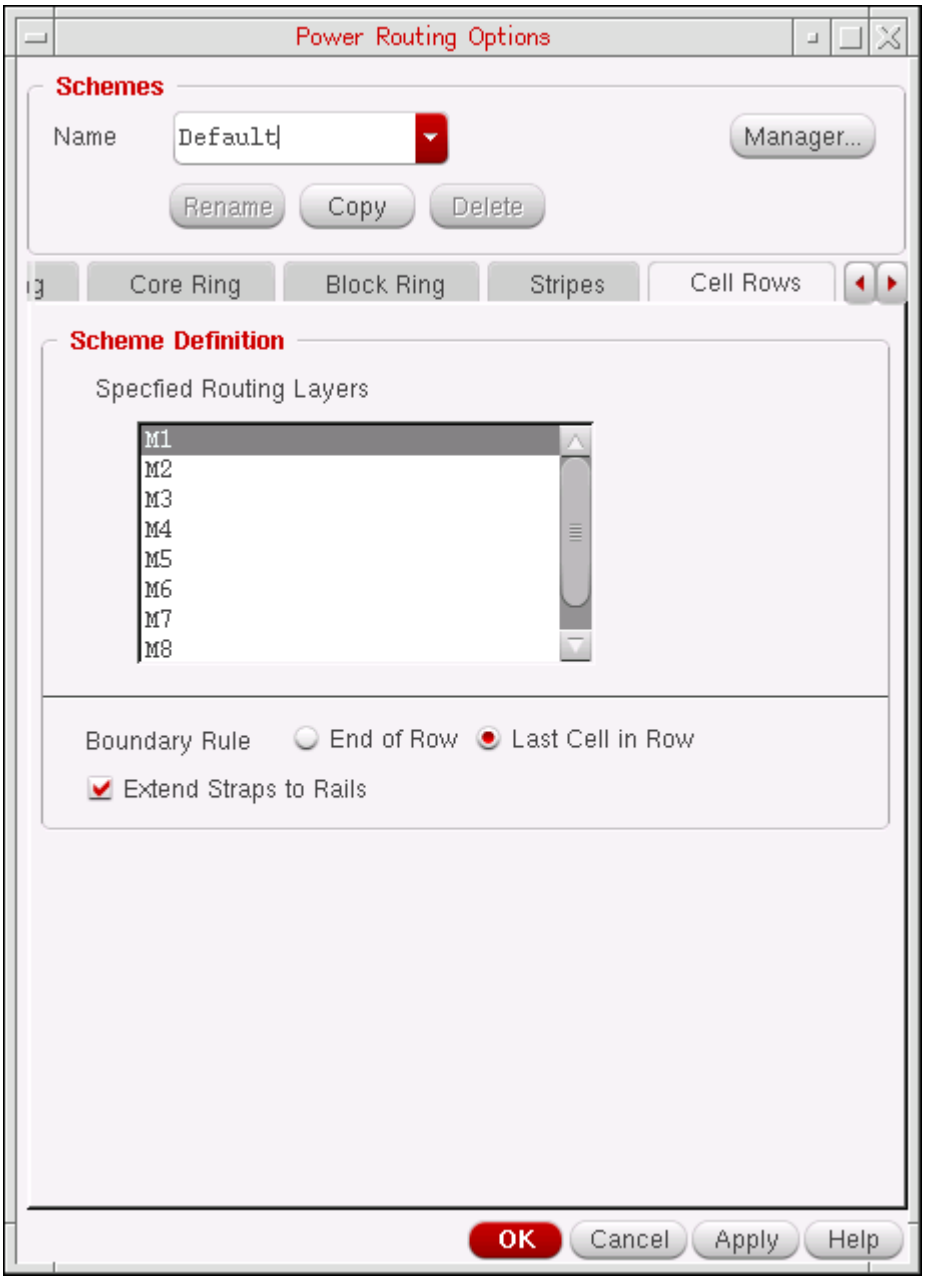
  Allows you to specify the region by doing one of the following:

  - Enter the values using the spin boxes for the lower-left (*XLo*, *YLo*) and the upper-right (*XHi*, *YHi*) coordinates for the region.
  - Click *Create*. Then draw the region in the workspace by dragging a rectangle from a lower-left coordinate to an upper-right coordinate. The coordinate positions are automatically entered in the form.Choose the nets to route cell row straps for.

3. To change or modify scheme options, do one of the following:

- Select a different scheme from the *Name* drop-down list box.
- Click *Edit...* to view or change the cell rows scheme definition.

The Power Routing Options form displays.



Follow the procedure in Changing the Cell Rows Scheme Definition.

4. Click *Route* to add standard cell row straps with the options in the scheme.

Changing the Cell Rows Scheme Definition

The *Cell Rows* tab of the Power Routing Options form lets you specify routing layers, the cell row strap boundary, and whether the cell row straps should be extended to the nearest power rail.

To change the Cell Rows scheme definition:

1. Select the scheme name you want to use from the *Name* drop-down list box.
2. In the *Scheme Definition* group box, choose the routing layers that you want to connect.
   - Use the Navigator or Search Assistant to select the nets.
   - Specify the routing layers in the *Specified Routing Layers* list box.

     Highlight a layer in the list box to select it.

     `Ctrl+`click to highlight additional layers. `Shift+`click to highlight layers inclusively.

     To deselect a highlighted layer, click the layer in the list box.
3. In the *Scheme Definition* group box, choose the *Boundary Rule*.
   - *End of Row*

     Routes cell row straps to the end of the defined rows.
   - *Last Cell in Row*

     Routes cell row straps to the last cell in the row.
4. Choose whether you want the cell row straps to *Extend Straps to Rails*.

   Extends straps to the farthest power rail.
5. To apply the settings without closing the form, click *Apply*. To apply the settings and close the form click *OK*.

Using the Pin To Trunk Form

Pin-To-Trunk routing allows you to add connections from the power pins of pad or macro instances to existing rings and rails. A macro instance is defined as any instance whose area is greater then 90 percent of the entire cell view area.

To route pin-to-trunk connections, do the following:

1. Click the *Pin To Trunk* tab.

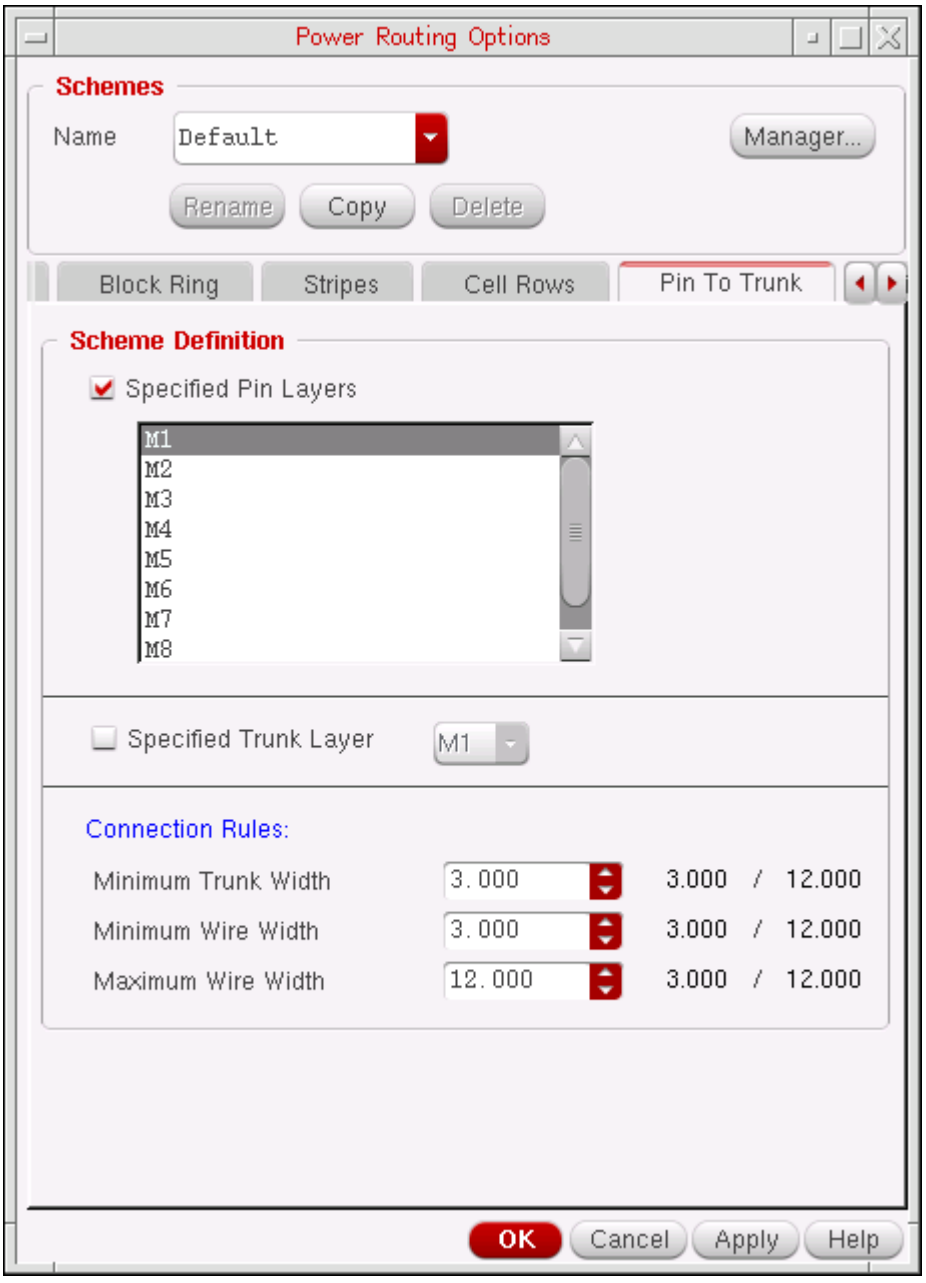
   The *Pin To Trunk* tabbed page displays.

2. Use the Navigator or Search Assistant to select the power or ground nets to route.
3. Use the Navigator or Search Assistant to select the pad or macro instances to which to add pin-to-trunk connections.

If no pad or macro instances are specified, the router uses all pad and macro instances.

4. To change or modify scheme options, do one of the following:
   - Select a different scheme from the *Name* drop-down list box.
   - Click *Edit...* to view or change the pin-to-trunk scheme definition.

   The Power Routing Options form displays.



Follow the procedure in Changing the Pin-To-Trunk Scheme Definition.

5. Click *Route* to add pin-to-trunk connections with the options in the scheme.

Changing the Pin-To-Trunk Scheme Definition

The *Pin To Trunk* tab of the Power Routing Options form lets you specify pin layers, trunk layers, and connection rules.

To change the Pin To Trunk scheme definition:

1. Select the scheme name you want to use from the *Name* drop-down list box.In the *Scheme Definition* group box, do one of the following to specify pin layers that you want to connect:
   - Deselect *Specified Pin Layers.*

     Connects all pin layers.
   - Select *Specified Pin Layers.*

     Highlight a layer in the list box to select it.

     `Ctrl+`click to highlight additional layers. `Shift+`click to highlight layers inclusively.

     To deselect a highlighted layer, click the layer in the list box.
2. In the *Scheme Definition* group box, choose the *Specified Trunk Layer* that you want to connect.
   - Deselect *Specified Trunk Layer.*

     Connects all trunk layers.
   - Select *Specified Trunk Layer.*

     Restricts pin-to-trunk connections to the trunk layer you select in the drop-down list box.
3. In the *Scheme Definition* group box, specify the *Connection Rules* that you want to use.
   a. Choose the *Minimum Trunk Width* in the spin box to specify the minimum trunk width for a trunk to be connected to a pin.
   b. Choose the *Minimum Wire Width* in the spin box to specify the minimum wire width for a wire to connect from a pin to a trunk.

      Note: If you specified a trunk layer in the *Specified Trunk Layer* drop-down list

box, the numbers to the right of the *Minimum Trunk Width* and *Minimum Wire Width* spin boxes are the value of the *minWidth* constraint of the selected layer.

If you did not specify a trunk layer, the numbers to the right of the spin boxes are the largest *minWidth* of all the Specified Pin Layers. If you then set a *Minimum Wire Width* value that is smaller than the number to the right of the spin box, you are likely to violate the *minWidth* on layers with smaller minimums. Best practice is to set a value greater than or equal to this number, in which case the number is black.

c. Choose the *Maximum Wire Width* in the spin box to specify the maximum wire width for a wire to connect from a pin to a trunk.

Pin to Trunk uses the largest wire width that does not exceed the *Maximum Wire Width* or the width of the pin.

Note: If you specify a trunk layer in the *Specified Trunk Layer* drop-down list box, the numbers to the right of the *Minimum Trunk Width*, *Minimum Wire Width*, and *Maximum Wire Width* in the *Connection Rules* section will be updated to the corresponding layer constraint values.

If you did not specify a trunk layer, the number to the right of the spin box is the smallest *maxWidth* of all the Specified Pin Layers. If you then set a *Maximum Wire Width* that is larger than this number, you are likely to violate the *maxWidth* on layers with smaller maximums. Best practice is to set a value less than or equal to this number, in which case the number is black.

4. To apply the settings without closing the form, click *Apply*. To apply the settings and close the form click *OK*.

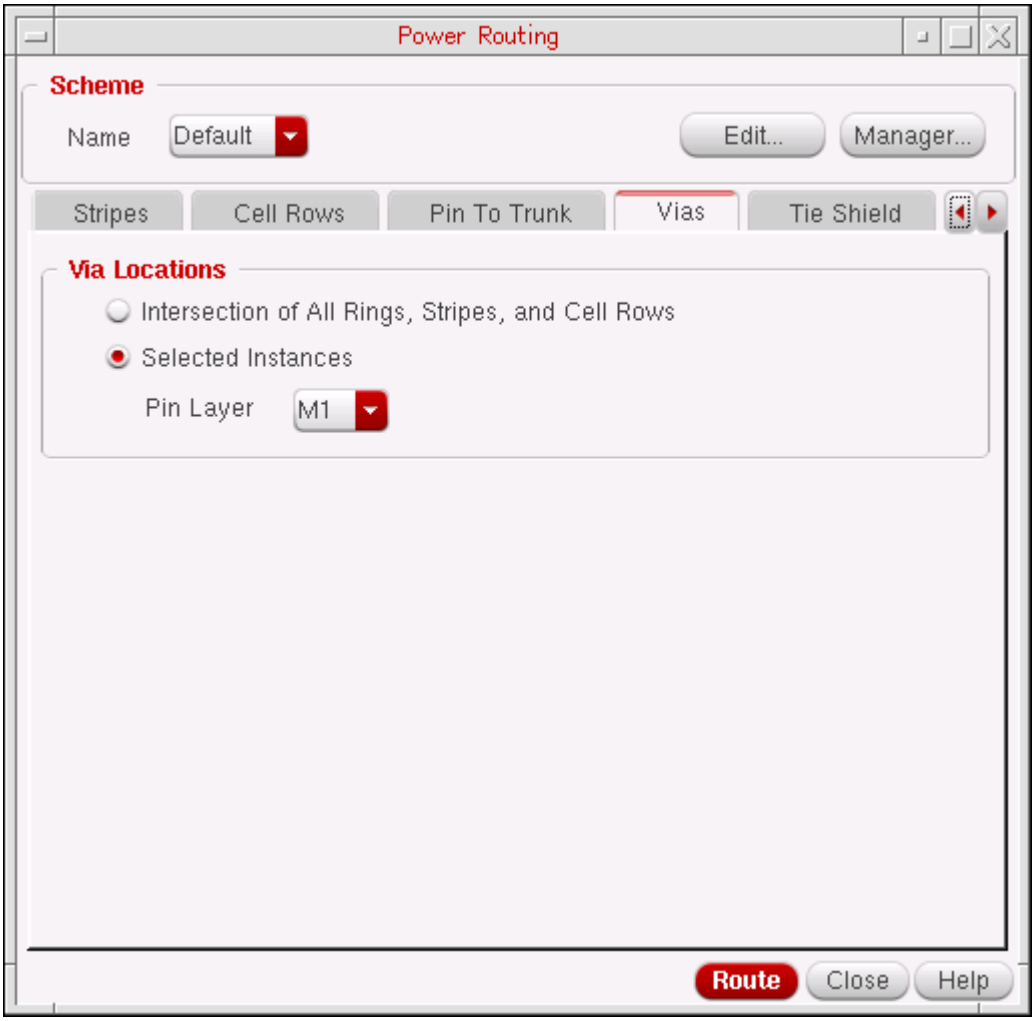
Using the Vias Form

Vias routing adds vias for interlayer connection at intersections of rings, rails, stripes, straps, and cell rows, or between instance pins and stripes.

To add vias in your design, do the following:

1. Click the *Vias* tab.

   The *Vias* tabbed page displays.



2. Choose the *Via Locations* to add.

   Inter-layer connections can take place at one of the following:

   - *Intersection of All Rings, Stripes, and Cell Rows* adds vias at the intersections of all rings, stripes, and cell rows.

- *Selected Instances* adds vias between stripes and the pins of selected block instances.

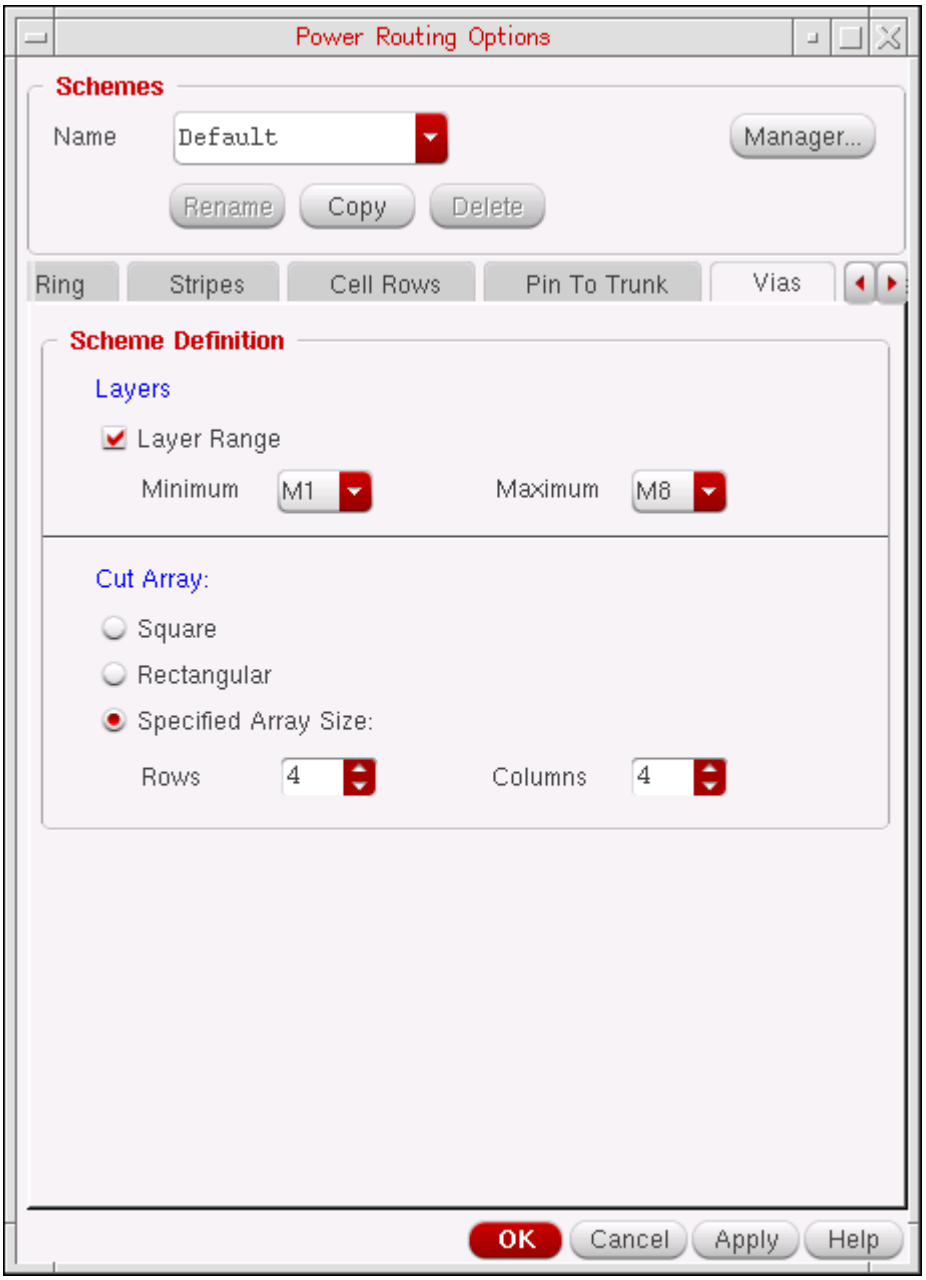
  a. Use the Navigator or Search Assistance to select the block instances.

  b. *Pin Layer* determines the layer to which the via should be inserted.

3. To change or modify scheme options, do one of the following:

- Select a different scheme from the *Name* drop-down list box.
- Click *Edit...* to view or change the vias scheme definition.

The Power Routing Options form displays.



Follow the procedure in Changing the Vias Scheme Definition.

4. Click *Route* to add vias with the options in the scheme.

Changing the Vias Scheme Definition

The *Vias* tab of the Power Routing Options form lets you specify layers and cut array types.

To change the Vias scheme definition:

1. Select the scheme you want to use from the *Name* drop-down list box.
2. In the *Scheme Definition* group box, choose the *Layers* that you want to connect.
   - Deselect *Layer Range.*

     Inserts vias to connect all layers.
   - Select *Layer Range*.

     Inserts vias to connect a range of metal layers. Use the *Minimum* and *Maximum* drop-down list boxes to select the bottom and top layers, respectively, to connect.
3. Choose the *Cut Array* type.
   - *Square*

     Specifies a square cut array for each via, with an equal number of rows and the number of columns. However, the cut array might not be square in size, depending on spacing rules. If the `minAdjacentViaSpacing` rule is set, then the spacing of the cuts can be different in the X and Y directions, producing a cut array that is rectangular in dimension.
   - *Rectangular*

     Specifies that the maximum number of rows and columns for the cuts be used for the available space.
   - *Specified Array Size*

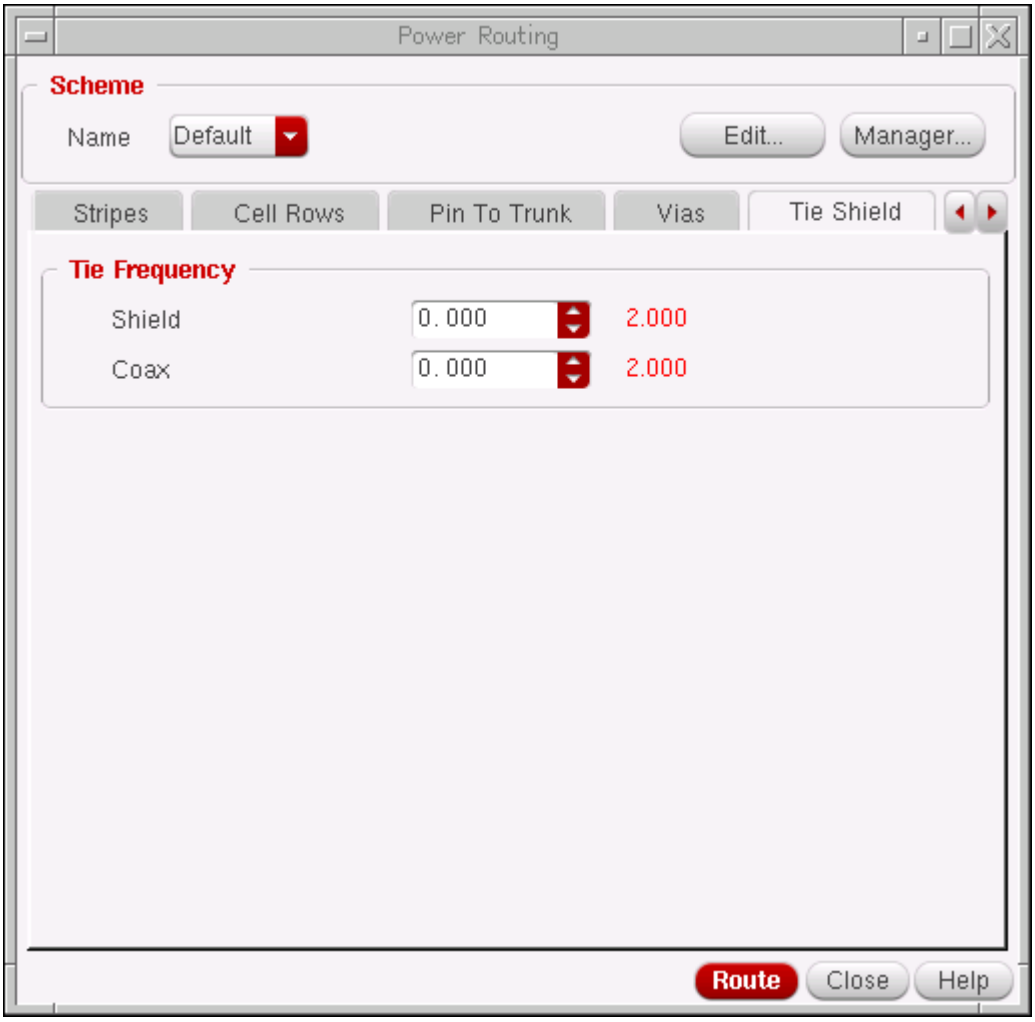
     Inserts vias with cut arrays of a specific dimension. If the cut array with these dimensions cannot fit within a via area, no via is created for that metal intersection. Choose the number of *Rows* and *Columns* from their respective spin boxes.
4. To apply the settings without closing the form, click *Apply*. To apply the settings and close the form click *OK*.

Using the Tie Shield Form

Ties shield wires to shield nets in the design. Do the following:

1. Click the *Tie Shield* tab.

   The *Tie Shield* tabbed page displays.



2. In the *Tie Frequency* section, choose the distance between the ties.

   - Choose the frequency of the *Shield* ties.

     Specifies the maximum distance between ties that must be inserted to tie the new shield wires to their respective shield nets. The *Shield* label next to the field indicates the maximum distance that should exist to tie the new shield wires to their

respective shield nets. If the *Shield* value is greater than the mentioned value, the label color is black; otherwise, the color is red.

- Choose the frequency of the *Coax* shield ties.

  Specifies the maximum distance between ties that must be inserted to tie tandem shield wires and parallel shield wires for coaxial shielding. The *Coax* label next to the field indicates the maximum distance that should exist to tie tandem shield wires and parallel shield wires for coaxial shielding. If the *Coax* value is greater than the mentioned value, the label color is black; otherwise, the color is red.

Note: The number to the right of the spin boxes is the largest *minSpacing* value in the design. If the color of the number is red, the current setting in the spin box is smaller than the largest *minSpacing* constraint value. You are then likely to violate the *minSpacing* on layers with larger minimums. Best practice is to set a value greater than or equal to this number, in which case the number is black.

3. Click *Route* to tie the shield wires to the nets.

Setting the Packages for the Scheme

To select the packages available in the scheme, do the following:

1. In the Power Routing form, select the scheme you want to use from the *Name* drop-down list box.Click *Edit* to display the Power Routing Options form.
2. Click the *Setup* tab.
3. In the *Scheme Packages* section, select the packages that are part of this scheme. The options are the following:
   - ❑ Pad Ring
   - ❑ Core Ring
   - ❑ Block Ring
   - ❑ Stripes
   - ❑ Cell Rows
   - ❑ Pin To Trunk
   - ❑ Vias
4. Click *Close* to apply the settings and close the form.

Managing Schemes

In the Power Routing Options form, you can manipulate schemes using the following options:

- Creating New Schemes
- Deleting a Scheme
- Loading a Scheme
  - Loading schemes using SKILL
- Saving a Scheme File
- Comparing Schemes

Creating New Schemes

To create a new scheme, do the following:

1. Click *Edit* to display the Power Routing Options form.
2. In the Power Routing Options form, ensure that the *Name* field displays the name of the scheme as `Default`.
3. Click the *Copy* button.

   A copy of the scheme now exists and the name is displayed as `Default_n`. The default name is the original scheme name with `_n`, where n is a number, automatically appended.
4. Optionally, you can rename the scheme by entering the new name in the *Name* field and clicking *Rename*. Specify the scheme name as `myBlockRing`.

   Note: You cannot use the name Default when renaming a scheme. Default is an internal scheme that should not be overwritten or removed. The form will not allow you to name a scheme Default or change the Default scheme.
5. Click the *Block Ring* tab and make some changs to the setiings
6. Specify the block clearance value as `1.200` in the *Block Clearance* field.
7. Click OK or Apply to save the `myBlockRing` scheme to the memory.

   A new scheme is created and saved in the memory.

Deleting a Scheme

To delete a scheme, do the following:

1. In the Power Routing Options form, display the name of the scheme to delete in the *Name* field.
2. Click the *Delete* button.

   Note: You cannot delete the Default scheme. Default is an internal scheme that should not be overwritten or removed. The form will not allow you to delete the Default scheme.

Loading a Scheme

You can load a scheme from a file.

1. Display the Power Routing Scheme Manager form by clicking the *Manager* button from either the Power Routing or Power Routing Options form.
2. Ensure you are in the *File* tab of the Power Routing Scheme Manager form.
3. Enter the name of the file containing the schemes in the *Name* field.

   Alternatively, you can use the *Browse* button to select the file containing the schemes.
4. Click the *Load* button to load all the schemes in the file into the power routing scheme database.

   All the loaded schemes appear in the list box.

   Note: The form will not duplicate schemes with identical names. If you load a scheme having the same name as an existing scheme, the existing scheme is overwritten.

When you save the design, schemes are written to the current directory by default. To save schemes to another location, see Saving a Scheme File.

Loading schemes using SKILL

You can load schemes automatically when starting **virtuoso**. For each scheme you want to load, add the following SKILL command to the .cdsinit file in your home directory.

```
(load "mySchemeFilePath/mySchemeFileName")
```

Saving a Scheme File

You can save schemes to a file.

1. Display the Power Routing Scheme Manager form by clicking the *Manager* button from either the Power Routing or Power Routing Options form.
2. Ensure you are in the *File* tab of the Power Routing Scheme Manager form. Also, see that the available schemes are listed in the *Schemes* list box.
3. In the *Name* field, enter the name of the file to which you want to save the schemes.

   Alternatively, you can use the *Browse* button to select the filename.
4. Click the check boxes of the schemes to save in the file. The *Save* button is enabled.

   Note: For the *Save* button to be enabled, ensure that you have specified a name of a file in the *Name* field and have at least one scheme defined and selected in the *Schemes* list box.
5. Click the *Save* button to save the selected schemes to the given file name.

   Note: If you save schemes to an existing file, the original contents of the file are removed and replaced by the newly saved schemes.

Comparing Schemes

You can compare the packages defined in two schemes.

1. Display the Power Routing Scheme Manager form by clicking the *Manager* button from either the Power Routing or Power Routing Options form.
2. Click the *Compare* tab in the Power Routing Scheme Manager form.
3. Select two schemes from the *Schemes* drop-down list boxes.

   The form indicates which packages are defined for each scheme.
4. Click the check boxes of the scheme packages you want to compare.

   You can compare Pad Ring, Core Ring, Block Ring, Stripes, Cell Rows, Pin To Trunk, and Vias packages.
5. Click the *Compare* button.

   The Power Routing Schemes Compare Result form displays. The form includes the following:

The first column lists the package options that have different values in the two packages. The Total number of Differences for the two packages also displays.

The second column lists the option data for the indicated scheme.

The third column lists the option data for the indicated scheme, so you can compare the two schemes side by side.

Using SKILL for Power Routing

To create power routing scheme using SKILL, use the functions in the following table.

SKILL Function	Function Description
`rteCreateBlockRingScheme`	Creates a scheme to add rings around the selected block instances or macros.
`rteCreateCellRowsScheme`	Creates a scheme to add straps along aligned pins of standard cells.
`rteCreateCoreRingScheme`	Creates a scheme to add rings around the core of a design that has pads, or around the entire design without pads.
`rteCreatePadRingScheme`	Creates a scheme to route pad rings for the given nets between pads on the periphery of the design.
`rteCreatePinToTrunkScheme`	Creates a scheme to add connections for power pins on macro blocks and power pads to existing rings and rails.
`rteCreateStripesScheme`	Creates a scheme to add net stripes at regular intervals.
`rteCreateViasScheme`	Creates a scheme to add vias for interlayer connections at intersections of rings, rails, stripes and straps, or between instance pins and stripes.

For more information on these commands, refer to "Space-based Router SKILL Functions" in the *Virtuoso Layout Suite SKILL Reference*.

4

Getting Started

This chapter describes how to get started with Virtuoso Space-based Router by exploring the Graphical User Interface (GUI). The Virtuoso Space-based Router GUIs aids designers in using different routing features. It provides access to the options for various steps in a routing flow and also provides a template of Tcl code that will run the steps so that the user can customize it as they become more familiar with the routing commands.

This section discusses the following.

- Virtuoso Space-based Router Commands
- Behavior of Virtuoso Space-based Router Toolbar Icons
- Synchronized Route Menu and the Toolbar
- Routing Scripts

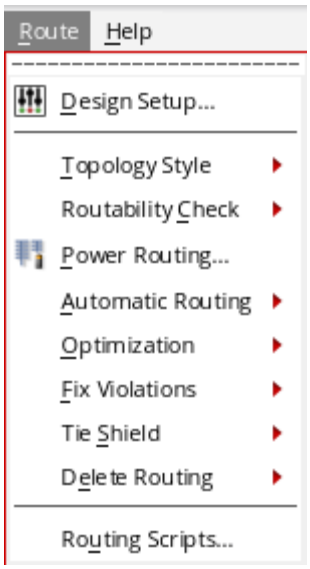
Virtuoso Space-based Router Commands

The Virtuoso Space-based Router commands are available in Virtuoso Layout Suite XL and higher tiers. You can access these commands by using either of the following methods:

- Route Menu
- Virtuoso Space-based Router Toolbar
- VSR Workspace

Route Menu

The *Route* menu is available by default in Layout Suite XL and higher tiers. It provides access to a set of commands that let you use the various available routing features. To view the routing commands, click the *Route* menu in the layout window. The *Route* commands are displayed as shown in the figure below.

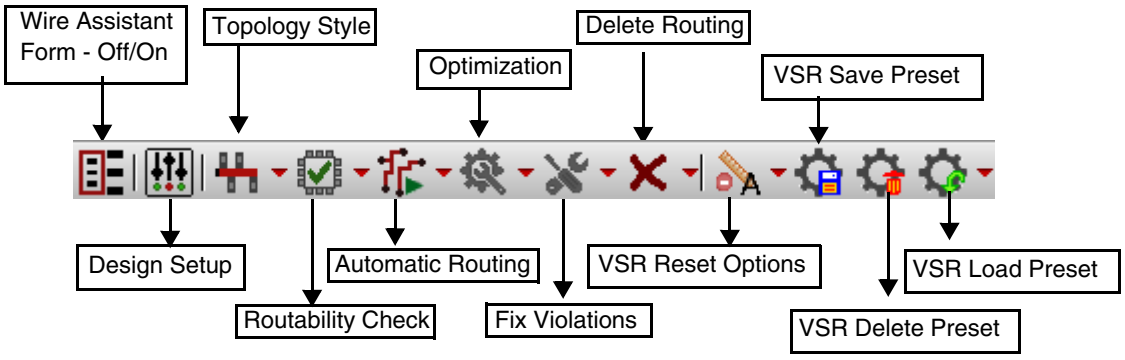


Note: The *Power Routing*, *Tie Shield*, and *Routing Scripts* commands are unavailable on the *Virtuoso Space-based Router* toolbar.

In addition to the menu commands, there is a Virtuoso Space-based Router toolbar that you can use to run the same set of commands. For more information about the commands and the toolbar, see Virtuoso Space-based Router Toolbar.

Virtuoso Space-based Router Toolbar

The *Virtuoso Space-based Router* toolbar is not available by default in the layout window. To display the toolbar, choose *Window – Toolbars – Virtuoso Space-based Router*. The *Virtuoso Space-based Router* toolbar is displayed. You can use this toolbar to access the routing features shown below.



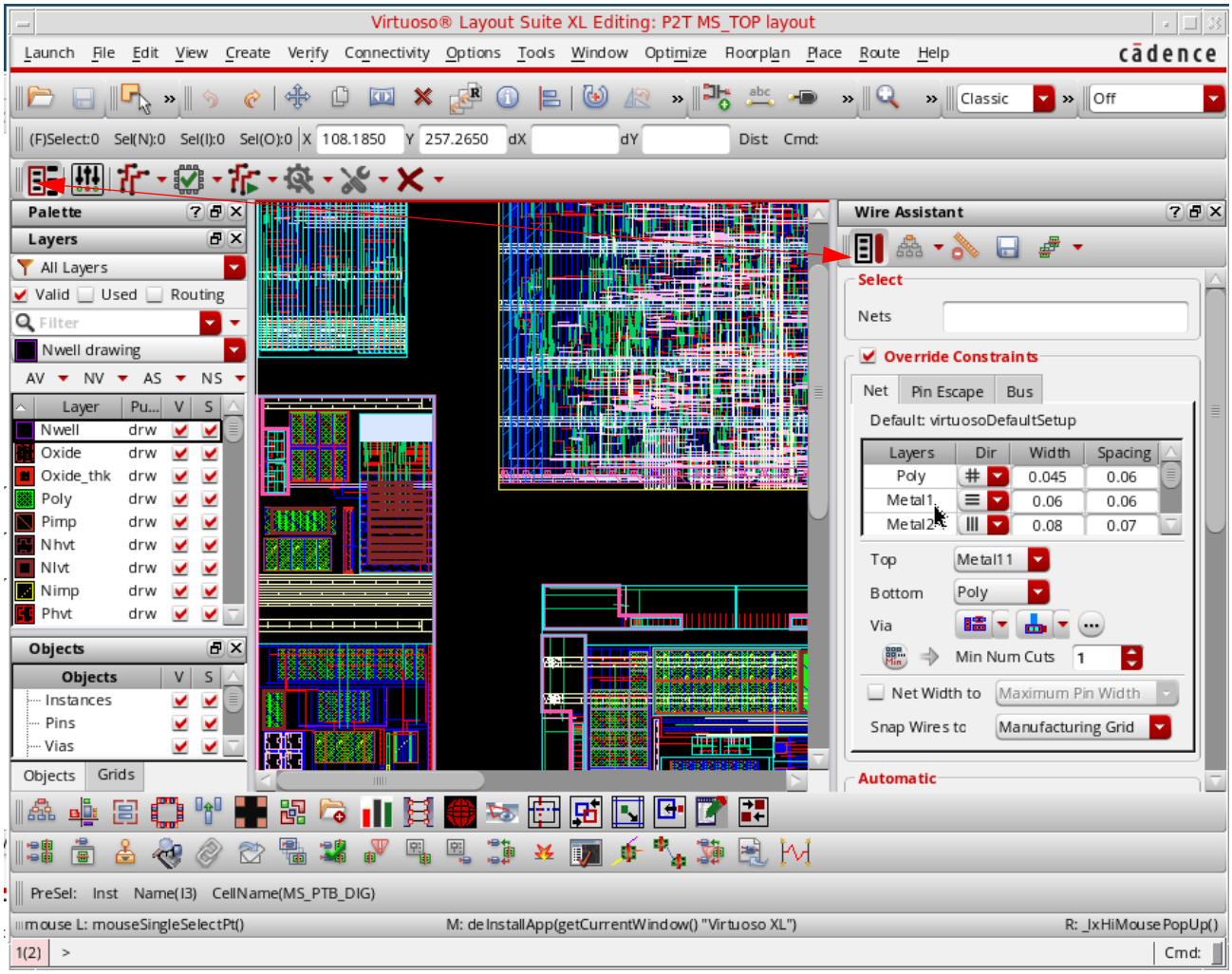
- Wire Assistant Form - Off/On
- Design Setup
- Route Flow
- Routability Check
- Automatic Routing
- Optimization
- Fix Violations
- Delete Routing
- VSR Reset Options
- VSR Save Preset
- VSR Delete Preset
- VSR Load Preset

The Virtuoso Space-based Router toolbar is by default available in the Routing Scripts.

Note: You can customize the *Virtuoso Space-based Router* toolbar by using the Toolbar Manager form. For more information about how to customize a toolbar, see Using Toolbar Manager.

Wire Assistant Form - Off/On

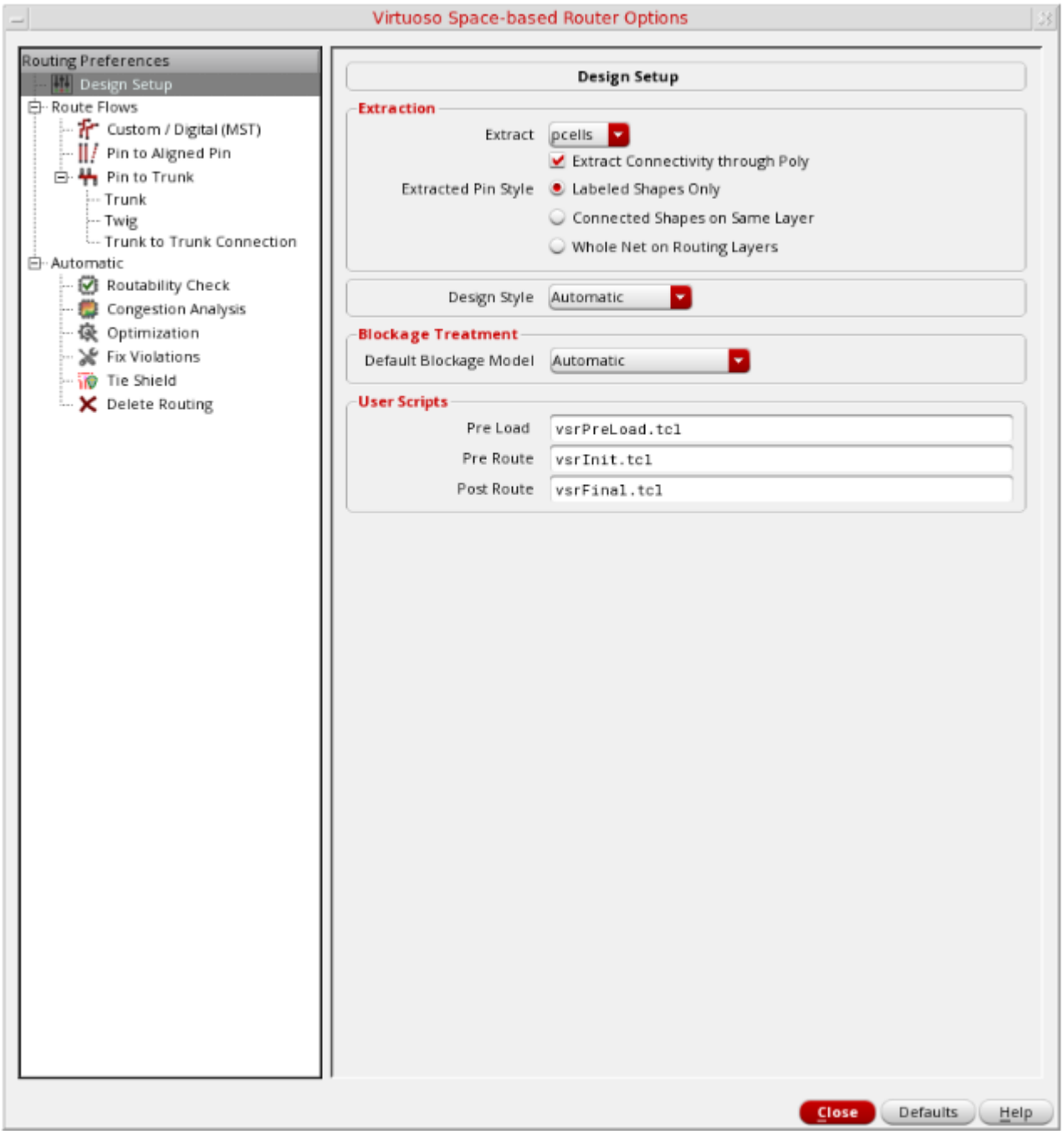
The *Wire Assistant Form* icon turns on or off the display of the Wire Assistant. By default, the Wire Assistant is not displayed. Click the icon to turn on the display.



Design Setup

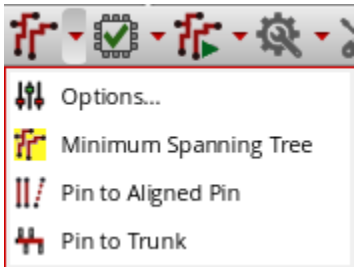
Click the *Design Setup* icon to view the Virtuoso Space-based Router Options form. Using the Virtuoso Space-based Router Options form you can access a set of hierarchically

arranged collection of routing features and options. For more information, see Using the Virtuoso Space-based Router Options Form.



Route Flow

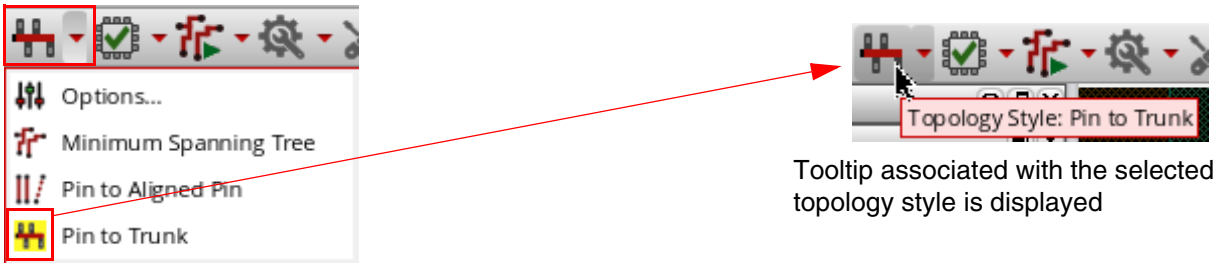
The *Route Flow* icon lets you select a routing style when running the automatic routing flow. Click the arrow next to the *Route Flow* icon to view the routing styles available on the drop-down menu associated with the icon.



- Click *Options* to view in the Virtuoso Space-based Router Options form all choices related to the routing style that is currently set.

  Note: The *Options* command on each menu provides quick access to a set of choices available for that feature in the Virtuoso Space-based Router Options form.

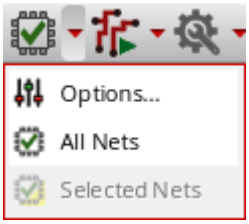
- Click a routing style, *Custom / Digital (MST)*, *Pin to Aligned Pin*, *Pin to Trunk*, or *Tree* to select a routing style for routing the design. You can choose only one routing style at any given time. A yellow highlight is displayed around the routing style that is selected, and the toolbar icon is updated to display the symbol and the tooltip associated with the selected routing style. For example in the following figure, the *Pin to Trunk* topology style is selected.



Tooltip associated with the selected topology style is displayed

Routability Check

Click the *Routability Check* icon to perform a set of routability checks for determining potential routing issues that can prevent the router from achieving optimal results. Click the arrow next to the *Routability Check* icon to view the options available on the drop-down menu associated with the icon.



Note: Except for *Wire Assistant Form - On/Off*, *Design Setup*, and *Topology Style* icons, the drop-down menu associated with all other icons is the same.

- *Options*

  Click *Options* to view in the Virtuoso Space-based Router Options form all choices related to the routability check feature.

- *All Nets*

  Lets you run routability checks on all nets in the current layout.

- *Selected Nets*

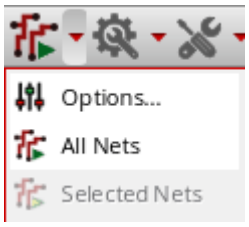
  Lets you run routability checks on the selected nets in the current layout. This menu item is enabled only if at least one net is selected in the layout. This functionality is the same as the *Selected* button in the *Automatic* section of Wire Assistant. To select one or more nets, use the Navigator Assistant. As soon as a net is selected in the Navigator Assistant, the *Selected Nets* menu item is enabled.

  Note: The router performs routability checks on all nets or the selected nets according to the override constraints (if the *Override Constraints* check box is selected in Wire Assistant) and the routability check options specified in the Virtuoso Space-based Router Options form.

Automatic Routing

Click the *Automatic Routing* icon to perform routing in the layout design. By default, automatic routing is performed on all nets in the layout design. Click the arrow next to the

Automatic Routing icon to view the options available on the drop-down menu associated with the icon.

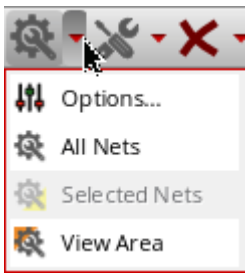


For information about *Options*, *All Nets*, and *Selected Nets*, see Routability CheckRoutability Check.

The router can operate on the current selected nets in the Navigator Assistant. The Navigator now supports user-defined sets that makes it easier to again select arbitrary sets of nets. For more information, see The Navigator Assistant.

Optimization

Click the *Optimization* icon to optimize the routing results in the layout design. By default, optimization is done on all nets in the layout design. Click the arrow next to the *Optimization* icon to view the options available on the drop-down menu associated with the icon.



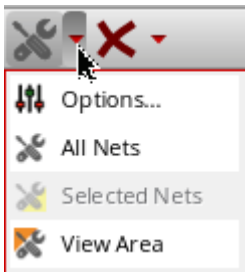
For information about *Options*, *All Nets*, and *Selected Nets*, see Routability CheckRoutability Check.

- *View Area*

  Lets you perform various wire optimization post processes on the current viewing area.

Fix Violations

Click the *Fix Violations* icon to perform all spacing-related checks including checks on merged shapes and fix violations for all nets in the layout design. Click the arrow next to the *Fix Violations* icon to view the options available on the drop-down menu associated with the icon.



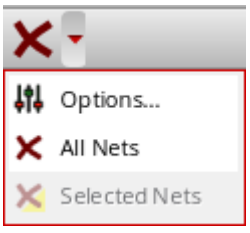
For information about *Options*, *All Nets*, and *Selected Nets*, see Routability CheckRoutability Check.

- *View Area*

  Lets you fix violations on the current viewing area.

Delete Routing

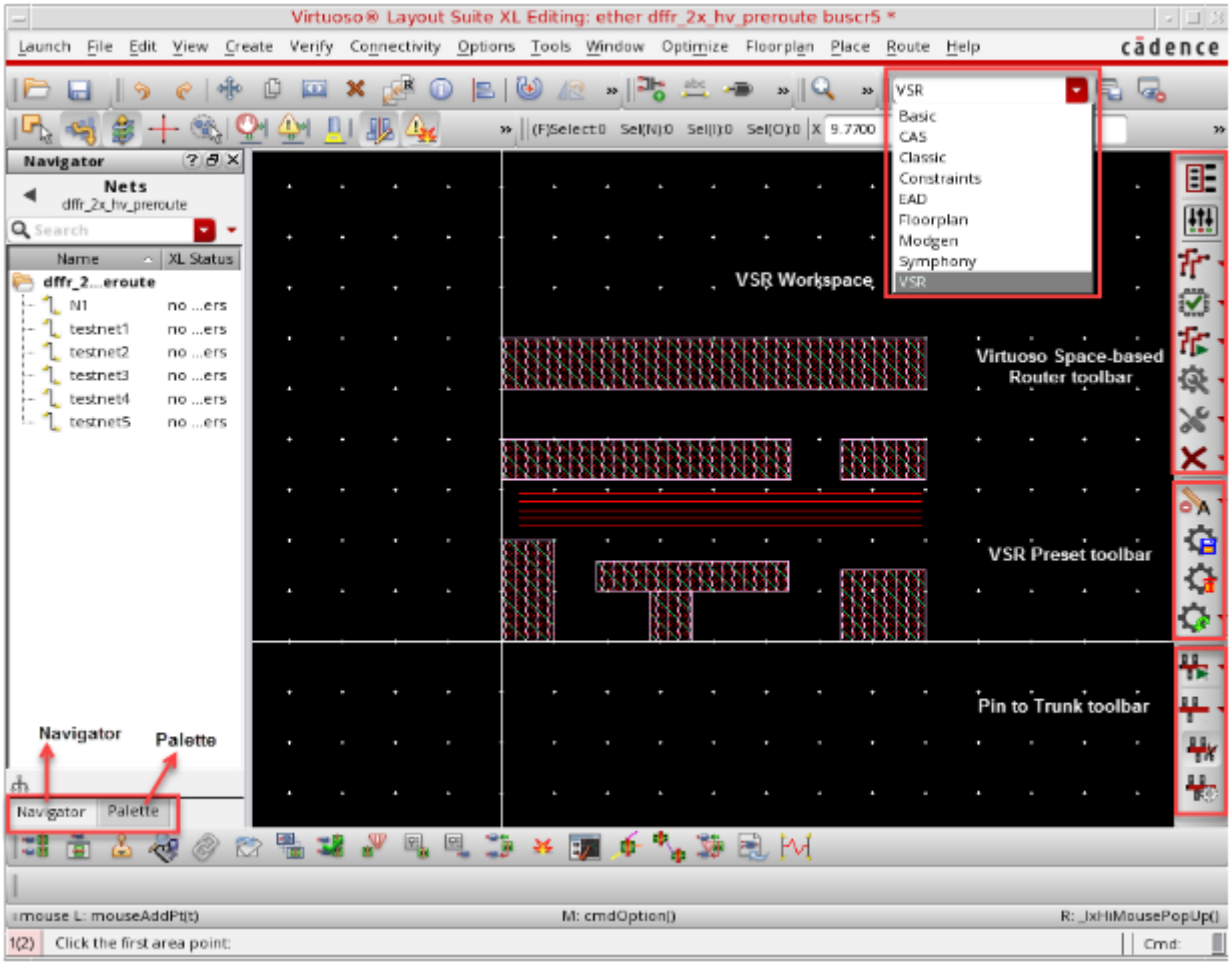
Click the *Delete Routing* icon to delete all the existing routes in the design. However, the routes in the LOCKED constraint are not deleted. Click the arrow next to the *Delete Routing* icon to view the options available on the drop-down menu associated with the icon.



For information about *Options*, *All Nets*, and *Selected Nets*, see Routability CheckRoutability Check.

VSR Workspace

Virtuoso Space-based Router has its own workspace in Layout Suite XL and higher tiers. The VSR workspace shows the canvas, the toolbars (*Virtuoso Space-based Router*, VSR Preset, and *Pin to Trunk*). These toolbars are available by default, vertically docked to right of the layout window. The VSR workspace also shows the Navigator and Palette assistants.



Behavior of Virtuoso Space-based Router Toolbar Icons

In the Virtuoso Space-based Router toolbar, the behavior of the toolbar icon is based on the last action of either *All Nets* or *Selected Nets*. By default, the selection of all nets is considered and the associated icon is displayed on the toolbar. This means that if you click the toolbar icon, it automatically performs the action of the routing feature on all nets in the design.

If you want to perform the routing feature only on selected nets, select one or more nets from the Navigator Assistant. The *Selected Nets* menu item is now enabled in the drop-down menu. When you click the *Selected Nets* menu item, the corresponding routing feature is performed on selected nets. In addition, the toolbar icon of the corresponding routing feature automatically changes to the *Selected Nets* icon and the tooltip on the icon informs you that the last action of the routing feature was on the selected nets.

For example, the *Automatic Routing* icon in Fig A shows that the default action for *Automatic Routing* is *All Nets*. The Fig B displays the enabled *Selected Nets* option after you have selected one or more nets from the Navigator Assistant. In Fig C, the tooltip of the icon changes from *Automatic Routing - All Nets* to *Automatic Routing - Selected Nets* after automatic routing is completed on the selected nets. In addition, the *Automatic Routing* icon on the toolbar changes to the *Selected Nets* icon.

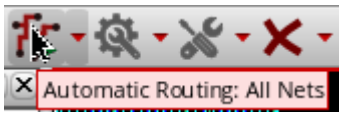


Fig A

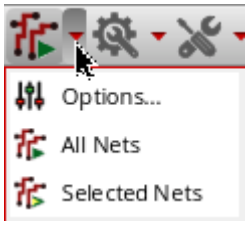


Fig B

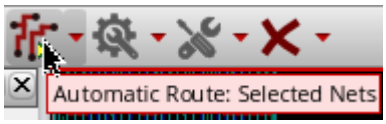
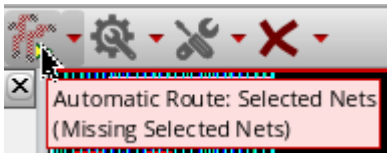
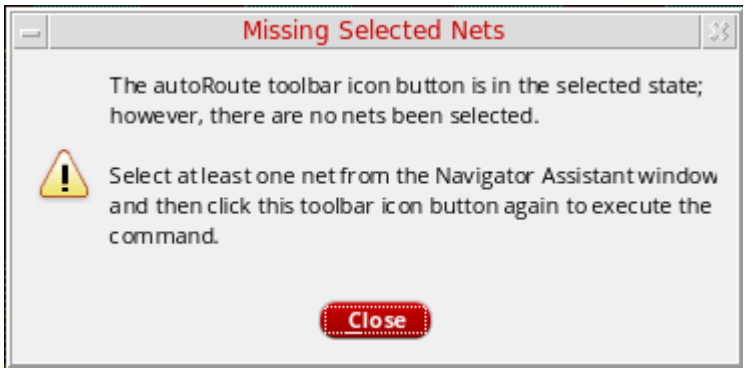


Fig C

If you now deselect all nets in the Navigator Assistant, the toolbar icon is automatically disabled, as shown in the following figure. In addition, the tooltip displayed on the icon is updated to indicate that the last routing action was performed on the selected nets, but currently there is no net selected in the Navigator Assistant.



Now, if you click the disabled Automatic Routing icon, a warning message is displayed stating that you are trying to perform the routing operation without any nets having been selected in the Navigator Assistant.

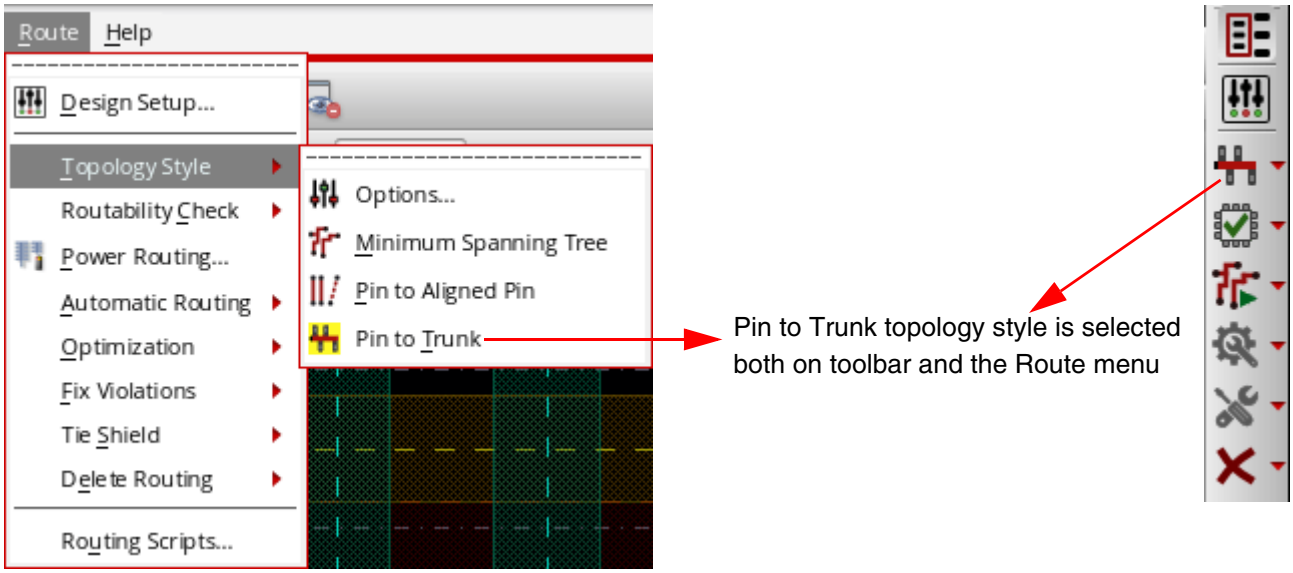


Note: Except *Wire Assistant Form*, *Design Setup*, and *Topology Style* icons, all other toolbar icons have the same behavior for All Nets and selected Nets menu item in the drop-down menu.

Synchronized Route Menu and the Toolbar

The commands on the *Route* menu and the Virtuoso Space-based Router toolbar are synchronized at any given time. This means that if there are no nets selected in the Navigator Assistant, then the *Selected Nets* option in both the Virtuoso Space-based Router toolbar and the *Route* menu is disabled. This option is enabled only if there is at least one net selected in the Navigator Assistant.

Similarly, for the *Topology Style* option, the icon of the topology style is synchronized to reflect the topology style that is currently selected. For example, when the Virtuoso Space-based Router toolbar has Pin to Trunk selected as the topology style for routing, the *Topology Style* submenu displays *Pin to Trunk* as the active topology style, as shown in the following figure.



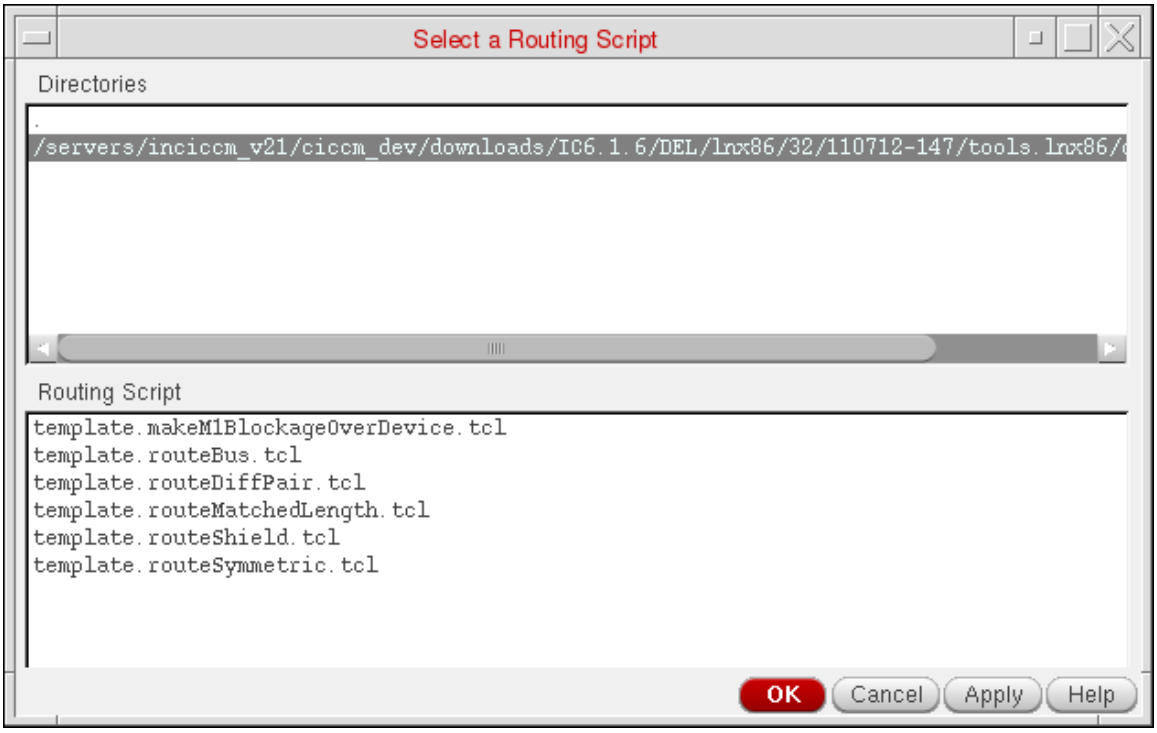
Routing Scripts

An existing routing flow can be captured in the form of Tcl routing commands within a Tcl script. You can run this Tcl script to view the routing results. To execute a Tcl script, perform the following steps.

1. Choose the *Route - Routing Scripts* menu item. Alternatively, with the *Route* menu displayed, you can press the *R* key on your keyboard to display the *Routing Scripts* form. The *Routing Script* form is a routing script specification and execute form.

2. In the *Routing Scripts* form, specify the Tcl script name that you want to execute. You can select an existing Tcl script by clicking the *Browse* button. When you click the *Browse* button, the *Select a Routing Script* form is displayed. This form displays a list of directories that are set using the `routingScriptingDir` environment variable.

   Note: The default directory is changed based on the `.cdsenv` setup that you have.



   a. Select a directory from the *Directories* box. All the `.tcl` files that exists in the selected directory are displayed in the *Routing Script* box.

   b. Select a `.tcl` script from the *Routing Script* box.

   c. Click *OK*. The selected script is displayed in the *Script* field and in the box below the field.

   The *Routing Scripts* form supports a search path mechanism. After entering the script name and clicking *OK* or Apply, it will look for the script in the look-up hierarchy. The look-up will stop after finding the script. The look-up hierarchy is as follows:

   - Current Directory
   - Current Directory /rde
   - Current Directory /.cadence/dfII/rde
   - Home Directory/rde

- $INSTALLDIR/samples/rde

When setting the INSTALLDIR environment variable, the `INSTALLDIR` environment variable must be defined as `<Virtuoso software installation directory>/tools/dfII`. The provided sample routing scripts are located in `<path to the Virtuoso software installation>/tools/dfII/samples/rde` directory. There are six sample scripts which are provided:

- template.routeMatchedLength.tcl
- template.makeM1BlockageOverDevice.tcl
- template.routeBus.tcl
- template.routeShield.tcl
- template.routeDiffPair.tcl
- template.routeSymmetric.tcl.

The script names will not appear in the GUI. However, if the `$INSTALLDIR` is defined then by typing the script name into the *Script* field it will be found, executed, and the full path and script name will appear in the box below the *Script* field.

3. Click *OK*. The routing commands contained within the specified routing script will be executed.

The *Routing Scripts* command also keep up with the values specified in the *Override Constraints* section of the Wire Assistant. For example, in the Wire Assistant, if you have specified the *Bottom* layer as `M1`, *Top* layer as `M2`, and have specified the *Min Num Cuts* value to `2`, and then run the customized Tcl script using *Route – Routing Scripts*, the routing layers and `minNumCuts` values are overriden.

5

Working with VSR Presets

A Virtuoso Space-based Router (VSR) Preset is a set of predefined routing options and user override constraint values, which can be saved to a file. For more information, see Preset File.

This chapter covers the following topics.

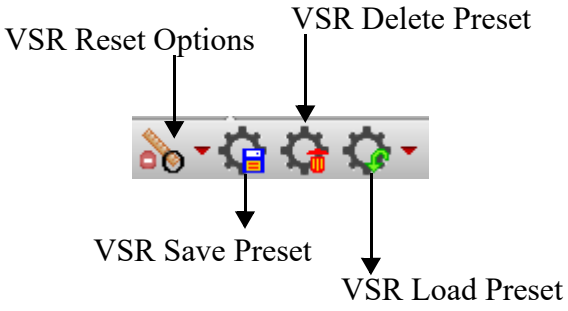
- Virtuoso Space-based Router Preset Toolbar
- Saving a Preset File
- Deleting a Preset File
- Loading a Preset File

Virtuoso Space-based Router Preset Toolbar

A Virtuoso Space-based Router (VSR) Preset is a set of predefined routing options and user override constraint values, which can be saved to a file. For more information, see Preset File.

The VSR Preset toolbar lets you save, load, restore, and delete routing-related settings and user override constraint values for all Automatic routing features. Using the VSR Preset toolbar, you can quickly switch between routing options with a single mouse click.

The VSR Preset toolbar is part of the Virtuoso Space-based Router toolbar. To display the toolbar, choose *Window – Toolbars – Virtuoso Space-based Router*. The figure below shows the various options available on the VSR Preset toolbar.



You can also access the VSR Preset options from the Wire Assistant toolbar, as shown below.

The VSR Preset toolbar consists of four icons.

- VSR Reset Options
- VSR Save Preset
- VSR Delete Preset
- VSR Load Preset

The state of the VSR Preset icons on the *Virtuoso Space-based Router* toolbar and Wire Assistant toolbar are synchronized at any given time. Because Wire Assistant does not have enough space to display all the user-defined presets as toolbar icons, the presets are displayed in the *VSR Load Preset* drop-down menu.

Video

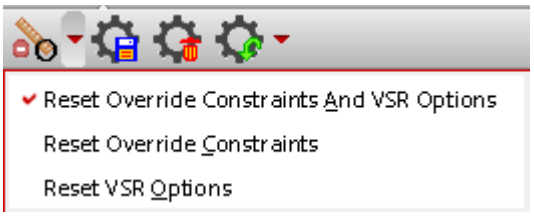
For a short demonstration on how to use VSR Preset features, see Introductio to Presets.

VSR Reset Options

VSR Reset Options lets you reset the override constraints that have been changed in the *Net* and *Pin Escape* tabs of the *Override Constraints* section of the Wire Assistant. It also lets you reset the VSR options that have been changed either in the Wire Assistant or in the Virtuoso Space-based Router Options form. The wire editing options that appear in the *Interactive* section of the Wire Assistant can also be reset.

Click the arrow next to the *VSR Reset Options* icon to select one of the three reset modes: *Reset Override Constraints And VSR Options*, *Reset Override Constraints*, or *Reset*

VSR Options. By default, the reset mode is *Reset Override Constraints And VSR Options*.



The icon corresponding to the mode selected last stays displayed on the toolbar. If you want to run a different reset mode, select it from the drop-down list. When you select a reset mode, the icon on the toolbar is automatically updated to reflect the change in mode.

Icon	Option Name	Description
	*Reset Override Constraints And VSR Options*	Resets the override values in both *Override Constraints* and *Automatic* sections of the Wire Assistant.
	*Reset Override Constraints*	Resets the override values in the *Override Constraints* section in the Wire Assistant.
	*Reset VSR Options*	Resets the override values in the *Automatic* section in the Wire Assistant.

VSR Save Preset

The *VSR Save Preset* icon lets you save the modified override constraint values and the automatic routing and interactive routing environment variables to a preset file. The VSR Save Preset form is displayed when you click this icon. For more information, see Saving a Preset File and VSR Save Preset Form.

Related Topics

Saving a Preset File

VSR Save Preset Form

Preset File

VSR Delete Preset

The *VSR Delete Preset* icon lets you delete a preset file. The VSR Delete Preset form is displayed when you click this icon. For more information, see Deleting a Preset File and the VSR Delete Preset Form.

Related Topics

Deleting a Preset File

VSR Delete Preset Form

Preset File

VSR Load Preset

The *VSR Load Preset* icon lets you load a preset. Clicking the icon lets you do the following:

- Refresh the presets found in the preset search paths.
- Reload a previously loaded preset file, if there is one.

You can click the arrow next to the *VSR Load Preset* icon to load a different preset file from the drop-down menu. For more information, see Loading a Preset File.

Related Topics

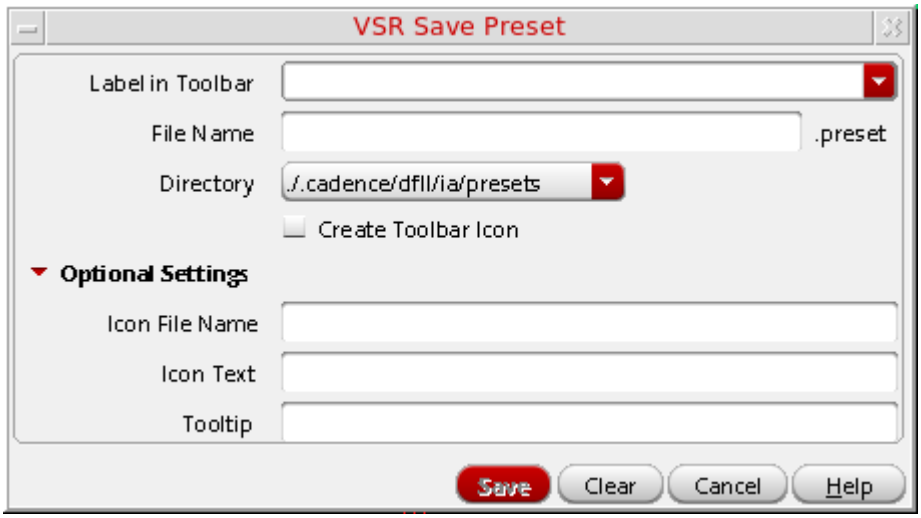
Loading a Preset File

Preset File

Saving a Preset File

Click the *VSR Save Preset* icon on the VSR Preset toolbar or on the Wire Assistant toolbar. The VSR Save Preset form displays. In this form, you can specify the name and location of

the preset file to which you want to save the current override constraint values and the values specified for various automatic routing and interactive routing environment variables.



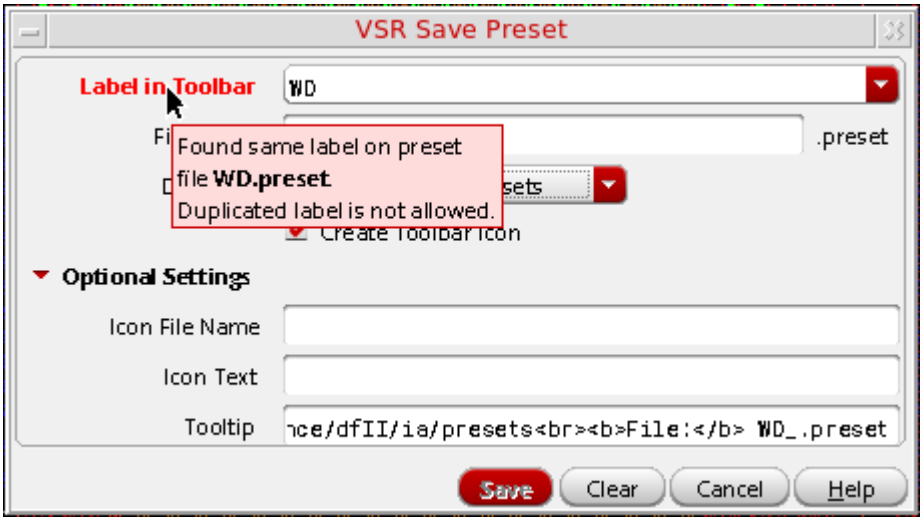
To save the preset file, do the following:

1. Specify a label for the preset file in the *Label in Toolbar* field. You will notice that the *File Name* field is automatically populated with the value that you specify in this field.

   If you also select the *Create Toolbar Icon* check box, an icon is added to the toolbar, labeled with the value that you specified. If you do not select the check box, the label appears as an option name on the VSR Load Preset drop-down menu. This label helps to identify a preset file.

   You cannot create a preset that has the label name same as an existing preset file in the same directory path. If the specified existing preset label already exists, the color of the *Label in Toolbar* field name changes to red and an appropriate tooltip is displayed, as

shown in the following figure. Because duplication of the preset label is not allowed, the *Save* button is also disabled.



2. Specify the name with which you want to save the preset in the *File Name* field. You can also retain the value that was automatically added to this field, which is the same as the label that you specified. The preset file is saved with the extension `.preset`.

   The name of the preset file is automatically generated based on the value specified in the *Label in Toolbar* field. For example, if you specify the label name as `Auto Route`, then the filename is automatically updated to `AutoRoute` in the *File Name* field.

   Similarly, when a preset file is selected from the *Label in Toolbar* drop-down list, the filename of the preset file is automatically updated in the *File Name* field.

   Note: Both *Label in Toolbar* and *File Name* are mandatory fields. The *Save* button at the bottom of the form is disabled if any of these fields is empty.

3. Select the location to which you want to save the preset file from the *Directory* drop-down list. You can select one of the following locations:

   - Current Virtuoso Invoking directory

     If the entries are specific to the current layout design, save the preset file to the current directory from where Virtuoso is run. The path from where Virtuoso is run is `./.cadence/dfII/ia/presets`.

   - $HOME

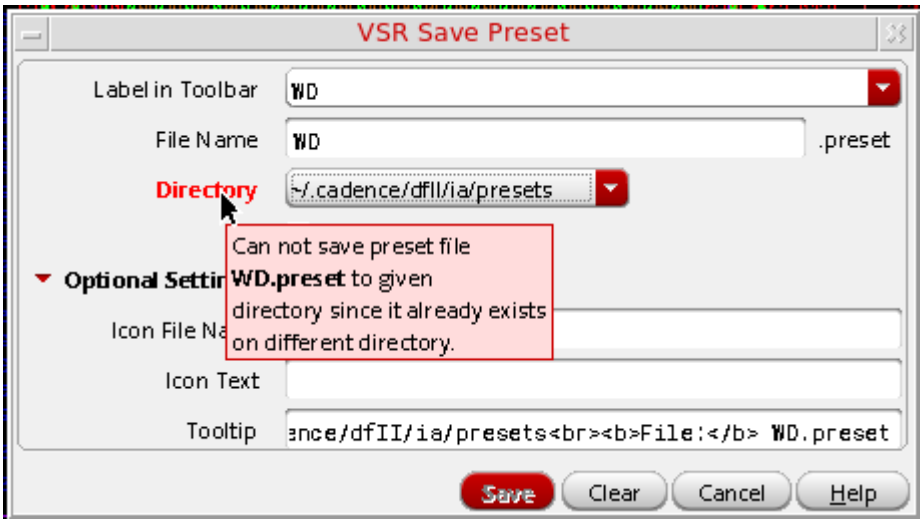
     If the preset entries are applicable to other layout designs as well, save the preset file to the HOME directory. The path for the HOME directory is `~/.cadence/dfII/ia/presets`.

- $CDS_PROJECT

  If the preset entries are general enough and can be shared with others in a design team, save the preset file to the directory to which the `CDS_PROJECT` shell environment variable points. The directory path is `$CDS_PROJECT/dfII/ia/presets`.

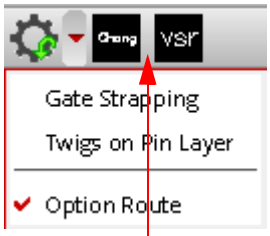
  Note: `CDS_PROJECT` must be specified so that the directory can be edited and can be accessed from the directory from where Virtuoso is run. If not, the *Directory* drop-down list displays only current Virtuoso invoking and HOME directories.

You cannot save a preset file to the specified directory location if the same preset file already exists on the different directory location. In this case, the color of the *Directory* field name changes to red and an appropriate tooltip is displayed, as shown in the following figure.



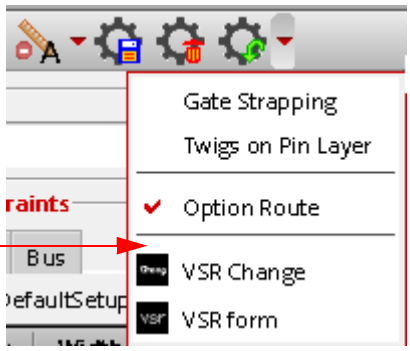
4. Select the *Create Toolbar Icon* check box to add the preset as an icon on the VSR Preset toolbar. If the *Create Toolbar Icon* check box is deselected, the preset is added as a menu option in the *VSR Load Preset* drop-down menu.

   In the Wire Assistant toolbar, all presets for which the *Create Toolbar Icon* check box is selected are automatically placed at the bottom of the *VSR Load Preset* drop-down menu and the icon for each preset is placed before the menu option. The presets for which the *Create Toolbar Icon* is deselected are placed in the middle of the drop-down menu, with only the option name.



VSR Presets added as icons on the VSR toolbar

Wire Assistant toolbar - VSR Presets added as menu options in the VSR Load Preset drop-down menu



5. Click *Optional Settings.* This enables the optional settings that are avialable in the form.

6. Specify the name of the icon file that you want to use for a particular preset in the *Icon File Name* field. The icon file name should be available in the `<cds-install-dir>/share/cdssetup/icons/24x24` directory or in the `icons/24x24` directory on any Cadence File System hierarchical lookup path as follows.

   ```
   ./.cadence/icons/24x24
   ~/.cadence/icons/24x24
   ```

   It is recommended to place the icon on CDS_PROJECT shell environment variable if it is shared among various users.

   ```
   $CDS_PROJECT/icons/24x24
   ```

   This option is applicable only if the *Create Toolbar Icon* check box is selected. If the specified filename is unavailable, a warning message is displayed in CIW, as shown in the following figure.

   ```
   ============================================================================================
   Virtuoso Space Router user preset file ./.cadence/dfII/ia/presets/OptionRoute.preset is saved
   Virtuoso Space Router user preset file ./.cadence/dfII/ia/presets/OptionRoute.preset is saved
   *WARNING* Unable to find the icon "abc.png"
   *WARNING* Make sure icon file abc.png is on the Cadence Search Paths.
   ```

   Note: The icon file should be a `.png` or a `.jpg` file.

7. Specify text in the *Icon Text* field. The specified text gets displayed on the VSR Preset toolbar icon. The icon created is of `24x24` pixels.

   If the specified text for the icon is one word, VSR Preset generates an icon with a single line text of maximum five characters. If the specified text has more than one word, the VSR Preset generates an icon with text in two lines. The icon text consists of maximum of five characters in each line.

If the *Icon Text* field is left blank, VSR Preset automatically uses the value from the *Label in Toolbar* field. The *Icon Text* field is ignored if the *Create Toolbar Icon* check box is deselected or if a valid *Icon File Name* is specified.

In addition, if an already existing icon text is specified, the *Icon Text* field is automatically set to empty once the value of either *Label in Toolbar* or *File Name* field is changed. The original value of the icon text cannot be restored if the value of the *File Name* field is changed after the value of the *Icon Text* field is set to empty. The only way to restore the icon text value is either by changing to the original value in the *Label in Toolbar* field or by again selecting the preset from the *Label in Toolbar* combo box.

8. Specify the text that you want to appear as a tooltip when the mouse pointer is placed on the preset icon in the *Tooltip* field. By default, the *Tooltip* field is prepopulated with three values: *Label*, *Directory*, and *File* names, as shown in the following figure.

You can either delete these values and add your own tooltip or append the tooltip to the existing string.

When the value of *Label in Toolbar*, *File Name*, and *Directory* fields are modified, the tooltip for the preset icon is automatically updated.

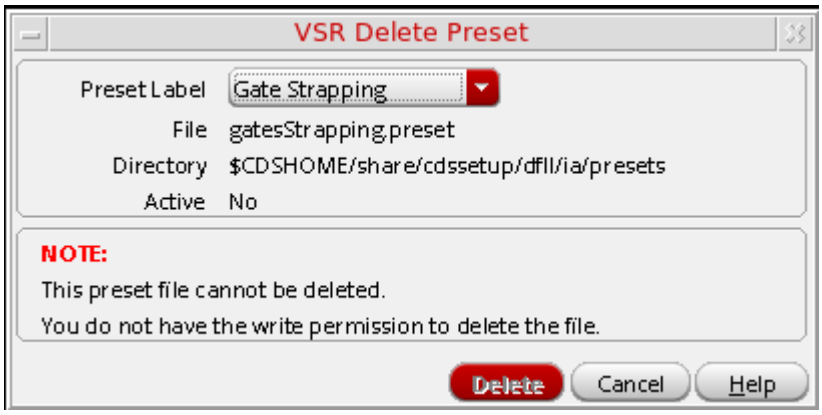
Note: The tooltip is not displayed when the mouse pointer is placed on the menu item in the *VSR Load Preset* drop-down menu.

9. Click *Save* to save the current settings to a preset file for future use.

Note: To clear all the fields in the VSR Save Preset form, click *Clear*.

Deleting a Preset File

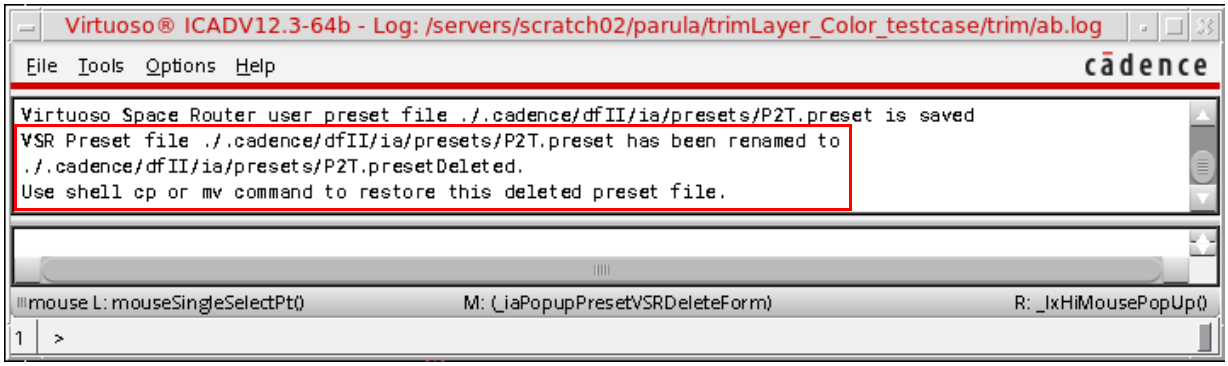
To display the VSR Delete Preset form, click the *VSR Delete Preset* icon on the VSR Preset toolbar or on the Wire Assistant toolbar.



To delete the preset file, do the following:

1. Select the preset file that you want to delete from the *Preset Label* drop-down list. This drop-down lists all preset files that are currently loaded.
2. Click *Delete*.

   When you click *Delete*, the preset file is really not deleted but is only renamed and saved as `<file-name>.presetDeleted`. As a result, you can restore and use the file later, if required. A message indicating the same also appears in CIW.



   The menu option and toolbar icon corresponding to a deleted preset is removed from the VSR Preset toolbar.

   Note: To prevent you from accidentally deleting another preset file, the VSR Delete Preset form automatically closes when the *Delete* button in clicked.

Loading a Preset File

To load a preset file, VSR Preset feature follows the Cadence Search Function Specificiation, except for the `@LIBRARY` entry in the `setup.loc` file. This means that you can copy the `setup.loc` file from `${CDS_INST_DIR}/share/cdssetup` to the directory from where Virtuoso is invoked. You can then edit the file and invoke Virtuoso.

The original `setup.loc` file has the following entries:

```
.
@LIBRARY        look in the design libraries for the file
$CDS_WORKAREA   user workarea if defined
$CDS_SEARCHDIR  this is set by various tools during tool startup
$HOME
$CDS_PROJECT    project storage area, ignored if not defined
$CDS_SITE       Site Setup Information. Default is $CDS_INST_DIR/share/local
$(compute:THIS_TOOL_INST_ROOT)/share    Cadence Default Setup Information
```

You can change the entries in the `setup.loc` file as follows:

```
.
@LIBRARY
$CDS_WORKAREA
$CDS_SEARCHDIR
$HOME
$TSMC       Add as a user defined search directory
$CDS_PROJECT
$CDS_SITE
$(compute:THIS_TOOL_INST_ROOT)/share
```

For every entry, except for `$(compute:THIS_TOOL_INST_ROOT)/share`) in the user's customized `setup.loc` file, the preset reads and writes the file to its `.cadence/dfII/ia/presets` directory. For example, `$CDS_SITE` shell environment points to `./myCDS`, then the preset reads and writes the preset files from `./myCDS/.cadence/dfII/ia/presets`.

The VSR Preset feature then searches and loads the preset file in the following order.

```
./.cadence/dfII/ia/presets
$CDS_WORKAREA/.cadence/dfII/ia/presets
$CDS_SEARCHDIR/.cadence/dfII/ia/presets
$HOME/.cadence/dfII/ia/presets
$TSMC/.cadence/dfII/ia/presets
$CDS_PROJECT/.cadence/dfII/ia/presets
$CDS_SITE/.cadence/dfII/ia/presets
```

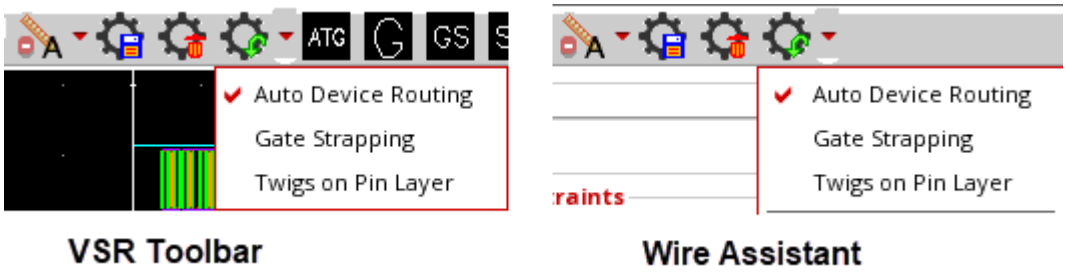
where `$TSMC` is a shell environment variable defined as follows:

```
setenv TSMC ~/TSMC
```

This means that if the same preset label is found in `.cadence/dfII/ia/presets` and `<HOME>/.cadence/dfII/ia/presets` directories, then the one in the `.cadence/dfII/ia/presets` directory is used. If the same toolbar label is found in two different files in the same directory, then the file that is read first is used. The VSR Preset feature also issues a warning if the same preset label is found in different directories.

When you click the *VSR Load Preset* icon, all the preset files found in the preset search paths are loaded and the list of available VSR presets is refreshed in the *VSR Load Preset* drop-down menu. Also, the last preset file, if any, is reloaded.

The VSR Preset feature has two built-in Pin to Trunk preset files, `gatesStrapping.preset` and `twigOnPinLayer.preset`. These preset files are located in `<CDS_INSTALL_DIR>/share/cdssetup/dfII/ia/presets` directory. The two preset files are available in the *VSR Load Preset* drop-down menu, as shown in the following figure.



VSR Toolbar **Wire Assistant**

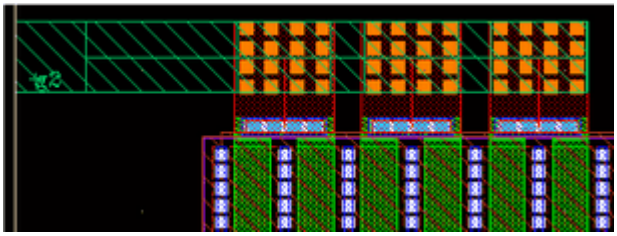
- Auto Device Routing

  Click *Auto Device Routing* to load the automatic device routing script and enable the *Tree Route* flow in the Wire Assistant. For more information, see Specifying the Tree Route Options.

- Gate Strapping

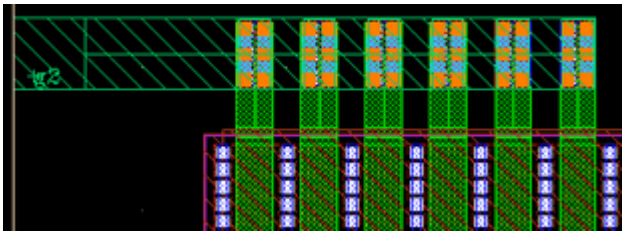
  Click *Gate Strapping* to load the gate strapping preset script with the `minWidth` strap width and enable the options in the *Pin Strapping* group box in the *Twig* subform of the

Virtuoso Space-based Router Options form. For more information on the options that you can specify for Pin Strapping, see Specifying Pin Strapping Options.

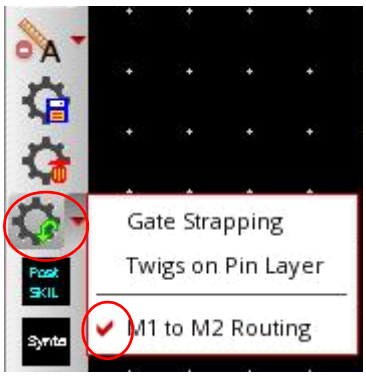


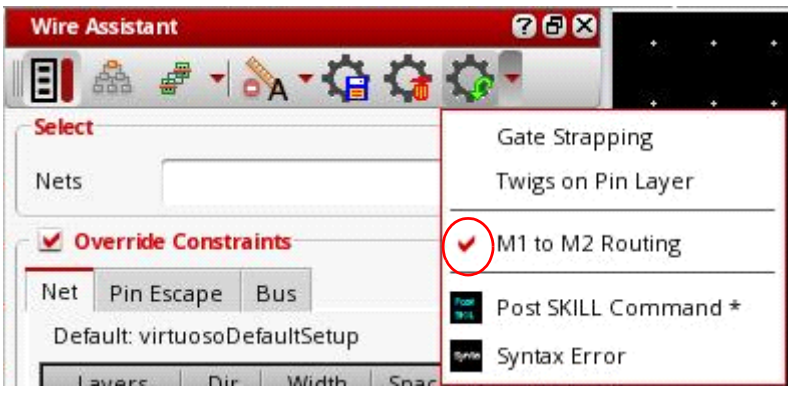
- Twigs on Pin Layer

  Click *Twigs on Pin Layer* to load the twigs on pin layer preset script, which, by default selects the *Prefer Pin Layer* menu item in the *Layer* drop-down list of the *Gate Twig* and *Non Gate Twig* group box in the *Twig* subform. The twigs use the same layer as pins. For more information, see Specifying Gate Twig and Non-Gate Twig Options and Specifying Via over Pin Coverage Options.



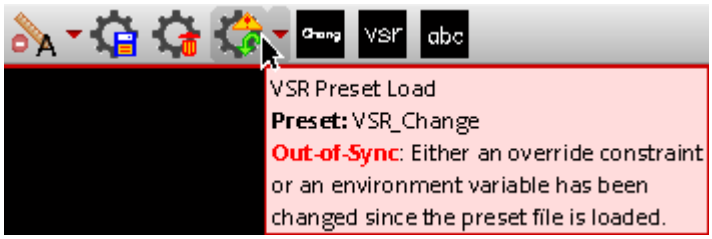
The presets are alphabetically placed on the *VSR Load Preset* drop-down menu. After a preset file is loaded, it automatically becomes the active preset. The active preset is indicated by a tick mark (✔) beside the preset label in the drop-down menu or by a slight depression of the icon, as shown in the following figure.





Alternatively, when you place mouse pointer on the *VSR Load Preset* icon, the tooltip lists the currently active preset, as shown in the following figure.

The active preset can become out of sync when you change any override constraint value or reset any environment variable in the Wire Assistant, Virtuoso Space-based Router Options form, or Via Configuration form. When the preset is out of sync, an exclamation mark (!) appears on the *VSR Load Preset* icon and the tooltip displayed on the icon is updated to indicate the *Out-of-Sync* state, as shown in the following figure.



Once the preset is out of sync, you cannot synchronize the preset by changing the override constraint or environment variable back to its original value. The only way to synchronize the preset is by reloading the preset file.

If the preset file has at least one override constraint value, then *VSR Load Preset* automatically clears all caches before processing the preset entries. This is required so that the override constraint values can be restored to a state that is just the same as when the preset is saved. However, if the preset file does not have any override constraint value, caches are not cleared. This ensures that the *VSR Load Preset* feature does not disturb the specified override constraint values after the preset is loaded. You can force *VSR Load Preset* to clear all caches by adding the following entry in the preset file:

```
clearCst nil nil nil
```

Execution Mode of Preset File

Because the preset is an ASCII file, you can edit the content of the preset file using any text editor. While editing a preset file, errors might get introduced. Therefore, before processing a preset file, the *VSR Load Preset* feature lets you check the syntax, value, and context of the entries. If the entries have errors, depending upon the execution mode, the *VSR Load Preset* behaves differently.

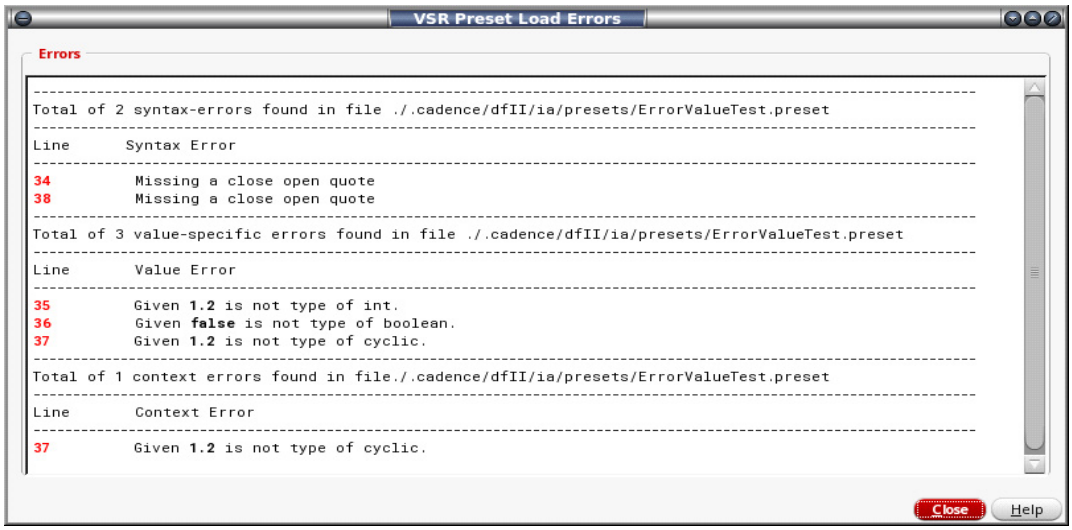
- Setting mode

Skips any preset entry that has an error and processes and loads all the entries that are error free.

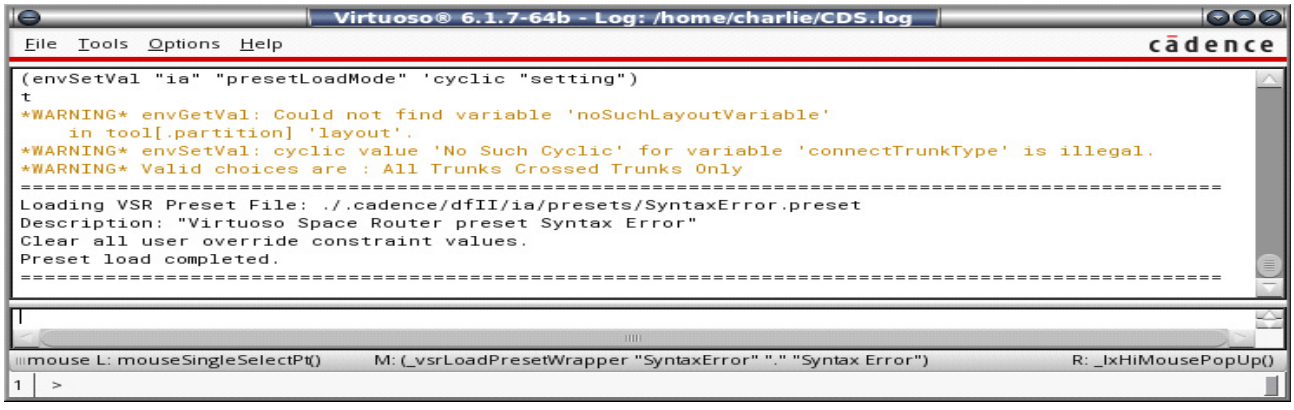
- Checking mode

  Processes the entries only if all the entries in the preset file are error free. If even one entry has an error, none of the entries are processed, even if all other entries are error free. Therefore, this mode is used to check the correctness of the preset file after the preset file is modified. This is the default mode.

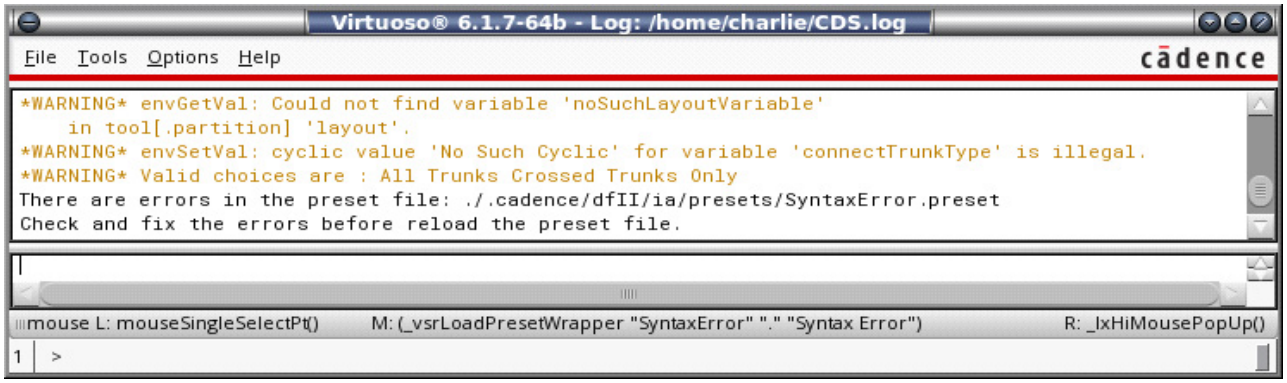
In both execution modes, the VSR Preset Load Errors dialog box is displayed to inform you about the errors that have been found. The errors are displayed along with the line number to indicate where the error has occurred. The following figure shows a preset file that has six errors. It also displays an error free entry at the end.



If the mode is `setting`, the last error free entry is processed, as shown in the following figure.



However, if the mode is `checking`, no entry is processed even though the last entry is error free. This can be seen from the output messages displayed in CIW.



Environment Variable: presetLoadMode

Errors in Preset File Entries

The four kinds of errors that can exist in preset entries are given in the table below.

Error Type	Description
`Syntax Error`	■ Missing a quote. ■ Missing parenthesis.
`Value Error`	■ The type of given value does not match with the type of the entry. ■ The given value violates the constraint lookup value. ■ The layer name does not exist in the technology file of the current layout design.
`Duplicated Error`	If two or more entries have the same override constraint identity or environment variable name, they are considered as duplicates. VSR Preset lists them as duplicate errors in the VSR Preset Load Error dialog box.
`Context Error`	■ The variable is not listed as one of the tool environment variable. ■ The variable is listed as an integer type, but has been specified as a string type.

6

Routing Your Design

This chapter describes the various routing options made available by Virtuoso Space-based Router.

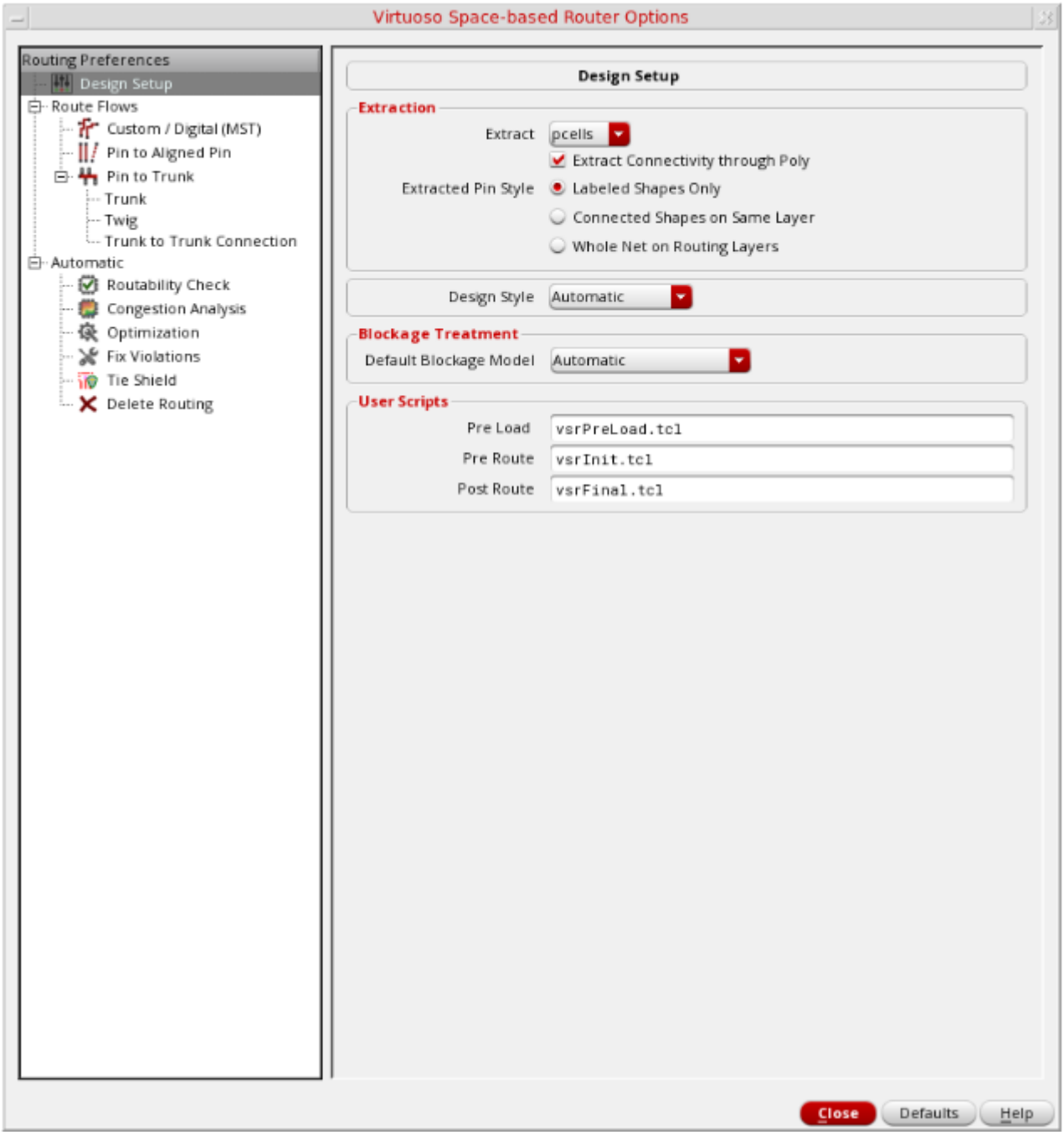
It includes the following sections:

- Support for Multiple Forms
- Inserting Colored TrimMetal Layers in Automatic Routing
- Interrupting Routing for Unresolvable Errors

Using the Virtuoso Space-based Router Options Form

Choose *Route – Design Setup* or click the *Design Setup* icon on the Virtuoso Space-based Router toolbar.

The Virtuoso Space-based Router Options form is displayed, as shown in the following figure.



Note: Alternatively, you can click the *Options* menu item. The *Options* menu item is available either in the drop-down menu when you click the icons on Virtuoso Space-based Router toolbar or when you place the mouse pointer on a menu item in the *Route* menu.

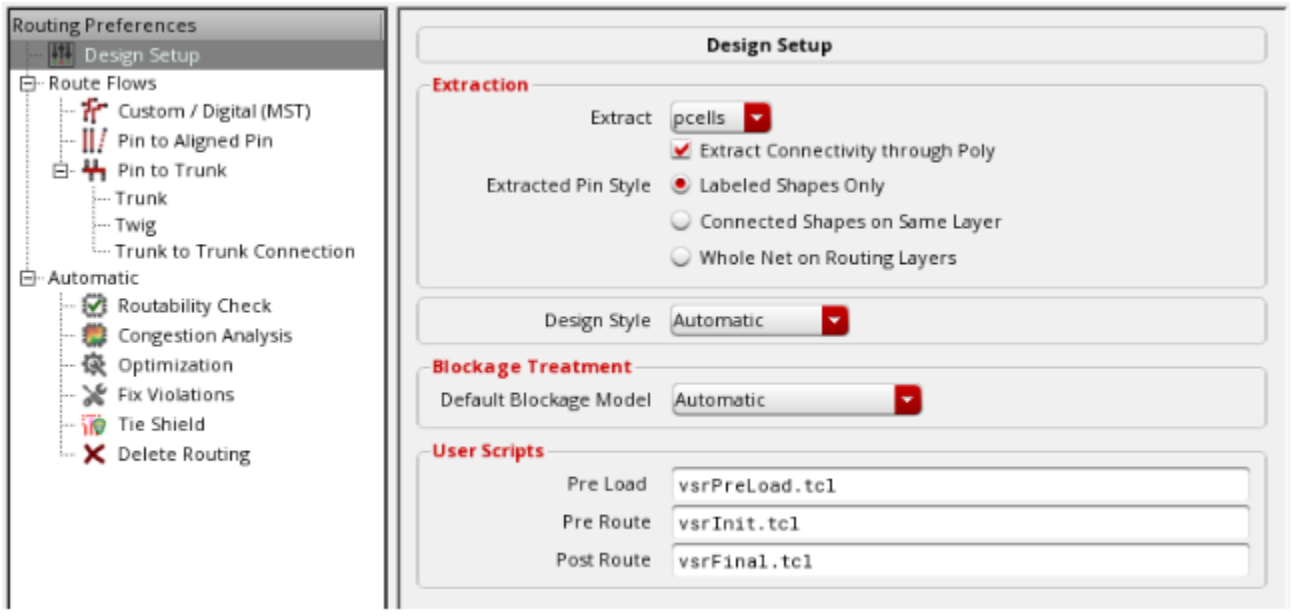
The Virtuoso Space-based Router Options form is divided into two panes, the left pane and the right pane. The left pane contains a hierarchically arranged collection of routing commands. When you select a command in the left pane, all options for that command are displayed in the right pane. The *Default*s button can be used to reset the options that are currently displayed in the right pane of the Virtuoso Space-based Router Options form to their default values.

Using the Virtuoso Space-based Router Options form, you can specify the following top-level routing features:

- Specifying Design Setup Options
- Specifying Routing Flow Options
- Specifying Automatic Flow Options

Specifying Design Setup Options

Click the *Design Setup* icon on the toolbar or choose *Route – Design Setup* to display the *Design Setup* subform in the Virtuoso Space-based Router Options form, as shown in the following figure.



Using the *Design Setup* subform, you can specify the following options:

- **Extraction**

- **Design Style**
- **Blockage Treatment**
- **User Scripts**

Extraction

Opening Virtuoso does not extract your design by default. You must set extractor depth (*Options – Layout XL*) and extract your design (*Connectivity – Extract Layout*).

The extraction options in the Virtuoso Space-based Router Options form lets you control the instances that are extracted using router specific extraction. The router specific extraction does not modify the OpenAccess database in any way. It modifies an internal data model used by the router. Therefore, it is recommended that your OpenAccess data follow the Virtuoso XL connectivity data model. If you are XL compliant then there is no need to run the router specific extraction, running the XL extractor is sufficient enough. However, there are some complex cases, such as Pcells that get automatically created when the Virtuoso XL extractor cannot fix non-compliant data. If the Pcells do not automatically create Virtuoso XL compliant data then you need to run the router-specific extraction on at least the Pcells (the default action). If you have Pcells in the lower levels of design hierarchy (for example, some Chip Assembly designs) you may want router-specific extraction on non-Pcells instances by selecting the *All* option.

- *Extract Connectivity through Poly*

  Extracts shapes on the poly layer. This option is selected by default.

- *Extract*

  Lets you extract *all* cells, only *pcells*, or disable the extraction altogether. By default, Pcells are extracted.

  Note: Extraction is done in the router's internal data structures and the original Pcell is not touched.

  Two other options are available, *none* and *all*.

  - *None*

    The *none* option bypasses the extraction process.

  - *All*

    The *all* option runs the extraction process on all instance masters. The *all* option can be used for designs that have Pcells nested two or more levels down in the

hierarchy because the space-based router extraction does not extract Pcell masters during hierarchical extraction.

- *Extracted Pin Style*

  Lets you control which shapes should be converted into a pin shape. The shapes that are labeled or connected are extracted as pin shapes.

  - *Labeled Shapes Only*

    Allows only the shapes that are marked by text or property to become pin shapes. This option is selected by default.

  - *Connected Shapes On Same Layer*

    Allows only the marked shapes and the shapes that are recursively connected to become pin shapes.

  - *Whole Net on Routing Layers*

    Allows all the shapes in the net to become pin shapes.

Design Style

The design style determines which steps in the Wire Assistant flow will run and also affects the heuristics of global and detail routes. Therefore, the value of design style needs to reflect the style of design you are routing. By default, the design style that was last specified is selected in the drop-down list.

If the design style is set to a value other than *Automatic* in the *Design Style* drop-down list then the specified value is used to route the layout. For example, if the design style is *Chip Assembly*, then the router uses the *Chip Assembly* style to route the layout. If the design style is now selected as *ASIC*, then the router uses the *ASIC* style to route the layout.

The following four design styles are available:

- *Automatic*

  The design style is automatically determined based on the size of the layout and the composition of macro instances. This evaluation happens when the design is first processed by the routing commands. The design style is retained until explicitly re-evaluated using the *Refresh* button to the right of the design style field.

- *Device*

  This design style is used when you need to route device-level for custom block creation and smaller designs that do not require congestion planning.

- *ASIC*

  This design style is used when you need to inter block routing for custom and digital blocks, standard cell, and ASIC designs that require congestion planning.

- *Chip Assembly*

  This design style is used when you need to route chip assembly with block abstracts and larger designs that contain large macros and require congestion planning.

Blockage Treatment

These options let you alter the treatment of the blockages.

- *Default Blockage Model*

  Ensures that all blockages have proper spacing properties and that the blockages model the metal on the edge of the block properly.

  - *Automatic*

    Automatically determines the blockage mode to be used according to the design style. This is the default value.

  - *Real Metal*

    Blockages are treated as metal and the spacing rules are applied based on blockage dimensions.

  - *Minimum Width Space*

    Blockages are treated as metal with minimum width, regardless of the real width of the blockage.

  Environment Variable: blockTreatmentTreatAs

User Scripts

An existing routing flow can be captured in the form of Tcl routing commands in a Tcl script. You can run the Tcl scripts to view routing results.

- *Pre Load*

  The specified pre-load script (`vsrPreLoad.tcl`) is used to set Catena environment variables, which alters the infrastructure setup for routing. For example, if you do not want the router to derive new vias from existing standard vias, in such a case, you can specify

the environment variable `db.enable_create_derived_vias` to `false` in the `vsrPreLoad.tcl` script file.

In addition, you can set Catena environment variables related to WSP routing. For example, you can specify the `db.allow_larger_tracks` variable to `true` in the `vsrPreLoad.tcl` script file. Other commands like `map_purpose` can also be specified in the `vsrPreLoad.tcl` script file.

The script is run before the Catena engine is loaded.

Note: The `vsrPreLoad.tcl` is not needed in usual routing cases. It is only needed in special cases.

- *Pre Route and Post Route*

  The specified pre-route and post-route scripts are used at the appropriate time during automatic and assisted routing flow of the Wire Assistant. You can specify scripts by using the Routing Scripts form that is displayed when you choose *Route – Routing Scripts*. For more information, see Routing Scripts.

  The `vsrFinal.il` file is now supported by Virtuoso Space-based router. You can customize this file by adding your own SKILL commands. After routing, the `vsrFinal.il` file is loaded and the customized SKILL commands are run.

Specifying Routing Flow Options

Click *Auto Route Styles*. This displays the hyperlinks that let you quickly navigate to the available routing flows. The router supports three routing flows:

- Custom / Digital (MST)
- Pin to Aligned Pin
- Pin to Trunk

Custom / Digital (MST)

The *Custom/ Digital (MST)* routing style results in a connection with the shortest overall length. This routing style is usually used for congestion-driven designs. If there is no trunk, the *Custom / Digital (MST)* router routes the nets. However, if one or more trunks exist, the following steps are performed in the routing flow:

- Pin to Trunk router routes the nets with trunks according to the options specified in the *Pin to Trunk* subform. For more information about the Pin to Trunk options, see Using the Pin to Trunk Routing

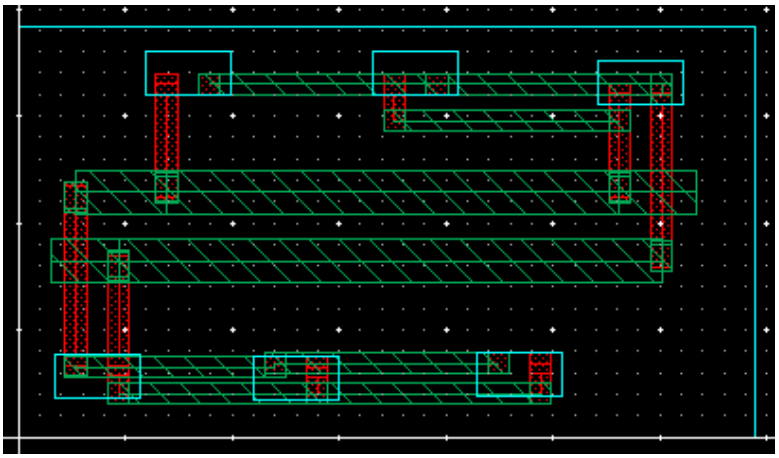
- *Custom/ Digital (MST)* router then routes the remaining opened pins on nets with trunks and also routes all the pins on nets without trunks.

  Note: Only standard cells designed with the area-based rules that exists in the PDK technology file are supported for the *Custom/ Digital (MST)* routing flow. For example, if the PDK technology file only contains area-based rules for the minimum spacing of one area, but the standard cells are designed with area-based rules for another area that allow smaller spacing values, the standard cells have false violations and do not route appropriately. Therefore, only standard cells designed with the same rules that are in the PDK are supported.

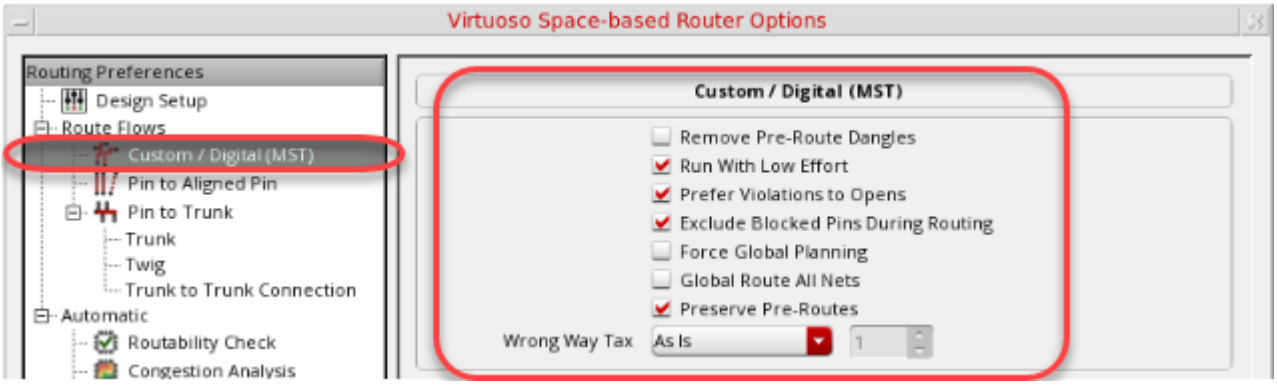
In the *Custom/ Digital (MST)* routing flow, power, ground, and clock nets are not routed when the *Route All* option is selected. To route these nets, you need to select them in the Navigator Assistant and then select the *Route Selected* option.

On a 7nm ASIC design style, an extended segment can be added at one end of a pin to repair the `minArea` and `minEndOfLineSpacing` violations. Pin extensions are generated for all the nets regardless of the nets that are selected for routing.

The following figure shows a layout design using the *Custom / Digital (MST)* routing flow.

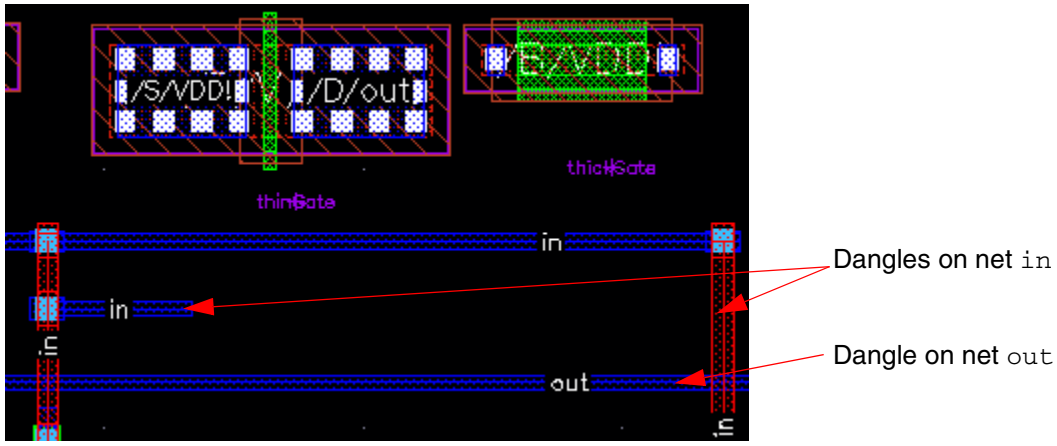


The following figure shows the options that are available for the *Custom/ Digital (MST)* routing flow.



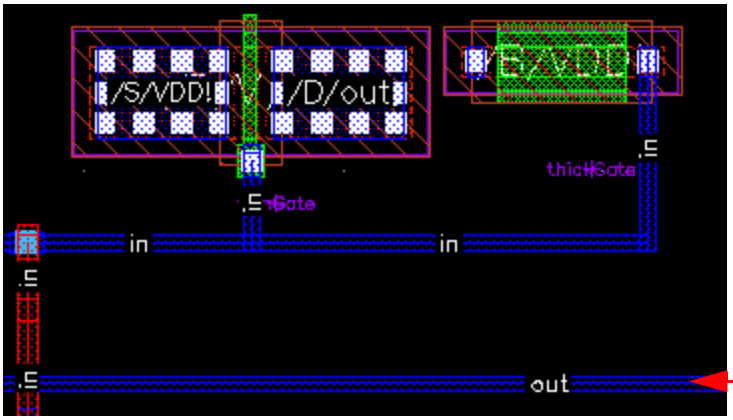
- *Remove Pre-Route Dangles*

  When selected, the *Auto Route* command removes any remaining dangles connected to the “from” and “to” points of the selected nets in the current command. This check box is deselected by default. The following figure shows three dangles that existed before routing the nets. Two dangles appear on net `in` and one on net `out`.



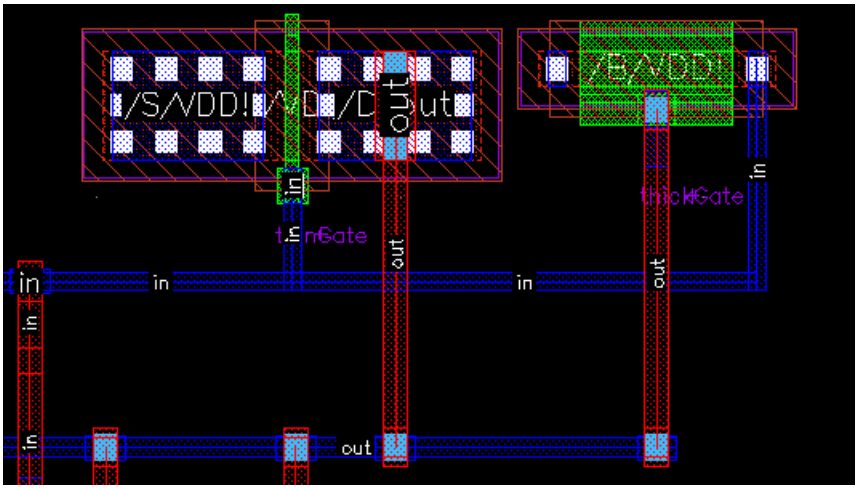
  All pre-routes are considered locked by Auto Route even if they are not explicitly locked and cannot be removed or modified. The *Remove Pre-Route Dangles* option allows you to remove leftover dangles that exist on a selected net and the net is modified as a result of auto routing. In the following figure, net `in` was selected for routing. As a result,

both the dangles on net `in` are removed. Since net `out` was not selected, the dangle on net `out` remains unchanged.



Dangle on net `out` remians unchanged

The following figure shows when both the nets `in` and `out` are selected for routing. As a result, dangles on both the nets are removed.



- *Run With Low Effort*

  Allows the detail router to exit early if the average reduction ratios for opens and errors do not converge in consecutive passes.

- *Prefer Violations to Opens*

  Allows the router to route nets even if a violation occurs while routing. The router will prefer to leave opens rather than creating violations. However, when the *Prefer Violations to Opens* option is selected, the router closes opens at the expense of leaving violations. The router still leaves opens for some violations such as shorts or for blocked pins.

Note: All preroutes, not just the preroutes of the selected nets, are locked.

The *Prefer violations to opens* option affects only automatic routing. It does not have any effect on interactive and assisted routing commands. To control whether violations are allowed in assisted and interactive routing commands, such as Finish Wire, Finish Bus, Finish Trunk, Point-to-Point, and Guided routing, select *Enforce* in the DRD Options form.

- *Exclude Blocked Pins During Routing*

  When selected, the pins on the nets to be routed are checked for planar and via accessibility. If a pin is inaccessible, the router does not attempt to route to it, and a marker appears on the *Routing* tab of the Annotation Browser. This option is selected by default.

  Note: Selecting the *Exclude Blocked Pins During Routing* option improves the router performance by preventing repeated attempts to access the pins that are inaccessible.

- *Force Global Planning*

  Allows forcing of global routing while in *Device* design style

  Environment variable: forceGlobalPlanning

- *Global Route All Nets*

  Forces the global router to use all nets even when routing a selected set of nets.

  Environment variable: globalAllNets

- *Preserve Pre-Routes*

  Allows for not locking the pre-routes. This is useful for ECO routes.

  Environment variable: lockPreRoutes

- *Wrong Way Tax*

  Controls whether wrong-way routing is permitted.

  - *As Is*

    Uses the default setting initialized by the router. This setting can allow certain wrong-way routing if required. This is selected by default and maps to the tax value `1`.

  - *Allow*

    Permits wrong-way routing freely. This maps to the tax value `0`.

  - *None*

Does not allow wrong-way routing. The wrong-way tax value is `100`.

- *Custom*

  Lets you control the extent of wrong-way routing that you want to permit. You can select this option and specify a value in the *Wrong Way Tax Value* field. The default value is 1. You can increase this value to increase the restriction on wrong-way routing.

Environment variable: allLayersWrongWay

- *Wrong Way Tax Value*

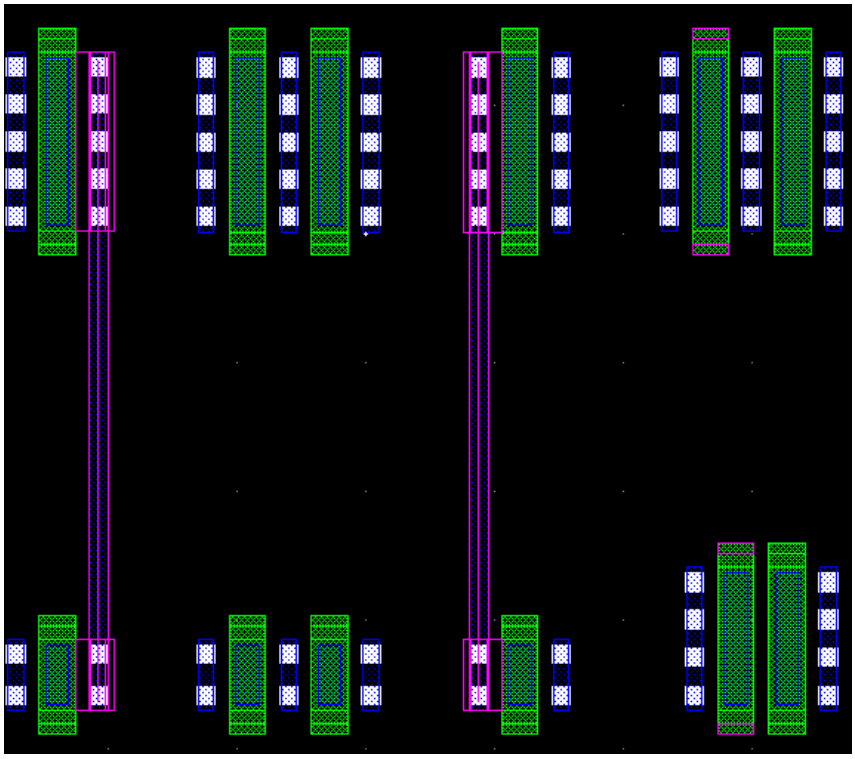
  Indicates the extent to which wrong-way routing can be created. This cyclic field is placed next to the *Wrong Way Tax* combo box. This field is editable only when *Wrong Way Tax* is set to *Custom*. You can specify a value between `0` to `100`: `0` being the level of maximum freedom, and `100` being the level of maximum restriction for creating wrong-way routing.

  Environment variable: allLayersWrongWayCustomValue

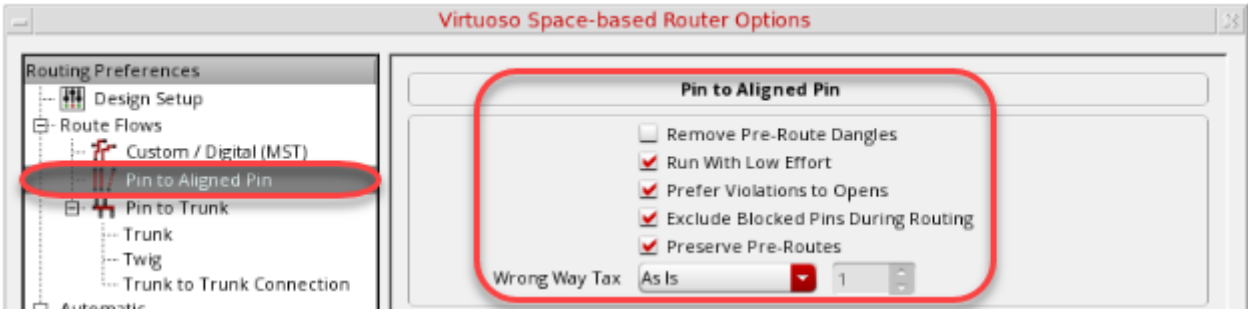
Pin to Aligned Pin

The *Pin to Aligned Pin* routing style is used to connect only the aligned pins on the same layer of net. The pins that are not aligned are not routed. This routing style is usually used

when devices are placed in rows to connect pins that are aligned. The following figure shows a layout design using the *Pin to Aligned Pin* routing style.



Similarly, the following figure displays the options that are available for the *Pin to Aligned Pin* routing flow.



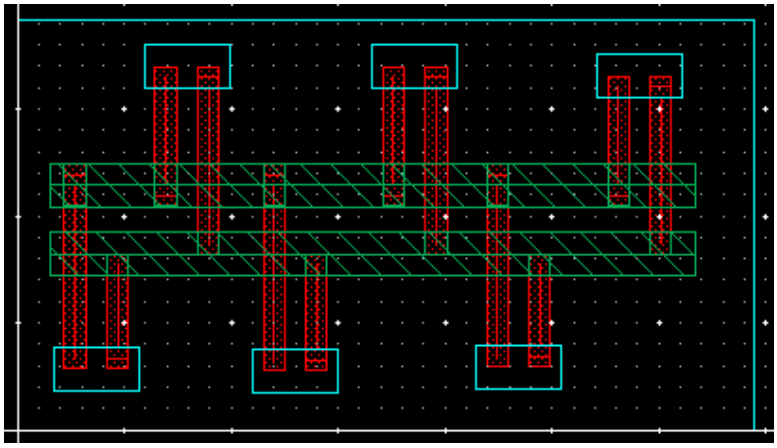
- *Remove Pre-Route Dangles*
- *Run With Low Effort*
- *Prefer Violations to Opens*
- *Exclude Blocked Pins During Routing*

- *Preserve Pre-Routes*
- *Wrong Way Tax*
- *Wrong Way Tax Value*

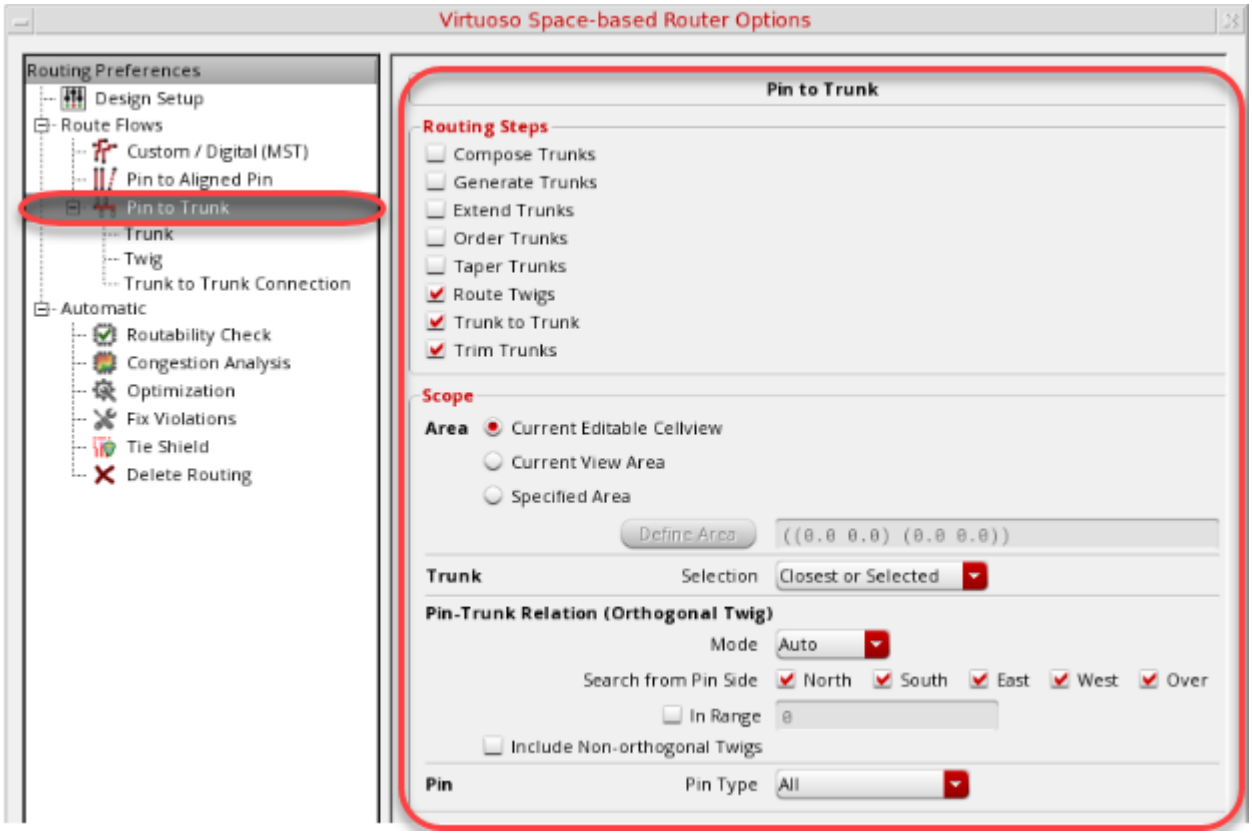
For more information about Pin to Aligned Pin options, see Custom / Digital (MST).

Pin to Trunk

The *Pin to Trunk* routing style is used to connect an individual pin to a trunk. Only the connections that are within the scope of a trunk are routed. The remaining opens are completed by using the *Custom / Digital (MST)* routing style or manually. This routing style is usually used when a spine structure is required.



The following figure displays the options that are available for Pin to Trunk routing flow.

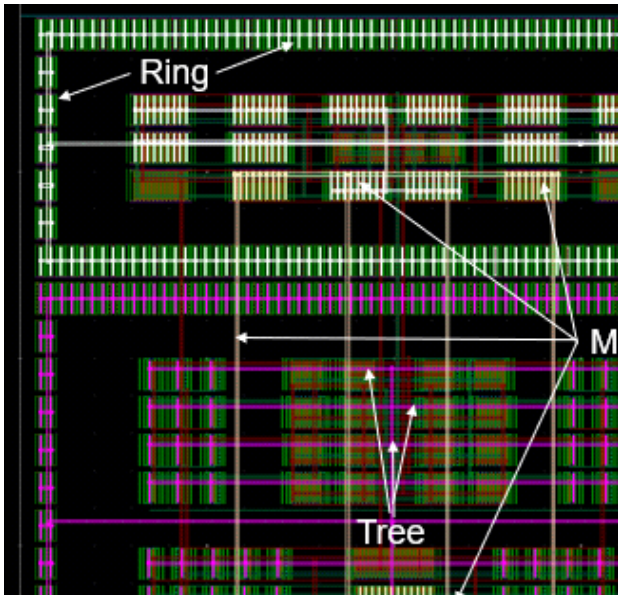


- Routing Steps
  - *Compose Trunks*
  - *Generate Trunks*
  - *Extend Trunks*
  - *Order Trunks*
  - *Taper Trunks*
  - *Route Twigs*
  - Trunk to Trunk
  - *Trim Trunks*
- *Scope*

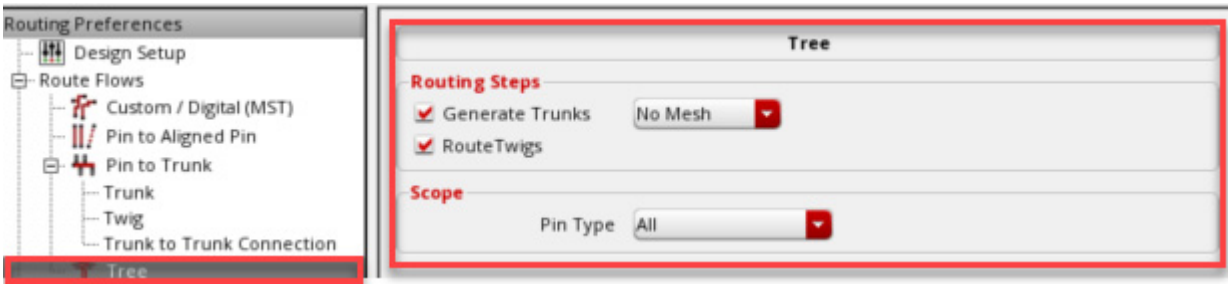
For more information about Pin to Trunk routing and its options, see Using the Pin to Trunk Routing.

Tree Route (ICADVM 18.1 Only)

The Tree Route flow lets you automatically route advanced node device-level designs with a minimum set of options.



The following figure displays the options that are available for the Tree Route flow.



For more information about Tree Route and its options, see Using the Tree Route Flow (ICADVM20.1 EXL Only).

Related Topics

Using Tree Routing

Tree Route

Specifying Automatic Flow Options

Click *Automatic*. This displays the hyperlinks that let you quickly navigate to other automatic routing flow subforms. The following routing flows are available in the Automatic subform.

- Routability Check
- Congestion Analysis
- Optimization
- Fix Violations
- Tie Shield
- Deleting Routing

Note: All options available in the subforms listed above are also available in the *Automatic* section of the Wire Assistant.

Routability Check

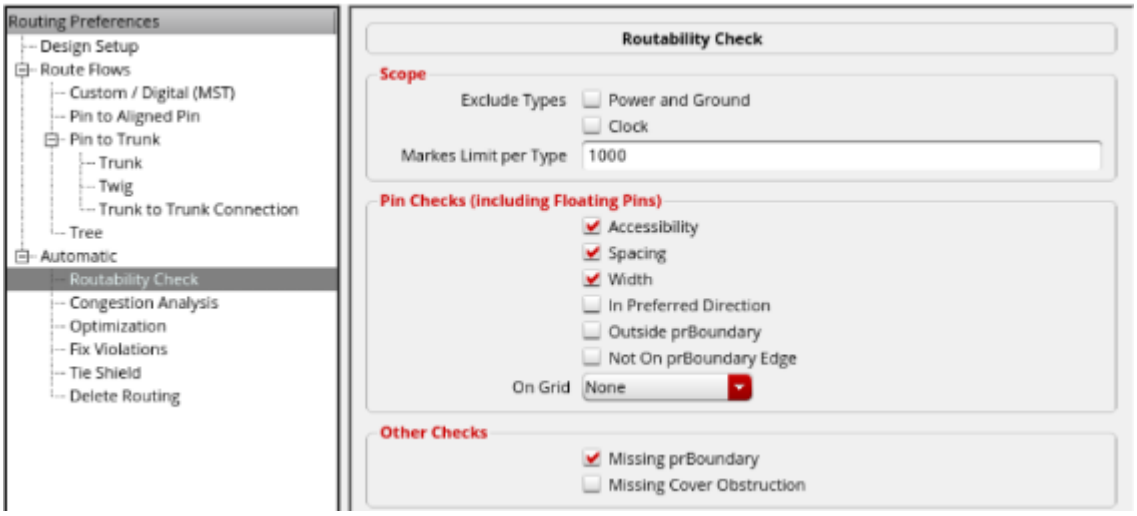
The *Routability Check* feature is used to debug the design and determine potential routing issues that can prevent the router from achieving optimal results.

Using the *Routability Check* options that are available in *Routability Check* subform, you can check the width, spacing, and accessibility of pins, and other issues that may affect routing.

To perform routability checks on a design, click *Routability Check* from the *Automatic* tree in *Routing Preferences*. The following routability check options are displayed in the Routability Check subform:

- Scope
- Pin Checks (including Floating Pins)

- Other Checks



Scope

Lets you choose the nets in the design on which routability checks can be performed.

- Exclude

  You can exclude certain signal types from routability checks.

  Environment variable: checkRoutabilityCreateExcludeSet

  - *Power and Ground*

    When selected, the router does not check during routing for the nets that are marked with signal type *Power and Ground*. This option is selected by default.

  - *Clock*

    When selected, the router does not check during routing for the nets that are marked with signal type *Clock*. This option is selected by default.

- *Markers Limit per Type*

  You can limit the number of violations reported in the Annotation Browser by specifying the required number in the *Markers Limit per Type* field. The default is 1000.

  Environment variable: checkRoutabilityMarkersLimit.

Pin Checks (including Floating Pins)

Lets you perform specific pin checks on the nets in the design. It also flags the pins without any connectivity information.

- *Accessibility*

  Checks if the pin is accessible through planar access (wire is on the same layer as the pin) or through via access from above or below. This option is selected by default.

- *Spacing*

  Checks whether the pin shape satisfies minimum spacing constraints and the overrides set in the Wire Assistant. This option is selected by default.

- *Width*

  Checks whether the pin dimensions are less than the minimum pin width and satisfy the minimum width constraints and the overrides set in the Wire Assistant. This option is selected by default.

- *In Preferred Direction*

  Checks whether the pins are in the preferred routing direction specified for the pin layer. It identifies the pins that can cause the router to use wrong-way metal. This option is deselected by default.

- *Outside prBoundary*

  Checks for the pins that are outside the instance's place and route boundary. This option is selected by default.

- *Not On prBoundary Edge*

  Checks for pins that are not on the edge of the place and route boundary. This option is deselected by default.

  Environment Variable: checkRoutabilityNotOnPRBoundaryEdge

- *On Grid*

  Checks whether the pin edge is on the grid or on the center of the grid. The grid is determined by the *Snap Wires to* field in the Wire Assistant and the *Wires* field in the *Snapping* group box of the Layout Editor Options form. If *Track Pattern* is selected from the *Snap Wires to* field in the Wire Assistant, then the *On Grid* checks determines if all instance pins, IO pins, and trunks have tracks running through their center.

  Note: The *Snap Wires to* field in the Wire Assistant and the *Wires* field in the *Snapping* group box of the Layout Editor Options form are synchronized.

You can select one of the following options from the *On Grid* drop-down list:

- *None*

  Checks whether all the pins have an edge or center on the grid or not.

- *Edge Only*

  Checks only the pins that have an edge on the grid.

- *Edge and Center*

  Checks the pins that have both an edge and center on the grid. This is the default option.

Other Checks

You can also perform the following additional checks before you route a design.

- *Missing prBoundary*

  Checks for instances with missing place and route boundary. This option is selected by default.

  Environment variable: checkRoutabilityMissingPRBoundary

- *Missing Cover Obstruction*

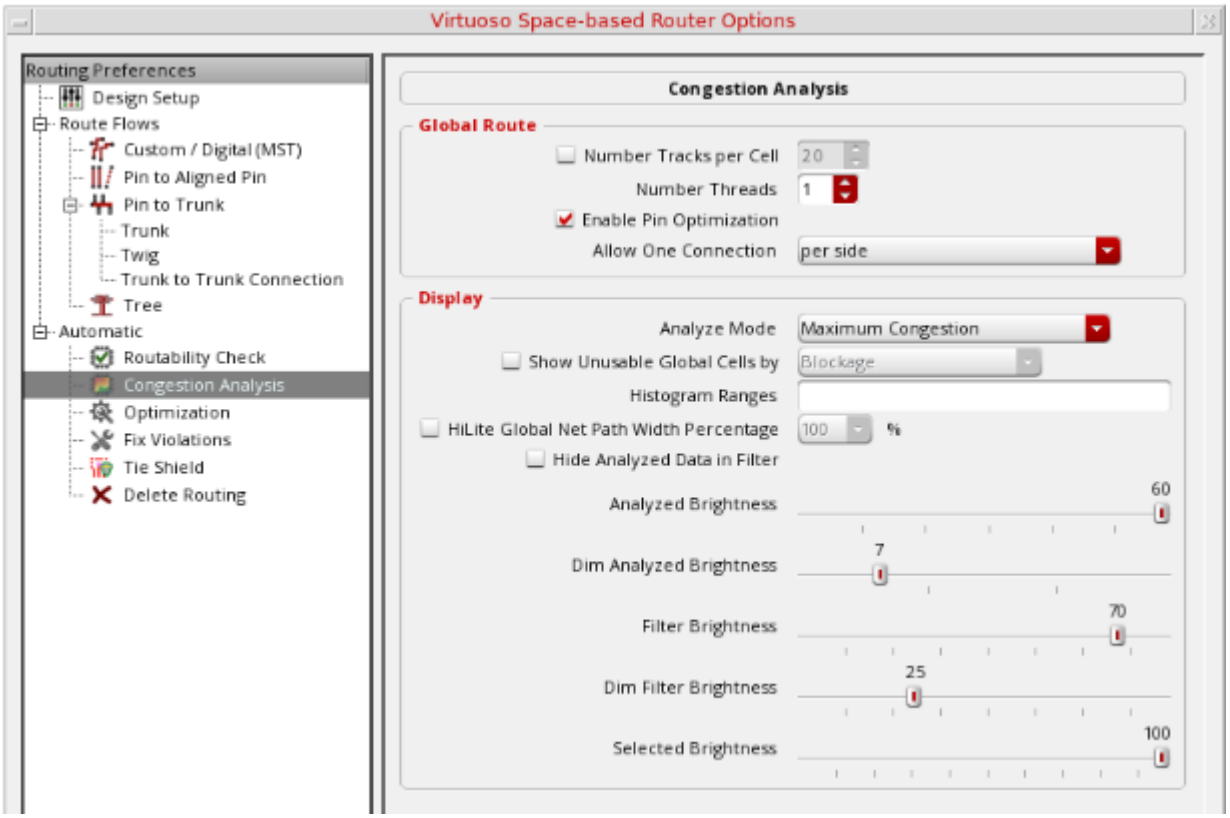
  Checks for instances with missing blockage objects and cover obstructions. This option is deselected by default.

  Environment variable: checkRoutabilityMissingCoverObstruction

The same routability checks are also available in the Batch Checker form. This allows you to run routability checks along with various other checks. To open the Batch Checker form, choose *Verify – Design*.

Congestion Analysis

Click *Congestion Analysis* from the *Automatic* tree under *Routing Preferences*. The following congestion analysis options are displayed in the Congestion Analysis subform.



Global Route Settings

- *Number Tracks per Cell*

  Overrides the number of tracks per gcell. This is suitable for a large design where routing is slow. Increasing the number of tracks passing through a gcell reduces the overall number of gcells in a design and therefore requires less resources to run global routing and congestion analysis.

- *Number Threads*

  Enables multiple threading of global routing and congestion analysis. By default, global routing and congestion analysis are run single threaded.

- *Enable Pin Optimization*

  Enables pin optimization for all soft blocks and opaque virtual hierarchies that are in the design. It also initiates the capabilities in the Virtuoso Design Planning assistant.

- *Allow One Connection*

  Controls the number of routing connections per side for congestion analysis with pin optimization.

  This option is enabled only when *Enable Pin Optimization* is selected. When *Enable Pin Optimization* is deselected, the *Allow One Connection* option is disabled and retains its previous value.

  - *per side*

    Enables one connection per side of each virtual hierarchy and soft block.

  - *per virtual hierarchy or soft block*

    Enables one connection for each virtual hierarchy or soft block.

Important

The *Allow One Connection* option is also added in the Design Planner Options form. The settings in the Congestion Analysis sub form and in the Design Planner Options form are synchronized. When the option setting is changed in one form, it is automatically updated in the other form. Similarly, when the `pinOptOneConnectionPerSide` environment variable is set, both the forms are updated.

Display Settings

- *Analyze Mode*

  Specfies the analysis mode. The recommended analyze mode is the *Maximum Congestion* mode. For more information, see Filtering by Analysis Modes.

- *Show Unusable Global Cells by*

  When selected, enables the display of the global cells that are blocked and are essentially not available for any global paths or interactive routing.

  - *Blockage*

    Shows or hides global cells in both histogram and Global cell track utilization table, for which the blockage is over 100%.

  - *Blockage and Preroutes*

    Shows or hides the global cells for which the blockage and preroutes are over 100%.

- *Histogram Ranges*

The values in this field are updated when the histogram customization is enabled.

- *Hilite Global Net Path Width Percentage*

  Enables you to change the width of the highlighted global path to be a percentage of the width of a gcell. It can widen or narrow down the displayed global path on the heat map.

Using the Hilite Global Net Path Width Percentage option provides a abnormal view when zoomed in to a small region.

- *Hide Analyzed Data in Filter*

  Hides gcells from the heat map that are not part of the histogram filter. This improves the visibility and makes it easier to visualize what has been filtered. In addition, when you select a congestion percentage bucket from the histogram, it automatically dims the congestion information of the gcells that are not in the selected bucket.

- *Analyzed Brightness*

  Controls the brightness in the heat map when filtering has not been applied.

- *Dim Analyzed Brightness*

  Controls the brightness of deselected gcells and deselected net global paths in the heat map.

- *Filter Brightness*

  Controls the brightness of the heat map when filtering is applied.

- *Dim Filter Brightness*

  Controls the brightness of deselected gcells and deselected net global paths in the heat map when filtering is applied.

- *Selected Brightness*

  Controls the brightness of selected gcells and net global paths in the heat map.

Optimization

Click *Optimize* from the *Automatic* tree under *Routing Preferences*. To optimize routing in the design, select any of the options from the subform.

Wires

- *Reduce Jogs through Vias*

  Attempts to reduce wire jogs, considering connection between multiple layers through vias.

- *Reduce Jogs within Single Layer*

  Attempts to reduce wire jogs for single layer wires.

- *Remove Pre-Route Dangles*

  Removes all segments with one or both dangling ends.

- *Adjust Pin Cover*

  Removes any small metal notches located near pins by shifting the wires slightly.

Vias

- *Align Via to Wire Edge*

  Moves vias so that the edge of the metal layers in the vias overlap or align with the edges of the wires. This can reduce the number of small notches around vias. This function is applicable only to vias that exist over route segments.

- *Maximize Cuts*

  Maximizes the number of via cuts for wide layer intersections to fit multiple cuts.

- *Use Double Cuts*

  Tries to replace single-cut vias with double-cut vias wherever it can, without creating a violation.

Fix Violations

Using the *Fix Violations* flow, you can perform all spacing-related checks including merged shapes and fix the violation operations, for all nets, selected nets, or a selected area.

Click *Fix Violations* from the *Automatic* tree under *Routing Preferences*. You can choose to fix any of the routing violations.

- *Minimum Spacing*

  Attempts to fix the opens that were caused by spacing violations.

- *Minimum Edge*

  Uses the setting of the *fixErrorsErrorTypesMinEdge* environment variable to fix minimum edge violations.

- *Minimum Area*

  The minimum area allowed for a shape on a specific layer. It uses the setting of the *fixErrorsErrorTypesMinArea* environment variable to fix minimum area violations.

- *Minimum Enclosed Area*

The minimum area allowed for a hole in the specified layer. It uses the setting of the *fixErrorsErrorTypesMinEnclArea* environment variable to fix minimum enclosed area violations.

- *Minimum Width*

  The minimum width for any shape on the layer. It uses the setting of the *fixErrorsErrorTypesMinWidth* environment variable to fix minimum width violation.

- *Minimum Number of Cuts*

  Uses the setting of the *fixErrorsErrorTypesNumCut* environment variable to fix minimum number of cuts violation.

- *Extension*

  Attempts to fix all extension violations based on the extension constraints defined in the design, such as `minOppExtension` and `minExtensionEdge` constraints.

- *Routing Grid*

  Uses the setting of the *fixErrorsErrorTypesRGrid* environment variable to fix routing grid violation.

- (IACDVM 18.1 Only) *Same Mask Spacing*

  Attempts to resolve same-mask spacing errors involving a double-cut via or a bar via by replacing it with a single-cut via. Even if you specify `minNumCut` as greater than 1 in the Wire Assistant, single-cut vias are used to resolve same-mask spacing errors, provided that `minNumCut` constraint in the technology file is not violated.

  Note: A pin is checked for same-mask spacing errors only when the pin is connected to an extra same-layer metal.

  Environment variable: fixErrorsErrorTypesSameMaskSpacing

Tie Shield

The *Tie Shield* command is used for tying the shielding wires together and to bring the connected shielding wires to the nearest power source, which can be part of a power pin, power ring, or trunk. The *Tie Shield* command is not used for routing the entire shielding net. It is recommended that before tying the shielding wires together to the routed power net, power routing is performed.

Note: While running the *Tie Shield* command, 12 tokens of Virtuoso Layout Suite GXL license are checked out.

- *Gap Space for Vias*

  Ignores the gap spacing required between vias on shield wires and the wires being shielded. The minimum spacing for the layer must still be met. By default, the *Gap Space for Vias* option is deselected.

- *Use Shield Width*

  Uses the shield width as the width for the tie off connections.

- *Connect Shield Only*

  Connects only same-net shields. When shields are added, they are tied to the shield net.

- *Add Redundant Vias*

  Adds redundant vias to tie shield wires to shield nets at every location where the shield wires overlap their respective existing power or ground rails.

- *Add Layer Redundant Ties*

  Adds redundant vias to tie shield wires to shield nets at only those locations where the shield wires overlap their respective existing power or ground rails on adjacent layers. For example, redundant vias are added to tie `Metal2` shield wires to respective `Metal3` power or ground rails, but not to `Metal6` power or ground rails.

- *Floating Shields*

  Preserves the floating shielding wire shapes when they cannot be tied.

- *Tie Frequency*

  Specifies the maximum distance between ties that must be inserted to tie the new shield wires to their respective shield nets.

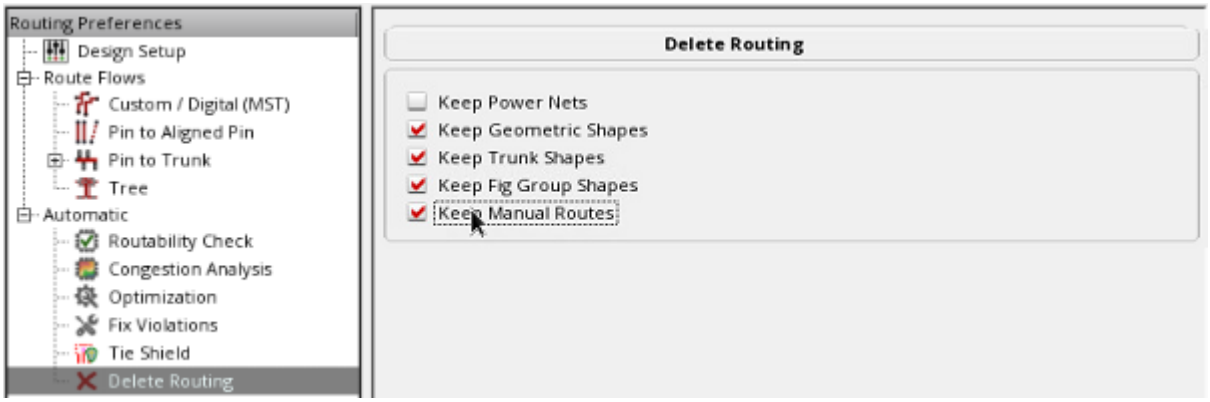
- *Minimum Number Cut*

  Controls the number of via cuts in the tie shield operation.

Deleting Routing

Delete Routing is an interactive command where you explicitly ask for the routing to be deleted. Clicking *Delete Routing* from the *Automatic* tree under *Routing Preferences* option deletes all the routing in FIXED constraints. However, the routes in the LOCKED constraint are not deleted.

Click *Delete Routing* from the *Automatic* tree under *Routing Preferences*. The *Delete Routing* subform displays the options to be considered when deleting routing from the design.



You can select any of the following *Delete Routing* options:

- *Keep Power Nets*

  Lets you retain the power nets. By default, this option is deselected.

  Note: If you choose to delete power routes, multi-part paths (MPPs) are also deleted.

- *Keep Geometric Shapes*

  To delete the geometric shapes, deselect the *Keep Geometric Shapes* option. The *Keep Geometric Shapes* option is selected by default.

- *Keep Trunk Shapes*

To delete the shapes that are marked as trunks, deselect the *Keep Trunk Shapes* option. The *Keep Trunk Shapes* option is selected by default.

- *Keep Fig Group Shapes*

  To delete the routing within fig groups while at the top-level, deselect the *Keep Fig Group Shapes* option. The *Keep Fig Group Shapes* option is selected by default.

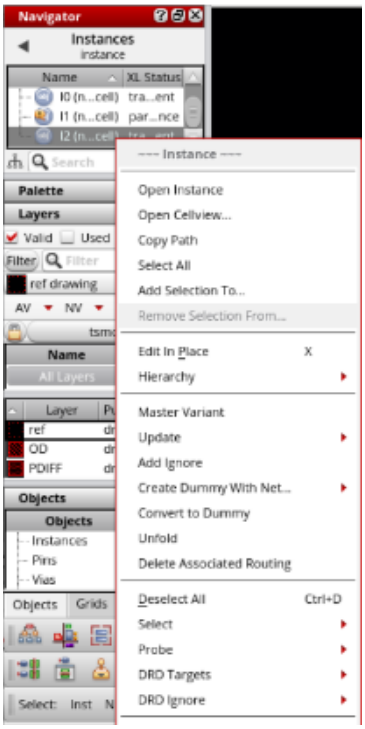
- *Keep Manual Routes*

  To keep the manual routes that are created by interactive and assisted routing and delete only the routes created by automatic routing, select the *Keep Manual Routes* option. The *Keep Manual Routes* option is deselected by default. In this case, all routes created by interactive, assisted, and automatic routing are deleted.

Deleting Partial Routing

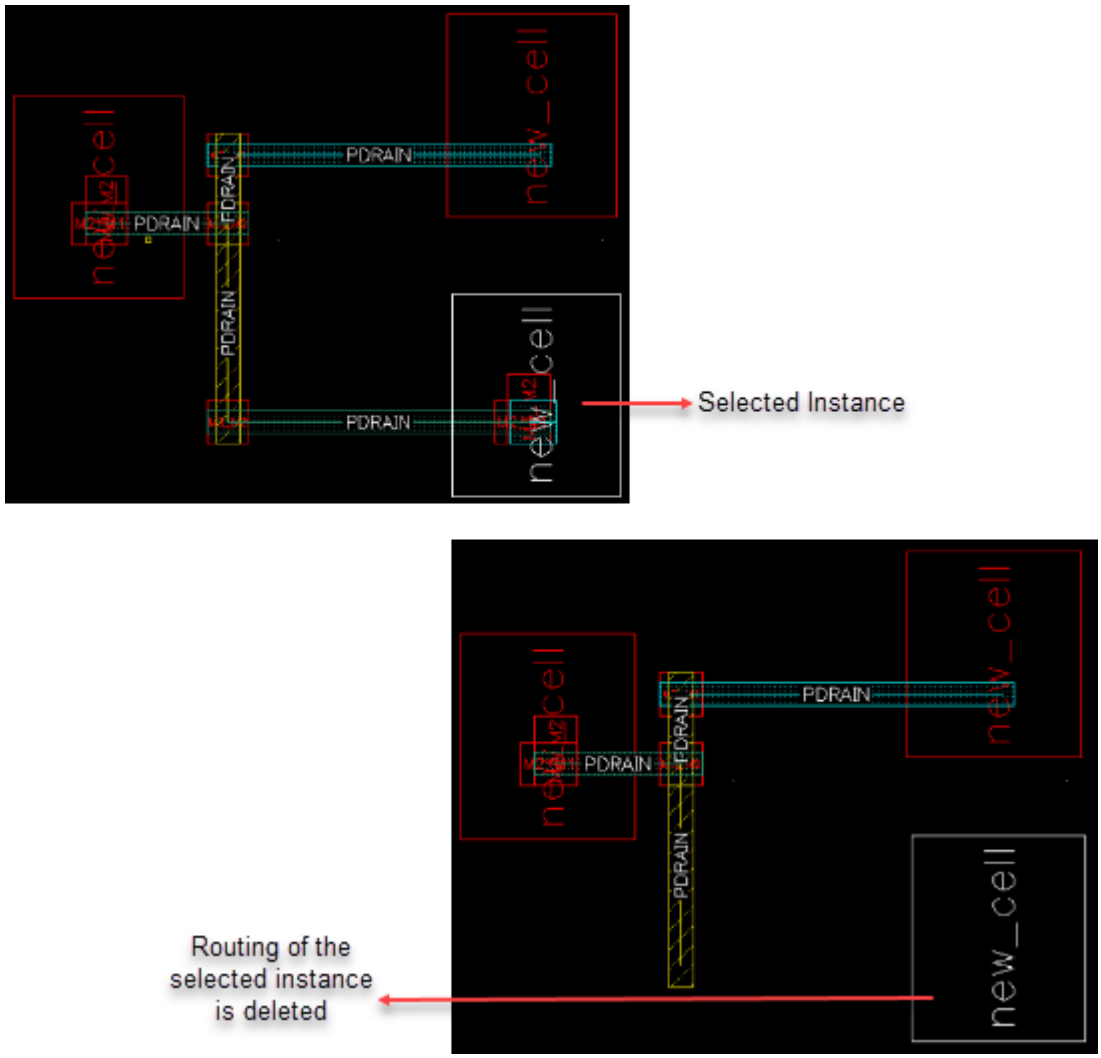
When certain instances or devices are changed in the schematic, you can delete only the routes that are connected to those devices, while ensuring that trunks and any fixed preroutes on that net are preserved, swapped, and rerouted. To do this:

1. Select an instance in the Navigator assistant or layout canvas.
2. Right-click the selected instance. The *Instance* shortcut menu appears.



3. Choose *Delete Associated Routing* in the shortcut menu.

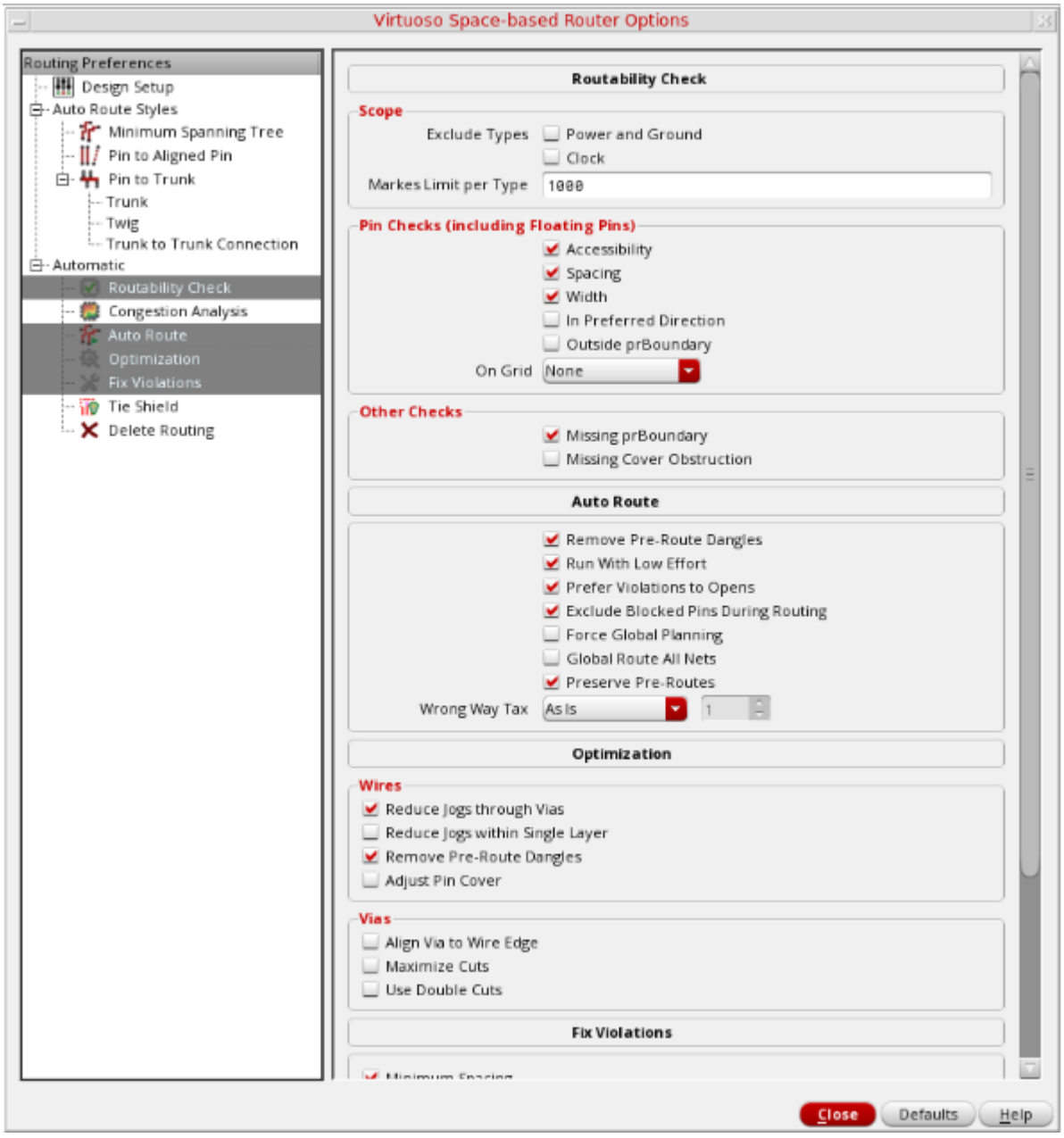
   The routing on the selected instance is deleted. The trunk and fixed preroutes are preserved.



Support for Multiple Forms

The Virtuoso Space-based Router Options form allows you to display multiple subforms in a single pane. To display multiple subforms, select any option from the *Routing Preferences* tree in the left pane, and then hold down the `Ctrl` key and select other options from the left

pane. All subforms corresponding to the options that you selected in the left pane are listed in the right pane in the order in which they are listed in the left pane, as shown below.



A vertical scroll bar is automatically added to the right pane if the pane cannot accommodate all the forms. To remove the subform from the right pane, hold down the `Ctrl` key and deselect the required option from the left pane. You can use the *Defaults* button to reset all options in the subforms displayed in the right pane to their default values.

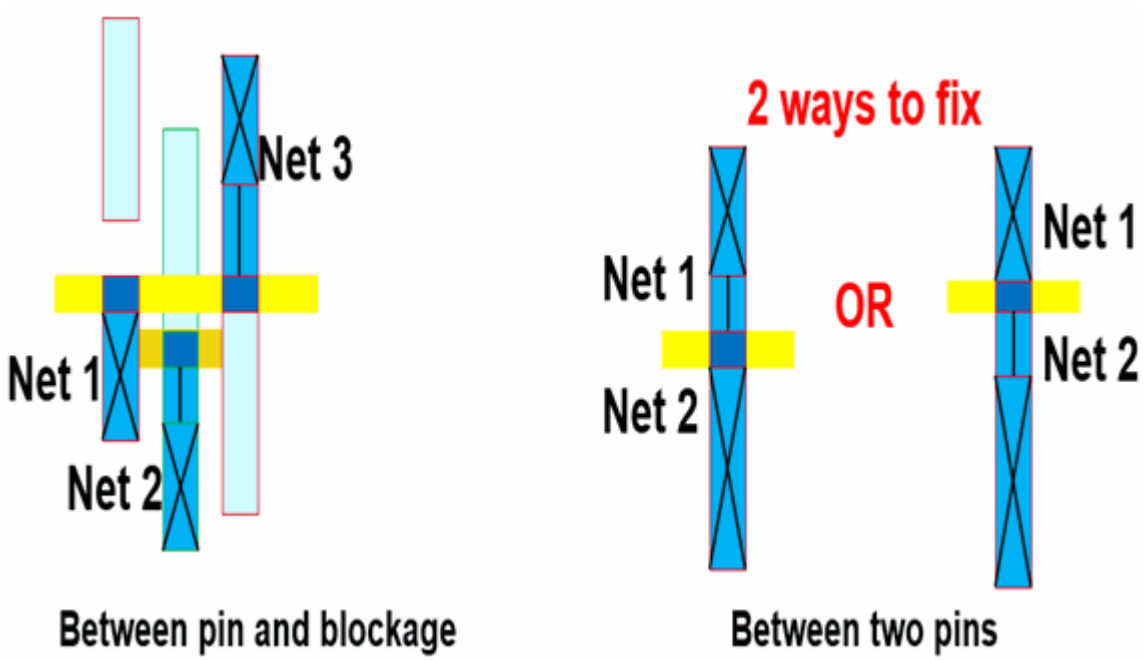
Inserting Colored TrimMetal Layers in Automatic Routing

Advanced node processes (SADP) with trim shapes might not have DRC clean instance pins (`minArea`) and post placement violations between other instance pins (`minEndOfLineSpacing`). This causes problems for automatic routing to access pins because of the violations generated during routing (pin are required to be DRC clean before routing).

To resolve this problem, automatic routing inserts wires (pathSegments) to fix `minArea` violation and trim shapes to fix `minEndOfLineSpacing` violation, while routing pins with connectivity. Automatic routing supports automatic insertion of trim shapes when the `trimShape` and `trimMinSpacing` constraints are defined in the technology file. In such cases, the `trimShape` and `trimMinSpacing` constraints are honored and the trim shapes are inserted.

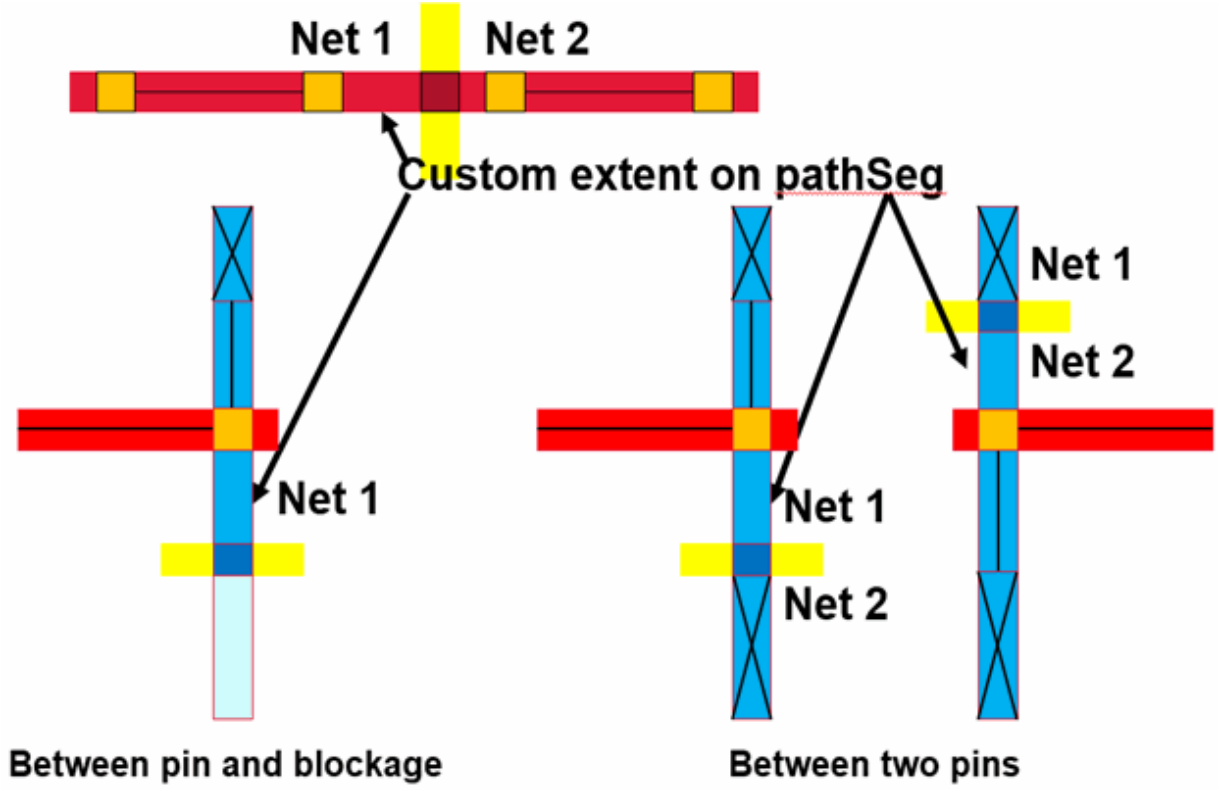
The automatic routing provides two methods of adding a trim shape.

- Static fixing of instance pin violations that are due to the `minEndOfLineSpacing` constraints. In this case, trim, bridge, and patch shapes are added to prevent `minEndOfLineSpacing` violations. This is only possible for pins with connectivity. However, if the checker is run, there may still be violations after fixing the static `minEndOfLineSpacing` fixing. For orthogonal grid alignment, the `orthogonalWSPGrid` constraint is supported.



Note: The routes created in this method are created with the route status as `Fixed`.

- During routing, when two wires are too close to each other, their end of line spacing creates a `minEndOfLineSpacing` violation. In this case, the router modifies the pathSeg and extends it (custom extent) until it comes into contact with trim and bridge shapes. A patch shape is not added.



In automatic routing, the width of the bridge is determined by the `exactEolWidth` parameter of the `minEndOfLineSpacing` constraint and not by the width of the pin.

Note: Two IO pins should not have a `minEndOfLineSpacing` violation between each other because that is considered as a valid violation when routing is performed.

You can also assign color to a trim layer and specify that a trim layer with a given color should cut a routing layer with that color. As a result, trim shapes are automatically assigned the correct color when they are added during automatic routing. For more information, see trimShape.

Interrupting Routing for Unresolvable Errors

While routing a design, there are some errors that can occur due to data or technology setup issues, which could not be recognized by the routability checker. If the router is taking too long to complete, you can stop the current run using `Ctrl + C` and investigate what the issues might be.

To prevent the router from running further, do the following

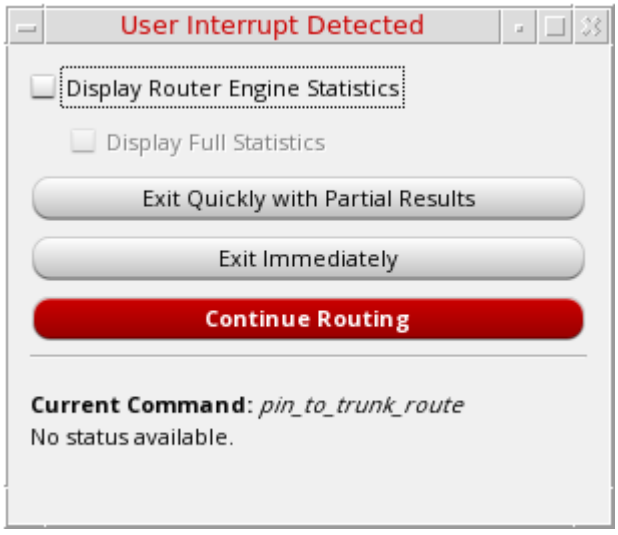
1. Select nets from the Navigator assistant.
2. From the Wire Assistant, select the *Auto Route* command.
3. Click the *All* or the *Selected* button next to *Route Net*.

   The routing for the nets selected in the Navigator assistant starts.

   Alternatively, choose *Route – Automatic Routing*.

   Select the *All Nets* or *Selected Nets* option from the *Automatic Routing* sub menu.
4. To interrupt routing, press `Ctrl + C`.

   The User Interrupt Detected form is displayed.



5. Select the *Display Router Engine Statistics* check box to collect and display statistics from the routing engine.
6. Select the *Display Full Statistics* check box to display all the violation statistics. By default, the *Display Full Statistics* check box is disabled and only the most relevant

statistics are displayed. The *Display Full Statistics* check box is enabled only when the *Display Router Engine Statistics* check box is selected.

7. In the form, click an option to discontinue the router from running. You can select one of the following options.

   - *Exit Quickly with Partial Results*

     Completes the current routing step as quickly as possible. The results are saved and routing exits without any further routing steps. You can view and edit the routing results that are saved. By viewing the routing results, you are able to identify the types of issues that are encountered by the router at the time of interruption.

     Note: The *Quickly Exit with Partial Results* button is enabled only when detailed routing step has already started.

   - *Exit Immediately*

     Exits the current routing immediately without saving any routing results. The design is unmodified and left in its pre-routing state.

   - *Continue Routing*

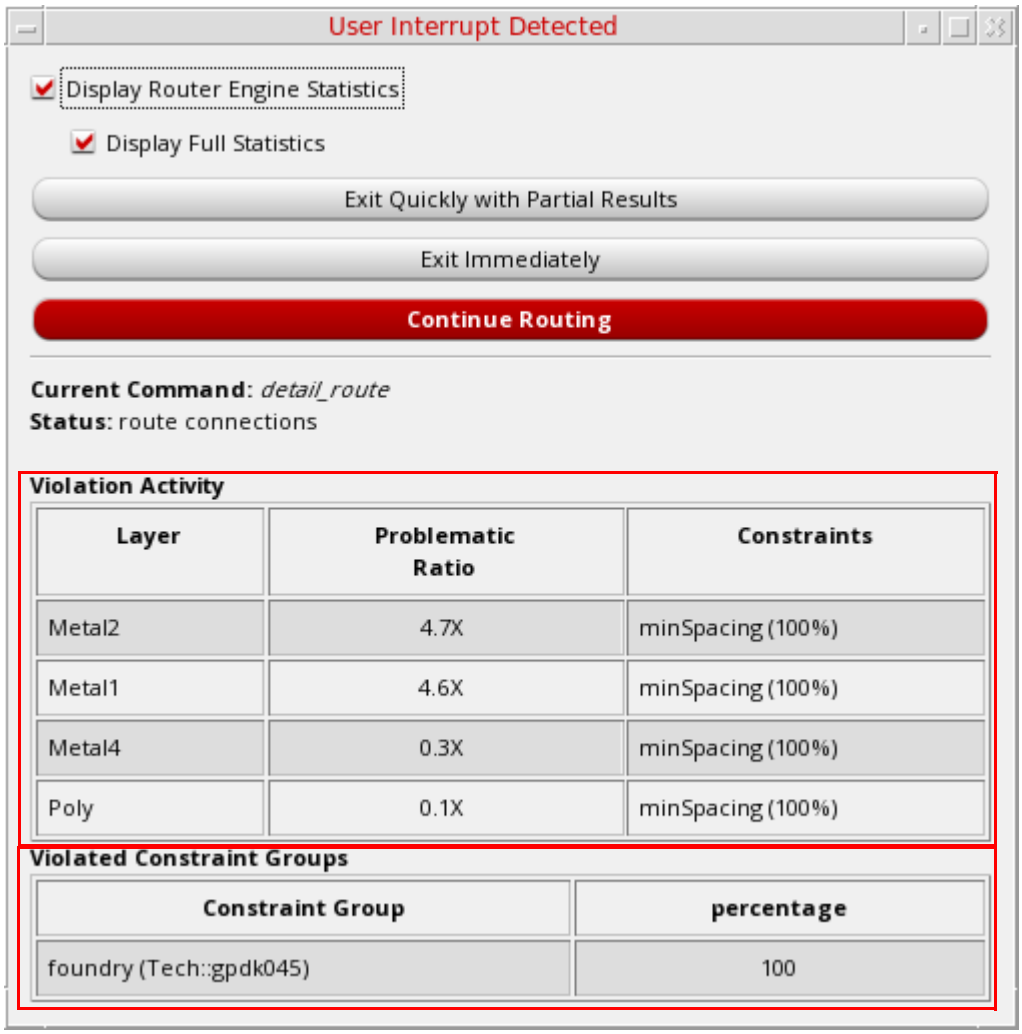
     Cancels the user interrupt and continues routing.

   The *Current Command* section provides information about the command that was running at the time of interruption.

8. To view the router statistics, again click the *Continue Routing* button.

9. Again, press `Ctrl + C`.

The following figure shows the routing engine statistics that get displayed in a tabular format.



Subsequently, clicking the *Continue Routing* button and then pressing `Ctrl + C` provides the routing statistics that have been completed between the two actions.

The *Violation Activity* table shows the router statistics, if they exist. The following table provides the description of various columns in the *Violation Activity* table.

Layer	Provides the layer name
Problematic Ratio	Provides the relative error ratio compared to a uniform distribution of equal violations on across all layers (1X is equal number of violations across all layers in a uniform distribution).

Constraints	Provides the constraint name and the percentage of the type of violation on that layer.

The *Violation Constraint* table shows the foundry constraint group rule violations. The following table provides the description of various columns in the *Violation Constraint* table.

Constraint Group	Excludes foundry CG for incomplete statistics and includes foundry violations if *Display Full Statistics* is selected.
percentage (excluding foundry)	Shows the percentage of violations based on the constraint group it is in.

7

Managing Cover Obstructions While Routing

You can now manage cover obstructions while routing using the Cover Obstruction Manager form. The Cover Obstruction Manager form allows you to create, edit, and display the cover obstruction information at the current level as well as for level-1 instances. For an editable master instantiated at the top, the cover obstruction information is stored directly into the master. However, for read-only masters, the cover obstruction information is stored in the top-level design. In the top-level design, some information, such as shapes, vias, and presence of instances lower in the hierarchy, contained in the instantiated masters is not necessary for the router. In addition, you can route an instance at the top-level even if there are instances down in the hierarchy, which have cover obstructions existing on them. Using this feature, you can:

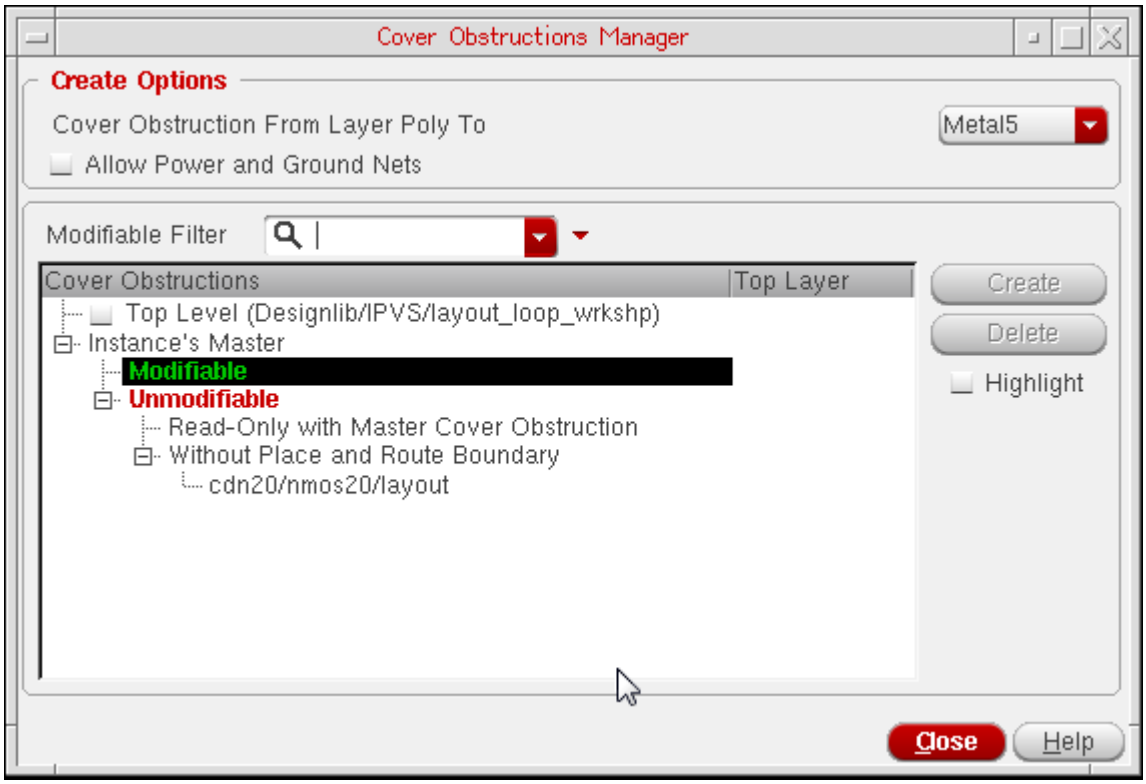
- Improve the routing performance in terms of memory usage and CPU time.
- Keep the layout views instantiated on top of the design without a need to abstract and re-master the views.
- Filter out all the unnecessary information for the router and also specify metal obstructions.

You cannot override any existing cover obstruction information; if you need to, then first remove it. For a read-only master, ensure that you change its write access before deleting any cover obstruction.

Defining a Cover Obstruction

To define a cover obstruction:

1. Choose *Tools – Cover Obstruction Manager* to open the Cover Obstructions Manager form.



   The Cover Obstructions Manager form automatically checks the masters of the currently selected instances that can either be modified or deleted. Also, in the layout window, the instances corresponding to the master instances selected in the tree widget automatically appear as selected.

2. Select the layer range to be obstructed by selecting the layer from the *Cover Obstruction From Layer Metal1 To* drop-down list in the *Create Options* group box.

3. To allow power and ground nets to route through obstructions, select the *Allow Power and Ground Nets* check box in the *Create Options* group box. Routing of power and ground nets can be allowed only if the cover obstruction is on the prBoundary of the master. It cannot be allowed if the cover obstruction is on the top-level layout. By default, the *Allow Power and Ground Nets* check box is deselected.

4. The top-level design can have many instances.To filter out the instances in the instance master tree, use the *Modifiable Filter* search widget. Specifiying the search criteria allows you to reduce the master list and quickly identify the instances to operate on.

5. Select an instance's master on which you want to create the cover obstruction from the Instance's Master tree widget. This tree widget displays all instance masters and top-

level cellviews that are there in the current design. The instance master in a design can be of the following three types:

- Modifiable

  An Instance's Master that allows creation and deletion of cover obstructions. In addition, it does not have a cover obstruction on both prBoundary and the instance headers.

- Unmodifiable

  An instance master that does not allow creation and deletion of cover obstructions. Such instances are further divided into the following two categories:

  - Read-only with Master Cover Obstruction

    The instance's masters that have cover obstruction defined on its prBoundary but the master itself is read-only. To delete the instance's master cover obstruction, you need to get the write permission on master files.

The read and write permissions of an instance's master are not determined by the master's shell permission on the UNIX (Linux) file system. The permissions are determined from the current master's mode in Virtuoso, which in turn is determined when the Virtuoso layout reads the master from the hard disk.

  - without Place and Route Boundary

    The instance's master that do not have prBoundary data and cannot define cover obstruction. Place and route boundary is required for creating and deleting cover obstructions.

- Deletable Only

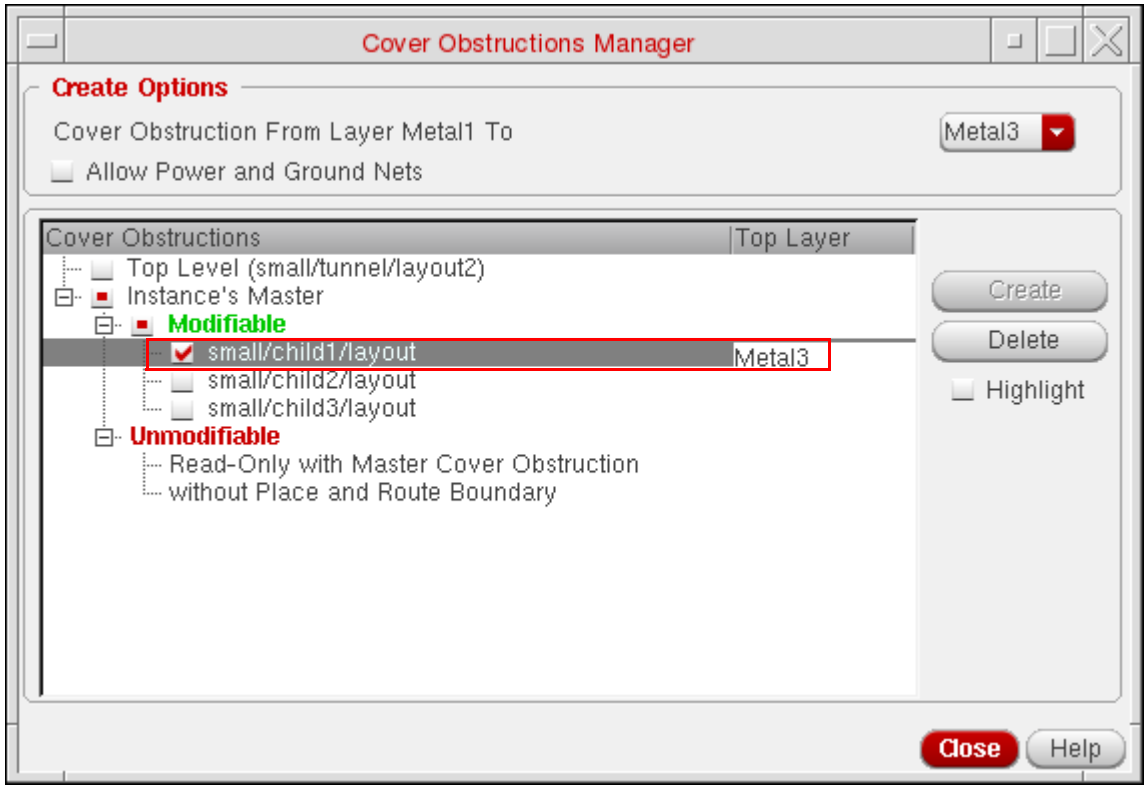
  Those master instances that have cover obstruction defined on both prBoundary and instance headers.

The Cover Obstructions Manager should never create cover obstruction on both master's PRBoundary and instance headers.

6. To add cover obstructions at the current level, that is, on the prBoundary shape, select the check box next to *Top Level* and then click the *Create* button.

   Note: The *Top Level* displays the master instance in the format `libName/masterInstanceName/viewName`. For example, `small/tunnel/layout`.

7. To add the cover obstruction on Instance's Master, select the check box next to the required instances from the Instance's Master tree widget and click *Create*. The following figure shows an example, where cover obstruction is created on instance `small/child1/layout`. After the cover obstructions have been defined, the related information gets displayed in the *Top Layer* column as shown in the following figure. If the *Top Layer* column is empty, it means no cover obstruction has been defined for that particular Instance's Master.



The state of the *Create* and *Delete* buttons gets updated according to the selection of the Instance's Master.

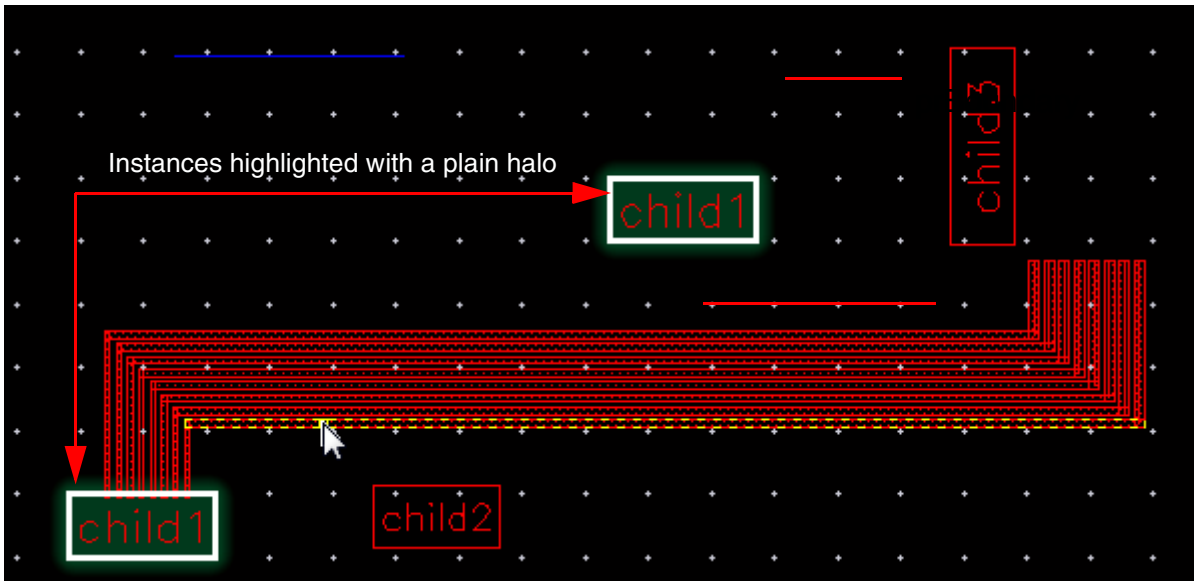
- If all selected masters already have cover obstructions, the *Create* button appearsas disabled.
- If all selected masters do not have cover obstructions, the *Delete* button appears as disabled.
- If some Instance's Master have cover obstruction and some do not, the *Create* button appears as disabled but the *Delete* button is enabled.
- If none of the Instance's Masters are selected, both the *Create* and *Delete* buttons are disabled.

Removing a Cover Obstruction

To remove cover obstruction, select the instance and then click the *Delete* button. The *Delete* button is disabled if the instance(s) on which the cover obstruction appears are not selected. Once the cover obstruction is deleted, the *Top Layer* column in the tree widget appears empty.

Viewing a Cover Obstruction

To view the defined cover obstruction in the layout canvas, select the *Highlight* check box. The instances that belong to the master with the defined cover obstruction get highlighted. The highlight in the layout design uses true color display based on the color of the top metal layer. The instances are highlighted with a plain halo. At the top level, the prBoundary shows a line halo, as shown in the following figure.



Note: When the Cover Obstructions Manager form is closed, the highlights on the instances automatically disappear.

8

Specialty Routing

This chapter describes,

- Creating Constraints With the Constraint Manager
- Symmetry Routing
- Differential Pair Routing
- Shield Routing
- Matched Length Routing

Creating Constraints With the Constraint Manager

Constraints can be created in the schematic or layout view. For information about how to create constraints see Creating Constraints, in the *Virtuoso Unified Custom Constraints User Guide*.

When creating a constraint in the schematic view, the constraint can be transferred to the layout view automatically or manually. See Constraint Transfer in the Virtuoso Layout Suite documentation.

The *Create Wire* command does not support retroactive application of constraints (for example, shielding, symmetry, differential pair) after the first digitized point.

For constraints to be applied properly, a net name must be provided at the first digitized point. The net name can be supplied at the first click by clicking on a shape stamped with the net name or by entering the name of the net in the Create Wire form before the first click.

Symmetry Routing

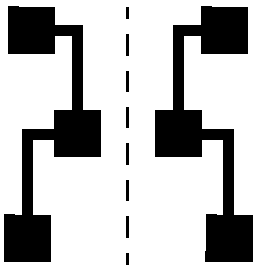
When routing a self-symmetric net, enable Snap to Pin Origin in the Create Wire form if you start the route from the pin on the symmetry axis. Otherwise, you may get two wires routing out from the pin.

- Types of Symmetry Routing
- Defining a Symmetry Constraint
- Automatic Symmetry Routing
- Differential Pair Routing

Types of Symmetry Routing

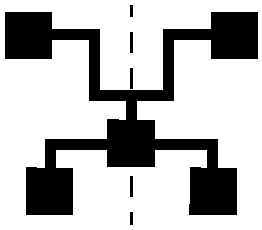
- Mirrored Symmetry

  Mirrored symmetry involves two nets mirrored over an axis. For a mirror symmetry constraint, the axis coordinate value must be determined by calculating the center coordinate between the two symmetric pins. When routing nets with mirror symmetry, ensure the axisLocation parameter is set to Fixed.



- Self Symmetry

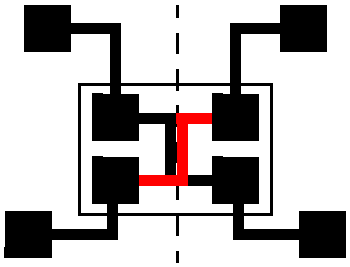
  Self Symmetry involves a single net that is mirrored over an axis. The entire geometry can reside on the symmetry line, in which case there are no symmetric objects. For a self symmetry constraint, the axis coordinate value is the center of the self symmetry pin.



- Cross Symmetry

  Cross symmetry involves two nets that are allowed to cross the symmetry line using one or more instances of crossover cells. This type of symmetry is comparable to two separate cases of mirror symmetry involving two pairs of nets which are part of the original nets.

Crossover cells are created automatically by the Analog Placer when a need for such a cell is detected. This happens when IO pins and instTerms associated with the symmetric net pair are placed such that a crossover can not be avoided.



- Partial Symmetry

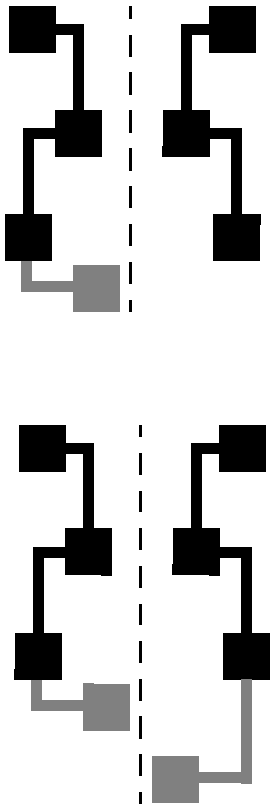
  Partial symmetry can apply to nets that are mirrored symmetric, self symmetric, or cross symmetric. Two nets that do not have the same number of terminals are considered partially symmetric. Partial symmetry also applies when a portion of the wires are unique on either side of the axis.

  Checks are not implemented to determine if every geometry in a symmetric net has a symmetric counterpart. Also, checks are not implemented as to whether corresponding pieces of geometry are symmetric.

  An asymmetric geometry in a symmetric net may cross the symmetry line or reside on the opposite side of the line because there are no symmetric counterparts to cause conflict.

  The parts of a net that have symmetric counterparts can be connected with symmetric wiring. The parts that are unique to one side of the symmetry line must be connected

with wiring that is not symmetric, as shown in the graphic below. To route an extra terminal, specify the specialty routing constraints in the Constraint Aware Editing mode.



Defining a Symmetry Constraint

The following steps describe how to create a basic symmetry constraint in the layout view.

These instructions assume that there is a schematic view or connectivity source associated with the layout view.

1. From the layout cellview, select *Window – Assistants – Constraint Manager*.

   The *Constraint Manager* assistant opens.

2. From the layout cellview, select *Window – Assistants – Navigator.*

   The *Navigator* assistant opens.

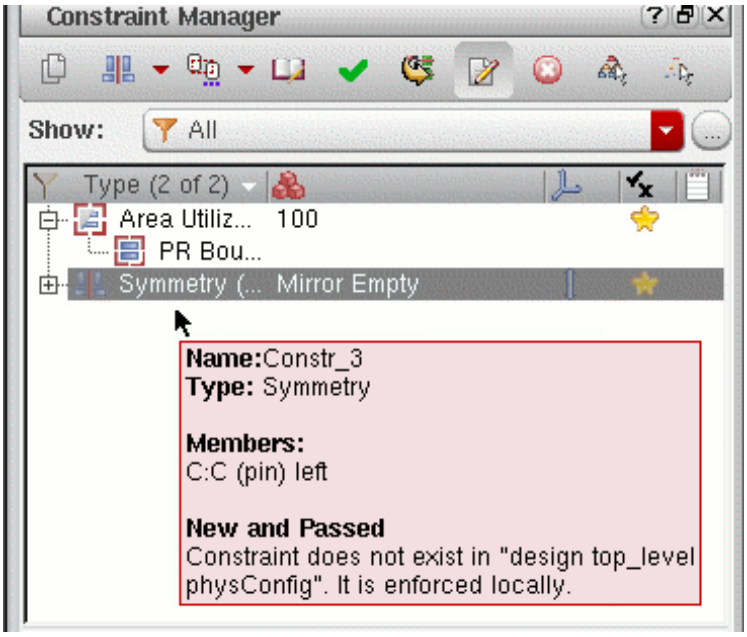
3. From the *Navigator* assistant, select one net or pin for self symmetry routing, or two nets or pins for mirrored symmetrical routing.

For mirrored symmetry, select the second net by pressing the `Control` key and clicking on the net or pin name. See Types of Symmetry Routing for a description of mirrored and self symmetry constraints.

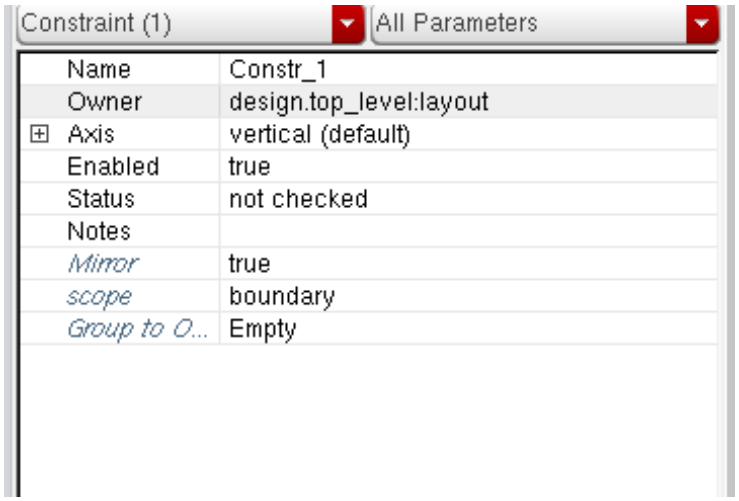
4. In the *Constraint Manager* assistant, left click the arrow next to the *Constraint Generator* icon and select *Routing – Symmetry* from the pop-up window.

   A symmetry constraint with default values is created.

5. You can confirm the members of the constraint by hovering the cursor over the *Symmetry* constraint in the Constraint Manager window.



6. In the lower section of the Constraint Manager window you will see the default symmetry constraint.



7. Click the *+* sign next to *Axis* to update the default constraint values.

   a. Specify the axis *direction* for the constraint; *horizontal* or *vertical*.

   b. Set the *axis Location* to *fixed*.

   c. Specify the axis *coordinate* parameters for the constraint. If you are not using the automatic placer then you should explicitly set the axis value. This is because the default value is typically incorrect for most designs.

      Note: The steps 7(b) and 7(c) are required only if user wants to specify an axis that is not the default, which runs through the center line of the prBoundary.

      For a mirrored symmetric constraint, the axis coordinate value must be determined by calculating the center coordinate between the two symmetric pins.

      For a self symmetry constraint, the axis coordinate value is the center of the self symmetry pin.

8. In order to interactively/assisted wire edit a partially symmetric net, you need to first route the symmetric shapes and then turn OFF the Constraint Aware Editing mode to complete the asymmetric shapes.The order of routing the symmetric shapes before the asymmetric shapes is important.

Automatic Symmetry Routing

When using automatic symmetry routing, it is important that your axis coordinate be correct. In addition, the surrounding shapes, such as instances, must be symmetrical.

Note: There should be no pre-routed asymmetric routes if there are any symmetric portions of the route left to be completed by the automatic router.

Use the *Detailed* run mode to route symmetry constraints with the automatic router. Symmetry constraints are not supported in the global, local, conduit, and power routers.

Be aware that the *Display Levels* settings *start* and *stop* control hierarchy depth for automatic routing.

See Defining a Symmetry Constraint for information about how to create a symmetry constraint.

Differential Pair Routing

Defining a Differential Pair Constraint

Before defining a differential pair constraint for a pair of nets, consider the topological relationship between the pins and existing wires and ensure that they meet the following requirements of differential pair routing:

- Pins must come in pairs
- The shape and appearance of pin pairs should be similar
- If there are pre-routed wires, these wires should be laid out in a pairing pattern. Also, corresponding segments must be laid out in the same layer.

If the nets do not meet the above set of requirements but you still want the nets to travel together as much as possible, then you should create a bus constraint for the nets.

The following steps describe how to create a basic differential pair constraint in the layout view.

These instructions assume that there is a schematic view or connectivity source associated with the layout view.

1. From the layout cellview, select *Window – Assistants – Constraint Manager*.

   The *Constraint Manager* assistant opens.

2. From the layout cellview, select *Window – Assistants – Navigator.*

   The *Navigator* assistant opens.

3. From the *Navigator* assistant, select two nets for the differential pair.

   Select the second net by pressing the `Control` key and clicking on the net name.

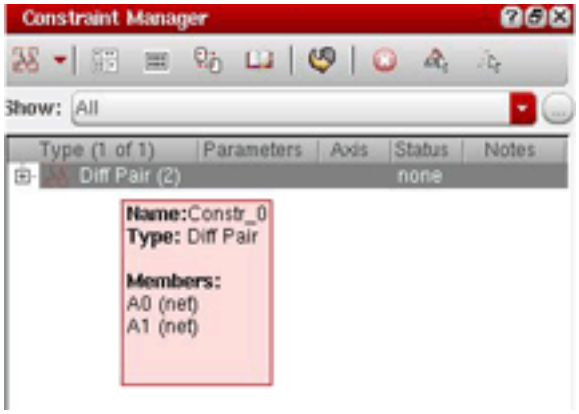
   Only nets are allowed as members of *Diff Pair* constraints.

4. In the *Constraint Manager* assistant, left click the arrow next to the *Constraint Generator* icon and select *Routing – Diff Pair* from the pop-up window.

   A differential pair constraint with default values is created.

   A *Diff Pair* constraint with default values is created. The *Diff Pair* constraint appears in the constraint browser tree under *Type*.

5. You can confirm the members of the constraint by hovering the cursor over the *Diff Pair* constraint in the Constraint Manager window.



Changing Diff Pair Values in the Process Rules Editor

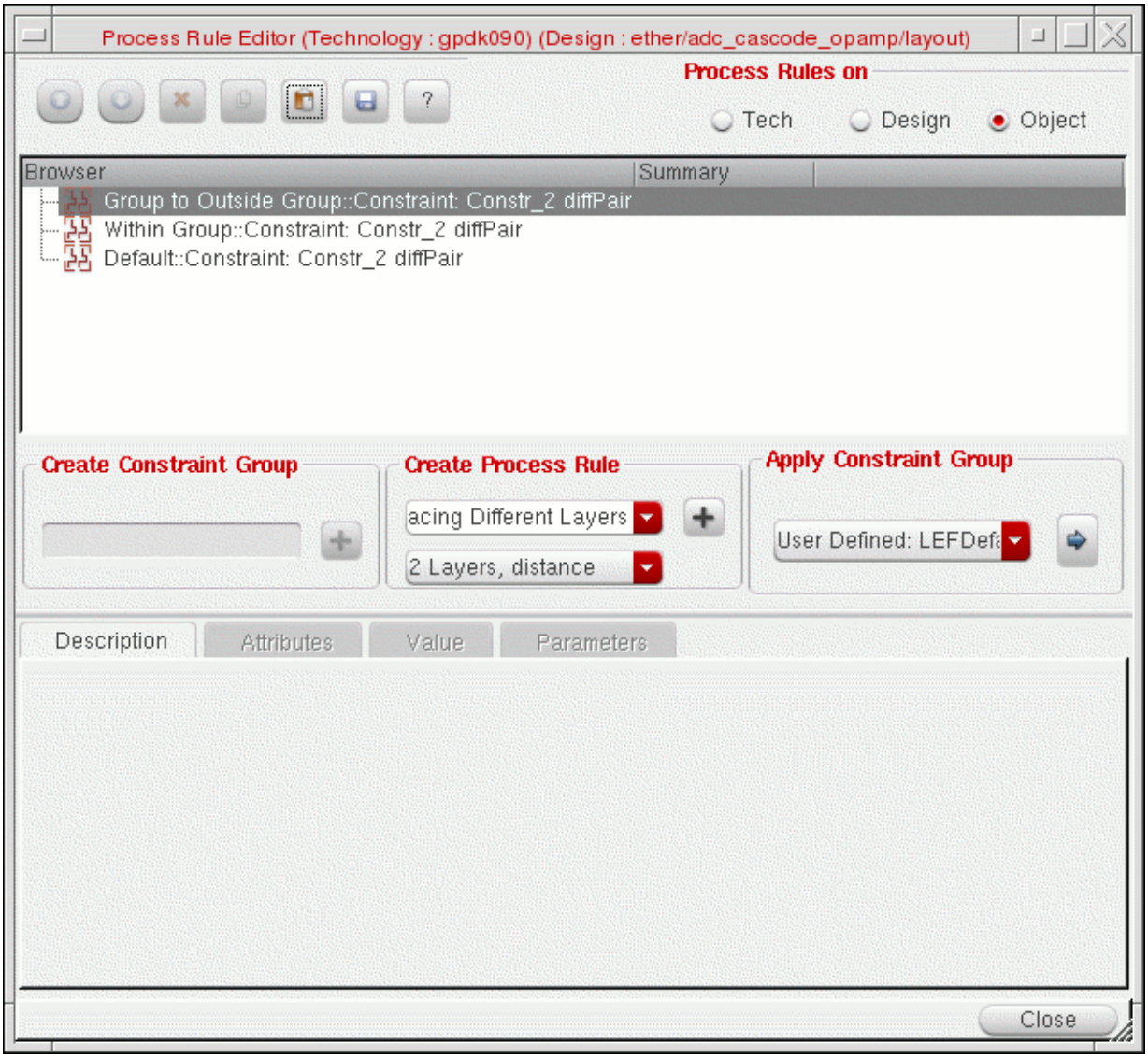
Once a *Diff Pair* constraint is defined you can optionally assign additional attributes in the Process Rule Editor.

Note: It is now possible to assign Constraint Groups to a differential pair net, 'Within Group' and 'Group to outside Group' rules, directly in the Constraint Manager that supports these parameters. You can just specify the layers and spacings, without using Process Rule Editor. It is also possible to automatically generate Constraint Groups for this purpose.

To set or change the values,

1. Select the *diff pair* constraint In the Constraint Manager window and click the Process Rules Editor icon.

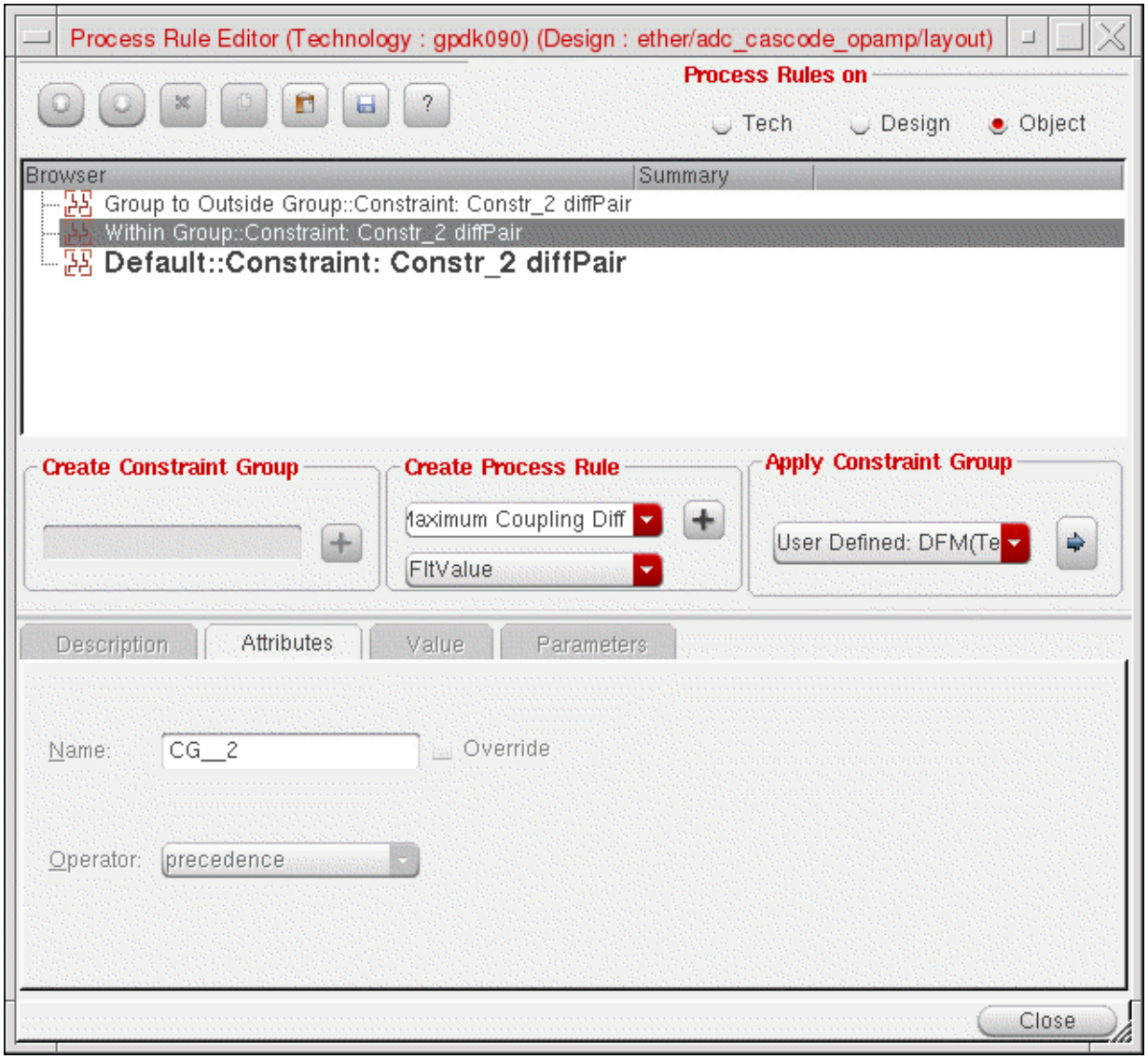
The Process Rule Editor form opens and the diff pair constraint is displayed in the browser window.



2. In the *Create Process Rule* field, select the type of rules and attributes you want to override or set. For example,

   a. To change the *Minimum Spacing Same Layer* for a diff pair,

b. Select *Within Group::Constraint: Constr_`Name`.*



c. Select *Minimum* Spacing Same Layer from the *Create Process Rule* cycle field.

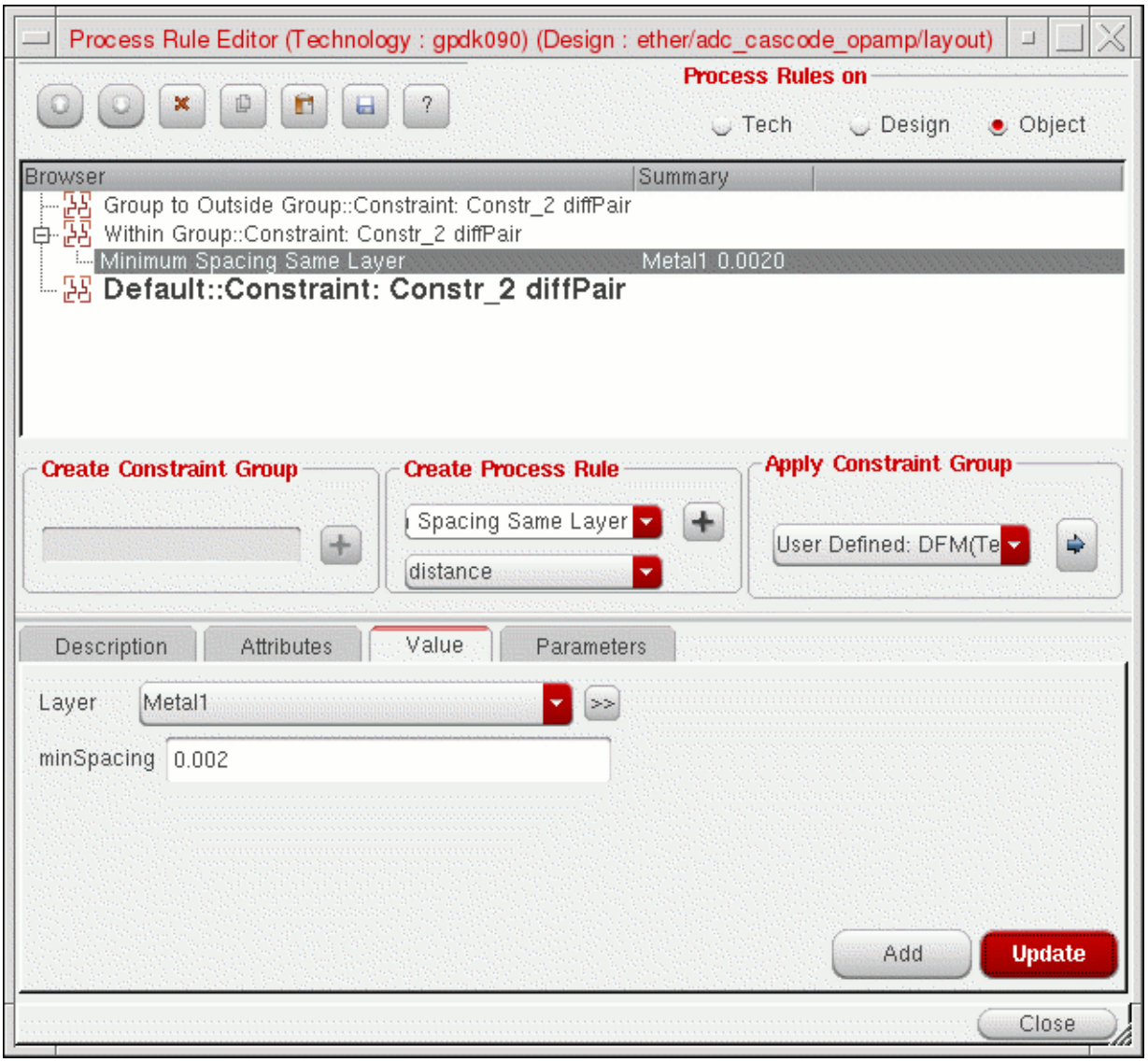
d. Click the (+) button next to the *Create Process Rule Cyclic* field. This adds the rule below the *Within Group::Constraint: Constr_`Name`.*

e. Select the *Minimum Spacing Same Layer* process rule and in the *Value* tab, select the layer of the pair from the *Layer* field.

f. In the *minSpacing* field, specify the minimum spacing for the pair.

g. Click the *Update* button at the bottom of the form.

The value of *Diff Pair* is updated in the Browser window as shown in the following figure.



3. When you are done setting attributes, click the *Close (X)* button.

Shield Routing

In order for shield routing to perform properly, the “shield” net must have a Signal Type of either “power” or “ground”. If not previously defined, you can change the default Signal Type (signal) of a net in the Property Editor.

- Defining Shield Constraints

- Shield Styles
- Default Shielding Types
- Changing Custom Shielding Values in the Process Rules Editor
- Tying Shield Wires
- Pin Escape Methods

Defining Shield Constraints

When defining a shielding constraint, the order in which you select two nets is important since it is order dependent. In the Navigator, first select the shield net, followed by the net to be shielded.

The following instructions assume that there is a schematic view or connectivity source associated with the layout view.

1. From the layout cellview, select *Window – Assistants – Constraint Manager*.

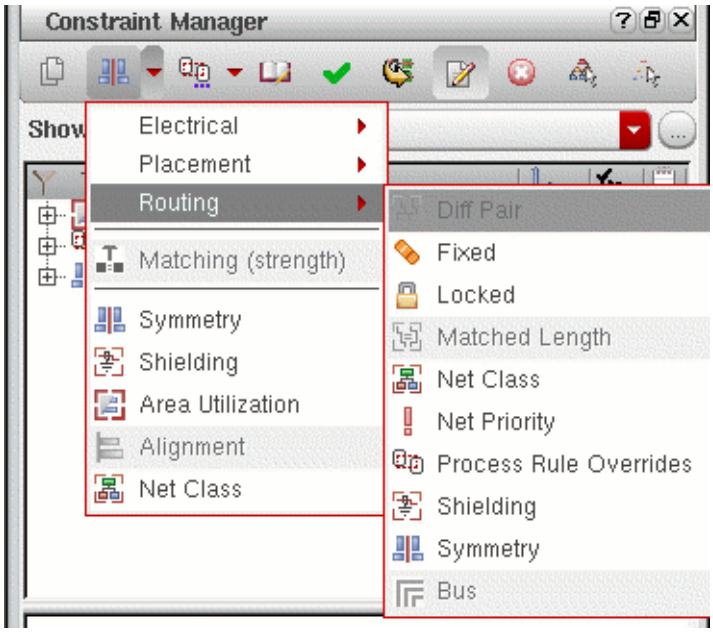
   The *Constraint Manager* assistant opens.

2. From the layout cellview, select *Window – Assistants – Navigator.*

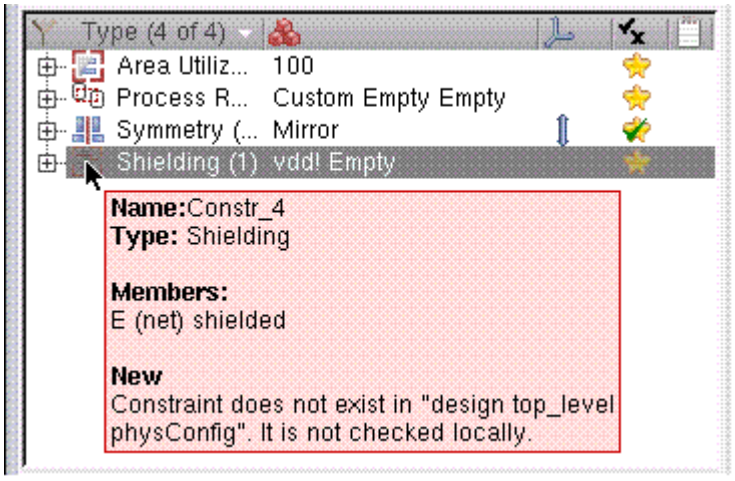
   The *Navigator* assistant opens.

3. In the *Navigator* assistant, select a net by clicking on the net name.

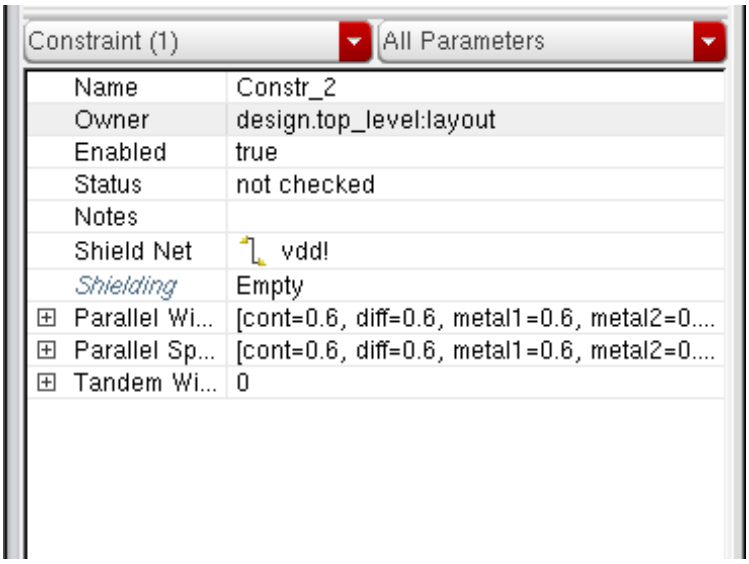
4. With the net selected, create the shield constraint by selecting *Routing – Shielding* from the Constraint Manager pull-down menu.



A shielding constraint is created. The Shielding constraint appears in the constraint browser tree under *Type*.



5. In the lower section of the Constraint Manager window you will see the parameters of shielding constraint.



6. For information about the attributes used to create different types of shield constraints, see Shield Styles.
7. For information about how to generate default shielding constraints, see Default Shielding Types.
8. For information about how to change values in the Process Rules Editor, see Changing Custom Shielding Values in the Process Rules Editor. You can even change values using the Constraint Manager.

   Note: When pushing wires where there are shielded wires, it is possible that the shielded unit will be broken apart by the push operation. To avoid pushing apart shielded units, lock the shielded unit (both shielded and shielding wires) using the Lock Navigator Nets command.

Shield Styles

Once a Shielding constraint is defined you may electively assign additional attributes in the Process Rule Editor. There are three types of shielding styles available depending on which attributes you choose to set in the Process Rule Editor.

- Parallel

Without adding any additional attributes in the Process Rule Editor, the default shield type is a basic parallel shield with a *shieldGap* equal to the applicable layer's `minSpacing` and parallel *shieldWidth* equal to the applicable layer's `minWidth`.

The default values for *shieldGap* and *shieldWidth* are not displayed in the Process Rule Editor. Once the default value is overridden with different values, they are displayed as an attribute.

- Tandem

  A tandem shield (metal above and/or below shield net) can be created by setting the *tandemLayerAbove* and/or *tandemLayerBelow* attributes along with the *tandemWidth* attribute in the Process Rule Editor.

  Note: If *shieldGap* and/or *shieldWidth* are also set, the result will be a coaxial not a tandem shield.

- Coaxial

  A coaxial shield is the combination of a parallel and tandem shield (metal above and/or below the shield net along with same layer shielding along side the shielded net).

  Note: In order to achieve coaxial shielding it is necessary to define the *shieldGap* and *shieldWidth* even if the values are equal to the default `minSpacing` and `minWidth` values.

In all cases, do not define shield values below a layer's `minWidth` value. If you do so, the shield may not appear and no warning message is given.

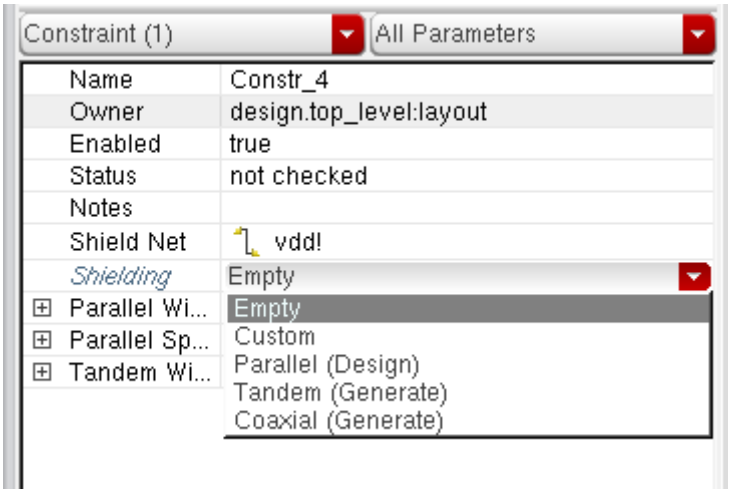
Default Shielding Types

Default shield types of parallel, tandem, and coaxial can be generated based on rule values set in the technology file.

To generate and modify default shielding constraints, follow these steps.

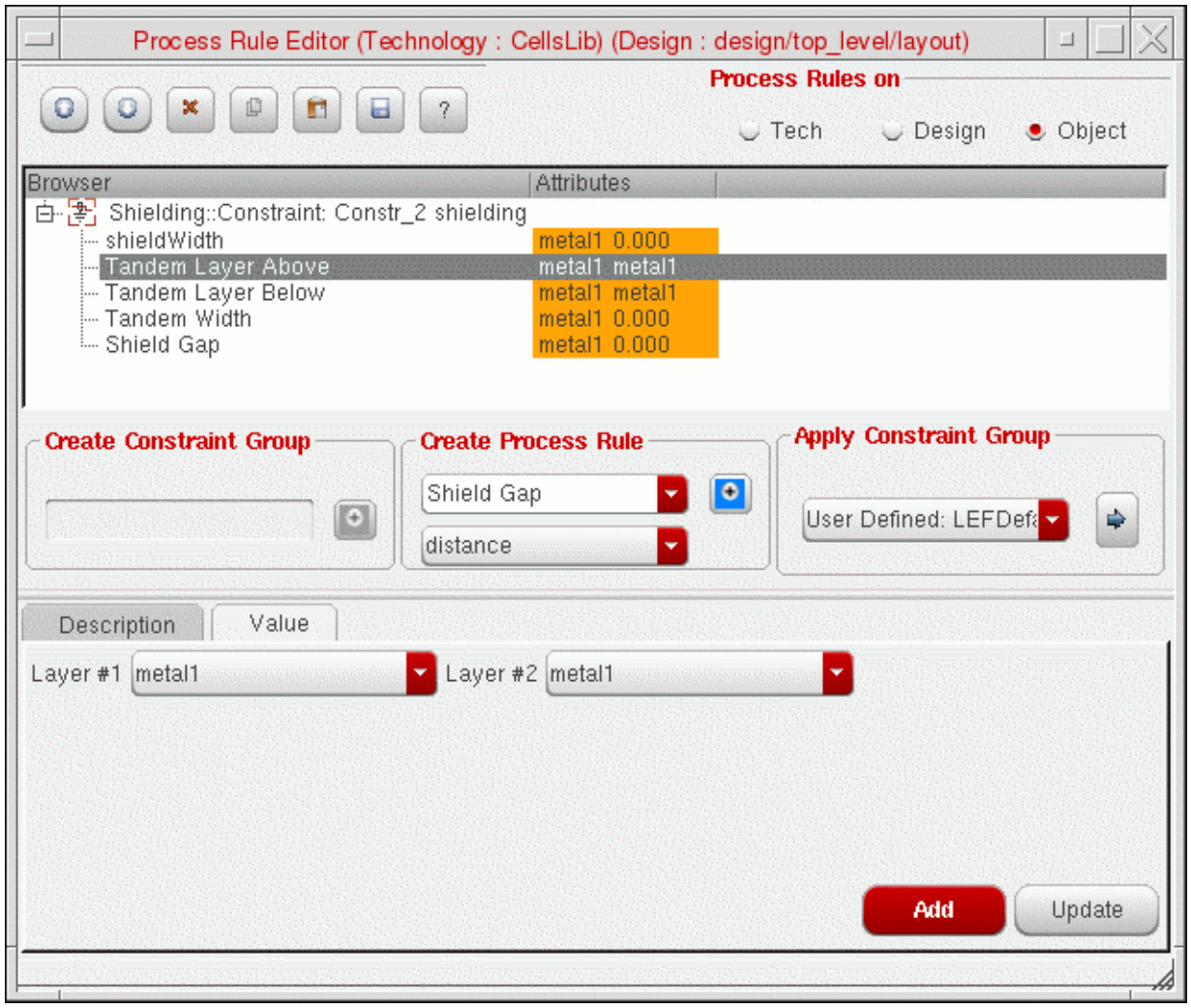
1. Select the *Shielding* constraint in the In the Constraint Manager constraint browser tree under *Type*.
2. In the lower Constraint Manager parameters window, click the *Shielding* pull-down window and select the desired type of shielding constraint.

The *Generate* label indicates that the predefined constraint group that has not yet been generated for the selected shielding constraint.

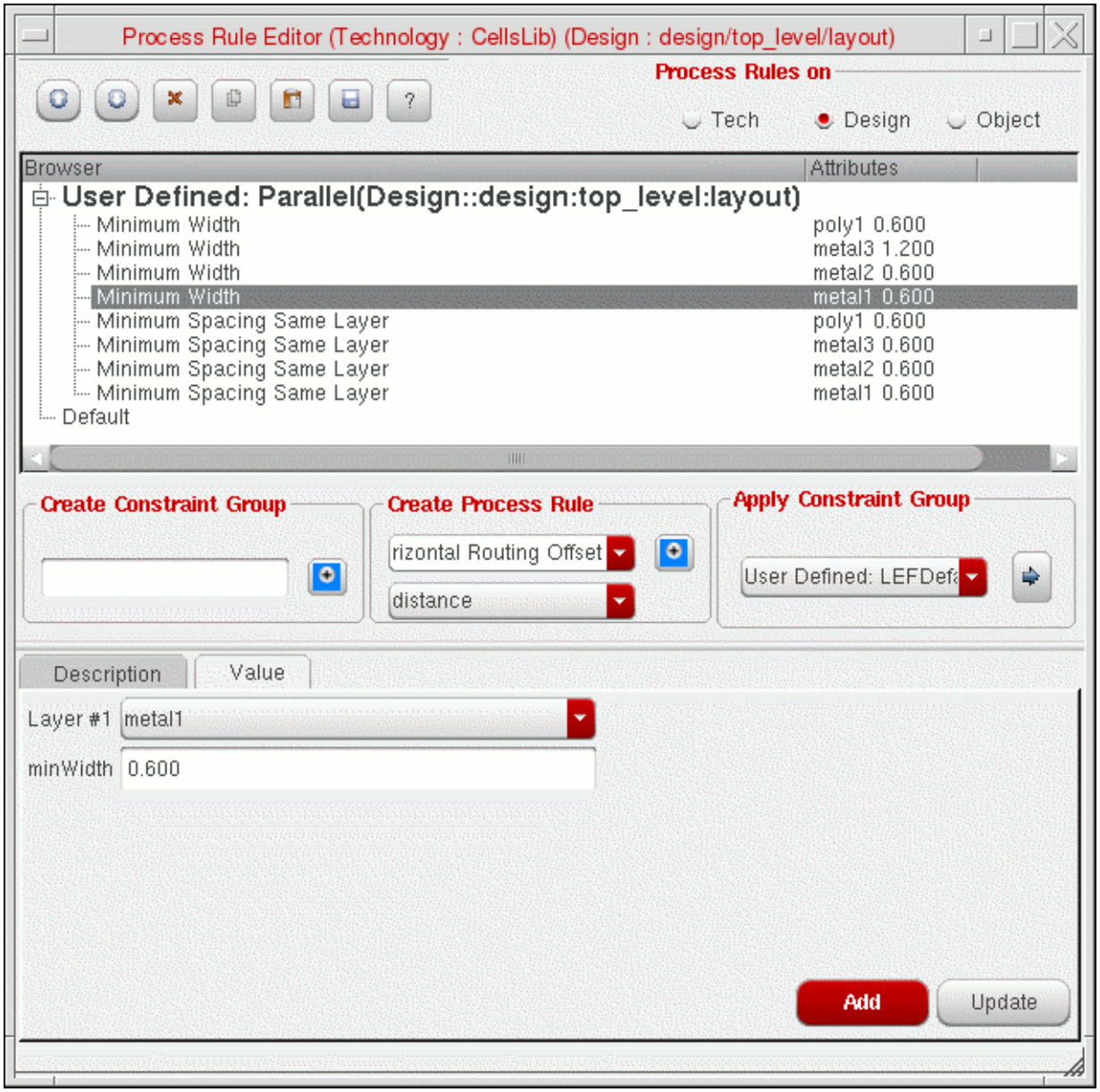


3. Open the *Process Rules Editor* to view or change the default values.

4. The default values are displayed for each type of shielding constraint that has been defined.



5. To edit values, click the *Design* button in the Process Rules Editor.



6. You can edit the constraint group values directly or use the copy and paste functions to make the predefined constraint group more specific for your design.

   Once a shielding type has been selected and applied, you can change the constraint type by clicking on the constraint in the Constraint Manager constraint browser tree under *Type*. Click the *Shielding* pull-down window and select a different type of shielding constraint to be applied.

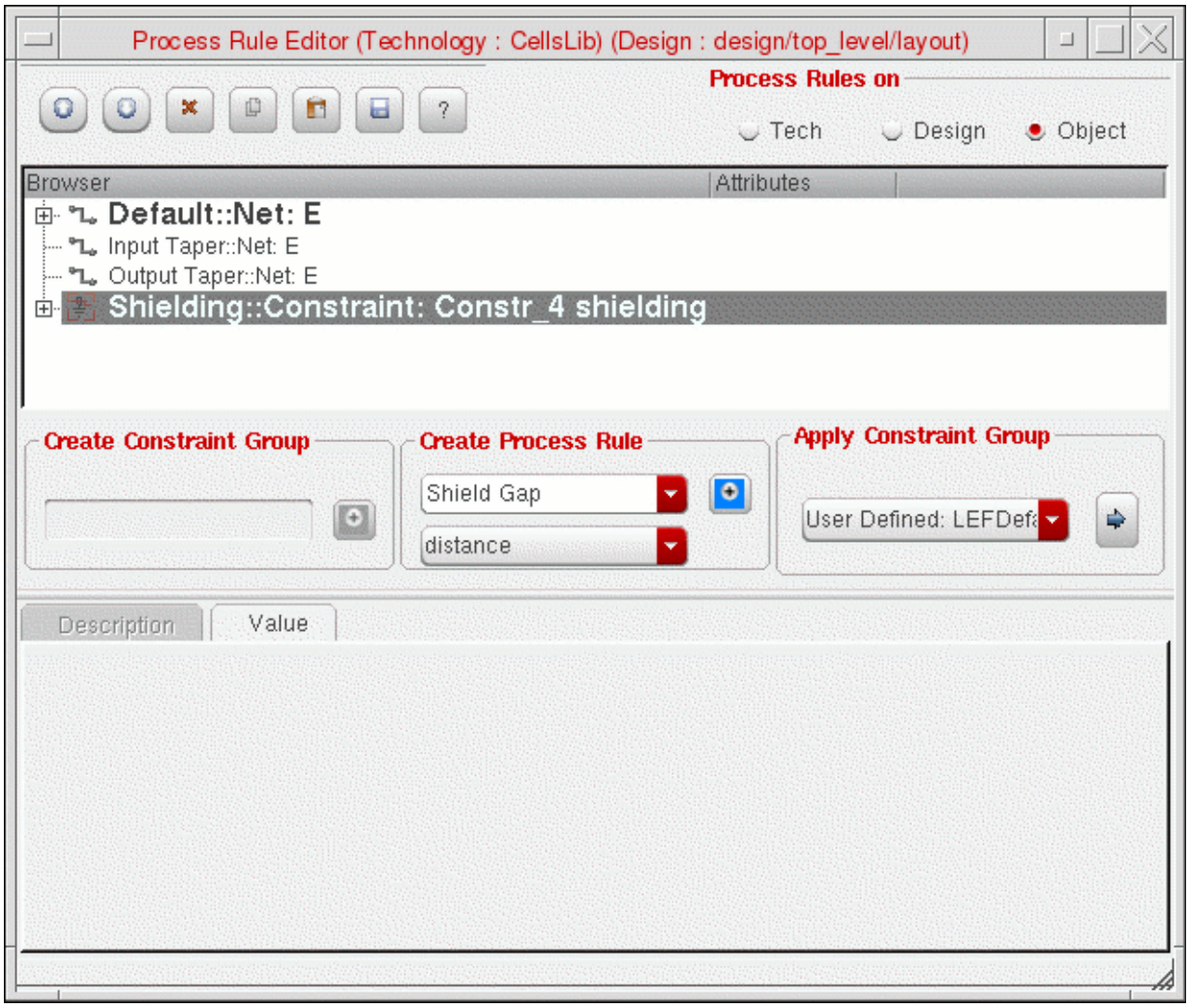
Changing Custom Shielding Values in the Process Rules Editor

Once a *Shielding* constraint is defined you can assign attributes in the Process Rule Editor.

To set or change the shielding style and values,

1. Select the shielding constraint In the Constraint Manager window and click the Process Rules Editor icon.

   The Process Rule Editor form opens and the shielding constraint is displayed in the browser window.



2. In the *Create Process Rule* field, select the type of rules and attributes you want to override or set. For example,

a. To change the *shieldGap* for a parallel shield, select *shieldGap* from the *Create Process Rule* cycle field.

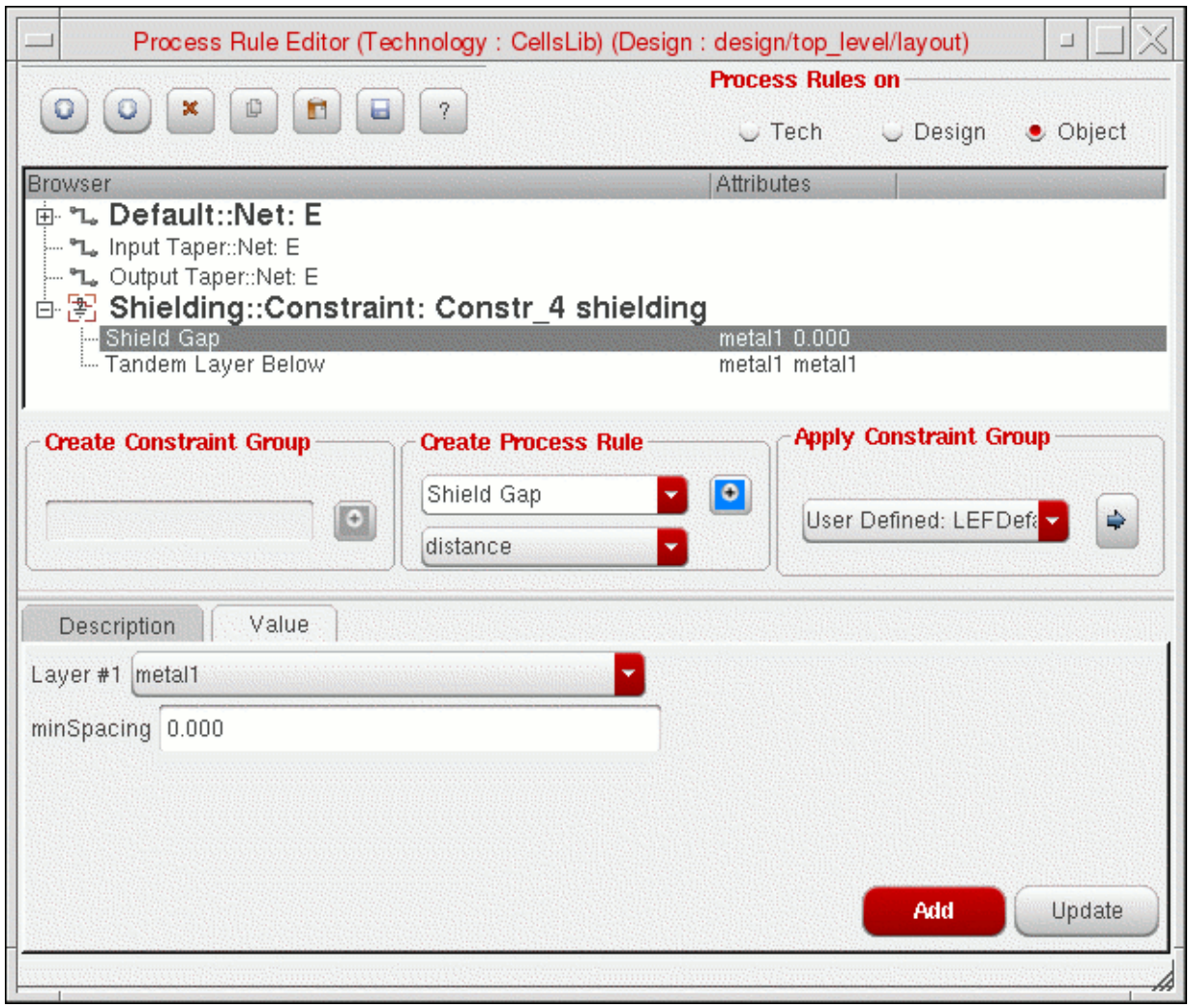
b. Leave *distance* as the value and click the blue + button to add the attribute to the shield constraint.

c. From the *Layer* field, select the layer of the shield.

d. From the *minSpacing* field, select the gap spacing for the shield.

e. Click the *Update* button at the bottom of the form.

The value of *shieldGap* is updated in the Browser window.

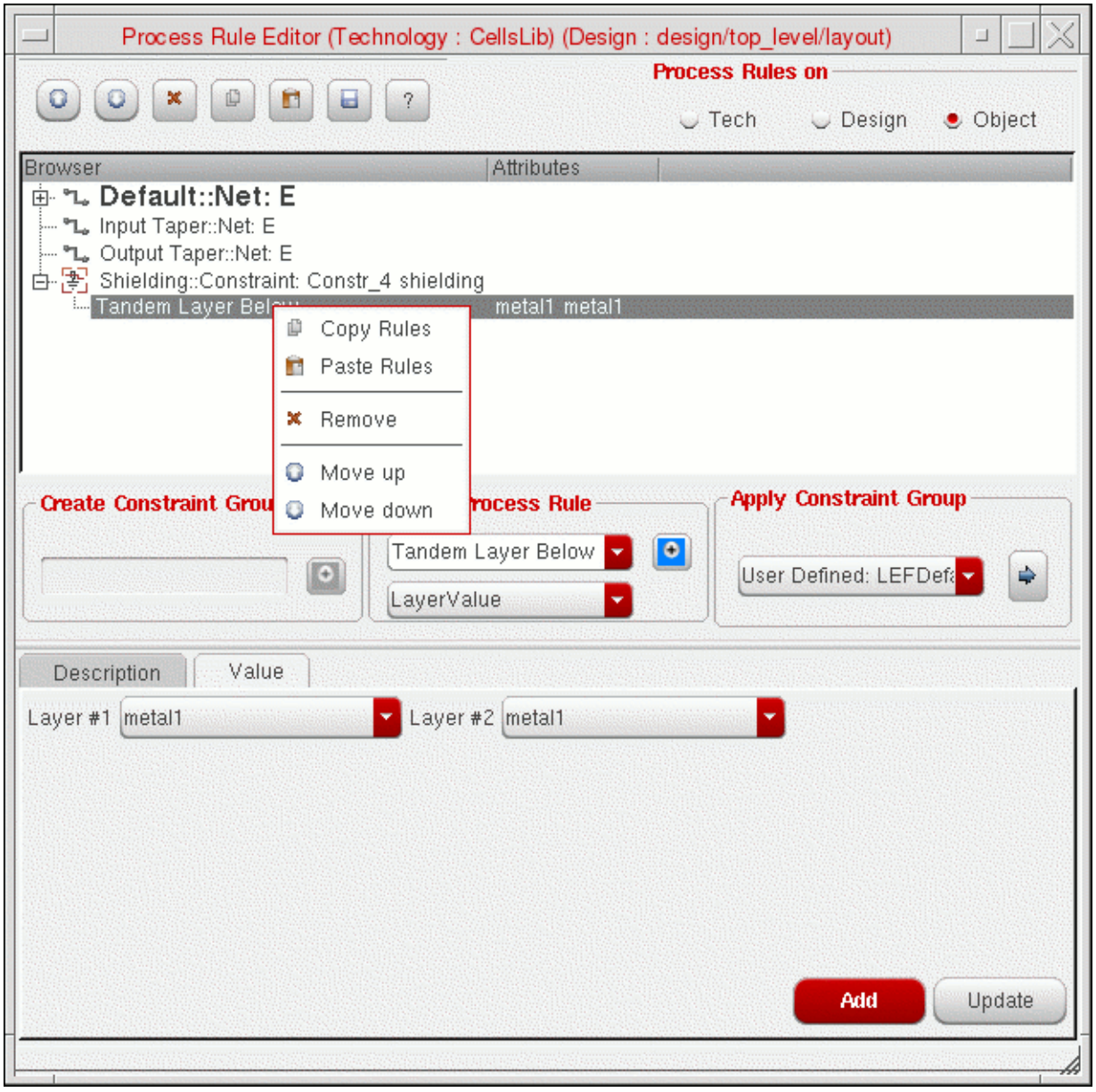


3. When you are done setting attributes, click the *Close (X)* button.

For information about what values to set to create different types of shielding constraints, see Shield Styles.

If you have many nets in your design that require shielding, you can optionally create a shielding constraint group in the technology file and apply the shielding constraint group, using the process rules editor, to the nets requiring the same set of constraints. See *Applying a Constraint Group* for more information.

To remove or copy attributes of a shielding constraint in the Process Rules Editor, click the right mouse button over the attribute to bring up the context sensitive form.



For more information about how to copy constraints, see *Copying an Existing Process Rule*.

Tying Shield Wires

Shield nets are not tied into power/ground by default. Once shield routing (automatic or manual routing) is complete, use the *Power Router – Via* options to add vias and wires to tie in the shields.

The *Create – Wire* command does not tie shield nets to shield wires in the design. To tie shield wires to shield nets, use the *Power Router – Tie Shield* option.

Pin Escape Methods

The router models a net with shielding as having a wide spacing requirement, as it takes the widths of the shields into account. When the router tries to escape from a pin, wide spacing is required if you do not specify a taper constraint group. If the neighboring pins have tighter spacing than is required by the shielded net, minimum spacing for example, the router has trouble escaping from the pin based on the default behavior of needing the wider spacing clearance.

To work around this default behavior, define a taper constraint group with a different name than the default constraint group used by the wire. The router will then use the spacing specified in the taper constraint group for pin escape rather than the wide spacing of the shielded net.

The following is an example of a taper constraint group.

```
constraintGroups(

 ;( group[override] )
 ;( --------------- )
  ( "virtuosoDefaultTaper"nil    "taper"

    interconnect(
     ( validLayers   (Poly Metal1 ) )
     ( validVias     (M1_PO ) )
    ) ;interconnect

    spacings(
     ( taperHalo  1.4 )
    ) ;spacings
```

```
  ) ;virtuosoDefaultTaper
) ;constraintGroups
```

Use the Technology File Manager to load the constraint group.

Matched Length Routing

Defining a Matched Length Constraint

The following steps describe how to create a basic matched length constraint in the layout view.

These instructions assume that there is a schematic view or connectivity source associated with the layout view.

1. From the layout cellview, select *Window – Assistants – Constraint Manager*.

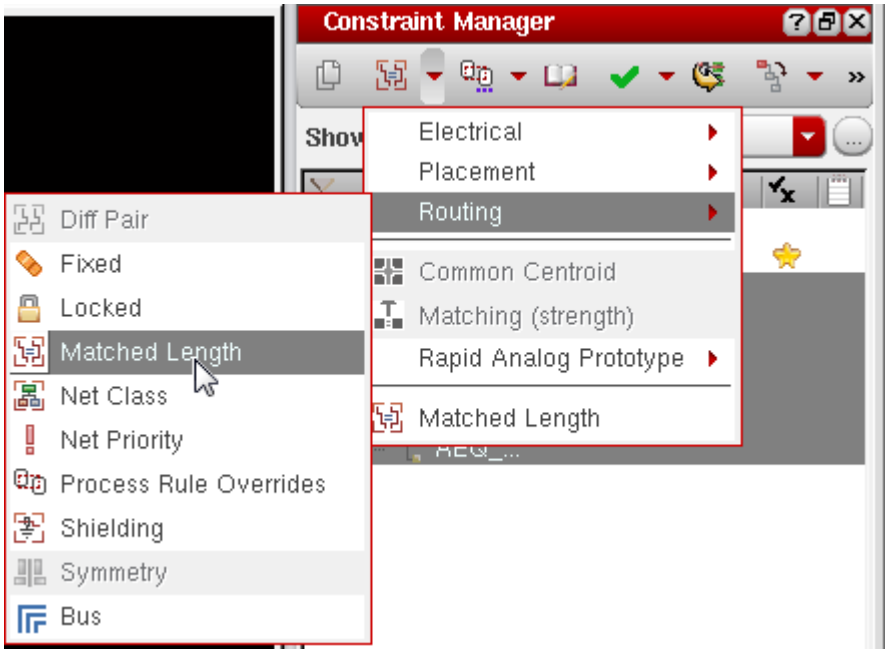
   The *Constraint Manager* assistant opens.

2. From the layout cellview, select *Window – Assistants – Navigator.*

   The *Navigator* assistant opens.

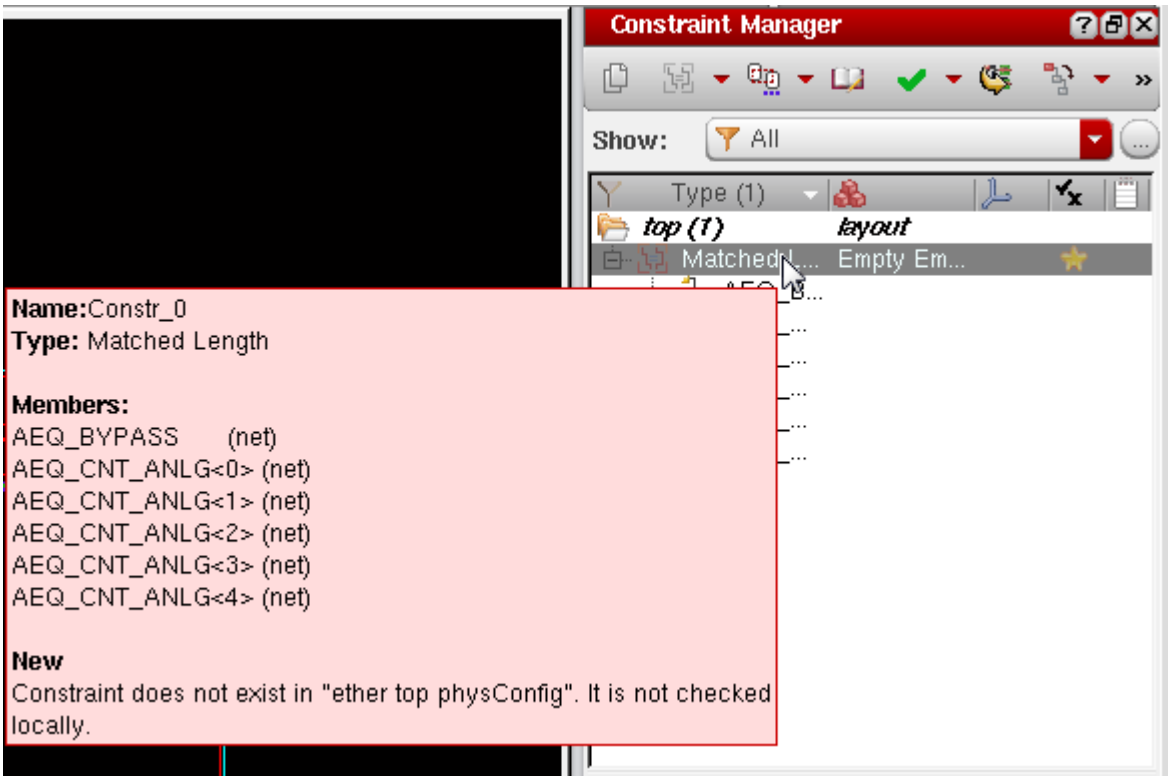
3. From the *Navigator* assistant, click a net name to select the net.

4. With the net selected, choose *Routing - Matched Length* from the Constraint Manager pull-down menu to create the matched length constraint.

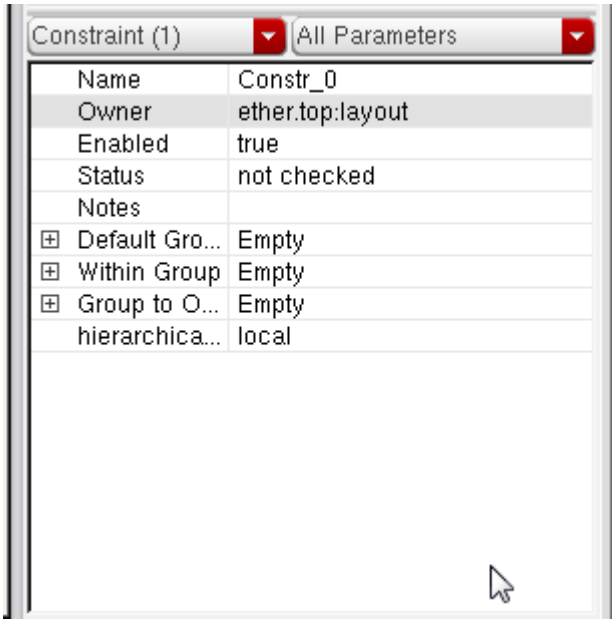


A matched length constraint with default values is created.

5. Hover the cursor over the *Matched Length* constraint in the Constraint Manager window to confirm the members of the constraint.



6. In the lower section of the Constraint Manager window you will see the default matched length constraint.



7. Click the plus(*+)* sign next to the *Within Group* parameter. It displays the *matchTolerance* and *Tolerance* parameters.

   - *matchTolerance*

     (Applies only for net groups of type net_match) Errors are reported on nets that are shorter than the longest net of the group minus the `matchTolerance`. Lengths are based on the net's routes. If both `matchTolerance` and `Tolerance` are set, `matchTolerance` is used. However, for interoperability, you should only use `Tolerance`, not `matchTolerance`. The default setting is two times the pitch or four times the gap space is used for the tolerance. The default value of this parameter in the Constraint Manager is `0`.

   - *Tolerance*

     (Applies only for net groups of type net_match) Errors are reported on nets that are shorter than the longest net of the group minus the `Tolerance` percentage. Lengths are based on the net's routes. If both `matchTolerance` and `Tolerance` are set, `Tolerance` is ignored. However, for interoperability, you should only use Tolerance, not `matchTolerance`. The default setting is two times the pitch or four times the gap space is used for the `tolerance`.

     For example, if `Tolerance` is 10, or 10%, and the longest net is 20 microns, then a net length of 18.2 microns would be acceptable because it is greater than the

longest net minus 10% of 20 (2 microns). A net length less than 18 microns would fail. The default value of this parameter in the Contraint Manager is 20.

You can now directly change the parameters of the Match Length constraint using the Constraint Manager. For more information on Match Length constraint and its parameters, refer to Matched Length Constraint in the *Virtuoso Unified Custom Constraints User Guide*.

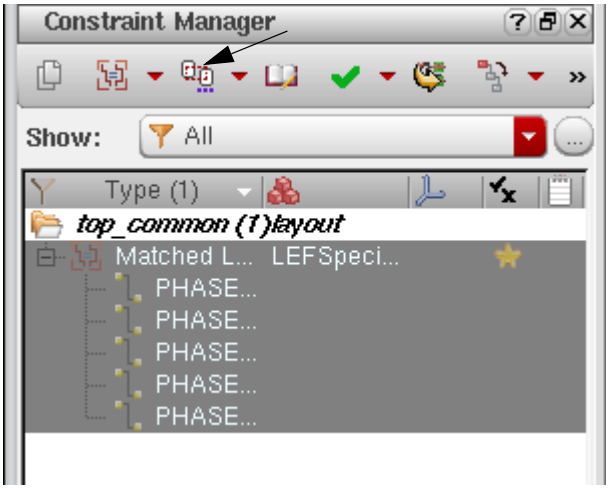
For information about how to change values in the Process Rules Editor, see Changing Matched Length Values in the Process Rules Editor.

Changing Matched Length Values in the Process Rules Editor

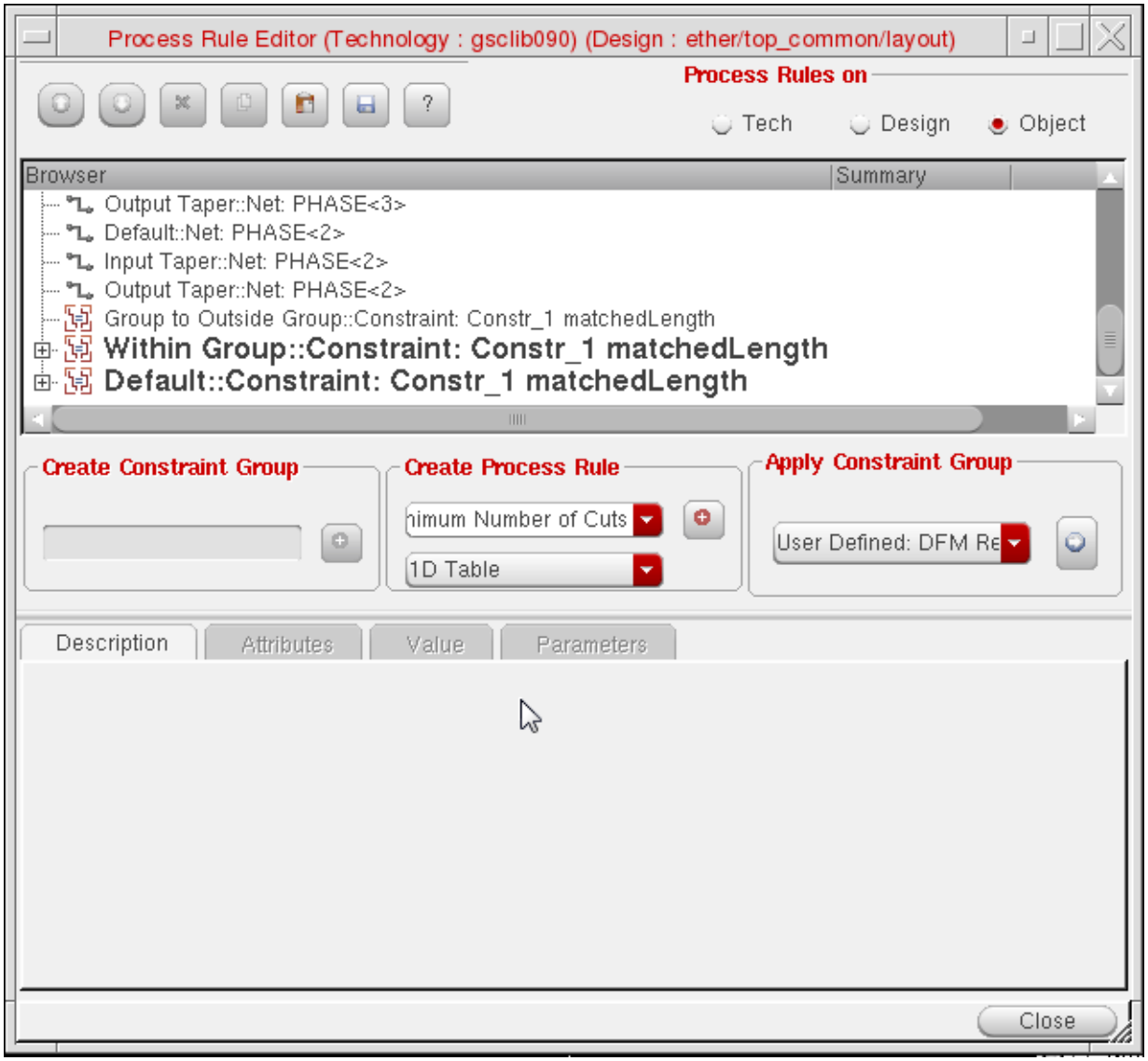
Once a *Matched Length* constraint is defined you can optionally assign additional attributes in the Process Rule Editor.

To set or change the values,

1. Select the *Matched Length* constraint In the Constraint Manager window and click the Process Rules Editor icon.



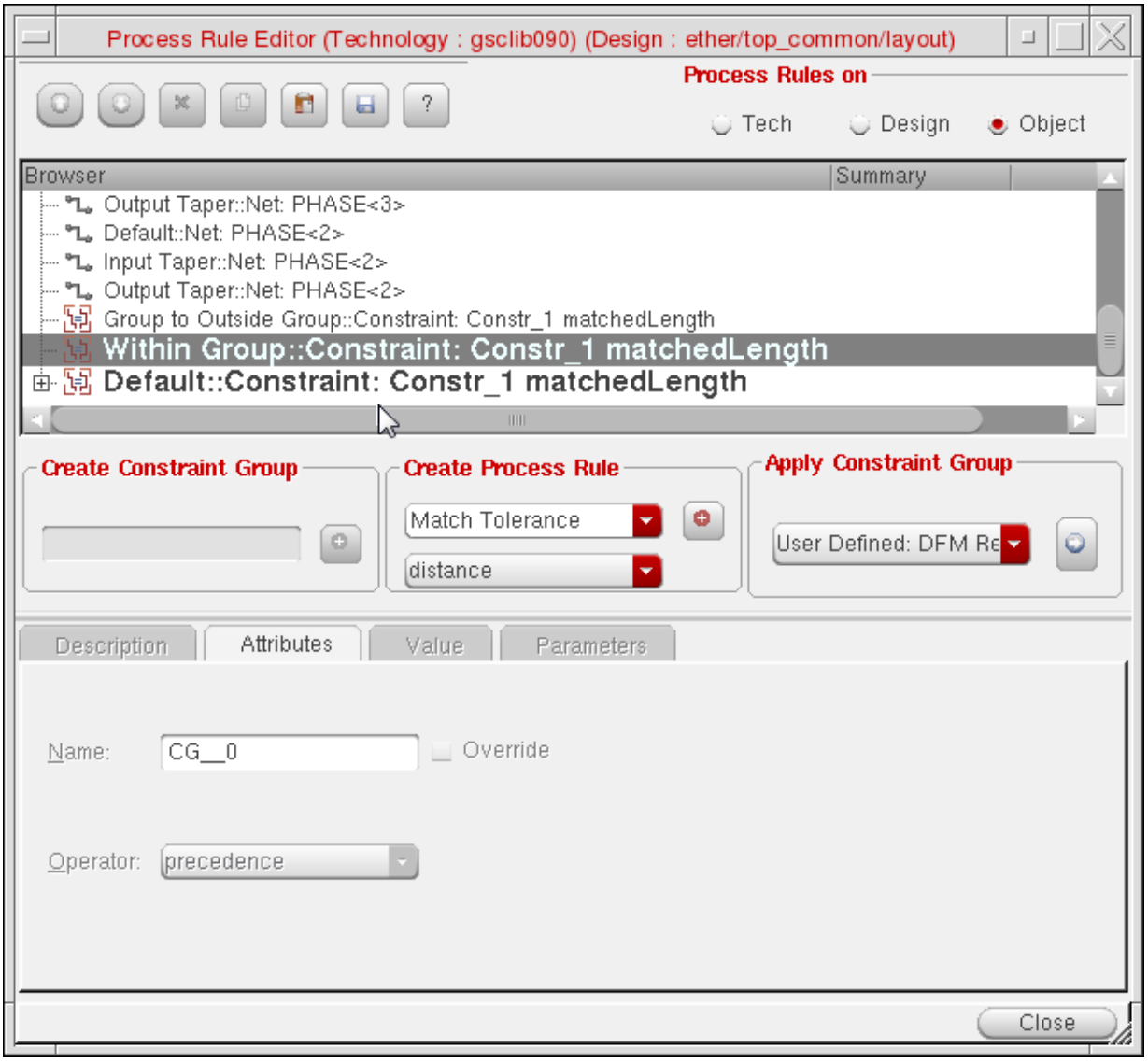
The Process Rule Editor form opens and the Matched Length constraint is displayed in the browser window.



2. In the *Create Process Rule* field, select the type of rules and attributes you want to override or set. For example,

   a. To change the *Match Tolerance* for a matched length constraint,

b. Select *Within Group::Constraint: Constr_Name*.



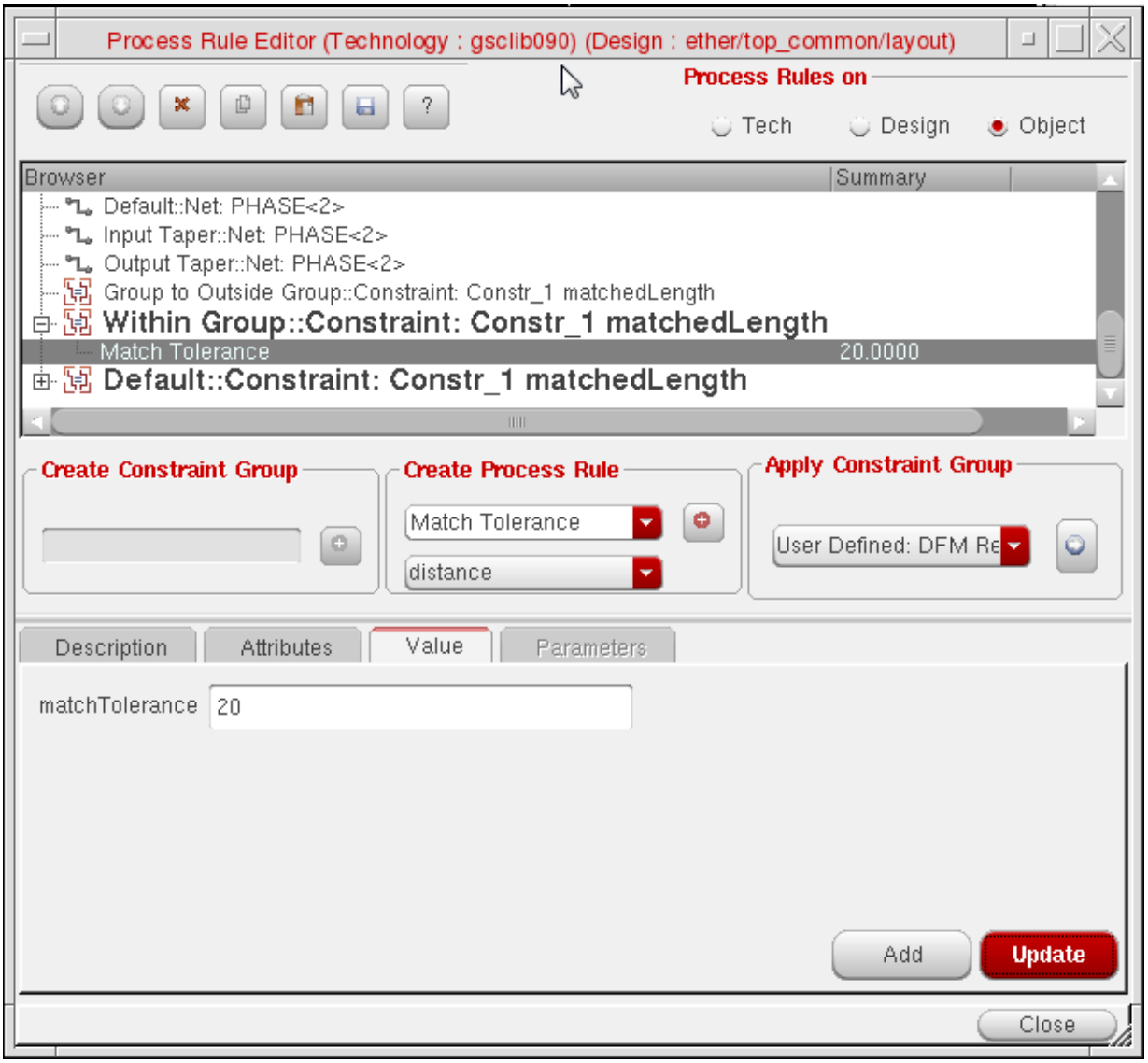
c. Select *Match Tolerance* from the *Create Process Rule* cycle field.

d. Click the (+) button next to the *Create Process Rule Cyclic* field. This adds the rule below the *Within Group::Constraint: Constr_Name*.

e. Select the *Match Tolerance* process rule and in the *Value* tab, specify the value in the *matchTolerance* field.

f. Click the *Update* button at the bottom of the form.

The value of *Match Tolerance* is updated in the Browser window as shown in the following figure.

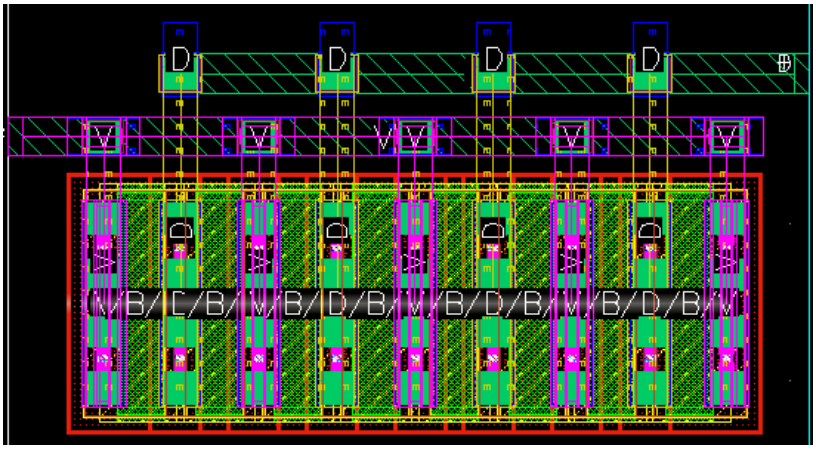


3. When you are done setting attributes, click the *Close (X)* button.

9

Using the Pin to Trunk Routing

The Pin to Trunk routing style is used to connect an individual pin to a trunk. Only the connections that are within the scope of a trunk are routed. The remaining opens are completed by the *Custom / Digital (MST)* routing style or manually. The Pin to Trunk routing style is usually used when a spine structure is required.



Pin to Trunk routing allows you to add connections from the pins of devices and macro instances to existing trunks. The Pin to Trunk routing is performed after composing trunks. For more information on composing trunks, see the Composing Trunks routing step.

This chapter describes the features and functionality of the Virtuoso Space-based Router Pin to Trunk routing. It covers the Pin to Trunk toolbar and the various options for controlling the behavior of Pin to Trunk routing. To specify the options, use the *Pin to Trunk* subform. The *Pin to Trunk* subform is displayed when the *Pin to Trunk* routing style is selected from the *Routing Preferences* tree in the Virtuoso Space-based Router Options form.

The following sections are included:

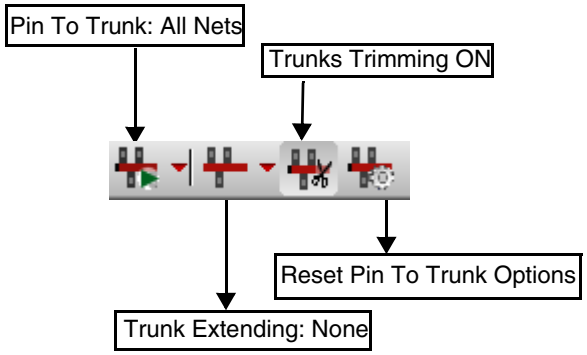
- Pin To Trunk Toolbar
- Behavior of Pin to Trunk Toolbar Icons
- Pin To trunk Routing Options

- Highlighting Trunks
- Using the Finish Trunk Command

Pin To Trunk Toolbar

The *Pin To Trunk* toolbar consists of a set of icons that provide quick access to commonly used options settings or action selections in Pin to Trunk routing, without having to use the *Pin To Trunk* subform in the Virtuoso Space-based Router Options form.

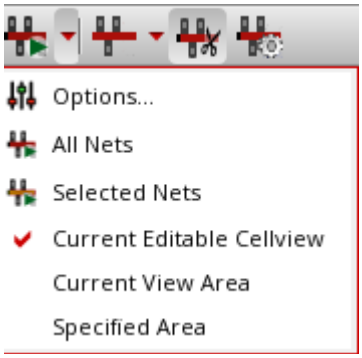
The *Pin To Trunk* toolbar is not available by default in the layout window. To display the toolbar, choose *Window – Toolbars – Pin To Trunk*. You can use this toolbar to access the options for Pin to Trunk routing shown below.



- Pin to Trunk: All Nets
- Extend Trunks
- Trunks Trimming
- Reset Pin To Trunk Options

Pin to Trunk: All Nets

The *Pin to Trunk* icon lets you set up the routing scope and run the *Pin to Trunk* command on the specified routing scope either on all the nets or selected nets. Click the arrow next to the icon to view the Pin to Trunk routing scope options available on the drop-down menu associated with the icon.



- Options

  Click *Options* to view in the Virtuoso Space-based Router Options form all the choices related to the Pin to Trunk routing style that is currently set. You can then select any of the option from the *Pin to Trunk* tree to set up specific settings.

- All Nets

  Lets you run Pin to Trunk routing style on all the nets in the current layout. This functionality is the same as the *All* button in the *Automatic* section of the Wire Assistant.

- Selected Nets

  Lets you run Pin to Trunk routing style on the selected nets in the current layout. This menu item is enabled only if at least one net or one trunk is selected in the layout. This functionality is the same as the *Selected* button in the *Automatic* section of the Wire Assistant. To select one or more nets, use Navigator Assistant. As soon as a net is selected in Navigator Assistant, the *Selected Nets* menu item is enabled.

- Current Editable Cellview

  The functionality of this option is the same as the *Current Editable Cellview* radio button in the *Scope* group box of the *Pin to Trunk (Default)* subform in the Virtuoso Space-based Router Options form. For more information, see Current Editable Cellview.

- Current View Area

The functionality of this option is the same as the *Current View Area* radio button in the *Scope* group box of the *Pin to Trunk* subform in the Virtuoso Space-based Router Options form. For more information, see Current View Area.

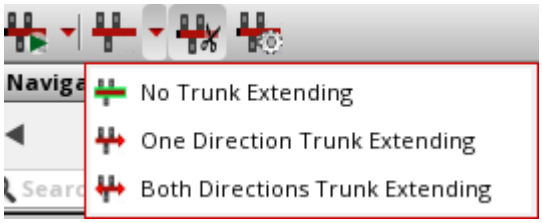
- Specified Area

  The functionality of this option is the same as the *Specified Area* radio button in the *Scope* group box of the *Pin to Trunk* subform in the Virtuoso Space-based Router Options form. For more information, see Specified Area.

Note: The *Current Editable Cellview*, *Current View Area*, and *Specified Area* options behave as a group of radio buttons; that is, you can only select one of the options at a given time.

Extend Trunks

The *Extend Trunk* icon lets you set up the options for the router to automatically extend a trunk. Click the arrow next to the icon to view the trunk extension options available on the drop-down menu associated with the icon.



- No Trunk Extending

  This is the default option. When this option is selected a trunk cannot be extended and the length of the trunks is preserved.

- One Direction Trunk Extending

  Click *One Direction Trunk Extending* to extend the trunk in a single direction until the trunk touches the PR boundary or is at a point that is at the minimum spacing away from an obstruction on the same layer. For *Finish Trunk* command, it means that the last point of the last trunk created by the *Create Wire* command is extended further. However, for auto routing, the option selected from the *One Direction for Auto Route* drop-down list in the *Extend Trunk* section of the *Trunk* subform determines which end of a trunk should be extended. For more information, see Extend Trunk.

- Both Directions Trunk Extending

  Click *Both Directions Trunk Extending* to extend the trunk in both the directions until the trunk touches the PR boundary or is at a point that is at the minimum spacing away from an obstruction on the same layer. Both ends of a trunk are extended. The functionality of this option is the same as the *Both* option selected from the *Direction* drop-down list in the *Extend Trunk* section of the *Modification* group box of the *Trunk* subform. For more information, see Extend Trunk.

Note: The *Extend Trunk* section in the *Modification* group box of the *Trunk* subform is enabled only when the *One Direction Trunk Extending* or *Both Directions Trunk Extending* option is selected from the *Trunk Extending* drop-down menu.

Trunks Trimming

The *Trunks Trimming* icon turns on or off the *Trunk Trimming* step in the *Pin to Trunk* subform. By default, the *Trunk Trimming* step is ON. The functionality of this icon is the same as the *Trunk Trimming* step in the *Routing Steps* group box of the *Pin to Trunk* subform.

Reset Pin To Trunk Options

Clicking the *Reset Pin to Trunk Options* icon resets all the Pin to Trunk related DFII environment variables to their default `.cdsenv` values.

Important

If you notice the mismatch icon () after you click an option of a *Pin to Trunk* toolbar, it indicates that the value of at least one of the options in the *Pin to Trunk* subform is no longer matching its default value. The tooltip on the icon is displayed as *Option value mismatched*. To reset all the Pin to Trunk environment variables to the default values, click the icon button.

Behavior of Pin to Trunk Toolbar Icons

The section covers the behavior of the pin to trunk toolbar icons.

- Behavior of the Pin to Trunk Options Preset Scripts Icon
- Behavior of the Extend Trunks Icon

Behavior of the Pin to Trunk Options Preset Scripts Icon

In the *Pin To Trunk* toolbar, the behavior of the *Pin To Trunk Options Preset Scripts* icon is based on the last action of either the *Default*, *Gate Strapping*, or *Twigs on Pin Layer* options preset script. By default, the *Pin To Trunk* toolbar resets all the Pin to Trunk options to their default values and the associated tooltip is displayed on the toolbar. However, after an options preset script is run, the tooltip on the icon automatically changes to the tooltip associated with the executed options preset script. The tooltip on the icon informs you of the last options preset script that was run.

For example, the *Pin to Trunk Options Preset Scripts* icon in Fig A shows the default action. The Fig B displays the selection of the *Gate Strapping* options preset script. In Fig C, the tooltip of the icon changes from *Pin to Trunk default options* to *Pin to Trunk Gate Strapping options script.*

Fig A

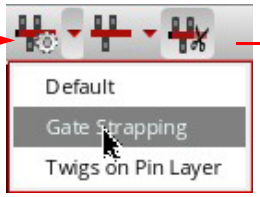
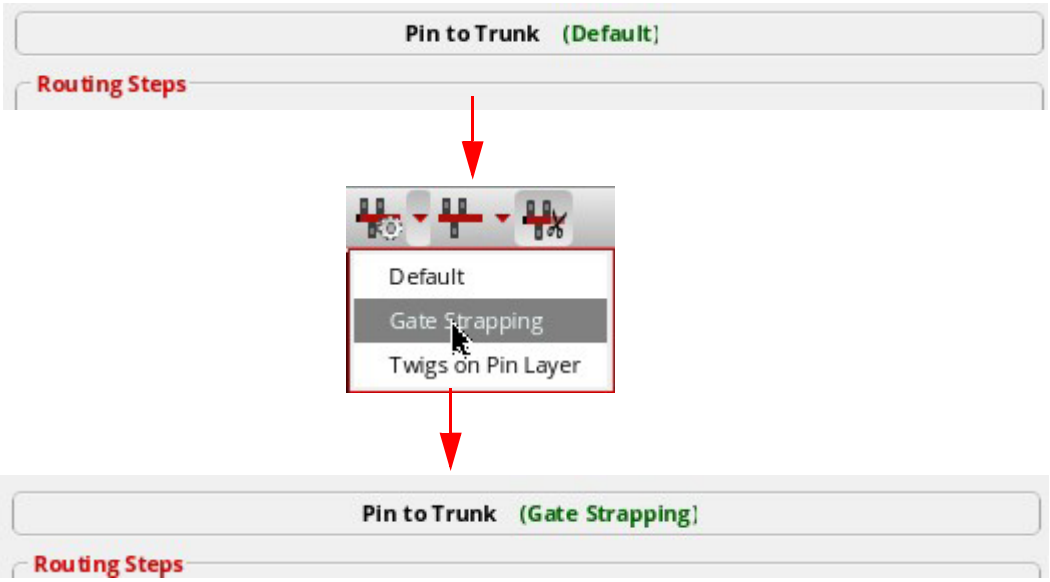


Fig B

Fig C

In addition, after the options preset script is run, the name of the options preset script is displayed along with Pin to Trunk on the Virtuoso Space-based Router Options form, as shown in the following figure.



Behavior of the Extend Trunks Icon

The behavior of the *Extend Trunks* icon is based on the last action of one of the following: *No trunk Extending, One Direction Trunk Extending, or Both Directions Trunk Extending*. By default, the trunk extension is set to *None* and the associated icon is displayed on the toolbar, as shown in Fig A. However, when you click the *One Direction Trunk Extending* menu item, the selected trunk is extended in one direction and the toolbar icon changes to *Extend Trunks: One Direction*, as shown in Fig B. Similarly, the toolbar icon changes to *Extend Trunks: Both Directions* when the *Both Directions Trunk Extending* menu item is selected, as shown in Fig C. The tooltip on the icon informs you of the last action that was performed for trunk extension.

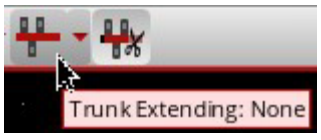


Fig A

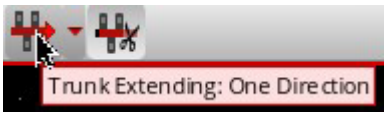


Fig B

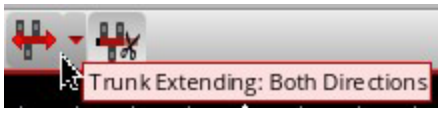
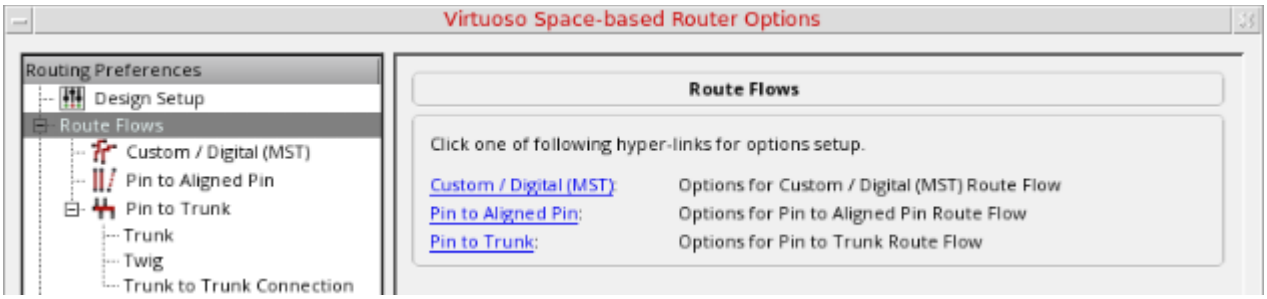


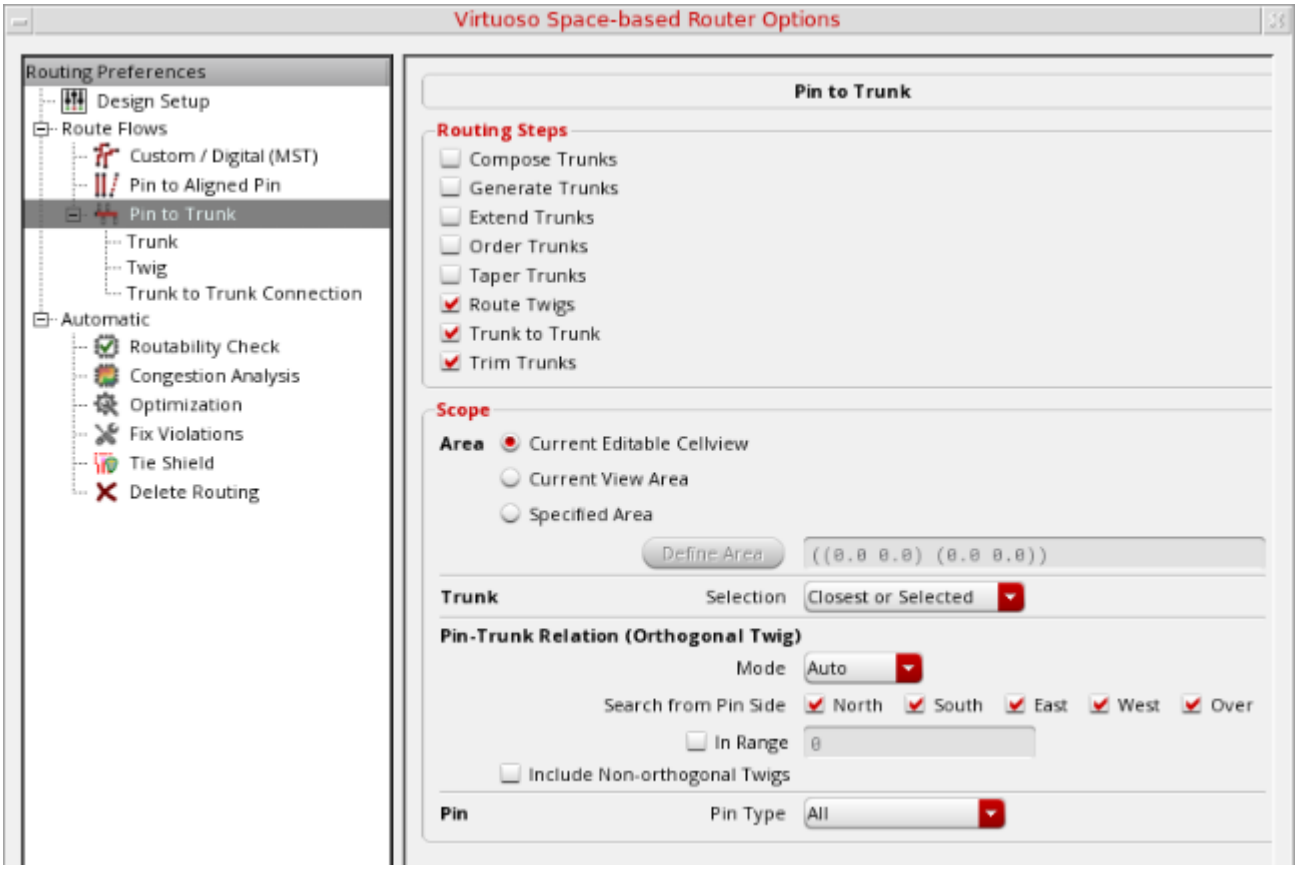
Fig C

Pin To trunk Routing Options

Click *Auto Route Styles*. This displays the *Auto Route Styles* subform. The subform consists of the hyperlinks to quickly navigate to the available routing styles, as shown in the following figure.



To specify different options for controlling the behavior of Pin to Trunk routing, click *Pin to Trunk* in the *Auto Route Styles* tree. The *Pin to Trunk* subform is displayed, as shown in the following figure.



Use the *Pin to Trunk* subform to specify the Pin to Trunk routing options. You can specify the required options for Pin to Trunk routing that are explained in the following sections:

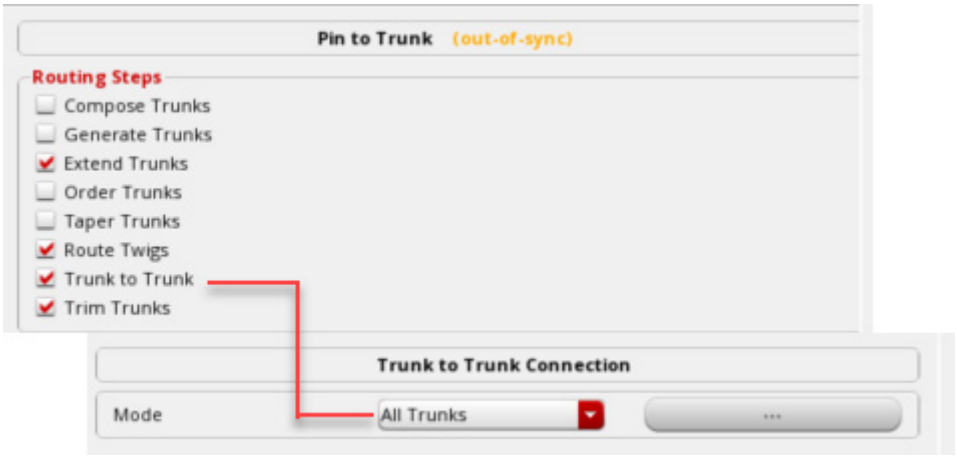
- Specifying Routing Steps
- Specifying Routing Scope
- Specifying Trunk Options
- Specifying Twig Options
- Specifying Trunk to Trunk Connections

Video

For a short demonstration on how to use the new and enhanced Pin to trunk interactive and automatic routing features, see Pin to Trunk Usage.

Specifying Routing Steps

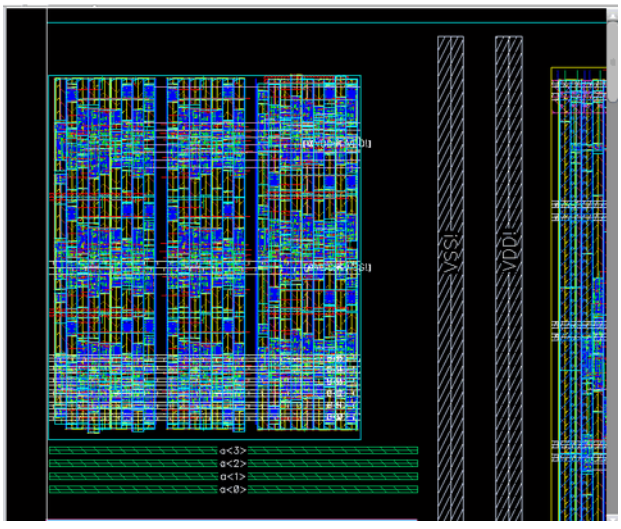
From the *Routing Steps* group box, you can select the routing steps that should be run sequentially. Each step controls the availability of the options on other *Pin to Trunk* subforms. If an option on a *Pin to Trunk* subform is disabled, it means that the option is dependent upon one of the routing steps which has been deselected. The tooltip on the disabled option informs you about the routing step that is required in order to enable the option. For example, when the *Trunk to Trunk Routing* step is deselected, the *Mode* option in the *Trunk to Trunk Connection* subform is disabled, as shown in the following figure.



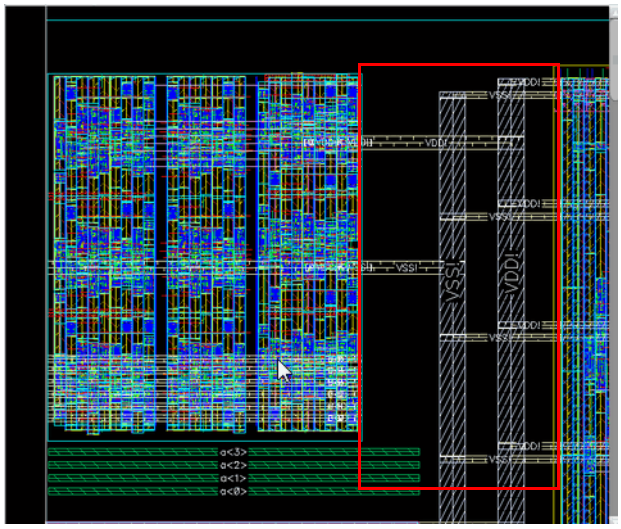
There are eight routing steps. Out of the eight routing steps only four are selected by default. You can select or deselect any one or more of the routing step at a given time. However, ensure at least one step remains selected.

Composing Trunks

Select the *Compose Trunks* routing step if you want to convert the selected set of shapes into a trunk object. A message in the CIW summarizes the objects that are converted to form a trunk. This trunk object can then be used to perform Pin to Trunk routing. The Pin to Trunk router is started by default when you run the route command from the *Net* context-sensitive menu after selecting the trunk net in the *Navigator* assistant. The Pin to Trunk routing step is performed after composing trunks, as shown in the following figure.



Compose Trunks step is OFF



Compose Trunks step in ON

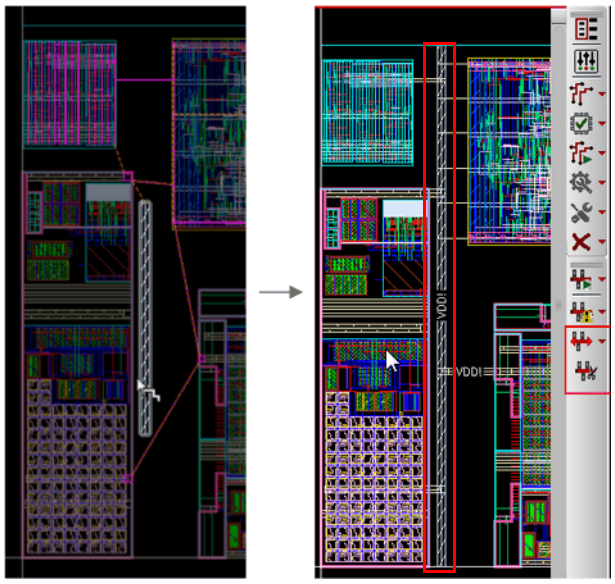
Environment Variable: autoComposeTrunk

Generating Trunks

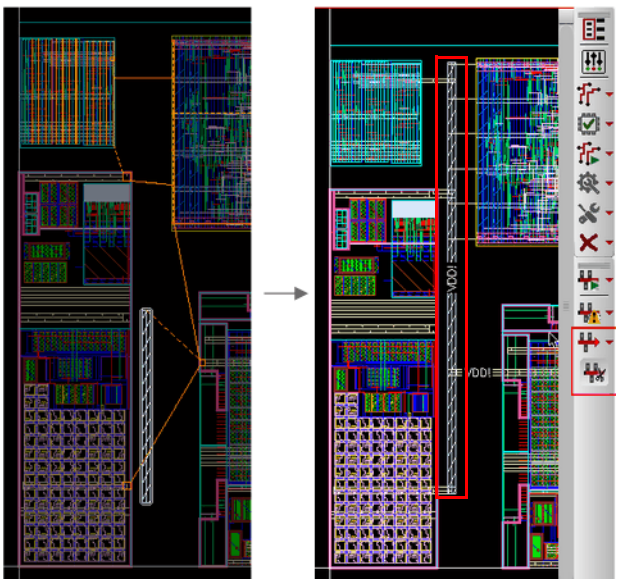
Select the Generate trunk routing step if you want to generate trunks for the selected or all nets in the design. For more information, see Specifying Trunk Generation Options (ICADVM18.1 EXL Only).

Extending Trunks

In this routing step, the selected trunk is extended while routing. The trunk extension behavior depends on the options that have been specified in the *Extend Trunk* section of the *Modification* group box in the *Trunk* subform. For more information, see Extend Trunk.



Trunk Extending - Trimming Off



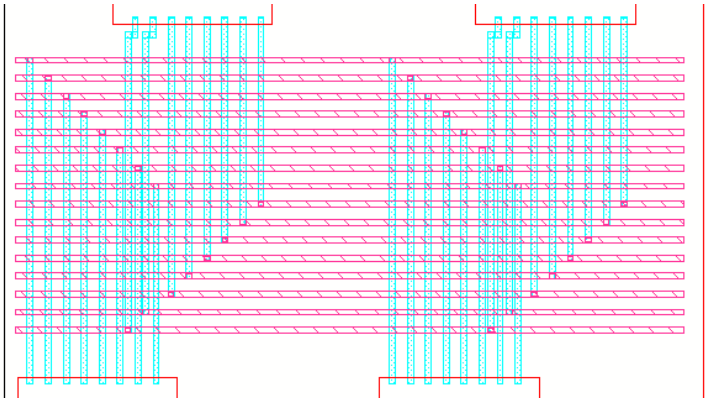
Trunk Extending - Trimming ON

Environment Variable: trunkExtending

Ordering Trunks

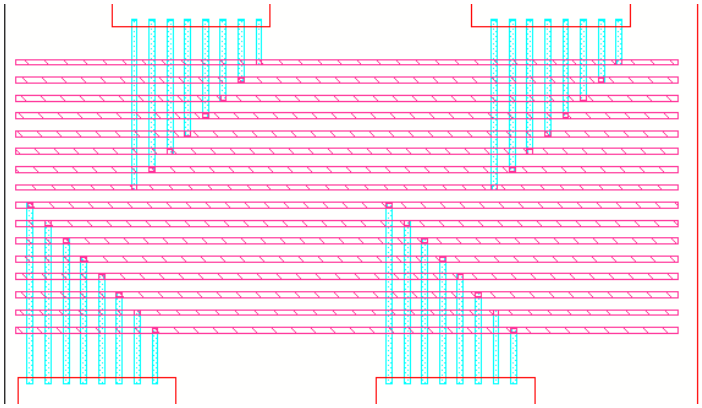
When this routing step is selected, the router optimizes the total routing length by re-ordering trunks not connected to a pin or another trunk. The trunks with more open twigs on the top are placed closer to the top end while the trunks with more open twigs on the bottom are placed closer to the bottom end. The trunks that have different widths retain the original width

after their order is modified. The following figure shows the optimized trunk ordering when the *Trunk Ordering* routing step is deselected as well as selected.



When Trunk Ordering is deselected

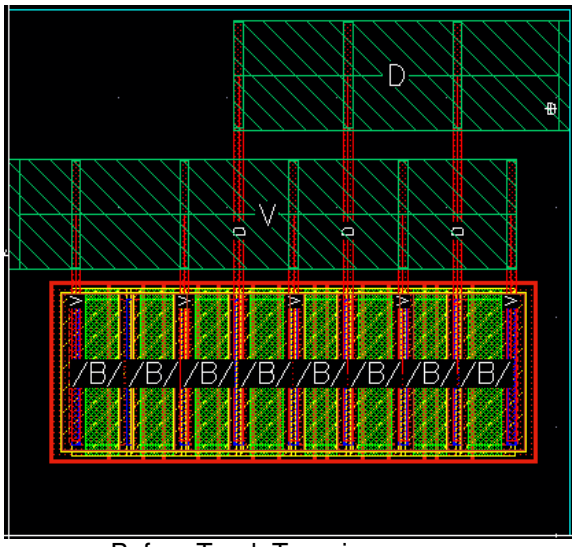
When Trunk Ordering is selected



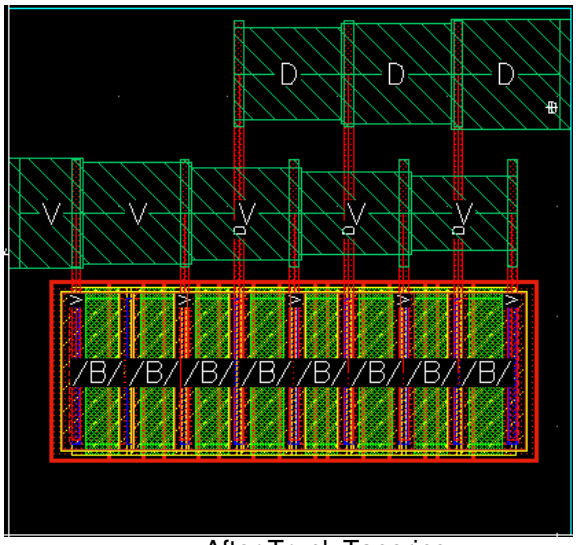
Environment Variable: orderTrunks

Tapering Trunks

The trunk tapering routing step lets you taper the trunks depending on the options that have been specified in the *Taper Trunk Width* section of the *Modification* group box in the *Trunk* subform. For more information, see Specifying Trunk Options.



Before Trunk Tapering

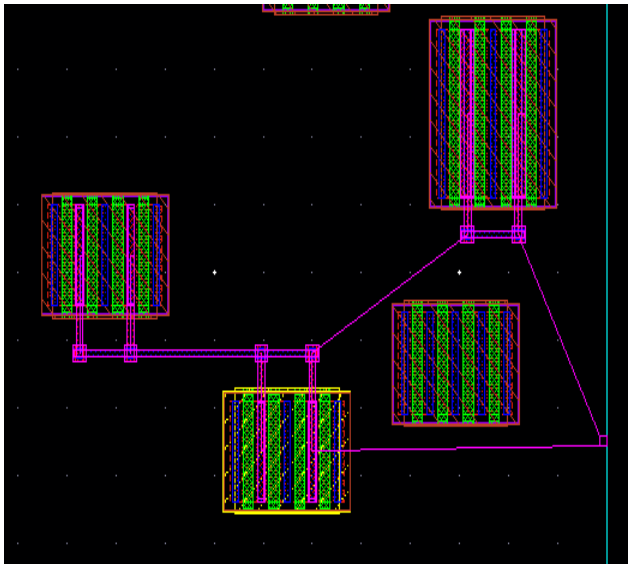


After Trunk Tapering

Environment Variable: trunkTapering

Routing Twigs

This routing step is selected by default, which ensures connectivity between the pins orthogonal to the trunks, as shown in the figure below.

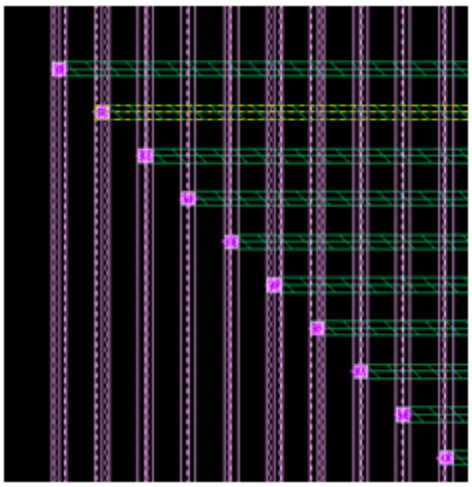


Also, when Pin to Trunk routing starts, the router, by default, completes the routing with a tap partially covering the trunk if complete coverage of the width of the trunk is not possible; ensuring that the number of vias used is not less than the value of the `minNumCut` constraint.

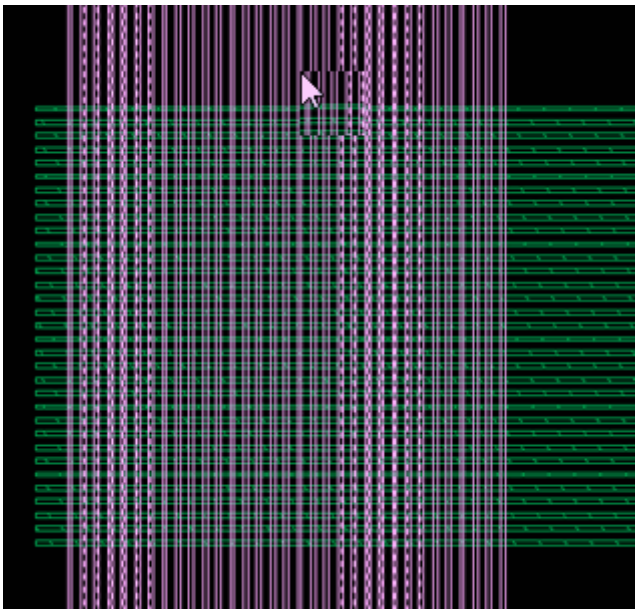
Environment Variable: pinToTrunk

Trunk to Trunk Routing

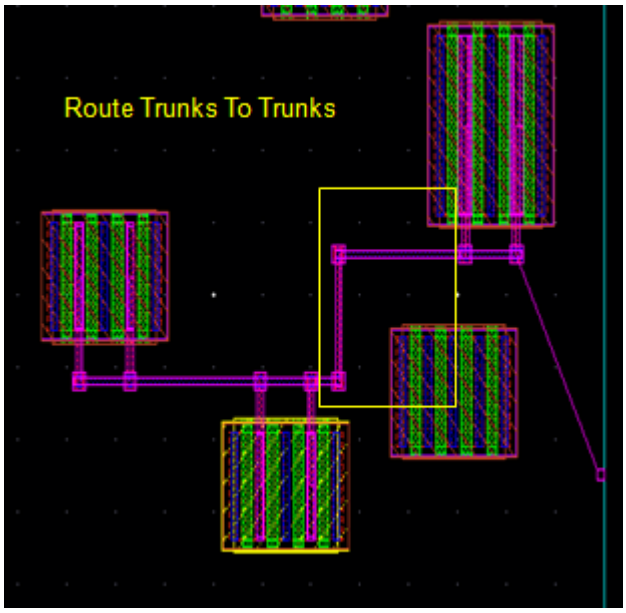
When selected, vias are added at the intersections of the trunks. This can also be used together with trimming, as shown in the following figure:



However, when the *Trunk to Trunk Routing* step is deselected, the trunks are preserved and no vias are added, as shown in the following figure:



The *Trunk to Trunk Routing* is selected by default and connects all the trunks, as shown in the figure below.



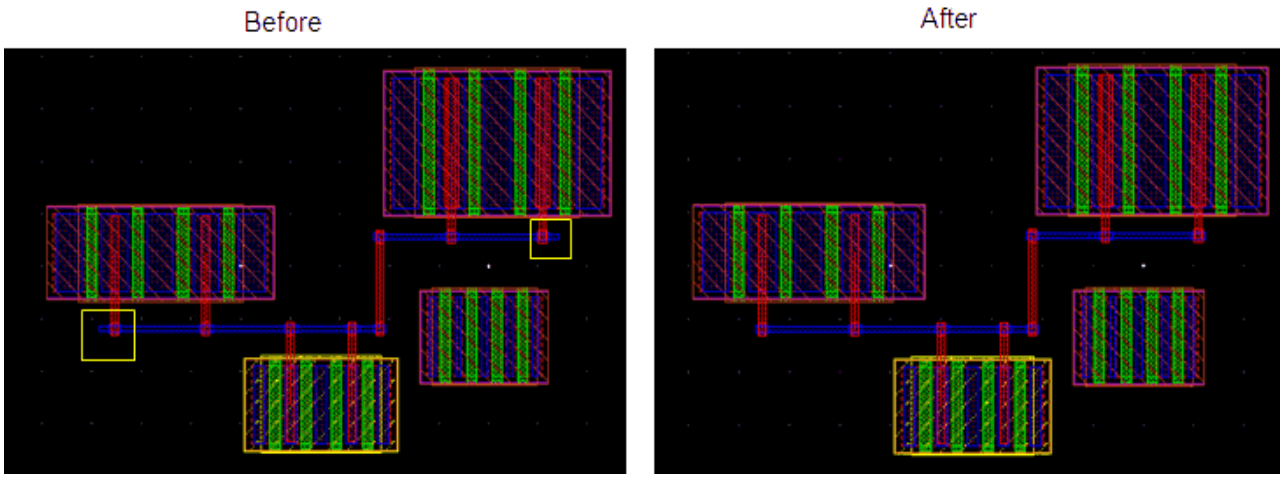
Note: When trunk to trunk is used to weld overlapping trunks, the layers and vias should be set up in the Wire Assistant in such a manner that only the trunks and vias between the selected layers are available.

Environment Variable: trunkToTrunk

Trimming Trunks

In this routing step, both ends of a trunk are trimmed while routing when the *Trunk Trimming* routing step is enabled. However, when the *Trunk Trimming* routing step is disabled, the

length of the trunks is preserved. By default, the *Trunk Trimming* routing step is enabled. The following figure shows routing when *Trunk Trimming* is disabled and when it is enabled.

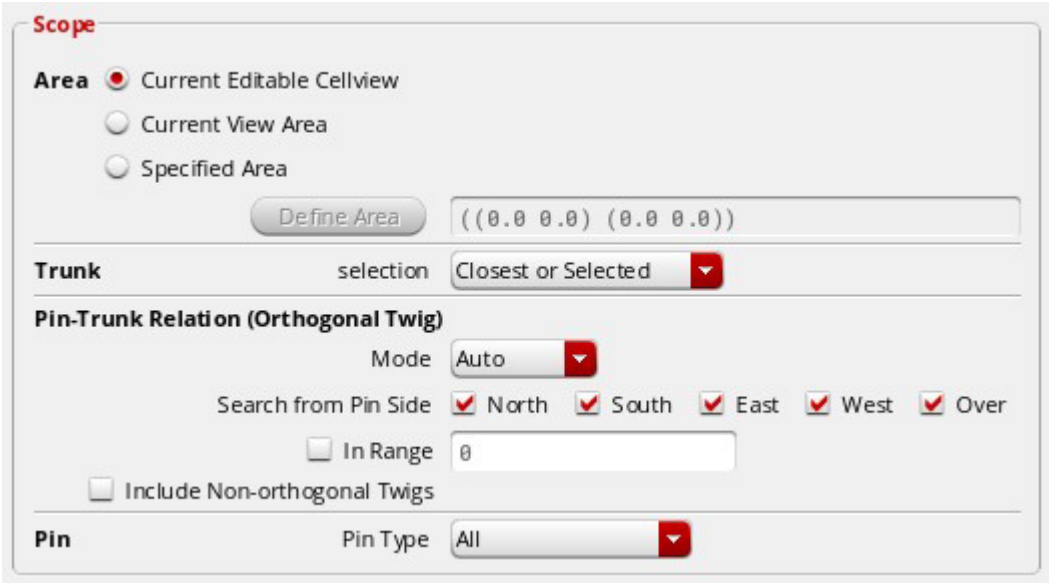


Note: The Trunk Trimming step can be run independently on pre-existing trunks.

Environment Variable: trunkTrimming

Specifying Routing Scope

The *Scope* group box allows you to specify the options to control the scope and region of Pin to Trunk routing. In addition, it allows you to specify the trunks that are to be used for Pin to Trunk routing and Pin to Trunk relation.



Specify the routing scope for Pin to Trunk routing, as explained in the following sections:

- Specifying Range or Area
- Trunk Selection
- Pin-Trunk Relation (Orthogonal Twig)
- Pin Selection

Environment Variable: routingScope

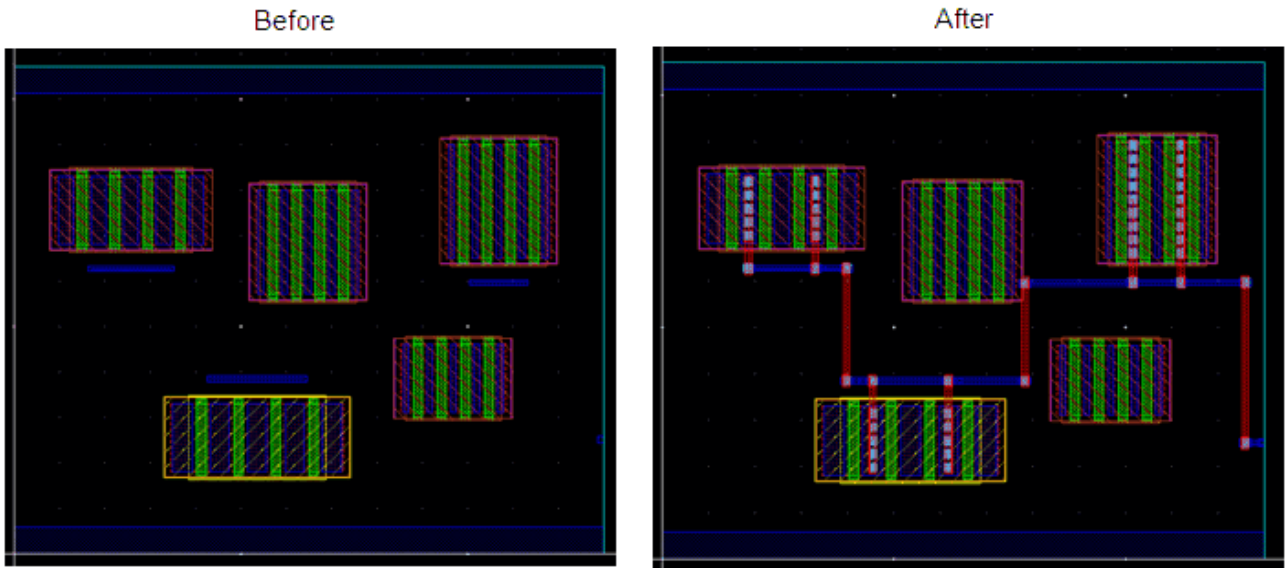
Specifying Range or Area

You can specify the range or area for Pin to Trunk routing. You can further specify the trunk selection and pin selection criteria for the trunks and pins enclosed in the specified range or area.

- Current Editable Cellview
- Current View Area
- Specified Area

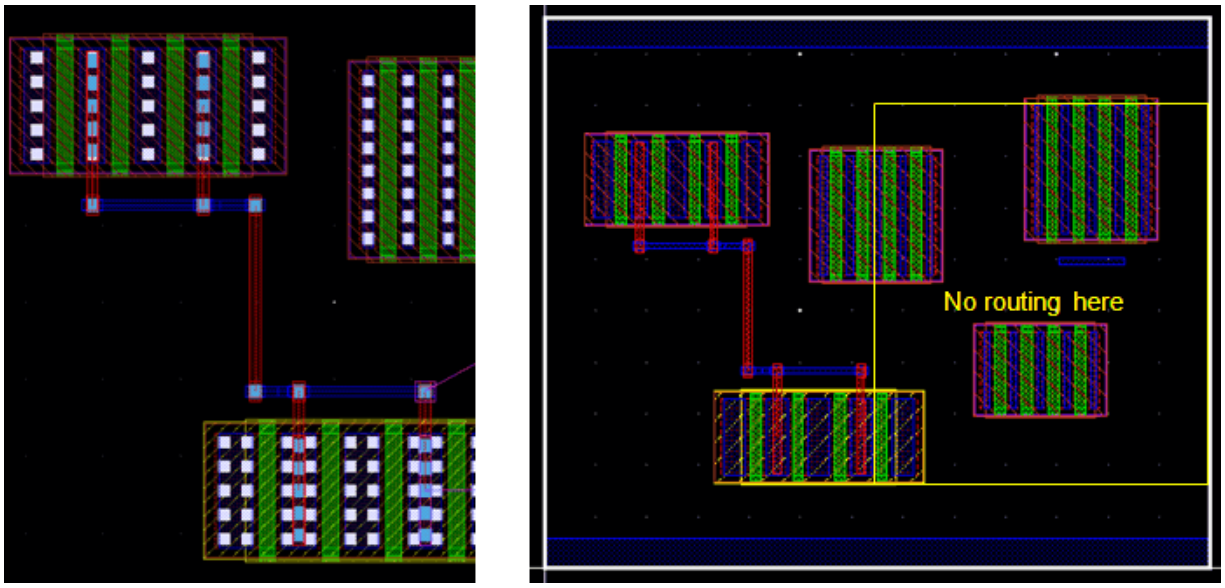
Current Editable Cellview

This is the default option. When selected, only the trunks and pins in the editable cellview are routed, as shown in the following figure.



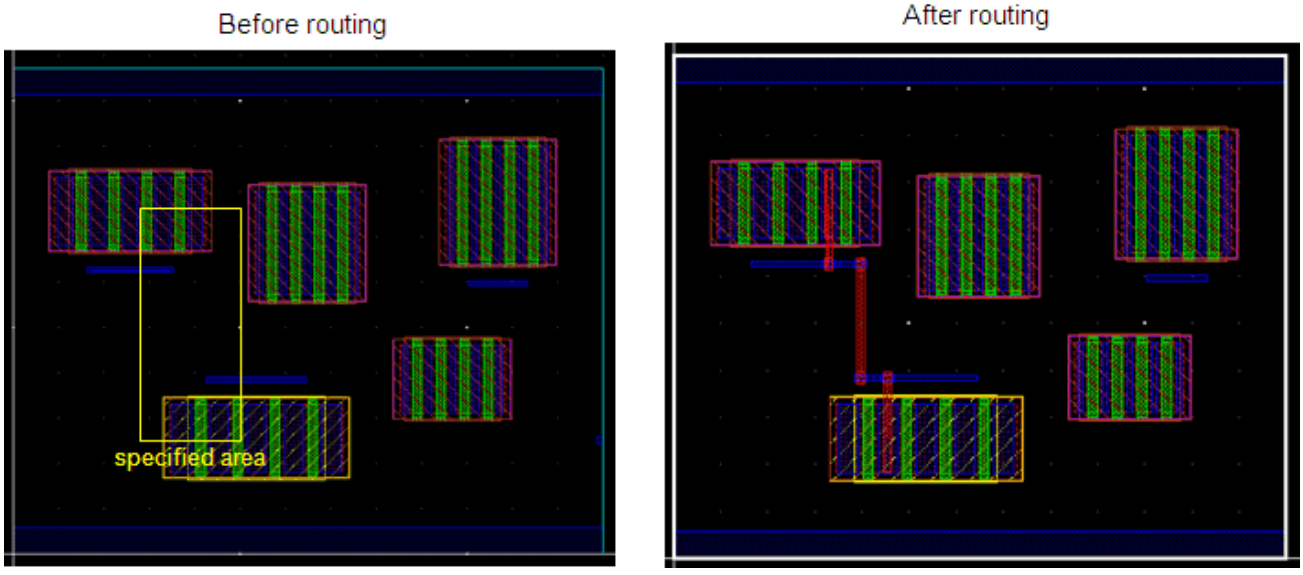
Current View Area

When this option is selected, the trunks and pins partially or fully enclosed in the current viewable area are considered for routing, as shown in the following figure.



Specified Area

When this option is selected, the *Define Area* field is enabled. You can specify the area in which the trunks and pins partially or fully enclosed will be considered for routing, as shown in the following figure.



Environment Variable: routingAreaLLx, routingAreaLLy, routingAreaURx, routingAreaURy

Trunk Selection

The *selection* drop-down list allows you to select the number and the trunks that are to be used by the Pin to Trunk routing engine. The two possible values in the *selection* drop-down list are *Closest or Selected* and *As Many As Possible*.

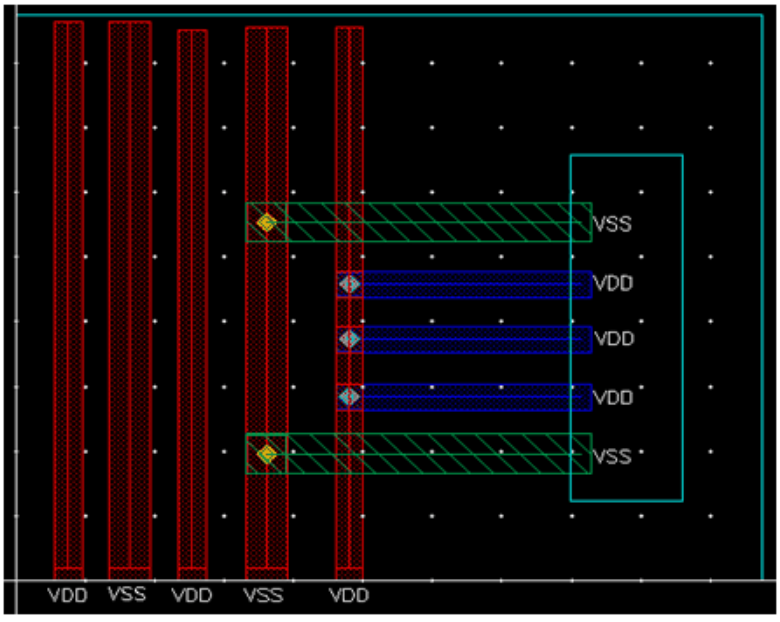
- **Closest or Selected**

  This is the default option. When this option is selected, one of the following happens:

  - If you have a set of trunks selected in the display canvas, the selected trunks are used for routing and no other trunks are selected.

    Note: Selected trunks are considered only if you click the *Selected* button adjacent to *Route Net* in Wire Assistant or select *Route With Default Lookup* or *Route With WA Overrides* from the context-sensitive menu in Navigator Assistant. Selected trunks are ignored if you click the *All* button adjacent to *Route Net* in Wire Assistant or select *Route All Nets* from the context-sensitive menu in Navigator Assistant.

  - If there is no selected trunk set, then the router automatically selects the closest trunk that is orthogonally accessible from the pin for routing.



- **As Many As Possible**

When this option is selected, all the trunks that are orthogonally accessible from a pin are selected by the router.

Environment Variable: selectTrunkMode

Pin-Trunk Relation (Orthogonal Twig)

You can select one of the following options from the *Pin-Trunk Relation (Orthogonal Twig)* section.

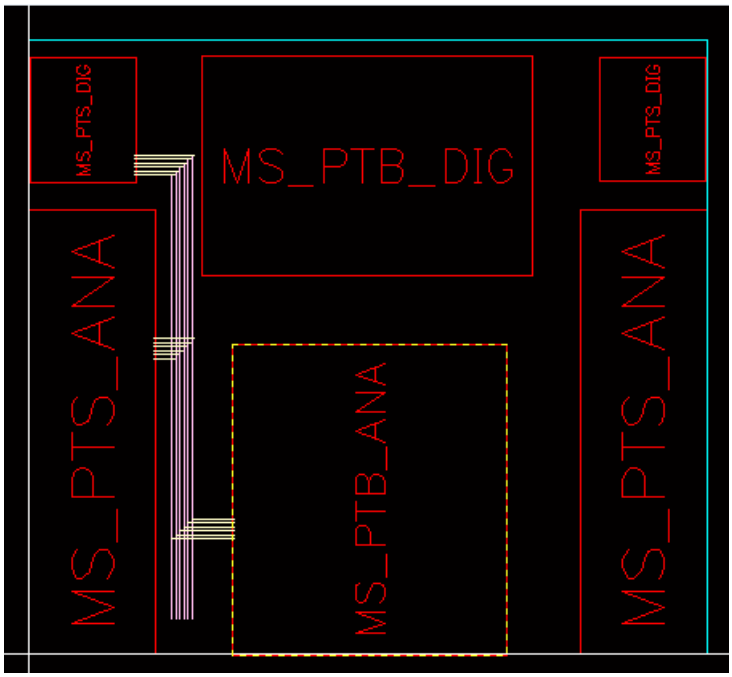
- Mode
- Search from Pin Side
- In Range
- Include Non-orthogonal Twigs

Mode

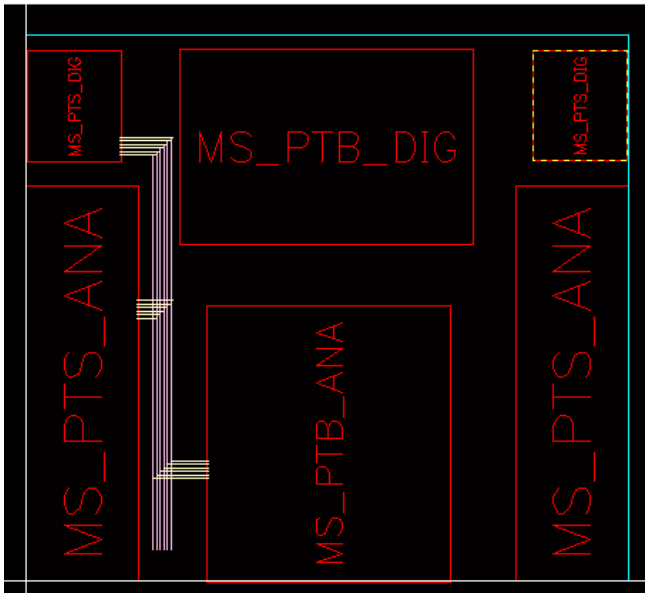
Determines the mode that should be used to route the orthogonal pins to trunks. The possible values are *Auto*, *Channel*, *Space*, and *All*. By default, *Auto* is selected.

- **Auto:** When selected, the router automatically determines the mode of pin selection for orthogonal pins. For Finish Trunk and chip-level designs, the router uses the *Channel*

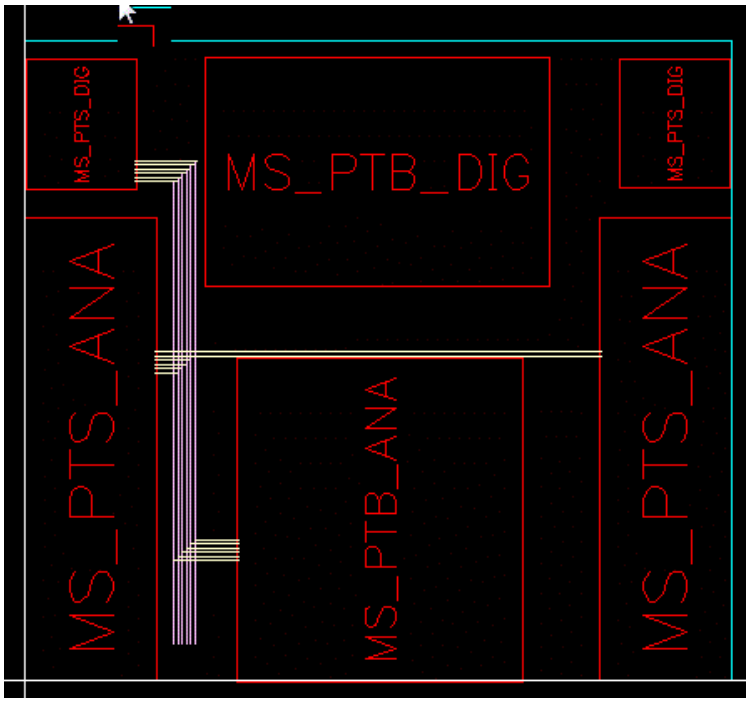
mode. For all other designs, the router uses *All*. The following figure displays the routing of orthogonal pins when the *Auto* mode is selected from the *Mode* drop-down list.



- **Channel:** When selected, the pins orthogonally accessible to a trunk in the same channel are selected. The *Channel* modes requires the pins to be visible to the trunk. Additionaly, the pin's parent must also be completely visible to the trunk.

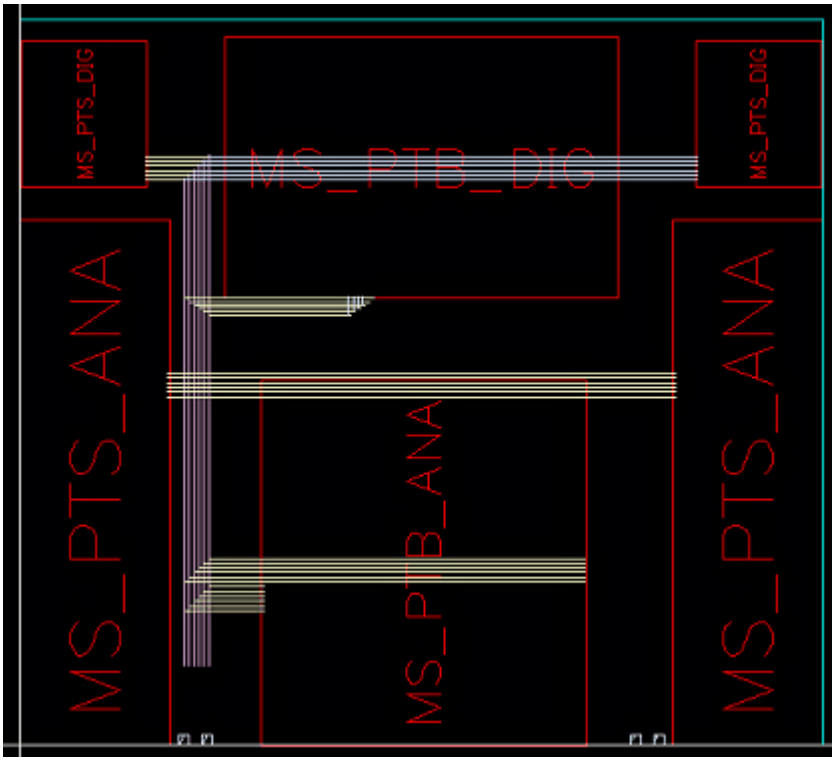


- **Space:** When selected, the pins orthogonally accessible to a trunk through one or more channels are selected. The *Space* mode also requires a pin to be visible to the trunk.



Note: A pin is said to be visible to a trunk if a perpendicular line can be extended from the pin to the trunk without the line crossing part of an instance or obstacle. If the crossed block is the pin's parent, then the line is allowed to cross the access side of the parent perpendicularly, but not running collinear to the access side.

- **All:** When selected, all the pins that are orthogonal to a trunk are selected and routed, regardless of whether the pins are visible from the trunk or not. If you want to route the non-orthogonal pins, select the Include *Non-orthogonal Twigs* option.

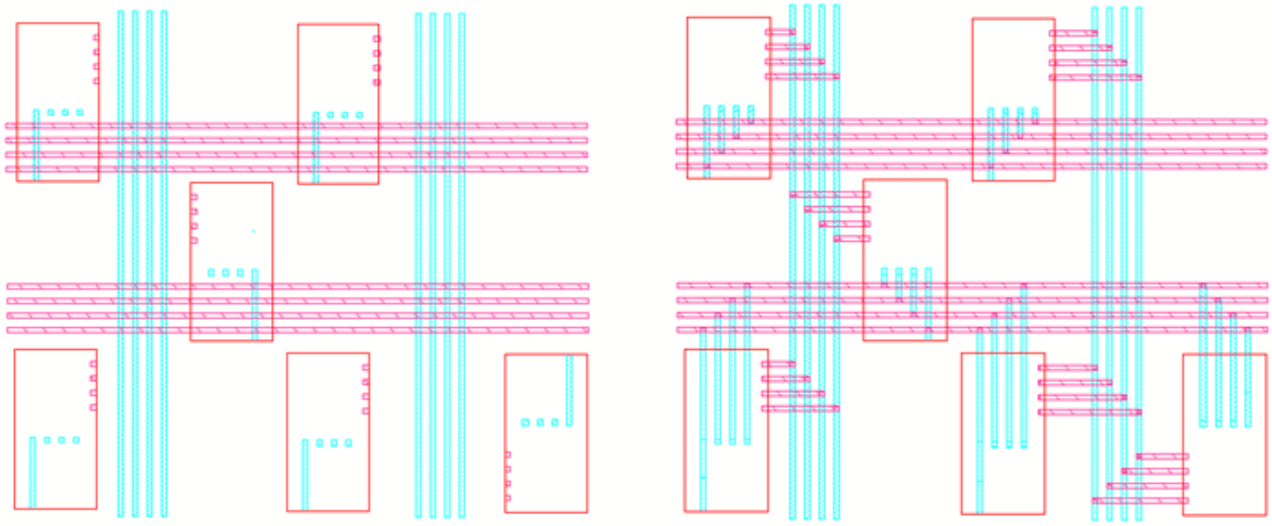


Environment Variable: selectOrthoPinsMode

Search from Pin Side

Specifies a search direction to control which trunks get selected for routing. You can select *North*, *South*, *East*, *West*, or *Over* as the search direction for the selected trunk. You can

select one or more search directions for a trunk. The default is to search for trunks in all directions. The following figure shows when all search directions are selected.



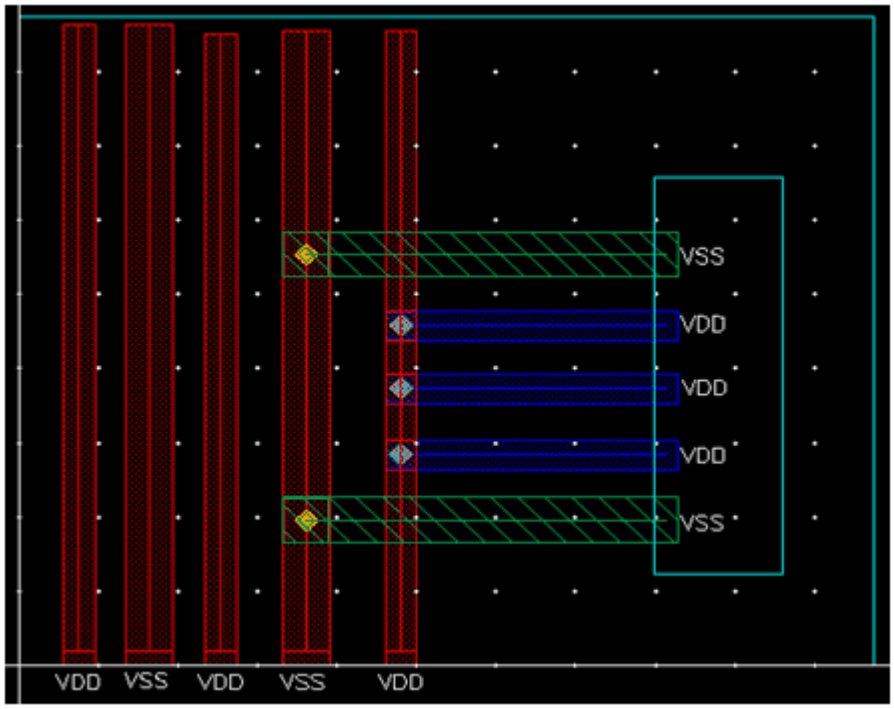
Before specifying the search direction for a trunk, a trunk should be selected and applied. For example, if you select the *Specified Area* option, and *North* and *South* as the search direction, only the trunks on the north or south side of the pins within the given area are used for routing.

Environment Variable: selectNorthTrunk, selectSouthTrunk, selectEastTrunk, selectWestTrunk, selectOverInstTrunk

In Range

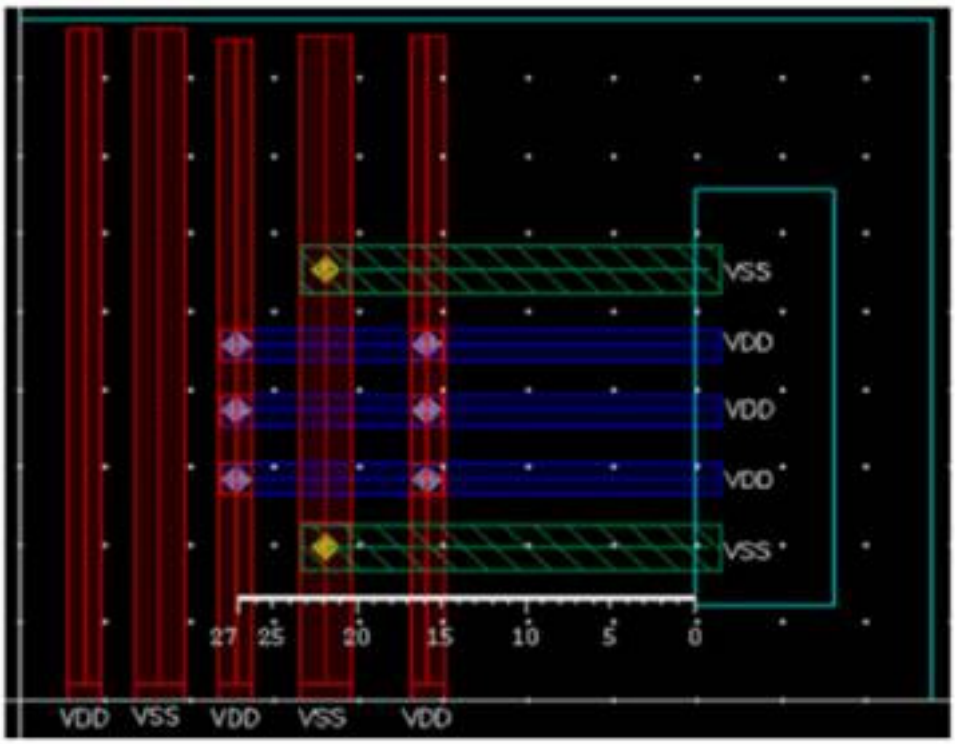
By default, the *In Range* check box is deselected and the adjacent field is disabled. When you select the check box, the field is automatically enabled allowing you to enter a range. The search for trunks is limited to the specified range. The number of trunks selected depends on the option selected from the *selection* drop-down list.

- When you select *Closest or Selected* from the *selection* drop-down list and specify a value in the *In Range* field, the trunk closest to a pin or a selected trunk, if there is one within the specified range, is selected.



- When you select *As Many As Possible* from the *selection* drop-down list and specify a value in the *In Range* field, all trunks within the specified range are selected. The following figure shows a situation where the *As Many As Possible* option is selected

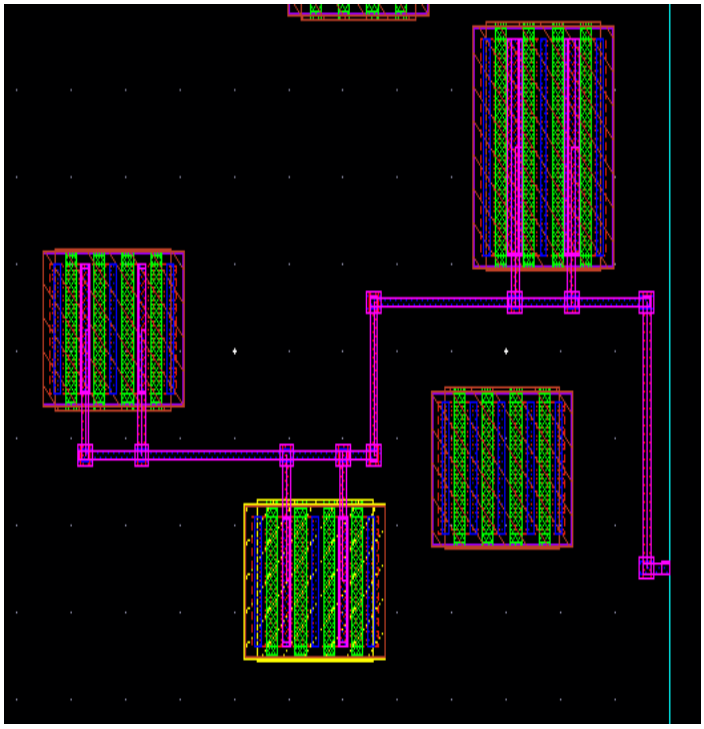
and the *In Range* check box is selected with a range value of `30` specified in the adjacent field.



Environment Variable: selectTrunkRangeValue, selectTrunkWithinRange

Include Non-orthogonal Twigs

For routing the top-level pins, select the *Include Non-orthogonal Twigs* check box. The top-level pins are routed as shown in the figure below.



Note: The router connects the top-level pins to only those trunks that are placed orthogonally to the pin.

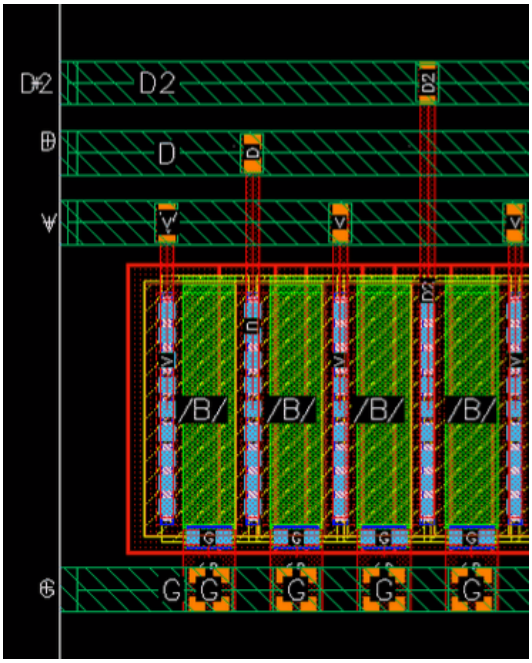
Environment Variable: includeNonOrthogonalTwigs

Pin Selection

The *Pin Type* drop-down list allows you to select the type of pin that should be used for Pin to Trunk routing. The possible values are *All*, *Gate*, and *Source And Drain*.

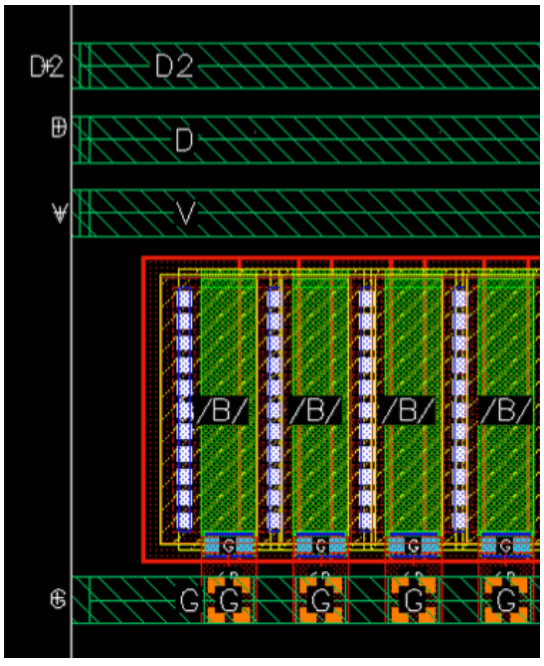
- **All**

When selected, the router connects the trunk to all the existing pins, as shown in the following figure.



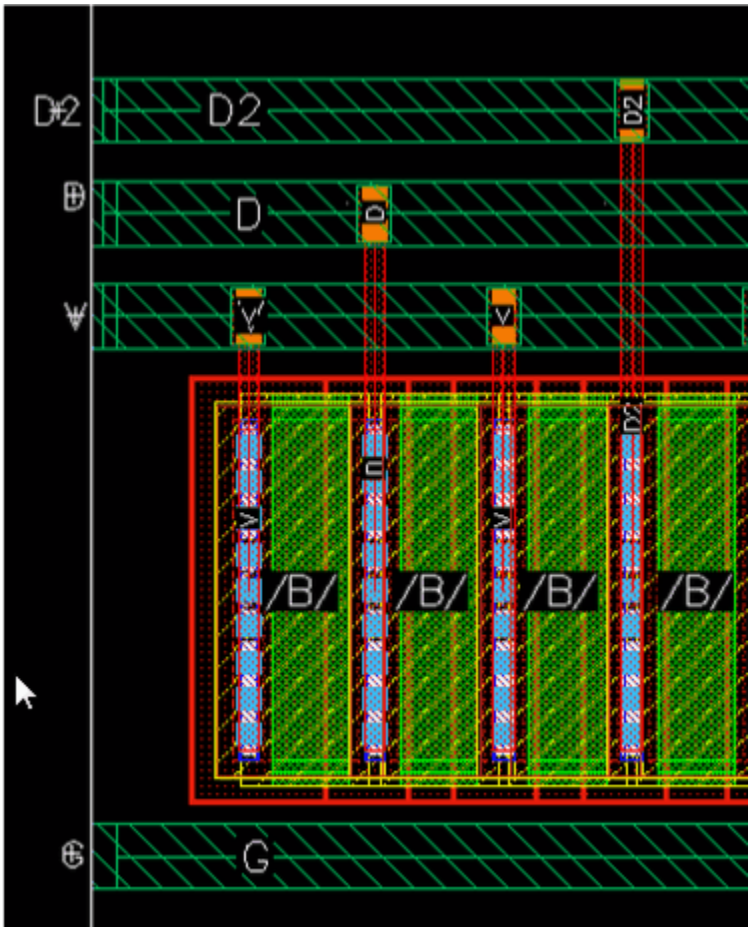
- **Gate**

  When selected, the router connects the selected trunk to the gate pins, as shown in the following figure.



■ Source And Drain

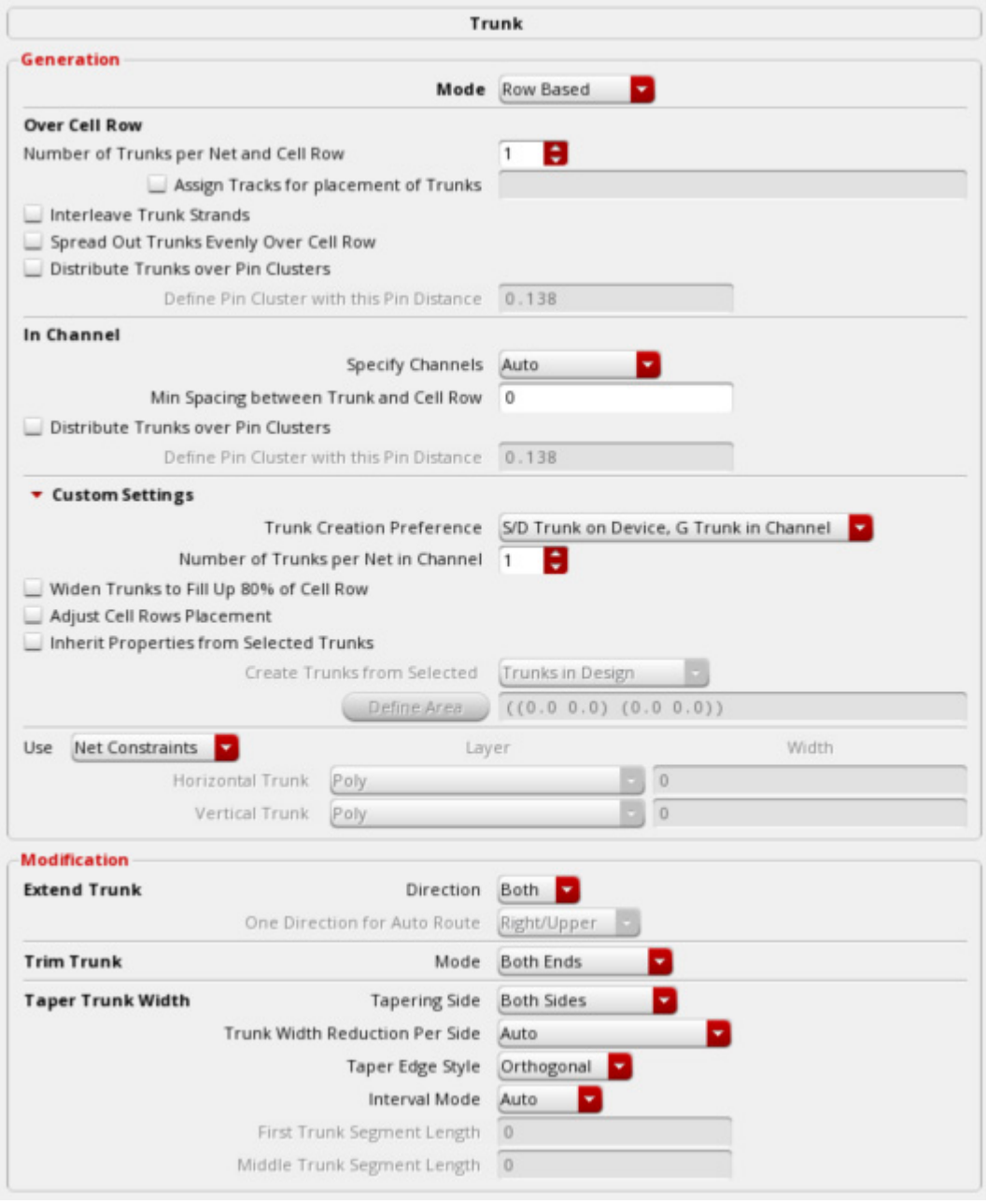
When selected, the router connects the trunk to the source and drain pins, as shown in the following figure.



Environment Variable: connectPinType

Specifying Trunk Options

The *Trunk* subform is displayed when you select *Trunk* from the *Pin to Trunk* tree in the left pane of the Virtuoso Space-based Router Options form.



Use this subform to specify the options for creating new trunks and modifying the existing trunks. To enable all the options available in the *Trunk* subform, you must select the

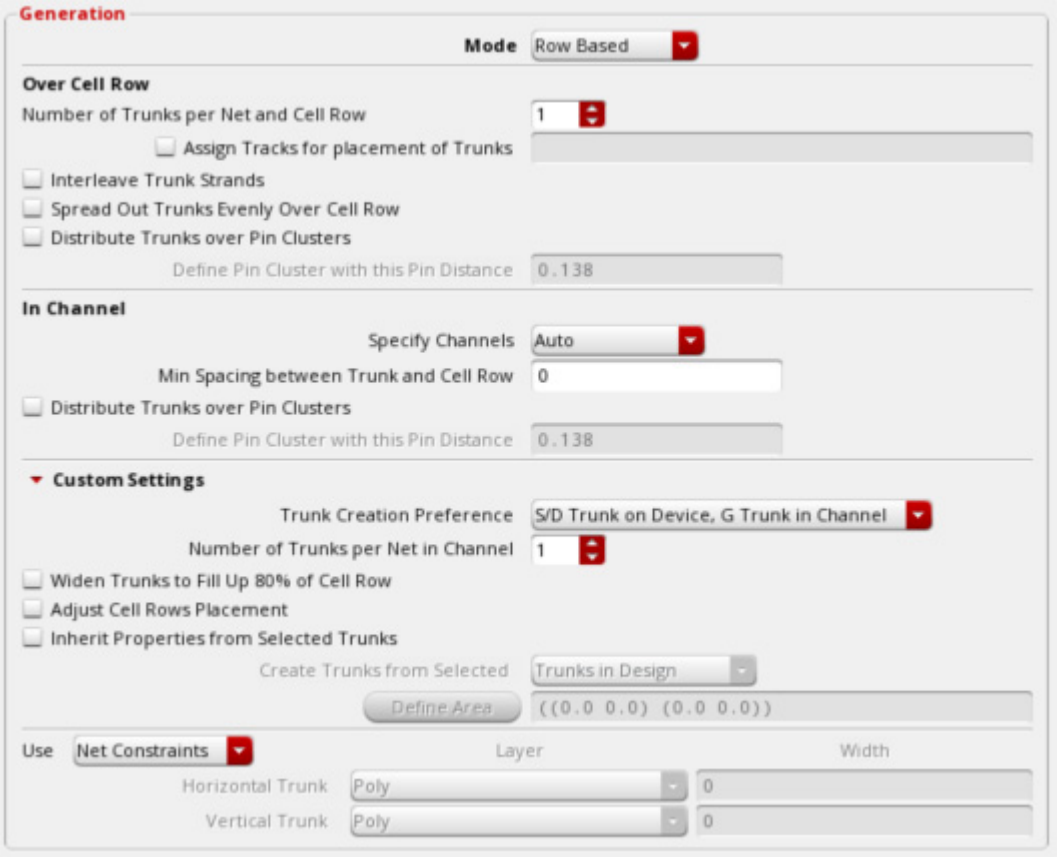
Generate Trunks, Extend Trunks, *Taper Trunks*, *Trunk to Trunk*, and *Trim Trunks* routing steps in the *Pin to Trunk* subform.

The options in the *Trunk* subform are categorized into *Generation* and *Modification* group boxes. Specify the trunk options as explained in the following sections:

- Specifying Trunk Generation Options (ICADVM18.1 EXL Only)
- Specifying Trunk Modification Options

Specifying Trunk Generation Options (ICADVM18.1 EXL Only)

You can specify the options for generating a new trunk in the *Generation* group box of the *Trunk* subform. Trunk generation can happen when the *Generate Trunks* routing step is selected in the *Pin to Trunk* subform.



Environment Variable: trunkSetup

You can specify the following trunk generation options in the *Generation* group box of the *Trunk* subform:

- Specifying the Trunk Mode
- Specifying Row Based Trunk Generation Options
- Specifying Copy Options for Trunk Generation

Specifying the Trunk Mode

From the *Mode* drop-down list, select either the Row Based, or the Copy option. By default, *Row Based* option is selected.

- Row Based

  This mode is used for creating new trunks for row based device level designs.

- Copy

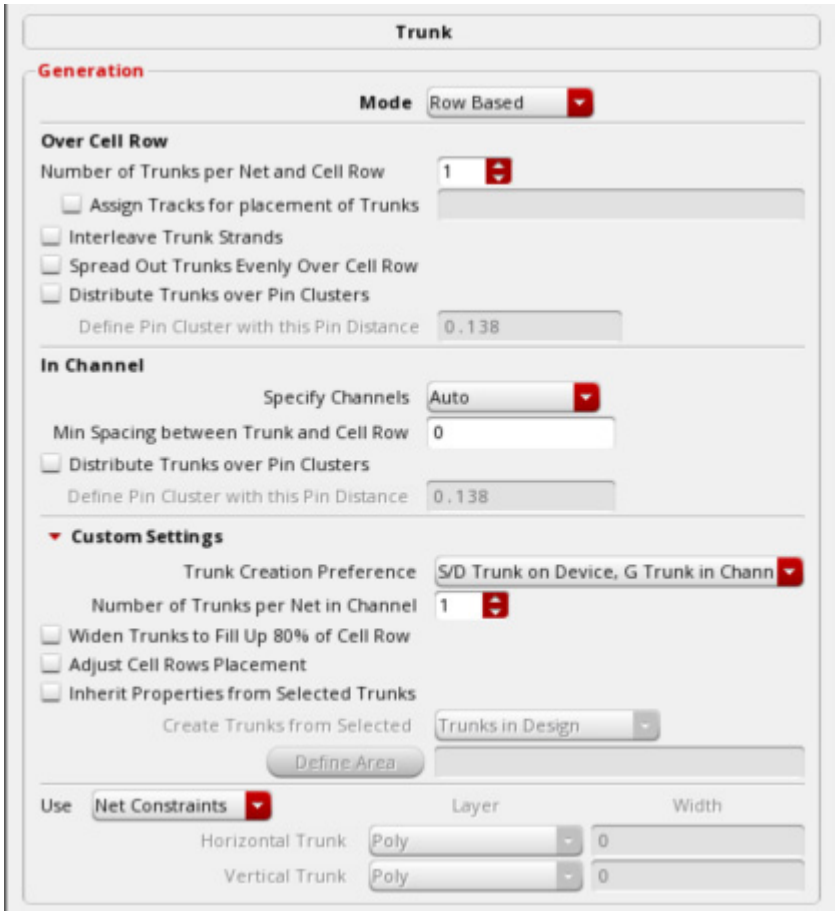
  This mode is used to copy existing trunks in a design.

Specifying Row Based Trunk Generation Options

The following sections lets you specify the options for row based trunk generation.

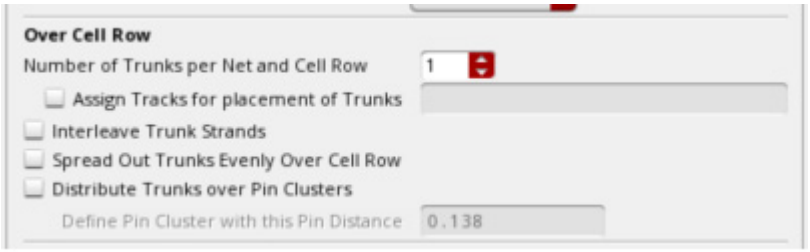
- Specifying Over Cell Row Options
- Specifying In Channel Options

- Specifying Custom Settings



Specifying Over Cell Row Options

The options grouped below *Over Cell Row* are only applicable for source and drain pins.



- Number of Trunks per Net and Cell Row

  Lets you specify the number of tracks per net for each cell row. Default is `1`.

- Assign Tracks for placement of Trunks

  When selected, lets you assign a track (1 for the bottom track) for the placement of each trunk.

- Interleave Trunk Strands

  Interleaves two trunks of different nets.

- Spread Out Trunks Evenly Over Cell Row

  Spreads out multiple trunks on the same net evenly over the span of the source and drain pin.

- Distribute Trunks over Pin Clusters

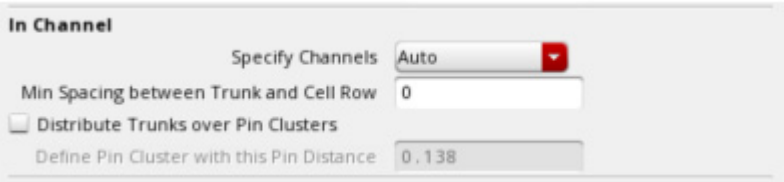
  When selected, divides the length of long trunks by pin clusters as defined by the space between the pins of the net.

  - Define Pin Cluster with this Pin Distance

    Specifies the space between the pins of the net. This field is enabled only when the *Distribute Trunks over Pin Clusters* option is selected. Default is `0.138`.

Specifying In Channel Options

The *In Channel* options are valid for gate trunks. The gate trunks are the only trunks that are automatically generated in the channel.



- Specify Channels

  Specifies how the trunks are placed in the channel between instances in different rows.

  - Auto

    Lets you create trunks in each channel between rows.

  - Odd Channels

    Lets you create trunks in odd channels.

  - Even Channels

Lets you create trunks in even channels.

- Min Spacing between Trunk and Cell Row

  Specifies the minimum spacing between the trunk and the cell.

- Distribute Trunks over Pin Clusters

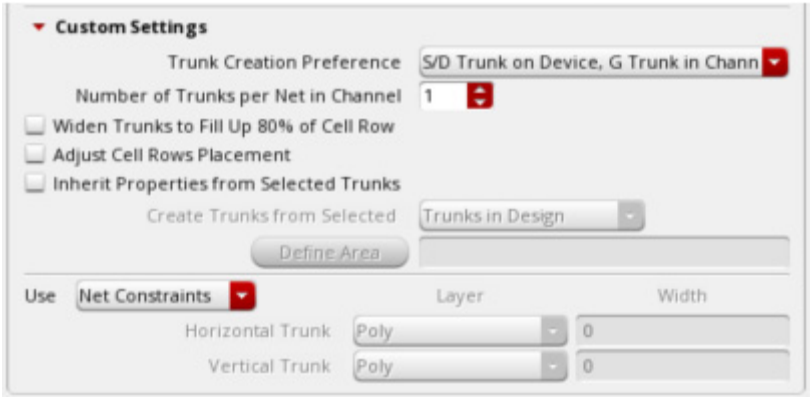
  When selected. divides the length of long trunks by pin clusters as defined by the space between the pins of the net.

  - Define Pin Cluster with this Pin Distance

    Specifies the space between the pins of the net. This field is enable only when the *Distribute Trunks over Pin Clusters* option is selected. Default is three times the minimum pitch (sum of width and spacing) of the highest metal layer.

Specifying Custom Settings

Note: This section lets you specify the settings that are applicable to mature nodes design as well.



- Trunk Creation Preference

  Trunks for source and drain pins are generated on devices and trunks for gate pins are generated in channels. You can specify one of the following preferences to generate trunks.

  - S/D Trunk on Device, G Trunk in Channel

    The trunks for source and drain nets are generated over the cell rows and the trunks for gate nets are generated in the channels.

  - S/D Trunk on Device, Skip G Trunk

Only the trunks for source and drain nets over the cell rows are generated.

- G Trunk in Channel, Skip S/D Trunk

  Only the trunks for gate nets in channels are generated.

- All Trunk in Channels

  Lets you generate all trunks for all or selected nets in channels.

- Number of Trunks per Net in Channel

  Lets you specify the number of tracks per net in each channel. Default is `1`.

- Widen Trunks to Fill Up 80% of Cell Row

  Use this option for relative widening. It is a quick way to widen to relative width of devices.

- Adjust Cell Rows Placement

  Adjusts the placement of cell rows to create space for trunks created in the channel between the rows.

- Inherit Properties from Selected Trunks

  Lets you generate trunks based on width and layer of the selected trunk.

- Create Trunks from Selected

  This field is only enabled when the *Inherit Properties from Selected Trunks* option is selected.

  - Trunks in Design

    Inherits width and layer from pre-existing trunks in the design.

  - Trunks on Same Net

    Inherits width and layer from pre-existing trunks on the same net.

  - Trunks in Area

    Inherits width and layer from pre-existing trunks in the specified area.

    - Define Area

      This field is enabled only when the *Trunks in Area* option is selected in the *Create Trunks from Selected* drop-down list. You can specify the area from which you want to inherit width and layer from pre-existing trunks.

- Use

You can control the layer and width used for creating a new trunk.From the *Use* drop-down list, select either the *Net Constraints* or the *Specified* option.

Environment Variable: trunkSetup

- Net Constraints

  When selected, the constraint lookup method is used to ensure that the layer, direction, and width of the created trunk adheres to the layer and width constraints for the net.

- Specified

  When selected, you can specify the layer and the width for creating a horizontal or a vertical trunk.

  - Horizontal Trunk

    To create a horizontal trunk, select the layer from the *Layer* drop-down list. The horizontal trunk is created on the selected layer. In the *Width* field, specify the width of which the horizontal trunk should be created.

    Environment Variable: trunkSetupHorTrunkLayer, trunkSetupHorTrunkWidth

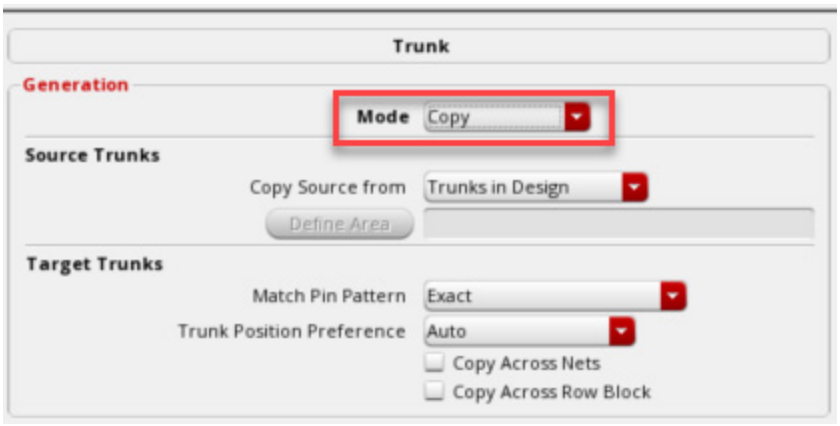
  - Vertical Trunk

    To create a vertical trunk, select the layer from the *Layer* drop-down list. The vertical trunk is created on the selected layer. In the *Width* field, specify the width of which the vertical trunk should be created.

    Environment Variable: trunkSetupVerTrunkLayer, trunkSetupVerTrunkWidth

Specifying Copy Options for Trunk Generation

The *Copy* mode lets you generate trunks for all pins on a net, based on the properties of existing trunks. The following sections lets you specify the options for *Copy* mode for trunk generation.



- Specifying Source Trunk Options
- Specifying Target Trunk Options

Specifying Source Trunk Options

- Copy Source from
  - Trunks in Design

    Inherits the width and layer from all existing trunks, as a template for copying, in the design.
  - Trunks on Same Net

    Inherits width and layer from pre-existing trunks on the same net as a template for copying.
  - Trunks in Area

    Inherits width and layer from pre-existing trunks in the specified area as a template for copying.
    - Define Area

      This field is enabled only when the *Trunks in Area* option is selected in the *Create Trunks from Selected* drop-down list. You can specify the area from which you want to inherit width and layer from pre-existing trunks. For example,

you can create some trunks out of prBoundary and specify an area to use it as a template for copying.

Specifying Target Trunk Options

Note: The *Match Pin Pattern* and *Trunk Position Preference* fields enable you to accommodate various symmetry preferences.

- Match Pin Pattern

  Lets you match two flipping or mirroring pin-patterns and accordingly copy trunk over. It lets you choose what pin context detection is allowed. For example, sometimes you might only want to copy when the pin context is exact. You can select one of the following matching pin pattern.

  - Exact
  - Allow X-Symmetric
  - Allow Y-Symmetric
  - Allow X and Y Symmetric

- Trunk Position Preference

  Controls how the trunks are copied over to the target pins. It gives preference when multiple copy methods are applicable.

  When a target can be matched only to X-symmetric, Y-symmetric, or XY-symmetric, then there is no ambiguity, as the trunk is copied in a corresponding way. However, when the target pattern is exactly the same when flipping across X or Y-axis, you have a choice to copy the trunk in either of the exact, X-symmetric, or Y-symmetric way. You can select one of the following trunk position.

  - Exact Matching
  - X Symmetric
  - Y Symmetric
  - X and Y Symmetric
  - Auto

- Copy Across Nets

  Lets you use template of one net to copy to other net as long as the pin context matching is noticed.

- Copy Across Row Block

  Lets you use row context and row block context to decide whether we should apply X-symmetric, Y-symmetry, none, or both from the source pin pattern to an exact matching pin pattern.

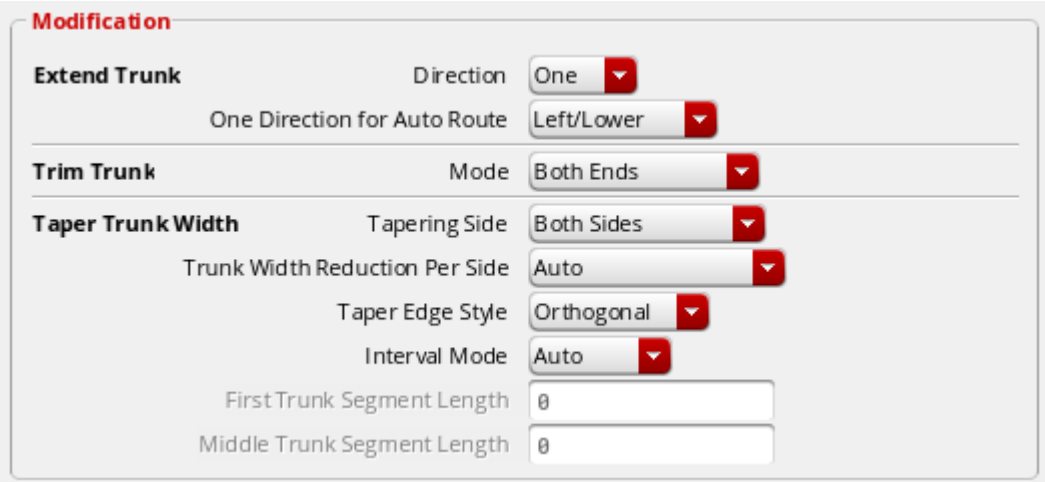
Related Topics

Using Automatic Trunk Generation in Pin to Trunk Route Flow

Specifying Trunk Modification Options

You can specify the following trunk modification options in the *Modification* group box of the *Trunk* subform:

- Extend Trunk
- Trim Trunk
- Taper Trunk Width



Extend Trunk

The *Extend Trunk* section is enabled only when the *Trunk Extending* routing step is selected in the *Pin to Trunk (Default)* subform. You can extend only those trunks that do not have any connection to a pin or another trunk.

Video

For a short demonstration on how to extend trunks and view the result of trunk extension, see Extending Trunks.

In this section, you can specify how to extend the trunks during pin to trunk routing.

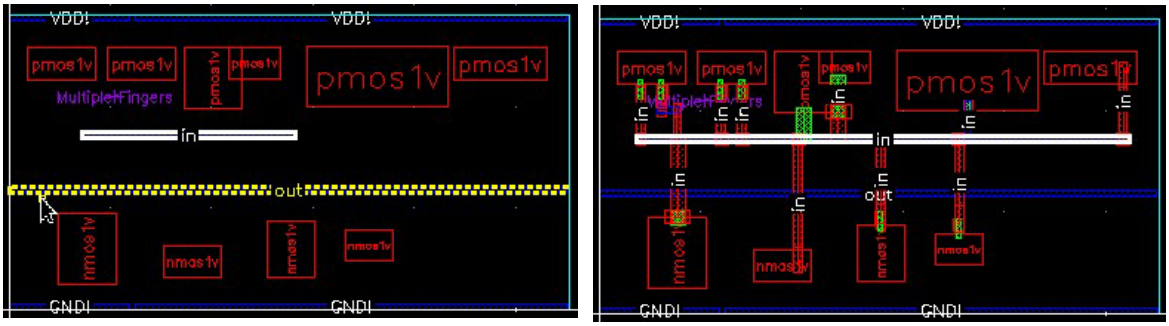
- Direction

  From the *Direction* drop-down list, you can select the direction in which you want to extend the trunk. You can select the direction either as *Both* or *One*. By default, the trunk is extended in both directions.

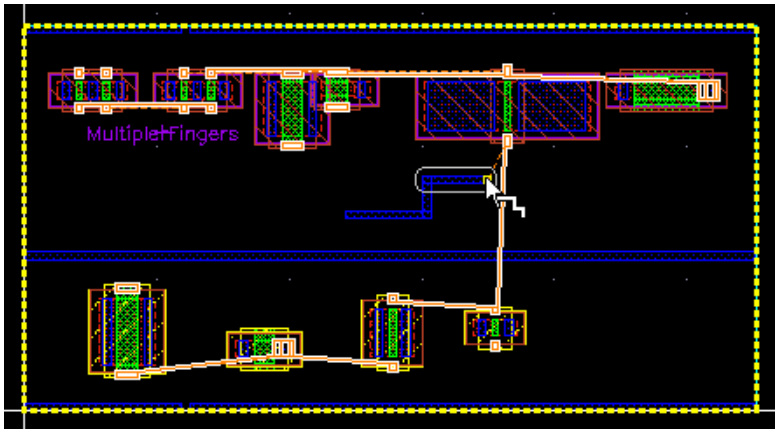
  Environment Variable: extendTrunkDirMode

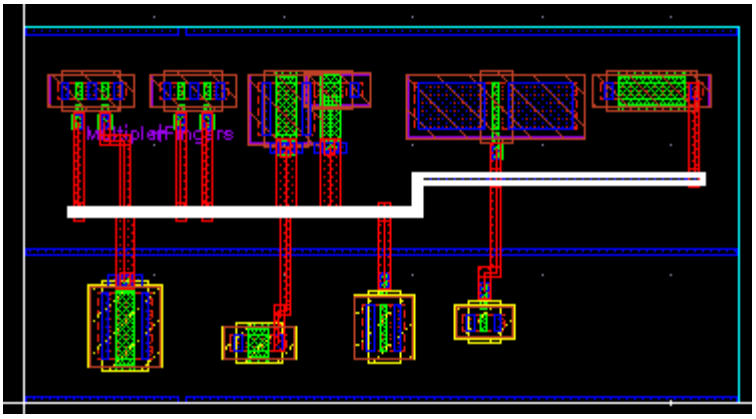
  - Both

    Extends the trunk in both directions up to the last pin when the *Trunk Trimming* step is selected, as shown in the following figure. This is when you select the *Route All* or *Route Selected* command in the Wire Assistant, or *Route With* command from the Navigator Assistant.

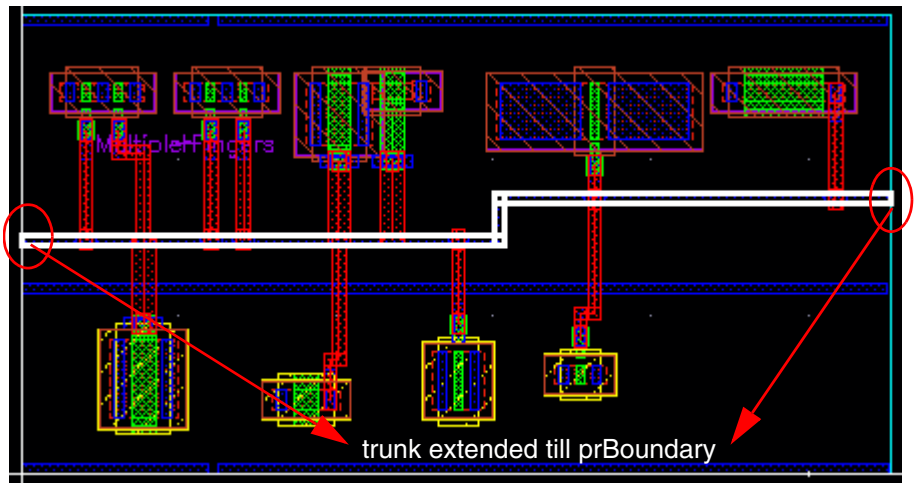


However, when you use the *Finish Trunk* command to complete the trunk extension, the first and the last trunk is extended up to the last pin when the *Trunk Trimming* step is selected, as shown in the following figure.





When the *Trunk Trimming* step is deselected, the trunk is extended up to the prBoundary, as shown in the following figure.



Important

If an obstructing instance is found while extending the trunk, the trunk is extended up to a minimum space from the obstructing instance.

- One

  Extends the trunk in a single direction for *Route All* or *Route Selected* command in the Wire Assistant, or for *Route With* command from the Navigator Assistant. The direction in which the trunk is extended depends on the value that is selected from the *One Direction for Auto Route* drop-down list. For *Finish Trunk* command, extending a trunk in one direction means to extend it from the last point of the last segment being created by the preceding *Create Wire* command.

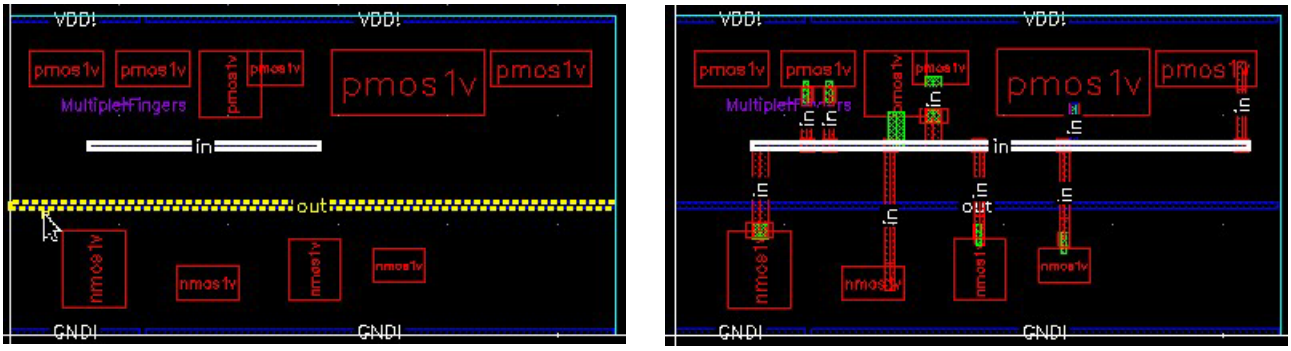
- One Direction for Auto Route

  This field is enabled when you select the direction for extending the trunk as *One*. You can extend the trunk either in the *Right/Upper* or *Left/Lower* direction. By default, the trunk is extended in the *Right/Upper* direction.

  Environment Variable: extendTrunkEndForOneDirMode

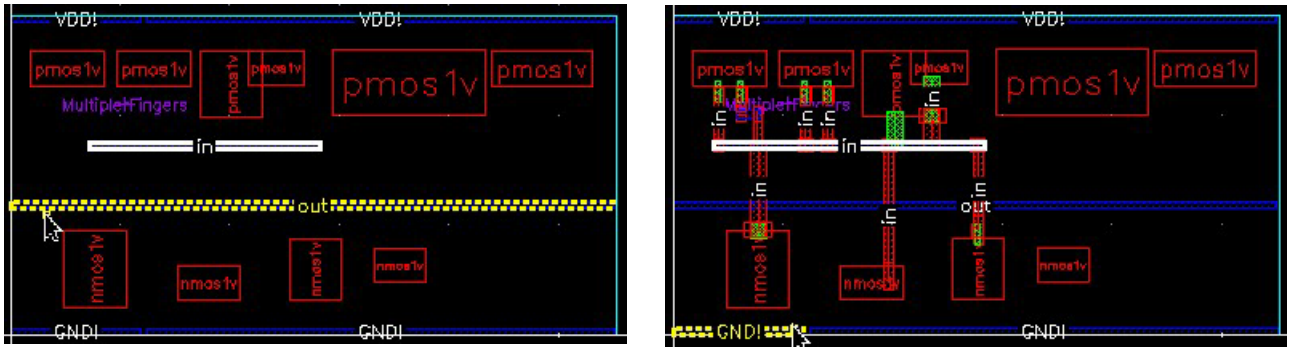
  - Right/Upper

Extends the right end of a horizontal trunk or upper end of a vertical trunk, as shown in the following figure.



- Left/Lower

  Extends the left end of a horizontal trunk or lower end of a vertical trunk, as shown in the following figure.



Trim Trunk

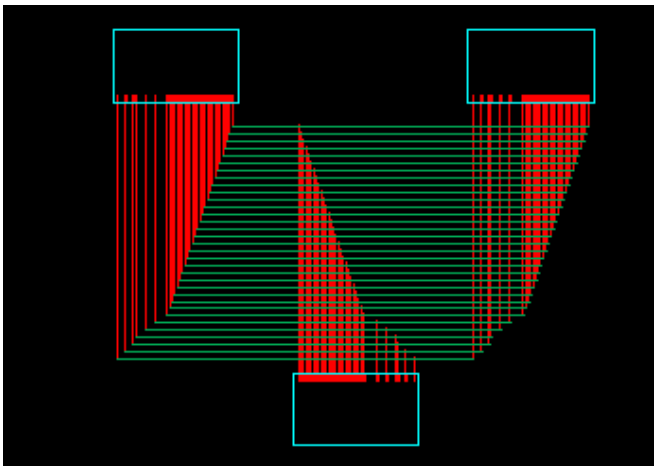
The *Trim Trunk* section is enabled only when the *Trunk Trimming* routing step is selected. The trunk trimming options are only available for trunks with at least two connections to either a pin or a trunk. It does not remove an entire dangling trunk. Also, if a trunk has one or no connection, trunk trimming does not trim or remove the trunk.

In this section, you can specify how to trim the trunks during Pin to Trunk routing. By default, the *Both Ends* option is selected. You can trim *Both Ends*, *Left/Lower End*, or *Right/Upper End* of a trunk.

Environment Variable: trimTrunkMode

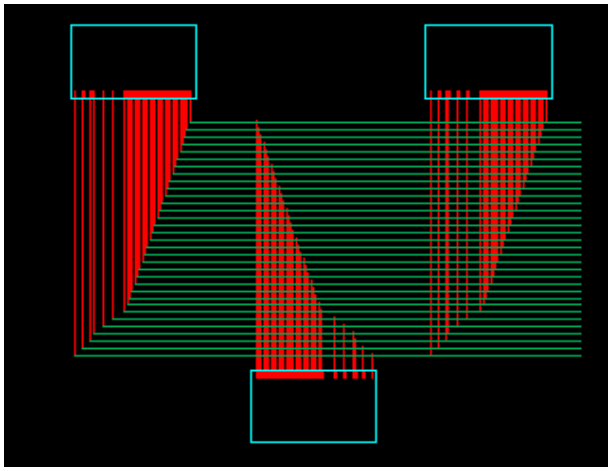
- **Both Ends**

When selected, trims both ends of the trunk, as shown in the figure below.



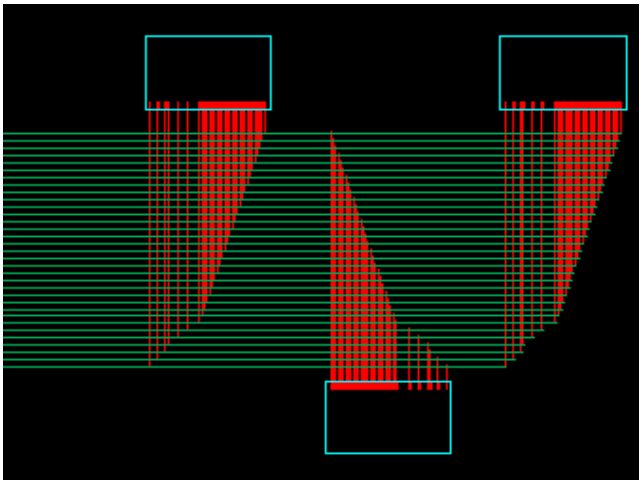
- **Left/Lower End**

  When selected, trims the left or lower ends of the trunk, as shown in the figure below.



- **Right/Upper End**

When selected, trims the right or upper ends of the trunk, as shown in the figure below.



You can also connect a multipart path guard ring to a trunk by changing the *Connectivity* attribute of a guard ring from net to pin. The Pin to Trunk routing then treats it as a pin and connects the guard ring to the trunk. For more information on MPP Guard Ring, see Creating and Editing Multipart Paths.

Taper Trunk Width

The *Taper Trunk Width* section is enabled only when the *Trunk Tapering* routing step is selected in the *Pin to Trunk* subform. When a trunk is tapered, the tapered trunk fragments remain connected and form a stair-like structure. By using the trunk tapering options, you can specify the appearance of the stair-like structure. The trunk tapering options are available only for trunks with at least two connections to either a pin or a trunk.

In this section, you can specify the options to taper the trunk widths during Pin to Trunk routing.

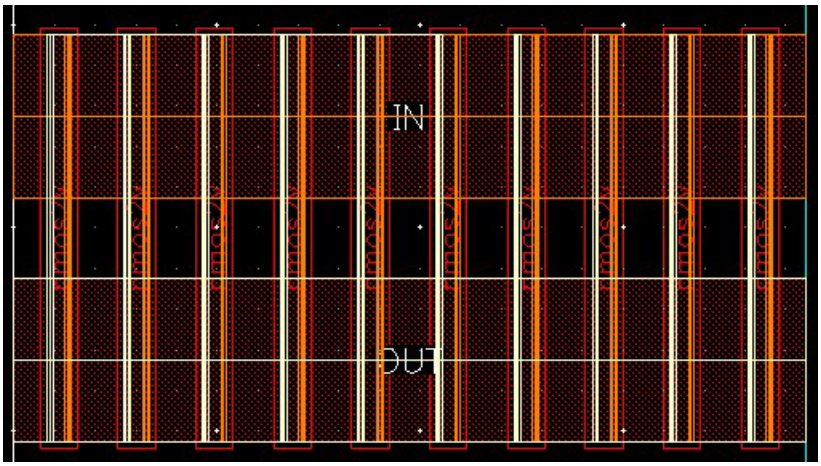
- Tapering Side

  You can reduce the width of a trunk from a particular side or from both sides. By default, the trunk is tapered from both sides.

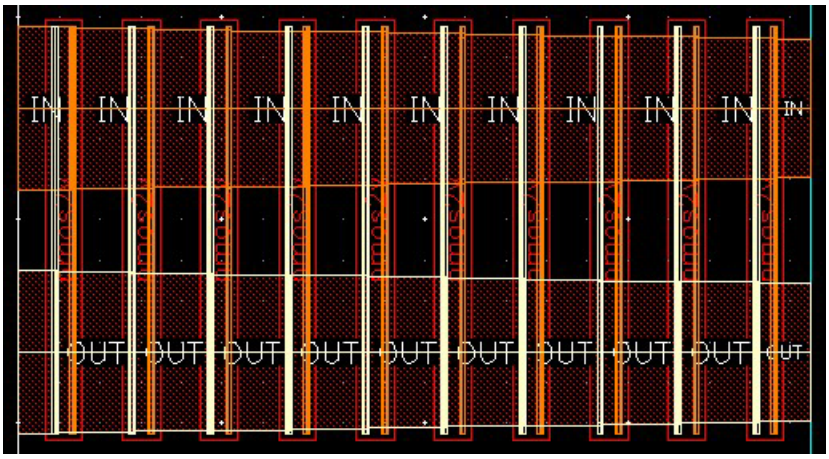
  Environment Variable: trunkTaperSideMode

  - Both Sides

When selected, tapers both sides of the trunk, as shown in the figure below.



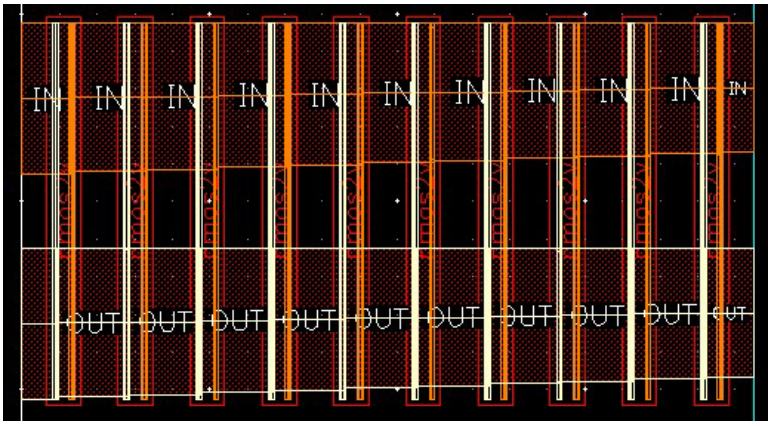
Untapered trunks



Tapered trunks from both sides

- Lower/Left Side

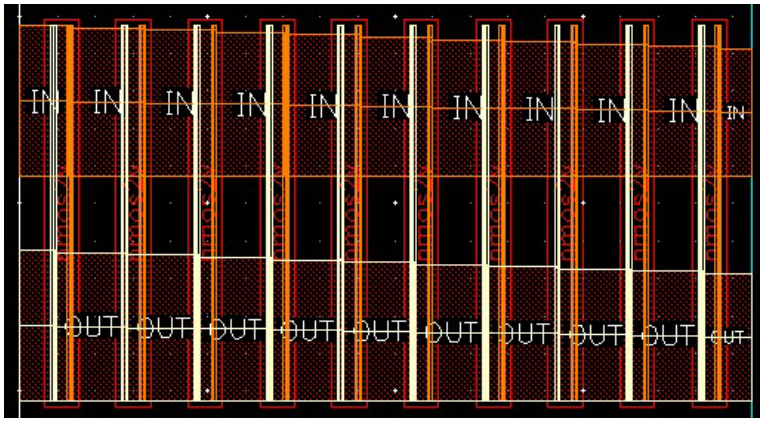
When selected, tapers the left or lower ends of the trunk, as shown in the following figure.



Tapered trunks from lower/left side

- Upper/Right Side

  When selected, tapers the right or upper ends of the trunk, as shown in the following figure.



Tapered trunks from upper/right side

- Trunk Width Reduction Per Side

  Use this option to specify the extent of width reduction on each side of a tapered trunk segment.

  Environment Variable: trunkTaperReductionMode

  - Auto

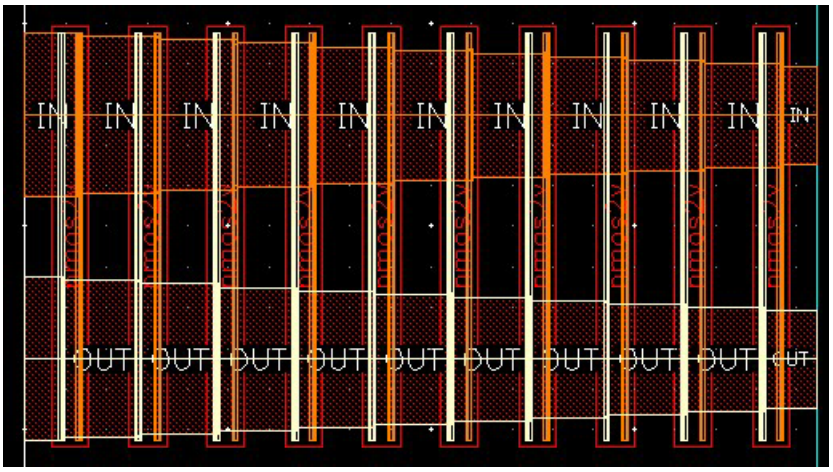
    When both *Trunk Width Reduction Per Side* and *Interval mode* are selected as *Auto*, the width reduction is equal to the width of the previous outgoing twig segment. However, when the *Trunk Width Reduction Per Side* is *Auto* and

Interval Mode is *Custom*, the trunk width is reduced by 5% of the original trunk width.

If the tapering side has been selected as *Both Sides*, the width of the tapered trunk segment is reduced by 2.5 % of the original trunk width on each side. However, if the *Right/Upper* or *Lower/Left* option is selected from *Tapering Side*, the width of the tapered trunk segment is reduced by 5% of the original trunk width on the selected side.

- Specify Percentage

  When selected, you can specify the percentage by which you want the width of the tapered trunk segment to be reduced. The reduction per tapered side is a percentage of the original trunk width. The following figure shows trunk tapering from both sides when the percentage is specified as 2.0.

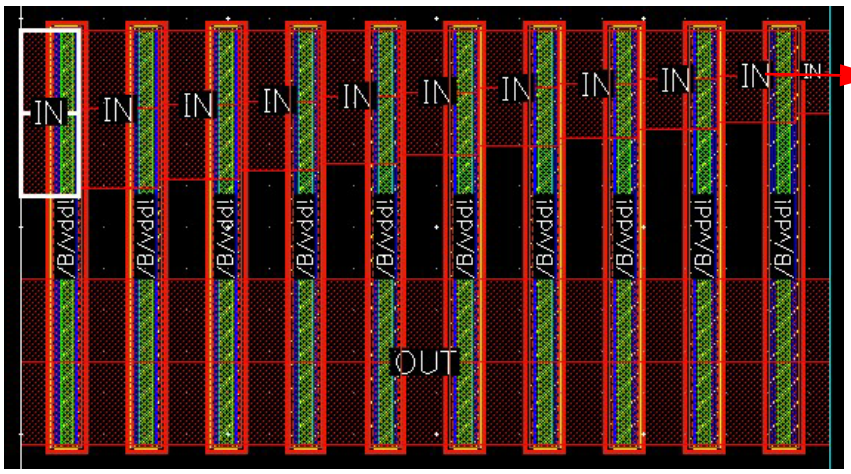


  Environment Variable: trunkTaperReductionPercent

- Specify Value

  When selected, you can specify a value by which the width of the tapered trunk segment is to be reduced. The reduction per tapered side is according to the

specified value. The following figure shows trunk tapering from lower/left side when the value is specified as 0.2 micron.



Environment Variable: trunkTaperReductionValue

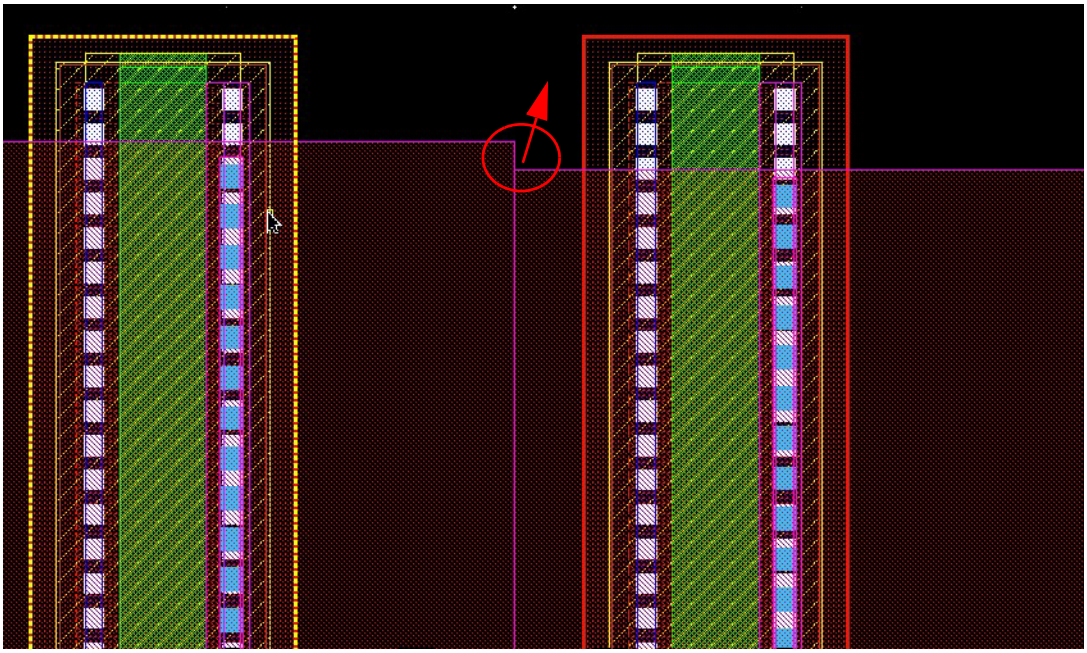
- Taper Edge Style

  Use this option to specify the edge style when the trunk width is tapered.

  Environment Variable: trunkTaperEdgeStyle

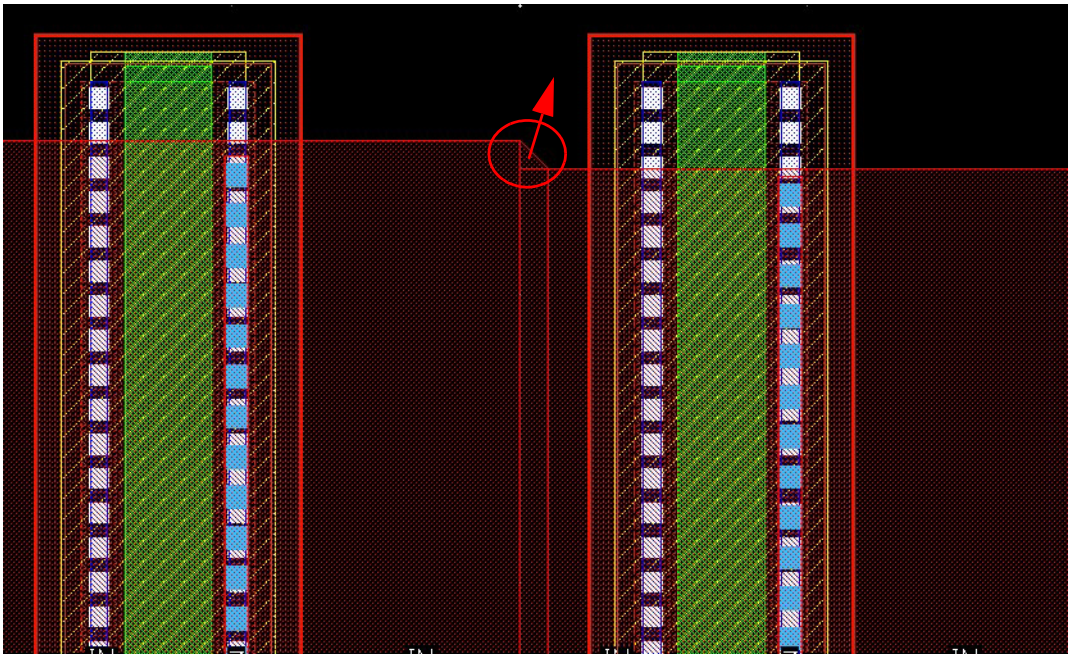
  - Orthogonal

When selected, the edge of the trunk connecting two steps is perpendicular to the steps, as shown in the following figure. This is the default option.



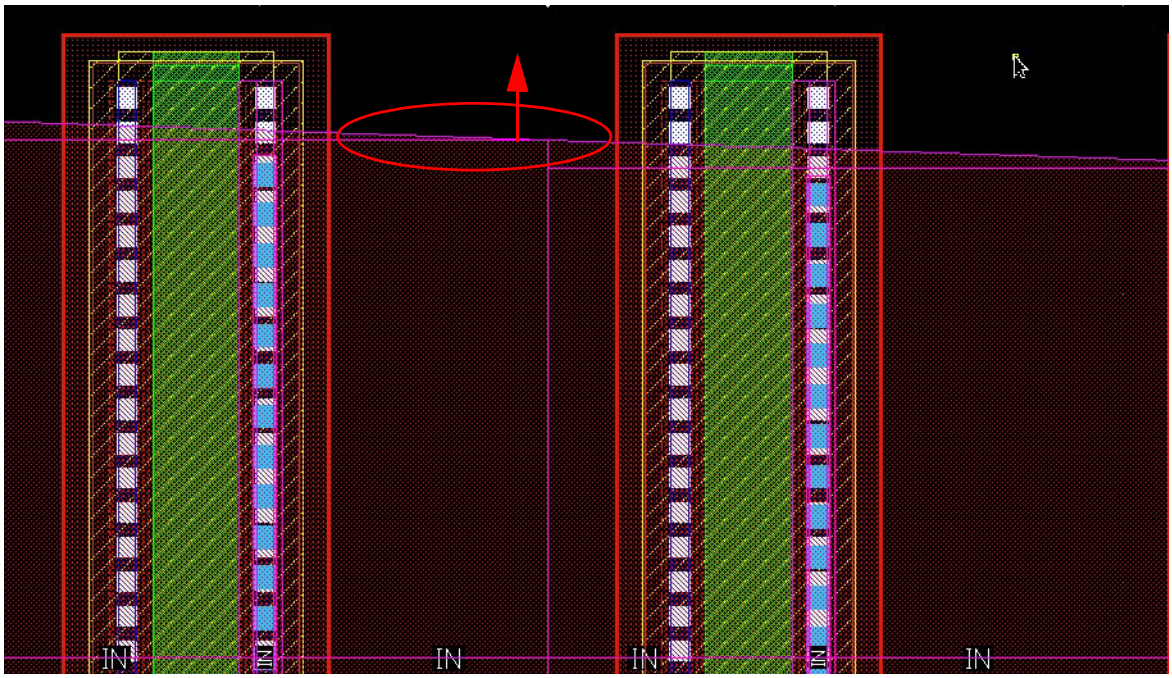
- Diagonal

  When selected, the edge of the trunk connecting two steps is slanted at 45 degrees from previous step to the next step, as shown in the following figure.



- Any Angle

  When selected, the stair-step appearance degenerates to form a smooth-slanting edge throughout the tapered trunks, as shown in the following figure.



- Interval Mode

  Use this option to specify the frequency of trunk tapering.

  - Auto

    The trunk tapering happens at each twig connection. This is the default option.

  - Custom

    The trunk tapering happens at the interval specified by you.

  Environment Variable: trunkTaperIntervalMode

- First Trunk Segment Length

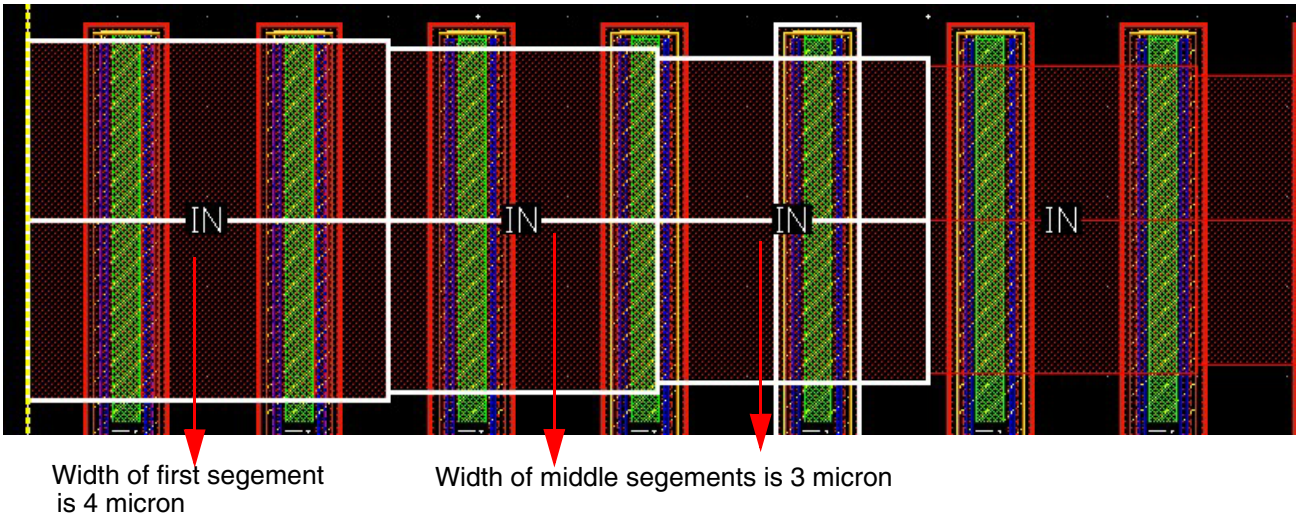
  In this field, specify the length of first tapered trunk segment.

  Environment Variable: trunkTaperFirstIntervalLength

- Middle Trunk Segment Length

  In this field, specify the length of all tapered trunk segments excluding the first and the last trunk segment. The following figure shows trunk tapering from both sides when the

interval mode is selected as *Custom* and the length of the first and middle trunk segments are 4 and 3 microns respectively.



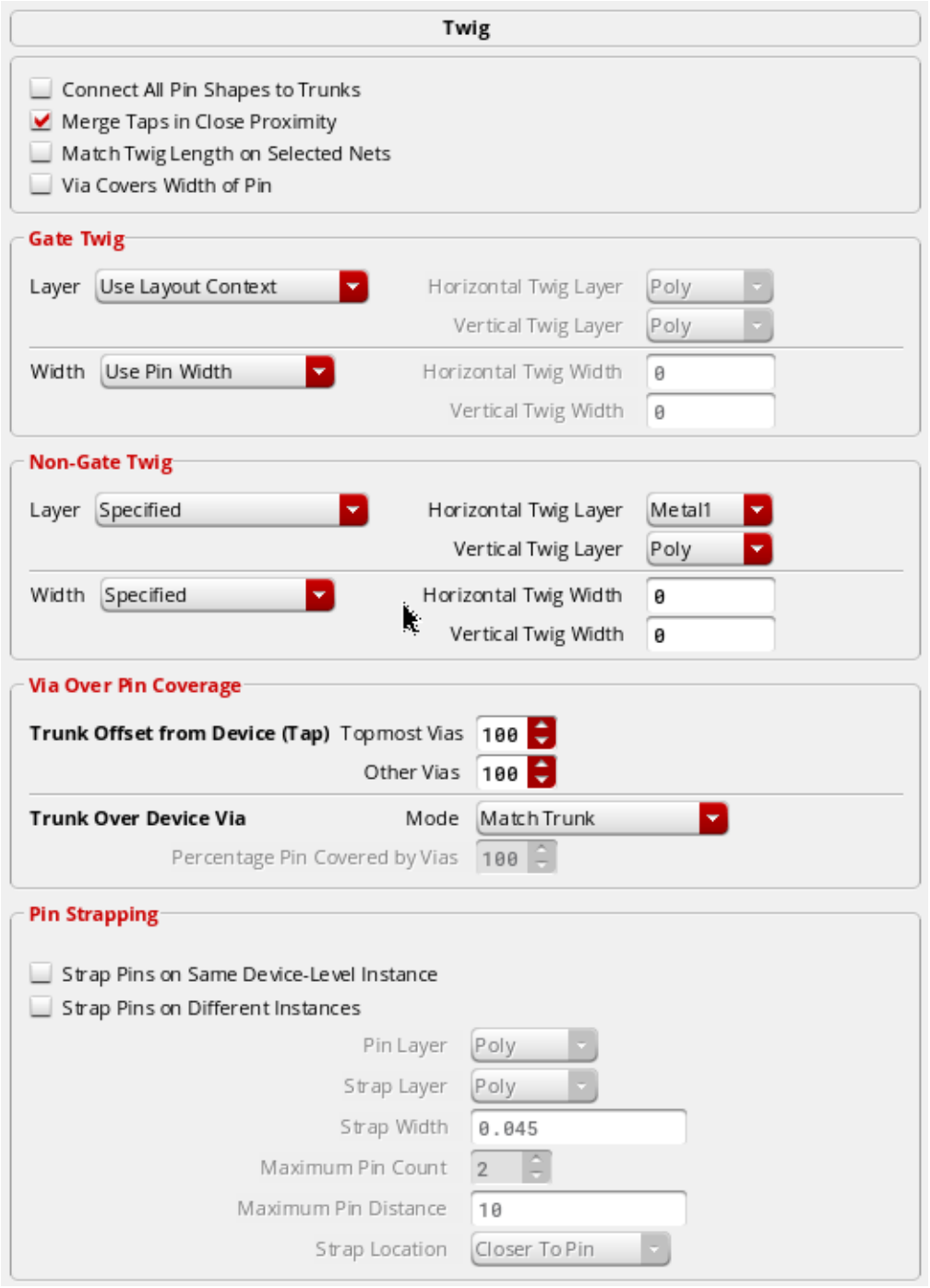
Environment Variable: trunkTaperMiddleIntervalLength

Specifying Twig Options

The *Twig* subform is displayed when you select *Twig* from the *Pin to Trunk* tree in the left pane of the Virtuoso Space-based Router Options form. In the *Twig* subform, specify the twig options for *Pin to Trunk* routing as explained in the following sections:

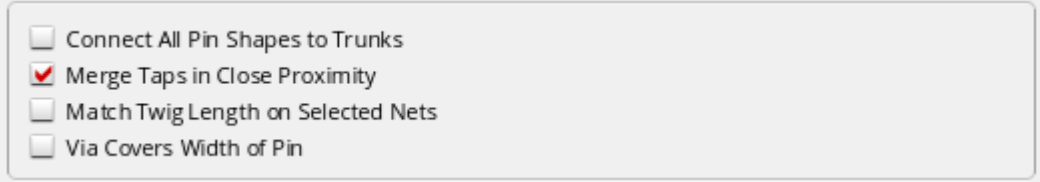
- Specifying General Twig Options
- Specifying Gate Twig and Non-Gate Twig Options
- Specifying Via over Pin Coverage Options

■ Specifying Pin Strapping Options



Note: All the options in the *Twig* subform are available only when the *Pin to Trunk* routing step is selected in the *Pin to Trunk* subform.

Specifying General Twig Options

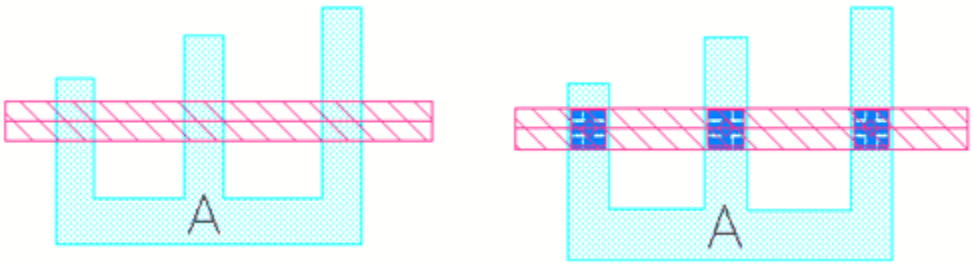


In the *Twig* subform, you can specify the following general options.

- Connect All Pin Shapes to Trunks
- Merge Taps in Close Proximity
- Match Twig Length on Selected Nets
- Via Covers Width of Pin

Connect All Pin Shapes to Trunks

A shape can be a simple shape (for example, rectangle) or a complex shape with multiple sub-shapes (for example, polygon). By default, the router connects to the closest simple shape, or a sub-shape of the closest pin of a terminal. However, if the *Connect All Pin Shapes to Trunks* check box is selected, the router connects to all the simple shapes and sub-shapes of all the pins of a terminal, as shown in the following figure.

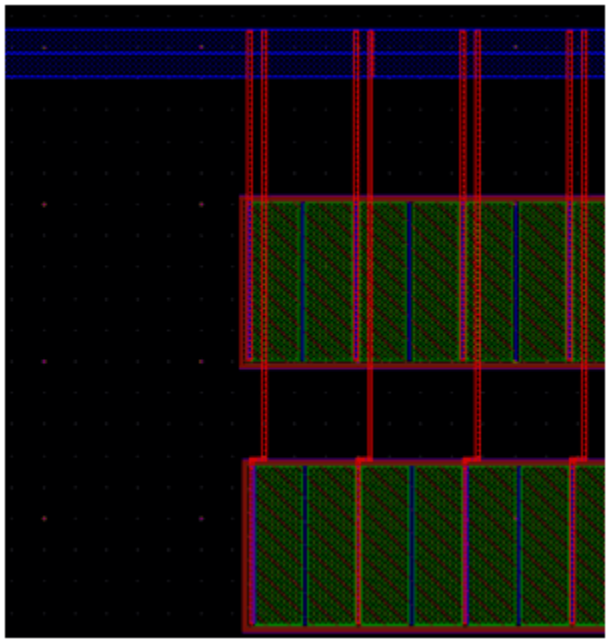


You can also specify a trunk range to control which shapes of a terminal should get connected. The *Connect All Pin Shapes to Trunks* check box is deselected by default.

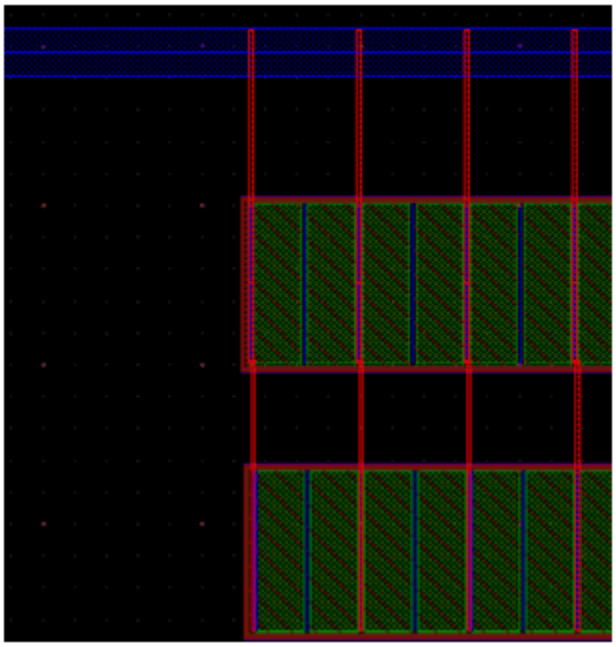
Environment Variable: connectMultiPinShapes

Merge Taps in Close Proximity

When selected, the router merges the taps that are close to each other. This means that the router merges adjacent parallel taps if the distance between them is less than the minimum spacing. When this check box is deselected, each tap is connected to the trunk. The *Merge Taps in Close Proximity* check box is selected by default.



When *Merge Taps in Close Proximity* is deselected

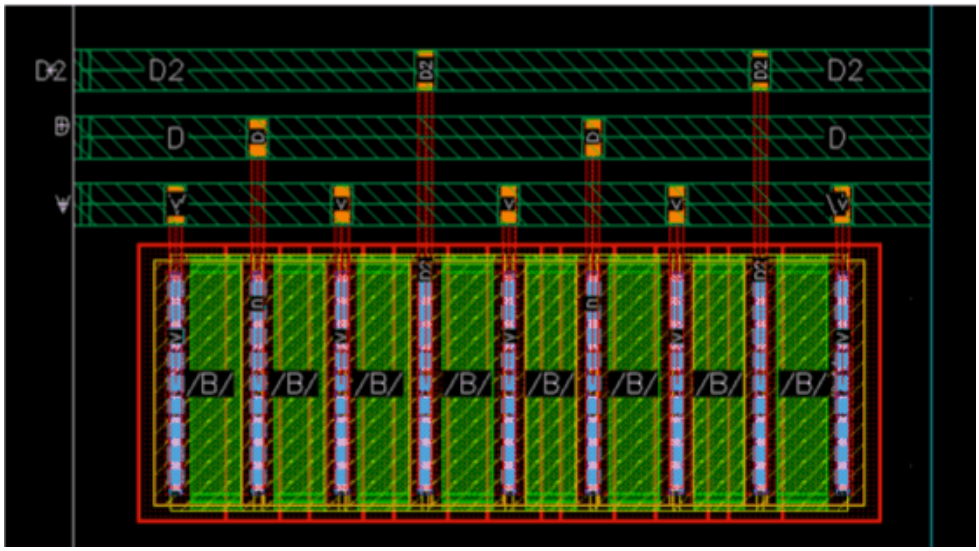


When *Merge Taps in Close Proximity* is selected

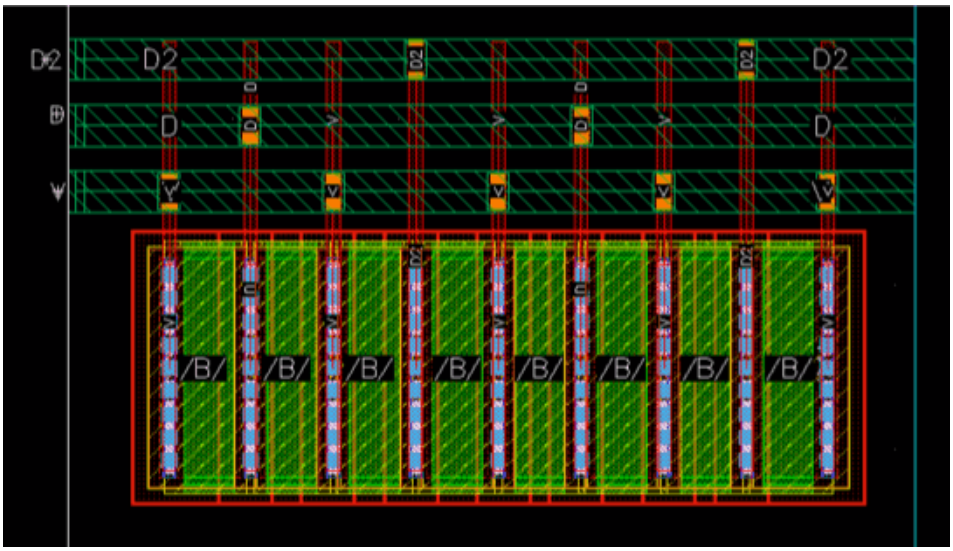
Environment Variable: mergeTapsInCloseProximity

Match Twig Length on Selected Nets

When selected, the shorter twigs are extended so that all twigs on the selected nets have the same length. The twig length is matched to the length of the largest twig irrespective of how far the trunk is placed from the pin, as shown in the following figure.



When *Match Twig Length on Selected Nets* is deselected

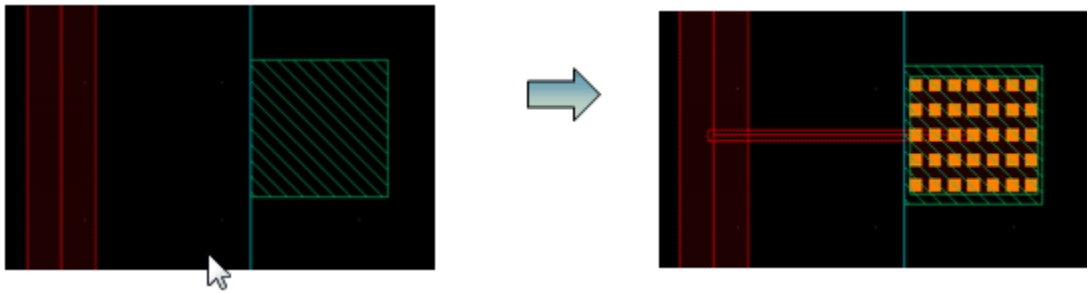


When *Match Twig Length on Selected Nets* is selected

Environment Variable: twigLengthMatching

Via Covers Width of Pin

When selected, covers the entire width of a pin with vias. Otherwise, covers pins using the width of the routed wire. By default, the *Via Covers Width of Pin* check box is deselected.



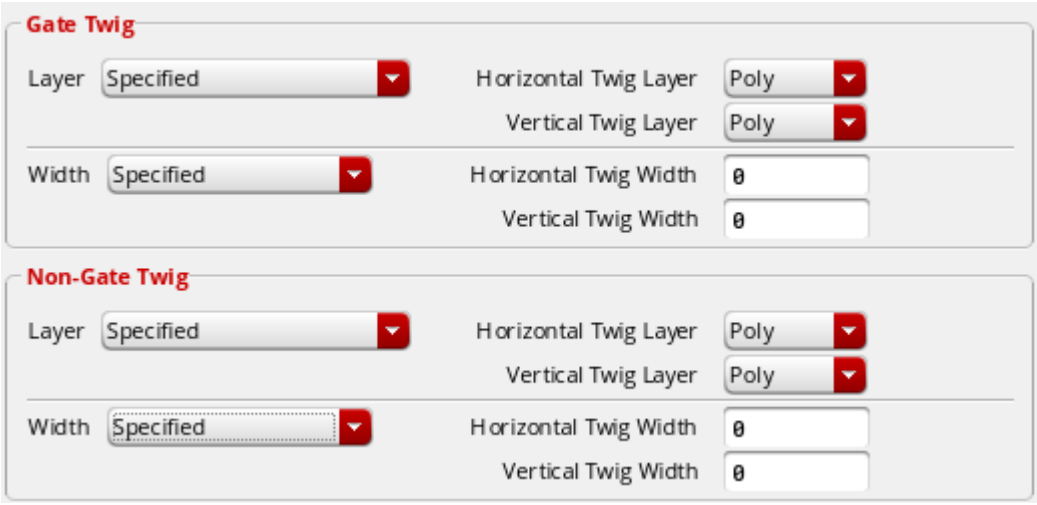
When *Via Covers Width of Pin* is deselected

When *Via Covers Width of Pin* is selected

Environment Variable: viaCoverPinWidth

Specifying Gate Twig and Non-Gate Twig Options

In this section of the form, you can specify the options for generating twigs between a trunk and poly (gate) or a trunk and source and drain pins.



- Layer

  Use this option to control how the router selects a layer for generating gate and non-gate twigs.

  Environment Variable: gateTwigLayerMode, nonGateTwigLayerMode

- Use Layout Context

  The router selects a layer to insert a twig from a set of valid routing layers.

- Prefer Routing Direction

  The router considers the preferred routing direction of the layer while creating a twig. If the router is unable to produce a suitable connection using a layer whose preferred routing direction matches the twig direction, it can select a layer whose preferred routing direction does not match with the direction of the twig.

- Prefer Pin Layer

  The router selects a layer matching the layer of a pin shape to which the twig is to be connected. The twig is created on the same layer as the pin shape. However, if this causes violations on the pin layer, router selects another layer on which violations are not caused.

  The router may not use *Prefer Pin Layer* to connect the twig if the result violates the Pin Escape mode even though *Prefer Pin Layer* option is selected.

- Specified

  Specify the twig layers to be used by the router for creating horizontal and vertical twigs.

  - Horizontal Twig Layer

    The router uses the selected layer for creating horizontal twigs.

    Environment Variable: horGateTwigLayer, horNonGateTwigLayer

  - Vertical Twig Layer

    The router uses the selected layer for creating vertical twigs.

    Environment Variable: verGateTwigLayer, verNonGateTwigLayer

- Width

  Use this option to control the width that the router uses to route a gate and non-gate twigs. The width can be controlled/specified using the following options:

  Environment Variable: gateTwigWidthMode, nonGateTwigWidthMode,

  - Use Net Constraints

    The twig width is determined by the width constraints specified for the net by using the constraint lookup method.

  - Use Pin Width

The router considers the width of the gate and non-gate twigs equal to the width of the pin.

- Specified

  You can specify a value for the twig width.

  - Horizontal Twig Width

    The router uses the specified width value for creating horizontal twigs.

    Environment Variable: horGateTwigWidth, horNonGateTwigWidth,

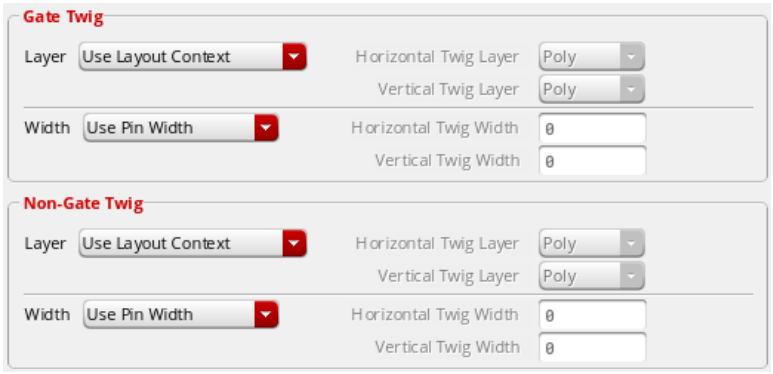
  - Vertical Twig Width

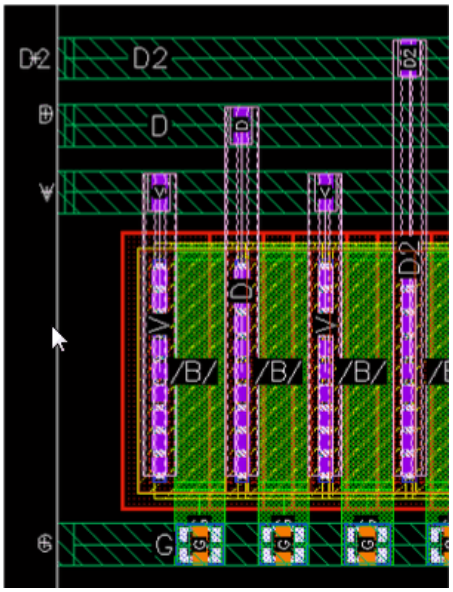
    The router uses the specified width value for creating vertical twigs.

    Environment Variable: verGateTwigWidth, verNonGateTwigWidth

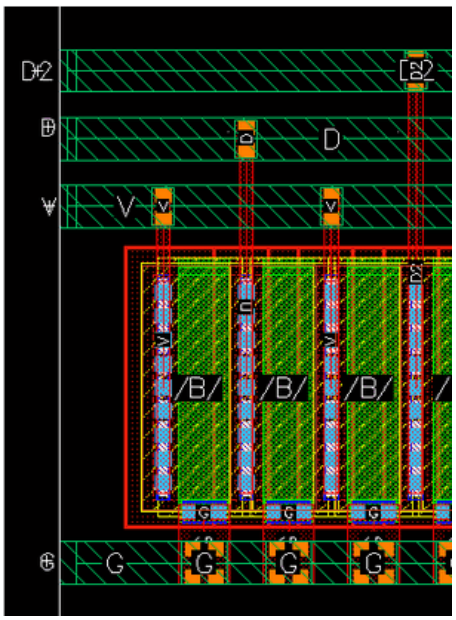
The following examples illustrate how gate and non-gate twigs are created for various options available in the *Gate Twig* and *Non-Gate Twig* sections.

Example 1: The following figure shows an example where the twig is created on a valid routing layer and the width of the twig is set equal to the width of the pin.





Example 2: The following figure shows an example where the twig is created on the same layer as the pin and the width of the twig is set equal to the width of the pin. For the non-gate twig, the layer is specified as `Metal4` and the width is specified as `0.2`.



Example 3: The following figure shows an example where the layer for gate and non-gate twigs is selected as `Metal1` and the width of the gate and non-gate twigs is set equal to the width of the pin.

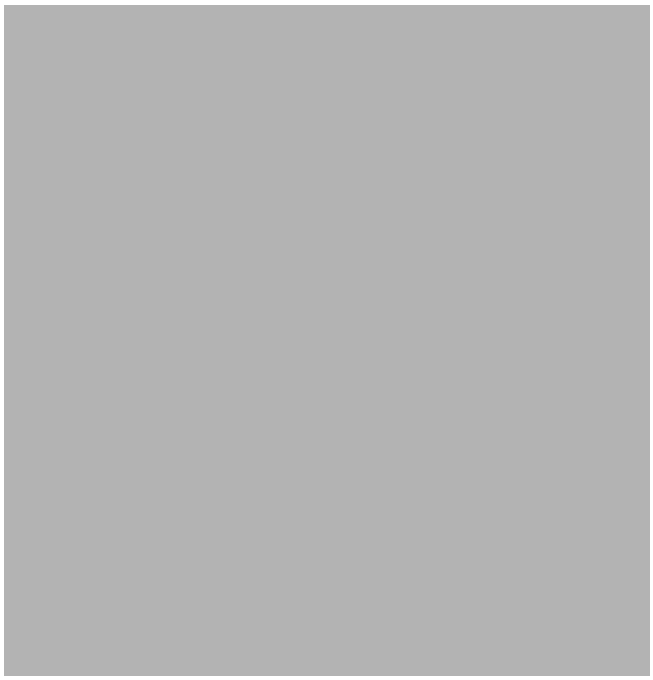
Specifying Via over Pin Coverage Options

You can specify the following pin coverage options:

- Trunk Offset from Device (Tap)
- Trunk Over Device Via

Trunk Offset from Device (Tap)

Specifies the percentage of a pin shape to be covered by a trunk tap when a trunk is not over a device. This reduces the parasitic capacitance effect across source and drain pins. In the following example, you can see that net `NetB` is routed with `M2` covering the `M1` pin shape and the intersecting area is flooded with vias. The percentage specified to be covered by a trunk tap is (45, 45). The percentage is specified in the *Topmost Vias* and *Other Vias* cyclic field.



- Topmost Vias

  Specifies the percentage of a pin shape to be covered by the topmost layer of the tapping vias.

  Environment Variable: tapCoverPinPercent

- Other Vias

  Specifies the percentage of a pin shape to be covered by the other layers of the tapping vias when the trunk is on a layer that is at least two layers above or below the pin layer.

  Environment Variable: tapLowerViasCoverPinPercent

Trunk Over Device Via

Controls the percentage of source and drain pin coverage with via stacks except for the top layer, when the trunk is entirely over the device. In the following example, you can see that `net A` has the trunk over the device and the selected mode is *Match Trunk*. Therefore, lower vias cover 100 percent of the trunk width.

Environment Variable: trunkOverDeviceViaCoverMode

- Match Trunk

  Covers the width of the trunk. This is the default option.

- Match Pin

  Covers the width of the pin.

- Match Pin Except Top

Covers the entire trunk on the top via layer and covers the entire pin on the lower via layers.

- Percentage Pin Covered by Vias

  Controls the percentage of a trunk or a pin to be covered with vias. This option is enabled only when the *Match Pin* or *Match Pin Except Top* option is selected from the *Mode* drop-down list.

Environment Variable: trunkOverDeviceViaCoverPercent

Specifying Pin Strapping Options

In the *Pin Strapping* group box, you can specify the following pin strapping options:

- Strap Pins on Same Device-Level Instance

  The router straps the gate, source, and drain pins of the same instance and same net together before making a connection to the trunk.

  Environment Variable: strapGateSourceDrainPins

- Strap Pins on Different Instances

  This option is applied to a set of pins only if they satisfy the following conditions:

  - The pins are on the same layer.
  - The pins are on the same side of the trunk to which they need to be connected.
  - The difference between the trunk-edge to pin-edge distances amongst the pins must not be larger than the width of the strap.

In addition, the edge-to-edge distance between adjacent pins must be within the maximum pin distance before router straps them together.

Environment Variable: pinStrapping

By default, the *Strap Pins on Different Instances* check box is deselected, which means that the strapping of pins on different instances is off. The following figure shows how routing is done when this check box is deselected.

The following figure displays the routing when the check box is selected.

When the *Strap Pins on Different Instances* check box is selected all the options in the *Pin Strapping* group box are enabled.

- Pin Layer

  Lets you select the layer on which pins should be strapped.

  Environment Variable: pinStrapPinLayer

- Strap Layer

  Lets you select the layer on which strapping should take place during Pin to Trunk routing.

Note: The layers available in the *Strap Layer* drop-down list match the routing layers that are available in the *Top* and *Bottom* layers drop-down list in the Wire Assistant.

Environment Variable: pinStrapLayer

- Strap Width

  Specifies the width of the strap that is created during Pin to Trunk routing. The strap width is automatically updated when the strap layer is changed. The strap width is calculated using the minWidth of the strap layer that is selected.

  Environment Variable: pinStrapWidth

- Maximum Pin Count

  Specifies the maximum number of pins that can be strapped during Pin to Trunk routing.

  Environment Variable: pinStrapMaxPinCount

- Max Pin Distance

  Specifies the maximum edge-to-edge distance between two adjacent pins to be strapped.

  Environment Variable: pinStrapMaxPinDistance

- Strap Location

  You can select the location where the strap should be created.

  - Closer to Pin

    The strap is created closer to the pin than the trunk. This option is selected by default. The following figure shows pin strapping on `Metal1` layer, with the strap created closer to the pin.

  - Closer to Trunk

The strap is created closer to the trunk than the pin. The following figure shows pin strapping on `Metal1` layer, with the strap created closer to trunk.

Environment Variable: pinStrapLocationPreference

Specifying Trunk to Trunk Connections

The *Trunk to Trunk Connection* subform is displayed when you select *Trunk to Trunk Connection* from the *Pin to Trunk* tree in the left pane of the Virtuoso Space-based Router Options form.

Environment Variable: connectTrunkType

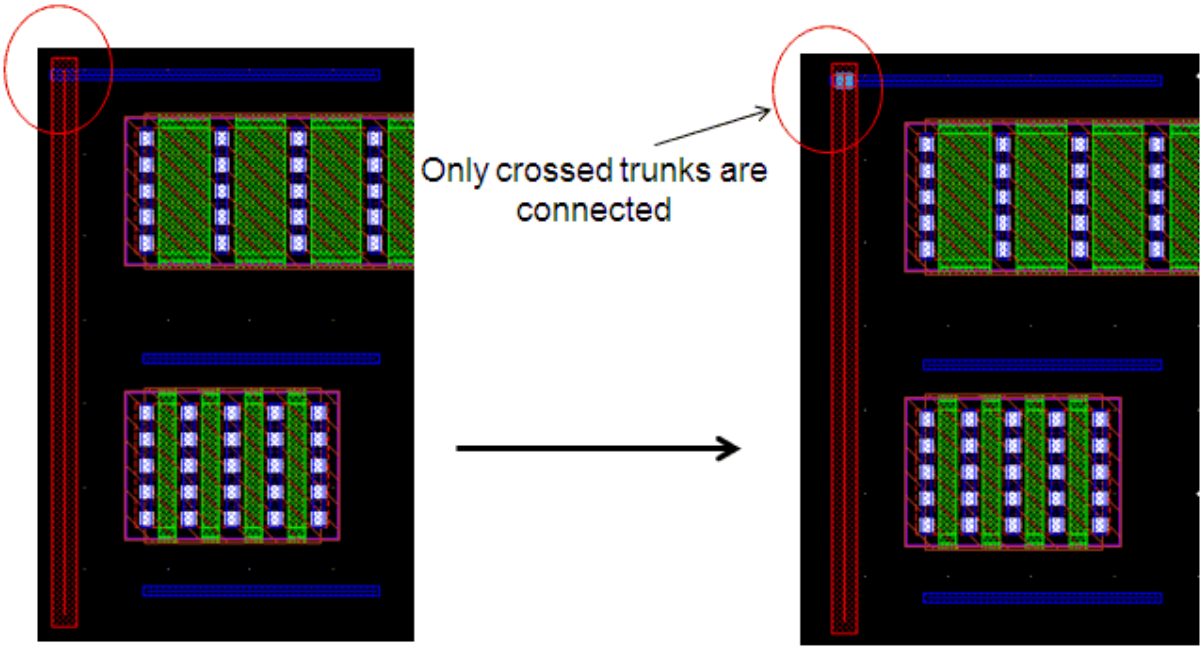
You can specify the following modes for the trunks:

- All Trunks

Connects all crossed and floating trunks, as shown in the following figure.

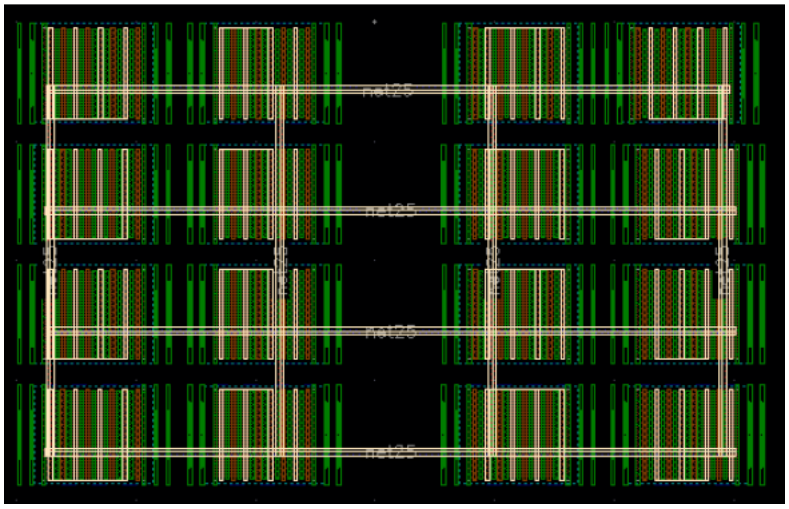
- Crossed Trunks Only

  Connects only crossed trunks, as shown in the following figure.



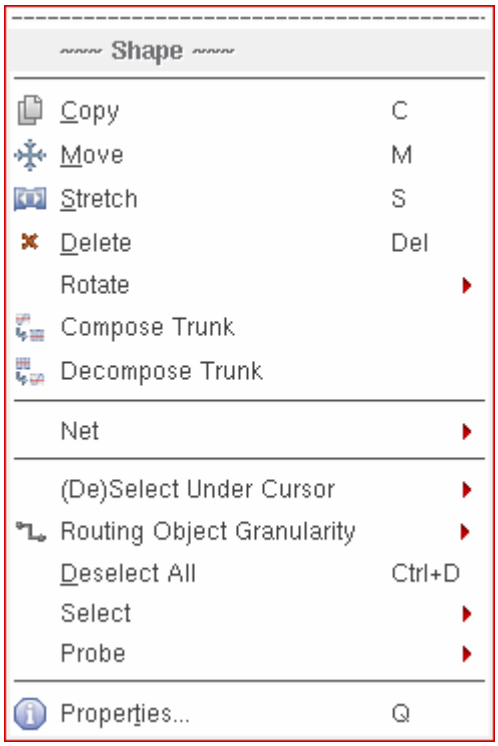
- Trunk Mesh Routing (ICADVNM18.1 Only)

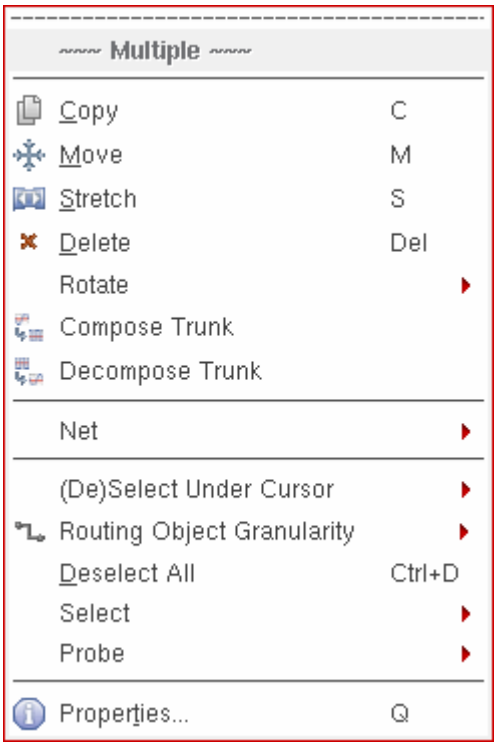
  Lets you customize the way meshes are routed. When you click the ellipses button next to the *Mode* field, the Trunk Mesh Routing Configuration form displays. This form displays a table of layer settings and a set of options that you can use to specify the way each layer is to be routed in the mesh. For more information on the options, see Trunk Mesh Routing Configuration Form (ICADVM18.1 EXL Only).



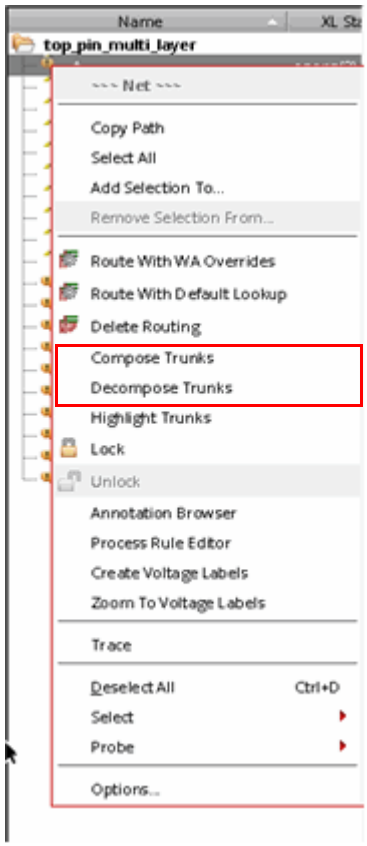
Composing and Decomposing Trunks

The *Compose Trunks* and *Decompose Trunks* commands are available in VLS XL in the *Edit – Wiring* menu. You can also access these commands from the *Shape* and *Multiple* context-sensitive menus that display if you right-click in the design display area after selecting one or more objects.





Alternatively, you can access the *Compose Trunks* and *Decompose Trunks* commands from the Navigator context-sensitive menu. The context-sensitive menu is displayed when you right-click the net after selecting it in the Navigator Assistant.



Use the *Compose Trunks* command to convert the selected set of shapes into a trunk object. Only pathsegs, vias, and rectangles are converted into trunk objects. Other shapes, such as paths and polygons, are not converted. For each object in the selected set that fails to be converted into a trunk object, the command creates an OA marker. This OA marker is visible in the Annotation Browser. A message in the CIW summarizes the objects that are converted to form a trunk. These trunk objects can then be used to perform pin to trunk routing. The pin to trunk router is started by default when you run the route command from the *Net* context-sensitive menu after selecting the trunk net in the *Navigator* assistant.

Use the *Decompose Trunks* command to convert the selected set of trunk, comprising selected set of shapes, to the original shapes from which it was formed. A message in the CIW summarizes the objects that are converted to original shapes. If the selected shapes are not part of a trunk and you run the *Decompose Trunks* command, the number of converted objects reported in the CIW is `0`.

Highlighting Trunks

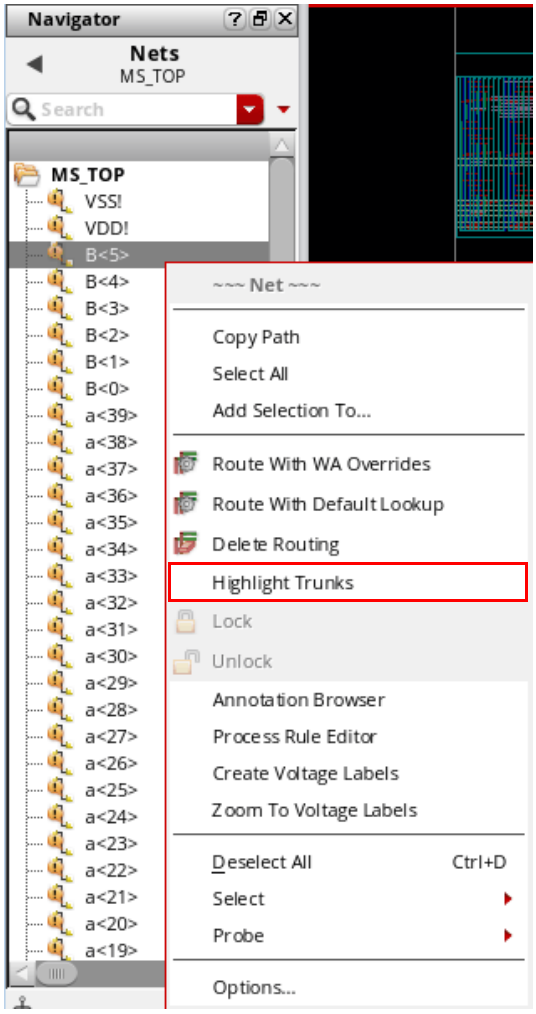
 Video

For a short demonstration on how to highlight trunks and view the twig lines, see Highlight Trunks.

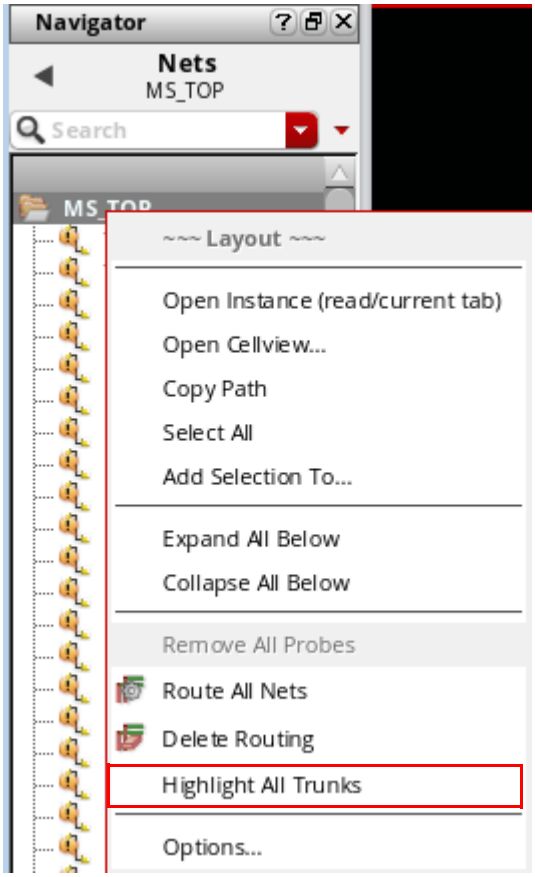
The *Highlight Trunks* option helps you to identify the trunk for which the twigs will be created. To identify composed trunks of a selected net and to highlight the trunks and twigs in the layout canvas, right-click the net in the Navigator and choose *Highlight Trunks* from the context-sensitive menu. The twigs are highlighted based on the options in the *Scope* section of the *Pin to Trunk* subform. The options that are considered while highlighting the trunks and twigs are as follows:

- *Area* options: *Current Editable Cellview*, *Current View Area*, and *Specified Area*,
- *Pin-Trunk Relation*: Pin mode (Channel, Space, All, Auto),
- *Pin side* (North, South, West, East, Over)

- *In Range*: the specified range of the cellview

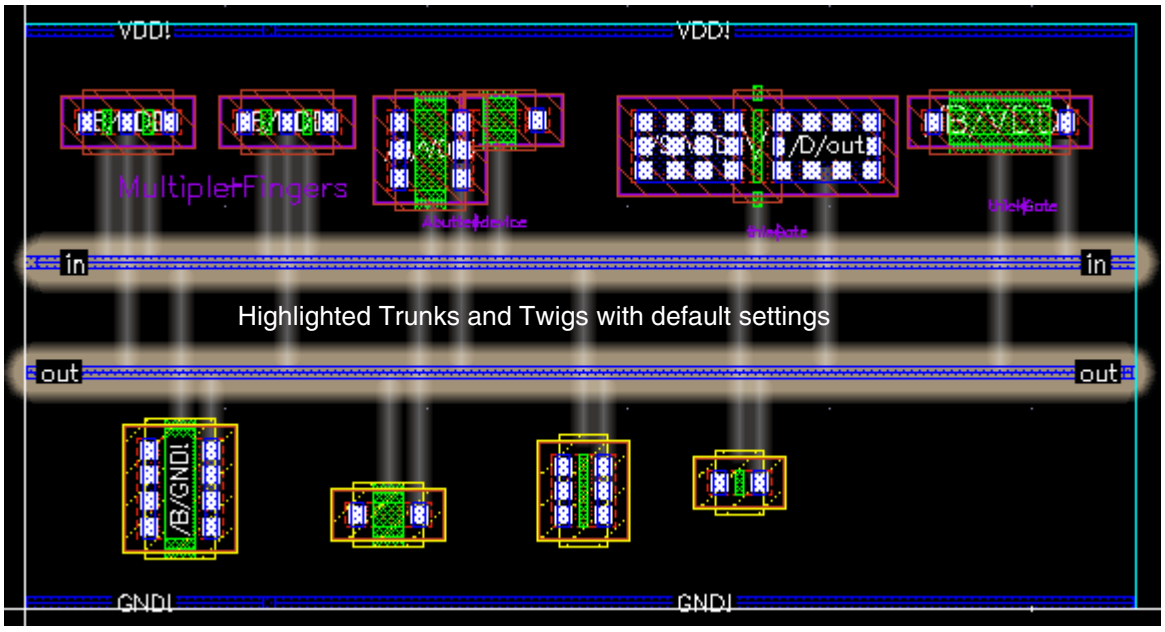


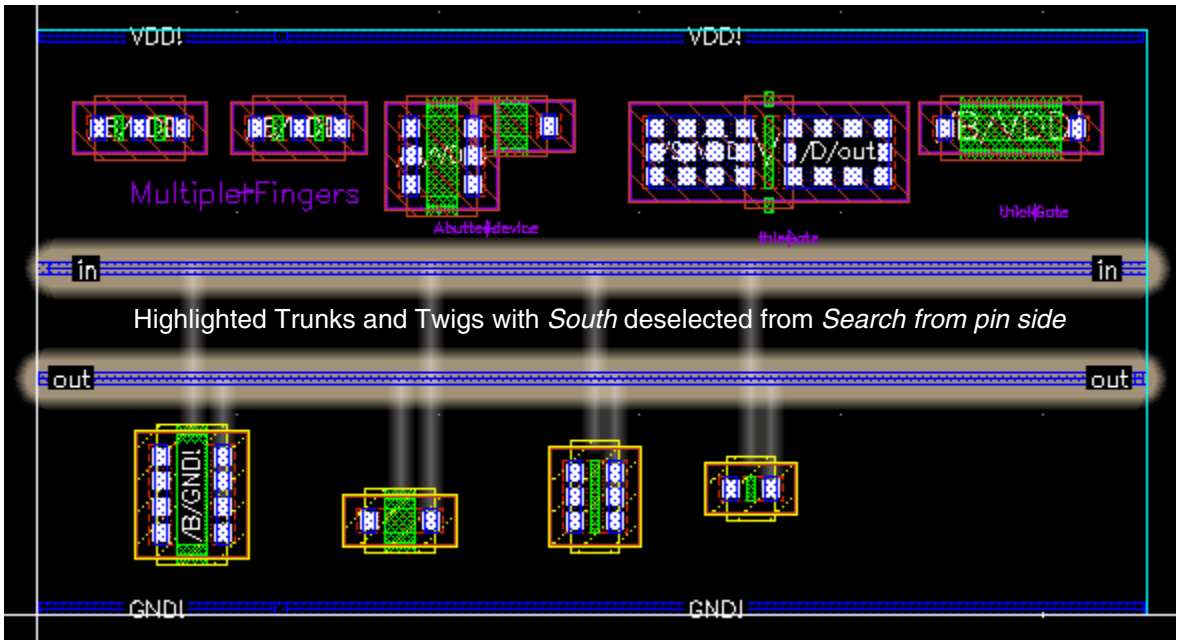
To highlight trunks for all the nets that have a valid composed trunk in a cellview, right-click the cell and choose *Highlight All Trunks* from the context-sensitive menu.



Trunks and twigs that are legal and can be recognized by the router are highlighted.The following figure shows an example where the trunks and twigs are highlighted when the

Current Editable Cellview option is selected and in the *Search from Pin Side*, *South* is deselected.





The trunks and twigs remain highlighted—even when you zoom in or out the layout canvas—until you change the selected set either in the Navigator or in the layout canvas.

Environment Variable: highLightTwigType

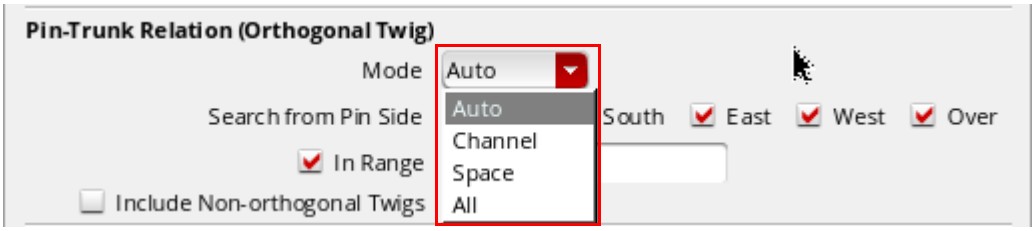
Using the Finish Trunk Command

The *Finish Trunk* command on the *Create Bus* and *Create Wire* context-sensitive menu automatically converts all the pathSegs except for the first and the last pathSeg into trunks, regardless of whether they are connected to a pin or not. The first and the last pathSeg is considered as a regular pathSeg if it is connected to a pin at one end and to a trunk at the other end. Else, it is converted to a trunk. The default bindkey for running the *Finish Trunk* command is `2`.

The *Finish Trunk* command enables pin to trunk connections only when *Pin to Trunk Routing* is selected from the Pin to Trunk options in the Virtuoso Space-based Router Options form. If the *Pin to Trunk* routing step is deselected, the *Finish Trunk* command skips pin to trunk routing. In addition, if the *Compose Trunks* step is selected, then all the wire segments are converted into trunks, regardless of whether they are connected to the pins or not. By default, the *Compose Trunks* routing step is deselected.

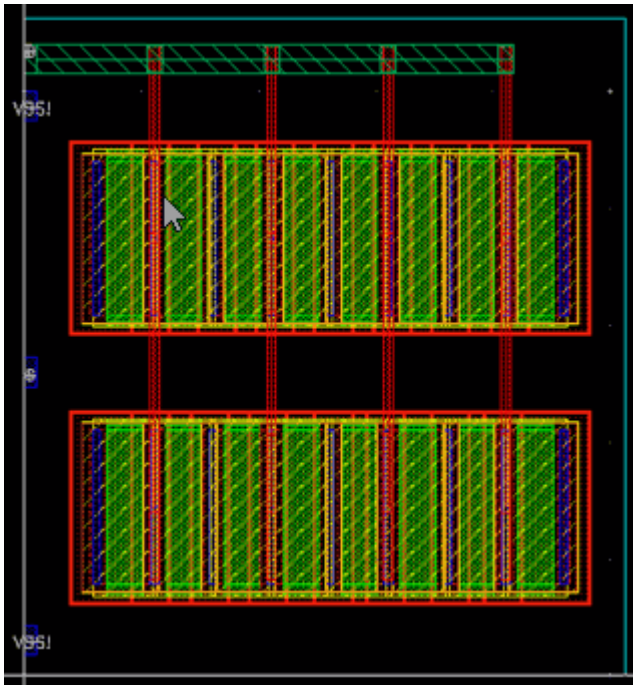
Note: The *Finish Trunk* command connects intersecting trunks only when the *Trunk to Trunk Routing* step is selected. The non-intersecting trunks are not connected by the *Finish Trunk* command.

To determine which pins should be connected, the *Finish Trunk* command uses the value selected in the *Mode* field in the *Pin-Trunk Relation* (*Orthogonal Twig*) section in the *Scope* group box of the *Pin to Trunk (Default)* subform.



The default value of *Mode* is *Auto*, which means that the *Finish Trunk* command selects only the pins that are in the same channel as the trunks. To allow the *Finish Trunk* command to connect the pins that are not in the same channel as the trunk, select either the *Space* or

All option from the *Mode* drop-down list. The following figure shows a layout design on which the *Finish Trunk* command was run successfully with *Mode* set to *All*.



10

Using the Tree Route Flow (ICADVM20.1 EXL Only)

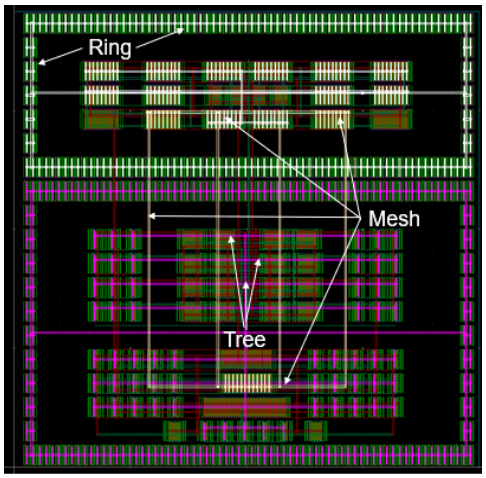
The license required for Tree Route flow to run on ICADVM20.1 is `95800 Virtuoso_Layout_Suite_EXL`. In addition, it checks out a 12 GXL tokens.

This chapter includes the following topics.

About Tree Route

The Tree Route is a new routing flow that helps you increase layout productivity through -automation. This routing flow lets you to automatically route advanced node device-level, row-based designs with a minimum set of options. The Tree router lets you quickly connect device

pins in a structured topology. It automatically identifies and routes mesh, ring (tap cell rings), and tree structures in the design.



Video

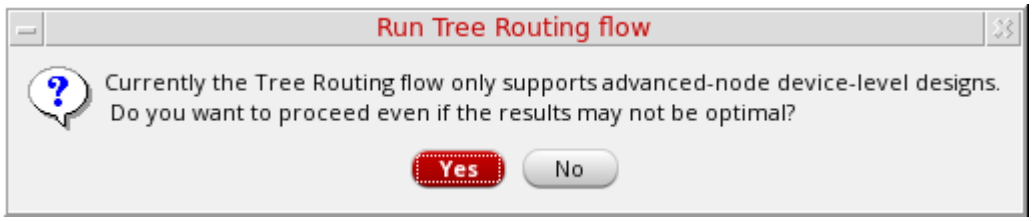
For a short demonstration on how to use the new tree route flow, see Automated Structured Routing using Tree Router.

Pre-requisites for Tree Router

The two main requirements to run the Tree router are as follows:

- You should be using Layout EXL.
- The technology file must be available to run the Pin to Trunk router.
- Ensure that *Design Style* in the Wire Assistant is selected as *Device Level*.

  In case, tree route is run on a layout with stdcells, a message box is displayed stating that the Tree route flow supports only advanced node device-level designs. Do you want to proceed even if the results may not be optimal? This message is also displayed when the design style in Wire Assistant is set to *ASIC* or *Chip Assembly*.



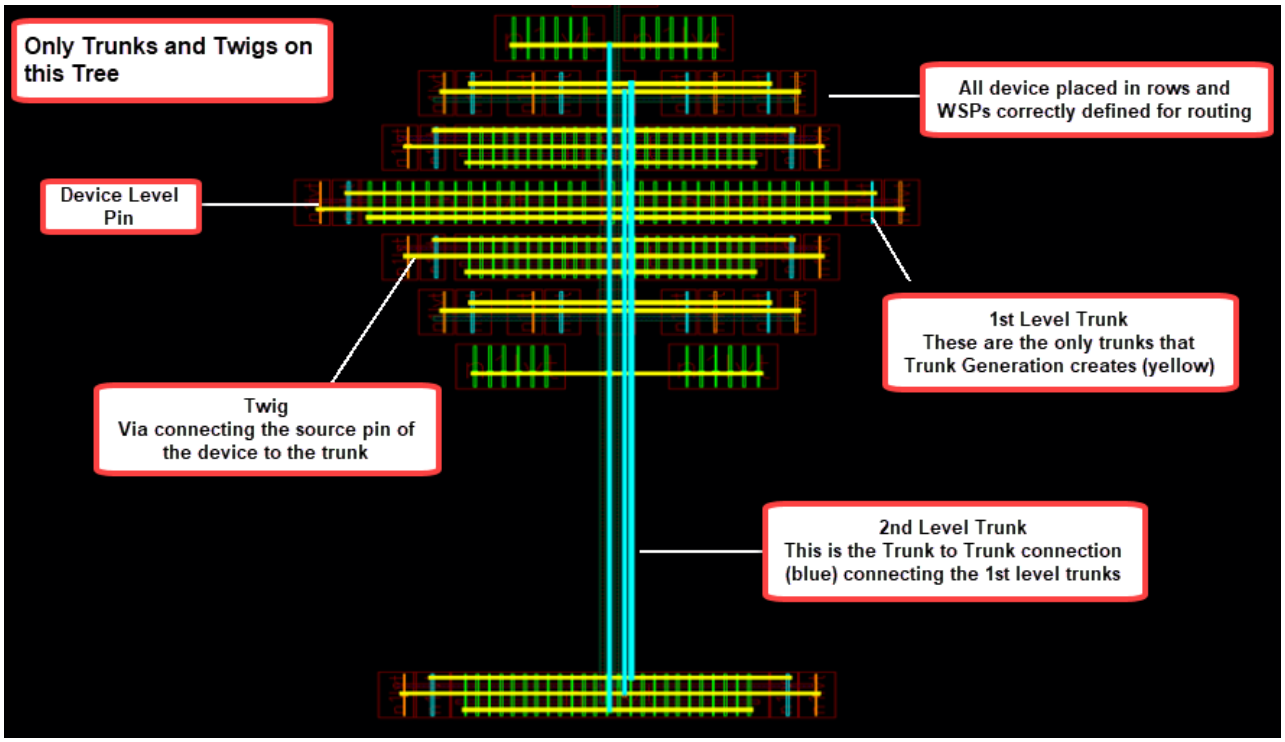
Clicking the *Yes* button in the message box, runs the Tree route flow and a pre-route check summary is displayed in CIW, as shown in the following figure.

```
*********************************************************************
************  Tree Routing Pre-Route Check Summary  **************
*********************************************************************
Design Statistics
    Total number of rows                : 3
    Number of long rows                 : 0
    Total number of instances           : 3
    Number of device instances          : 0
    Number of instances in row          : 3
    Number of instances in long row     : 0
    Percentage of device instances      : 0.00
    Percentage of instances in long row : 0.00
-------------------------------------------------------------
Design Check
    Device-level           : NO
    Row-Based              : YES
    Check Status           : FAIL
Tech Check
    Process Node           : Above 32nm
    Process Node Support   : NO
    Check Status           : FAIL
Pins Check
    Top-level Pins
        Not On Tracks      : 0
        Violate WSP Width  : 0
    All Pins
        Have shorts        : 0
    Check Status           : PASS
-------------------------------------------------------------
Overall
    Number of new markers  : 0
    Check Status           : FAIL
```

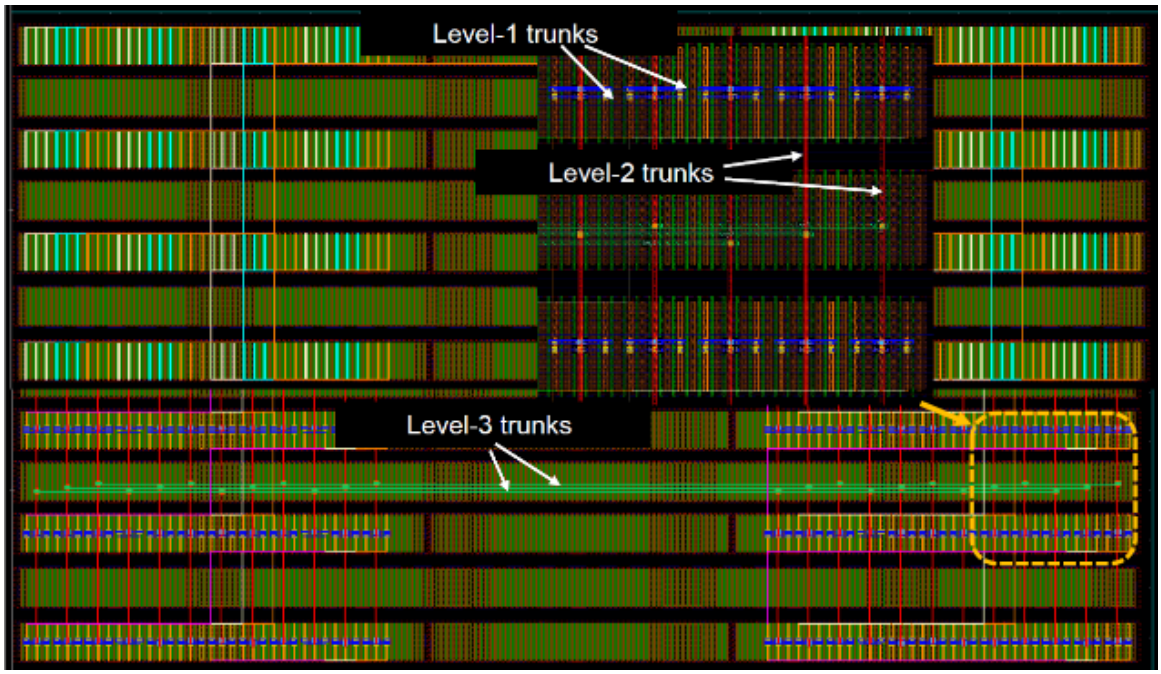
Analyzing the Tree Route Structure

When you highlight trunks after running the Tree router, the tree design displays different level of trunks in different colors. The level-1 trunks are highlighted in Yellow and the level-2 trunks

are highlighted in Cyan. The following figure explains the structure and composition of Tree Route.



In addition, the Tree router automatically creates and connects all levels of trunks.



Displaying the Tree Routing Options

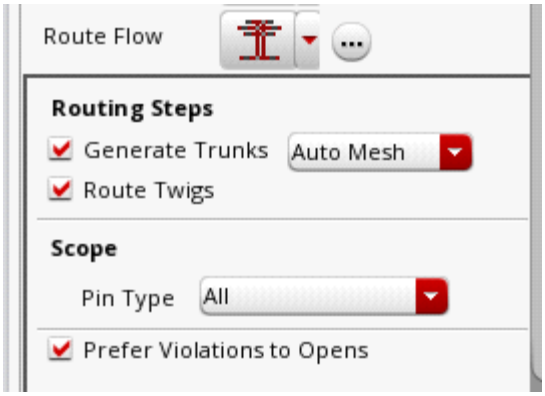
To display the Tree routing options, perform the following steps:

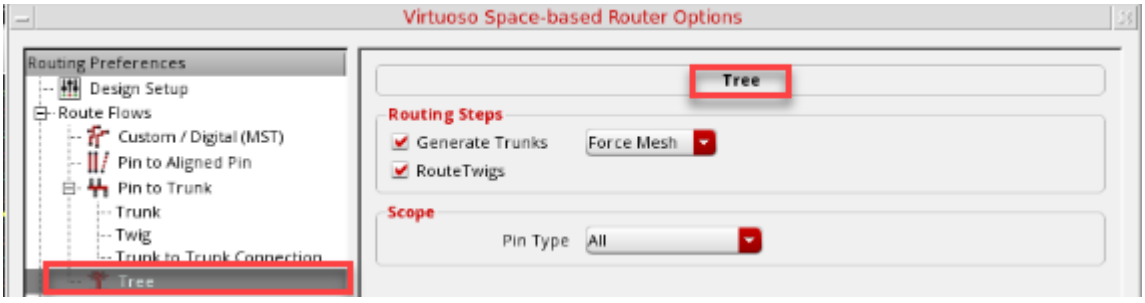
1. Choose *Window – Assistants – Wire Assistant.*
2. *In the Wire Assistant, select Auto Route from the Command drop-down list in the Automatic section*.
3. Select Tree from the Route Flow drop-down list. The Tree routing options are displayed in a group box just below the Route Flow field.

   Alternatively, select Tree from the Route Flow drop-down list and click the ellipses button next to it. The Tree routing options are displayed in the Tree subform of the Virtuoso Space-based Router Options form.

Specifying the Tree Route Options

You can specify the Tree Route options either in the Wire Assistant or in the Tree subform of the Virtuoso Space-based Router Options form.





- Generate Trunks

  Generates all level of trunks. If you want to use more options by using the Pin to Trunk routing for twig routing layer or if you want to move or add trunks before routing twigs, you can run Generate Trunks as a separate step.

  Generate Trunks has the following three options:

  - Auto Mesh

    Enables the router to automatically identify the regions for mesh routing. You can also force it on or off.

  - No Mesh

    Lets the router to perform routing without creating any mesh. This is the default option.

  - Force Mesh

- Forces mesh routing when *Auto Mesh* results in no mesh based on the properties of the net.

- Route Twigs

  Routes twigs, which are the connections between the pins of devices and the 1st level trunks.

Note: The *Generate Trunks* and *Route Twigs* options ignore all the Pin to Trunk options and route the design using the Tree routing algorithms.

- Scope

  Defines the scope of tree routing

- Pin Type

  Lets you specify the type of pins to connect to. Pin Type has three options: All, Gate, and Source And Drain. You select either one of the option to connect to all gate and source and drain pins, only gate pins, or only source and drain pins.

Using Auto Device Routing Preset

The *Auto Device Routing* preset option is available in the *VSR Load Preset* drop-down menu. It lets you automatically load the device routing option settings and enable the *Tree Route* flow in the Wire Assistant.

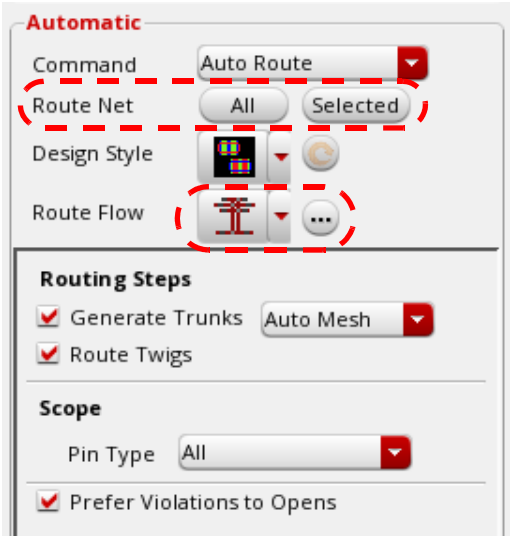
VSR Toolbar Wire Assistant

Running Tree Routing

When using Layout EXL, you can run Tree routing, using one of the following methods.

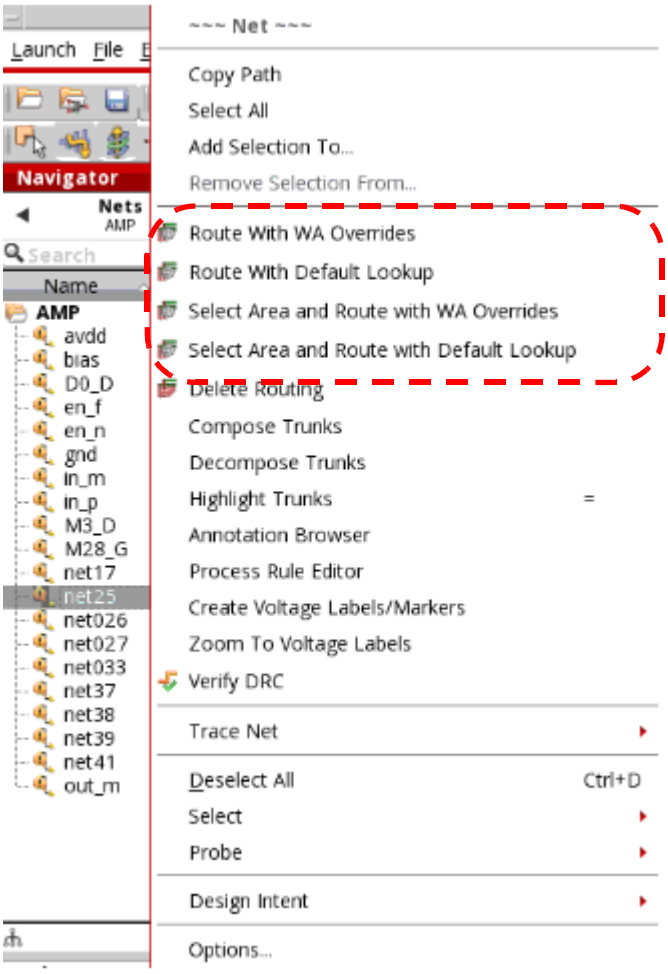
- Use the Automatic Routing icon on the Virtuoso Space-based Router toolbar.

- Using Wire Assistant
  - Select the Tree option from the Route Flow drop-down list.
  - Click the All or Selected button next to Route Net.



- Using Navigator Assistant
  - From the Navigator assistant, select a net.
  - Right-click the selected net.

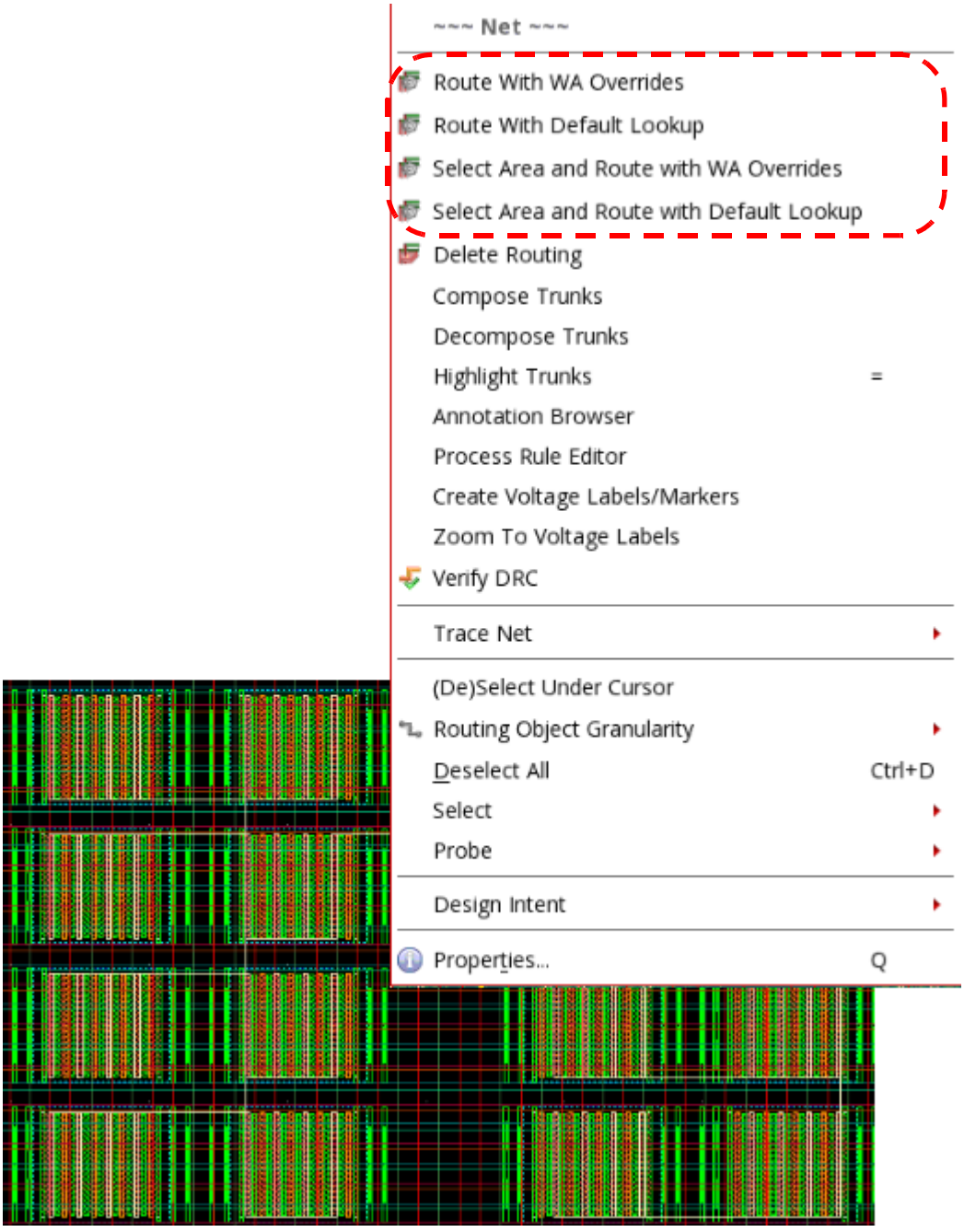
- Choose one of the routing options from the Navigator drop-down list, as shown in the figure below.



■ Using context-sensitive menu in layout

- From the layout canvas, select a net.
- Right-click the selected net.

- Choose one of the routing options from the Net context-sensitive menu, as shown in the figure below.



Related Topics

Using Tree Routing

Tree Route (ICADVM 18.1 Only)

11

Working with Assisted and Automated Structured Routing (ICADVM20.1 EXL Only)

The first step towards assisted and automated structured routing is more automation and less GUI options to be specified for routing the design.

For automated structured routing, you can use one of the following features.

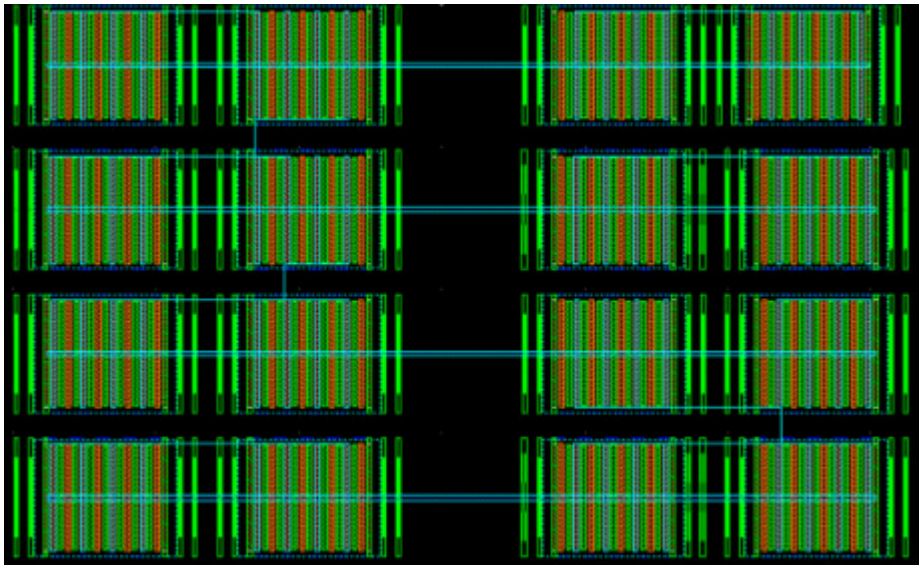
- Using Automatic Trunk Generation in Pin to Trunk Route Flow
- Using Trunk Mesh Routing in the Pin to Trunk Route Flow
- Using Tree Routing
  - Performing Area-based Tree Routing

Using Automatic Trunk Generation in Pin to Trunk Route Flow

The following steps show you how to use automatic trunk generation in the Pin to Trunk routing flow.

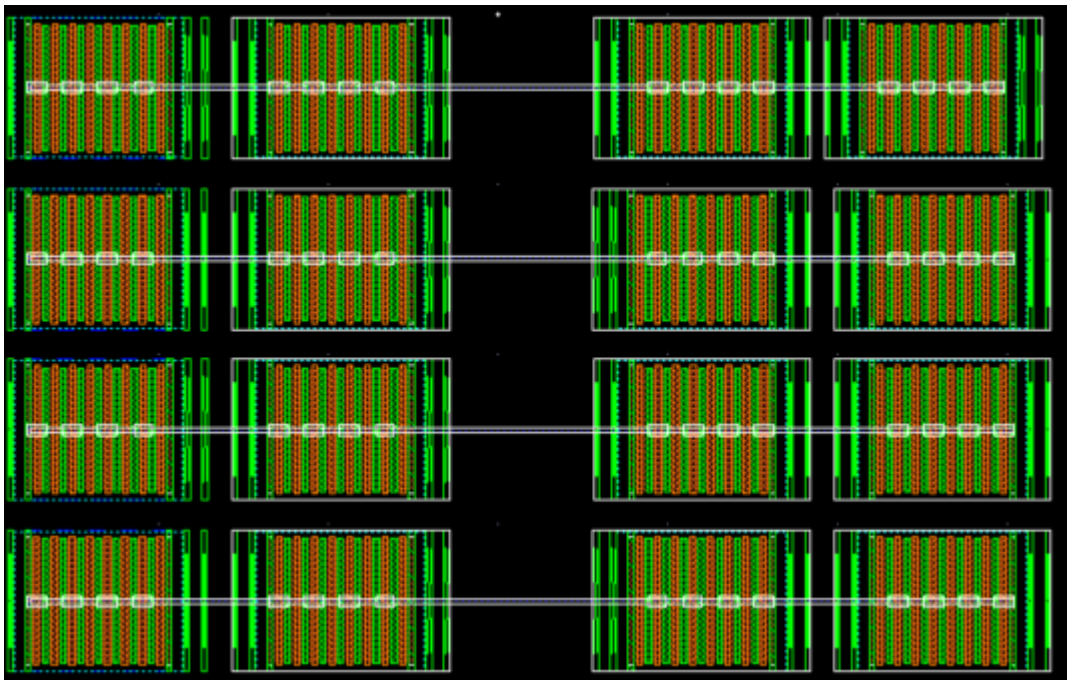
1. Open a design in the layout window.
2. In the Navigator Assistant, select *net25*. The instance pins on the selected net are highlighted.
3. Select the design style as *Auto* from the *Design Style* drop-down list.
4. Select the routing flow as *Pin to Trunk* from the *Route Flow* drop-down list.
5. Ensure that the *Generate Trunks* option is selected in the *Routing Steps* group box.
6. Click *Selected* in the *Route Net*.

The following figure shows the trunks that are generated for the routed net. You can see that there are no vias connecting the trunks to the pins. This is because the *Route Twigs* step is deselected.



7. Now, delete the routing on the selected net by clicking the *Undo* button on the *Edit* toolbar.
8. Again, perform step 2 to step 5.
9. Now, select the *Route Twigs* option in the *Routing Steps* group box.
10. Click *Selected* in the *Route Net*.

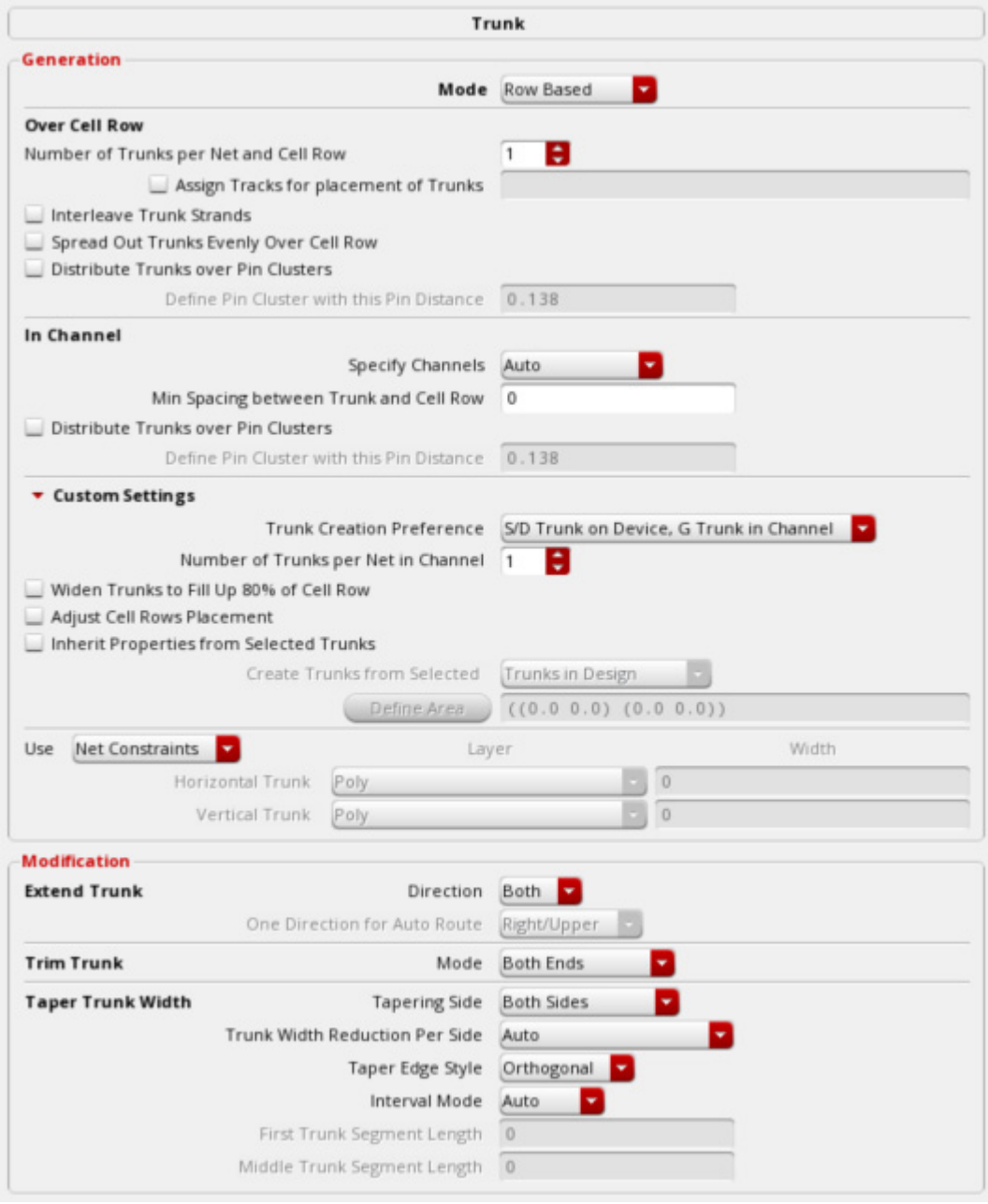
The following figure shows the trunks that are generated for the routed net. Observe that now there are vias connecting the trunks to the pins. This is because the *Route Twigs* step is selected.



Note: To limit vias stacking at the trunk intersection, go to the *Via Config*uration tab and deselect the layers that are not required.

11. In the Wire Assistant, click the *Options* button next to *Route Flow*. This displays the Virtuoso Space-based Router Options form.

12. To view the trunk generation options, click the *Trunk* option below Pin to Trunk.



13. Specify the options in the *Over Cell Row* section and see how the generated trunks are changed according to each option. For more information, see, Specifying Over Cell Row Options.

Note: When the pins of the selected net are source and drain pins only and no gate pins, only the *Over Cell Row* options are applicable for such a net in the design.

14. Specify the *In Channel* options for gate pins. For more information, see Specifying In Channel Options.

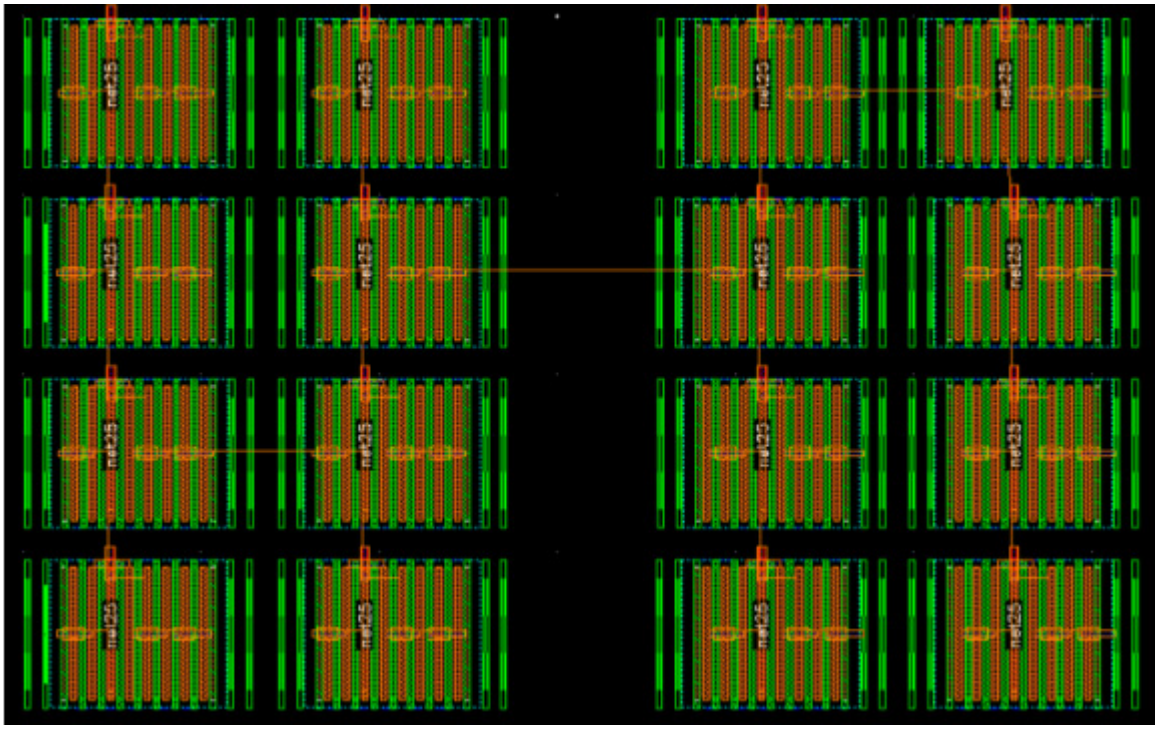
    Note: The *In Channel* options are applicable only for the gate pins of the selected net in a design.

15. If you want to customize the settings for the selected net, click *Custom Settings*. and specify the options. For more information, see Specifying Custom Settings.

16. Click *Close*.

17. Click *Selected* in the *Route Net*.

    The following figure shows the routing result when few of the over cell row options for source and drains pins have been specified in the Trunk subform.



Video

For a short demonstration on how to use the new trunk generation feature, see Automatic Trunk Generation using Pin To Trunk (P2T) Route Flow in Wire Assistant.

Related Topics

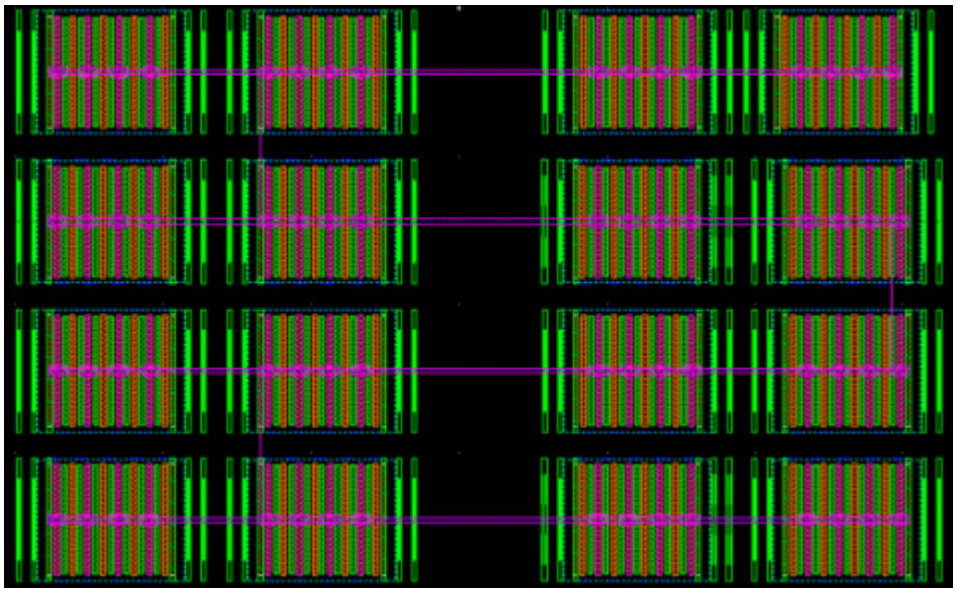
Specifying Trunk Generation Options (ICADVM18.1 EXL Only)

Using Trunk Mesh Routing in the Pin to Trunk Route Flow

The following steps show how to use trunk mesh routing using the *Pin to Trunk* routing flow.

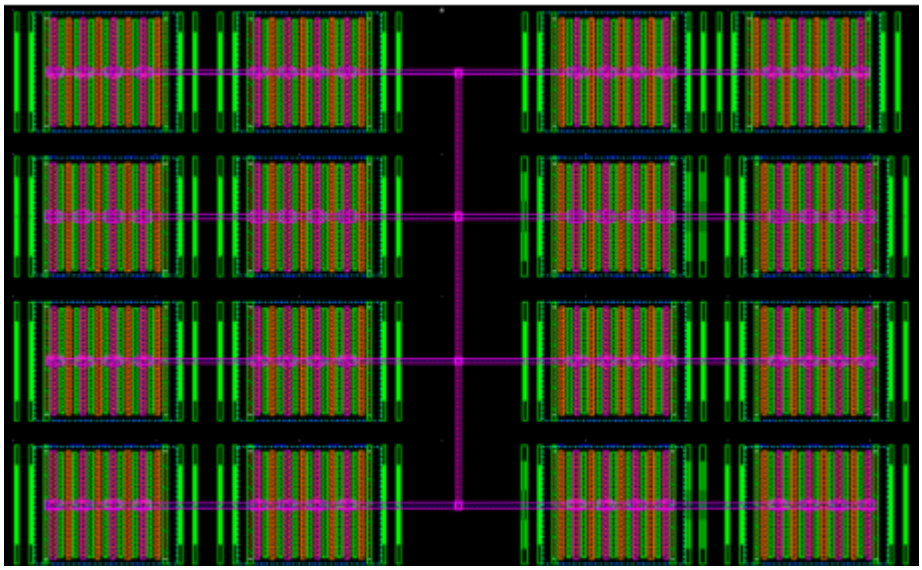
1. Open a design in the layout window.
2. In the Navigator Assistant, select *net25*. The instance pins on the selected net are highlighted.
3. Select the design style as *Automatic* from the *Design Style* drop-down list.
4. Select the routing flow as *Pin to Trunk* from the *Route Flow* drop-down list.
5. Ensure that *Generate Trunks* and *Route Twigs* options are selected in the *Routing Steps* group box.
6. To generate trunks and route the twigs for the selected net, click *Selected* in *Route Net*.

   You can view the generated trunks. Observe that there are vias connecting the trunks to the pins. This is because the *Route Twigs* step is selected. The following figure displays the routed net. However, the trunks are not connected to one another.



7. To connect the trunks, select *Trunk to Trunk* routing step. Ensure that the mode is selected as *All Trunks*.
8. Click *Selected* in *Route Net*.

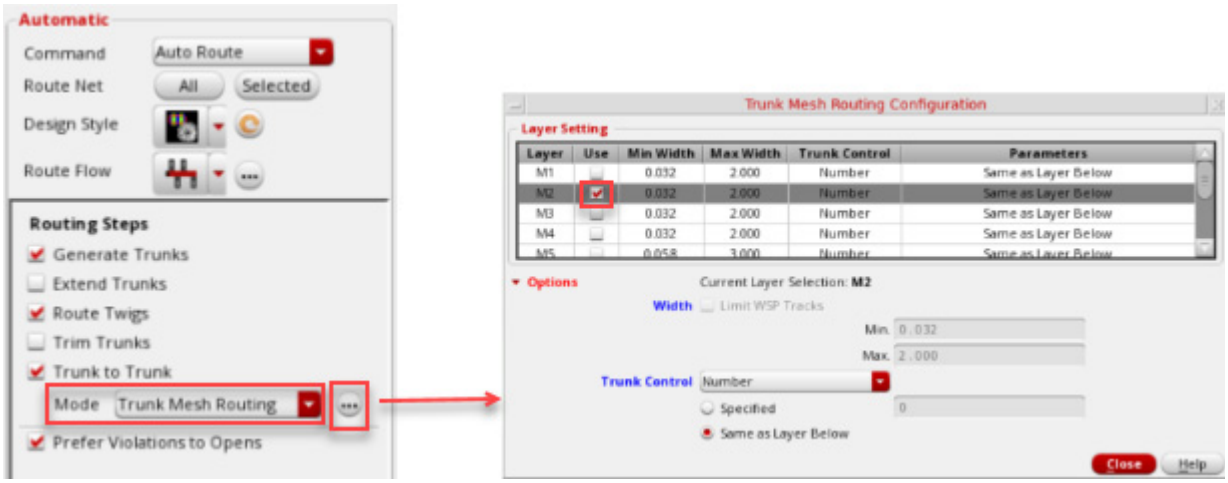
Observe that only one vertical trunk was created to connect all of the horizontal trunks together. However, this is not enough to handle electromigration rules for a net with high current.



To connect the trunks using multiple trunks, you can use Trunk Mesh routing.

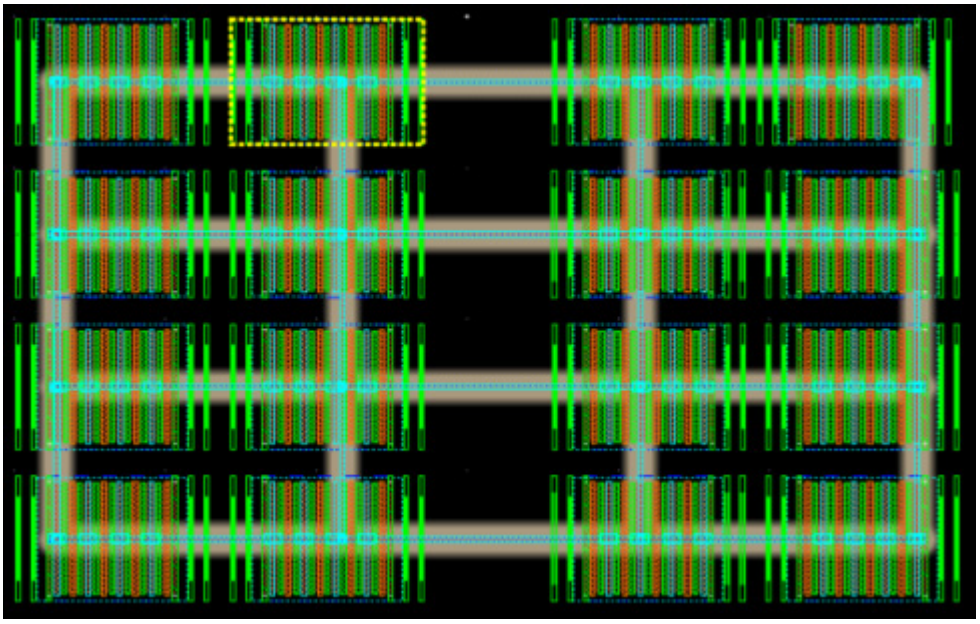
9. Delete the previous routing on the selected net by clicking the *Undo* button on the *Edit* toolbar.
10. In the *Trunk to Trunk* routing step, select the routing mode as *Trunk Mesh Routing* from the *Mode* drop-down list.
11. Click the ellipses button next to the *Mode* field.

The Trunk Mesh Routing Configuration form displays.



12. Select the check box next to `M2` metal layer in the *Use* column and click *Close*.
13. Click *Selected* in *Route Net*.

    The following figure displays the routed result.



Video

For a short demonstration on how to use the new trunk to trunk mesh routing feature, see Trunk To Trunk Mesh Routing using Pin To Trunk (P2T) Route Flow.

Related Topics

Trunk Mesh Routing (ICADVNM18.1 Only)

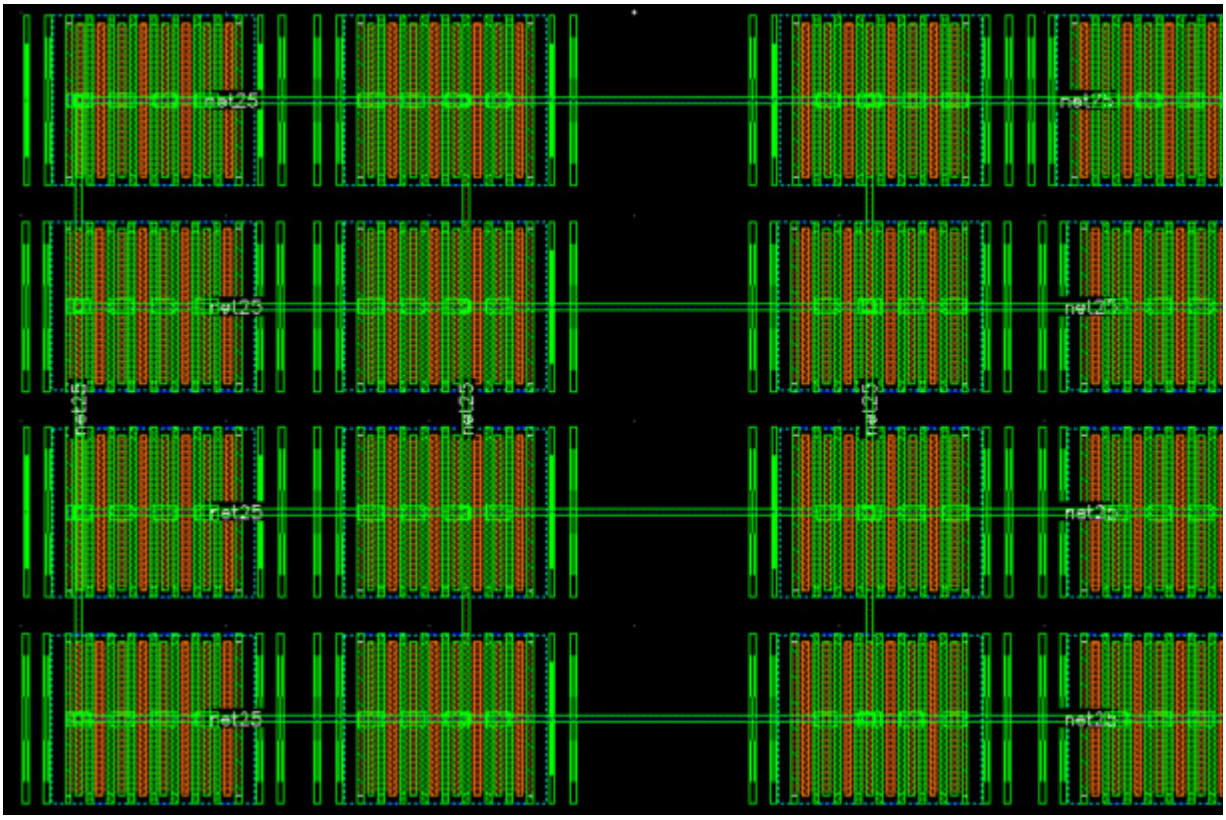
Trunk Mesh Routing Configuration Form (ICADVM18.1 EXL Only)

Using Tree Routing

1. Open a design in the layout window.
2. In the Navigator Assistant, select *net25*. The instance pins on the selected net are highlighted.
3. In the Wire Assistant, select the design style as *Automatic* from the *Design Style* drop-down list.
4. Select *Tree Route* as the routing flow from the *Route Flow* drop-down list. The *Routing Steps* for Tree Route flow displays, as shown in the following figure.
5. Ensure that *Generate Trunks* is selected in the *Routing Steps* group box.
6. Select *No Mesh* from the *Generate Trunks* drop-down list.
7. Click *Selected* in the *Route Net*.

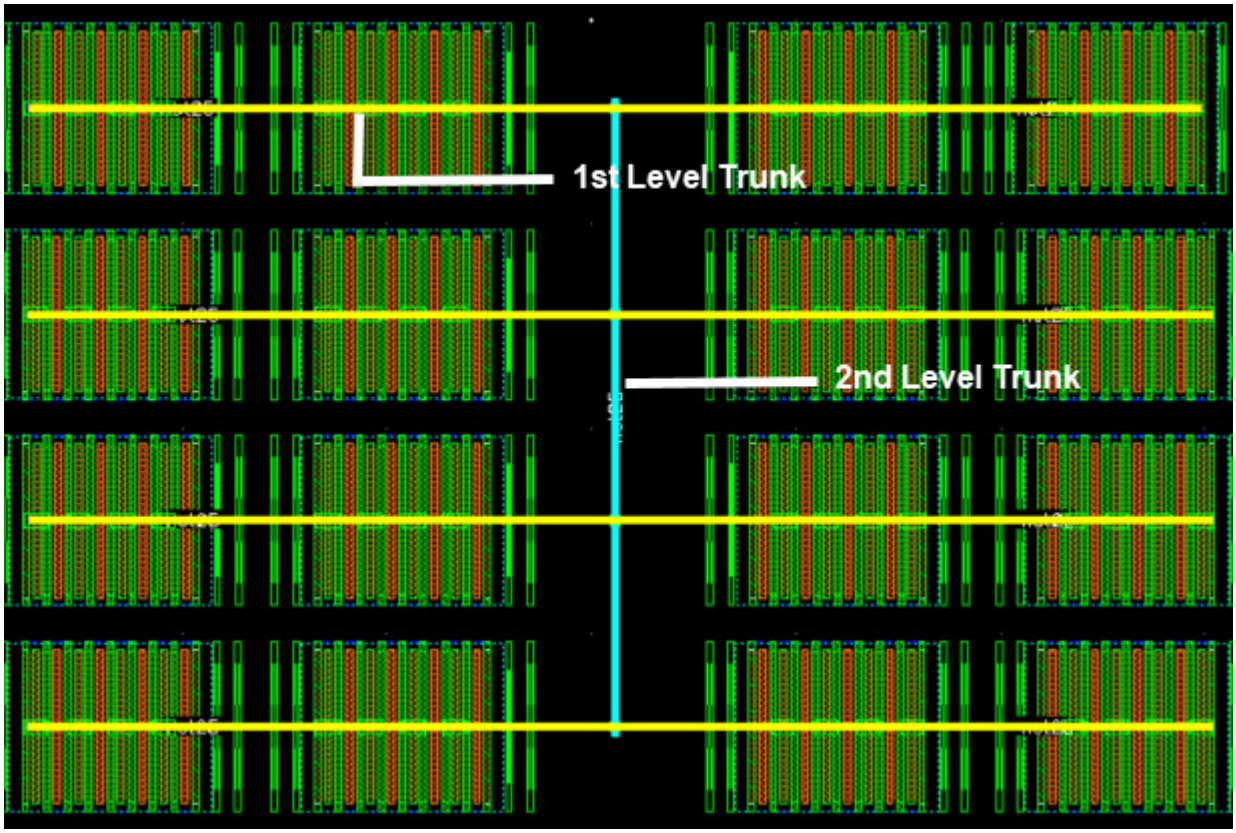
   The following figure shows the routing results for the selected net when *No Mesh* is selected.

Similarly, you can select the Auto Mesh option to automatically generate trunks for mesh routing.

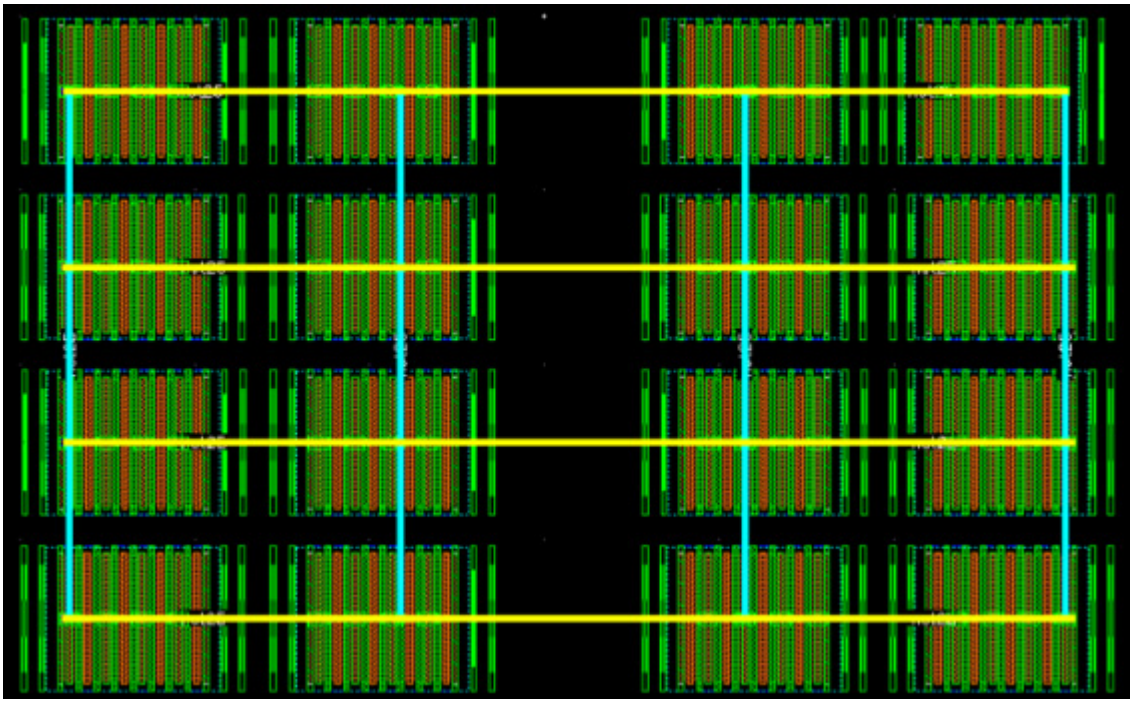


8. In the Navigator Assistant, right-click the selected net.

9. In the context-sensitive menu, click *Highlight Trunks*. The routed trunks are highlighted in different colors. The level-1 trunks are highlighted in Yellow and the level-2 trunk is highlighted in Cyan.

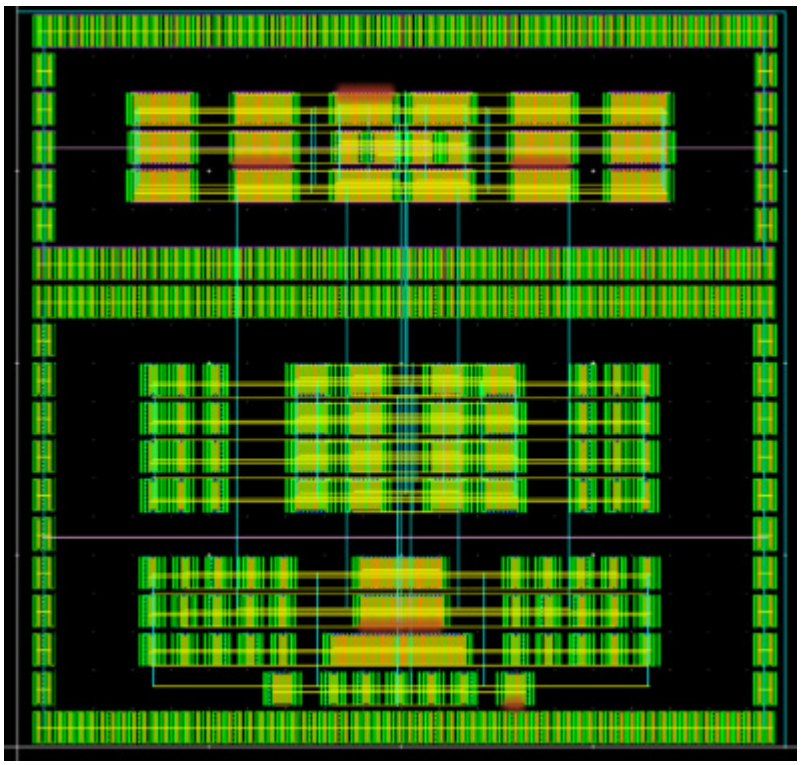


The following figure shows the highlighted trunks for auto mesh.



10. Now, delete the routing on the selected net by clicking the *Undo* button on the *Edit* toolbar.
11. In the Wire Assistant, select *Auto Mesh* from the *Generate Trunks* drop-down list.
12. Click *All* in the *Route Net*.

13. Highlight the trunks and observe the routed results.



Related Topics

Using the Tree Route Flow (ICADVM20.1 EXL Only)

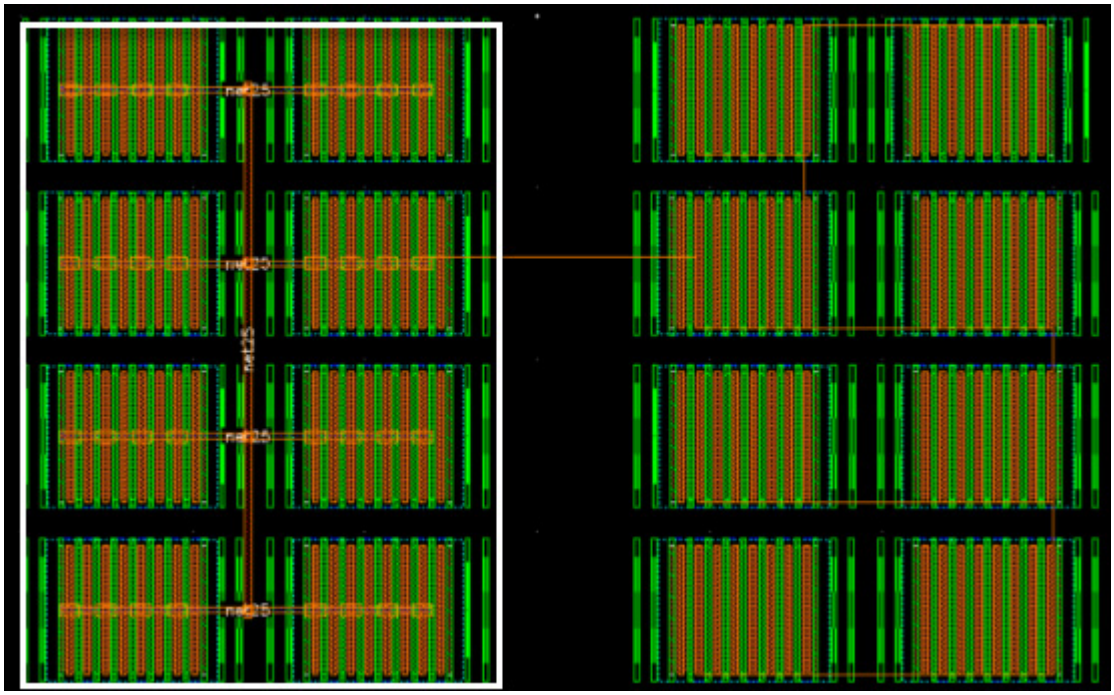
Tree Route (ICADVM 18.1 Only)

Performing Area-based Tree Routing

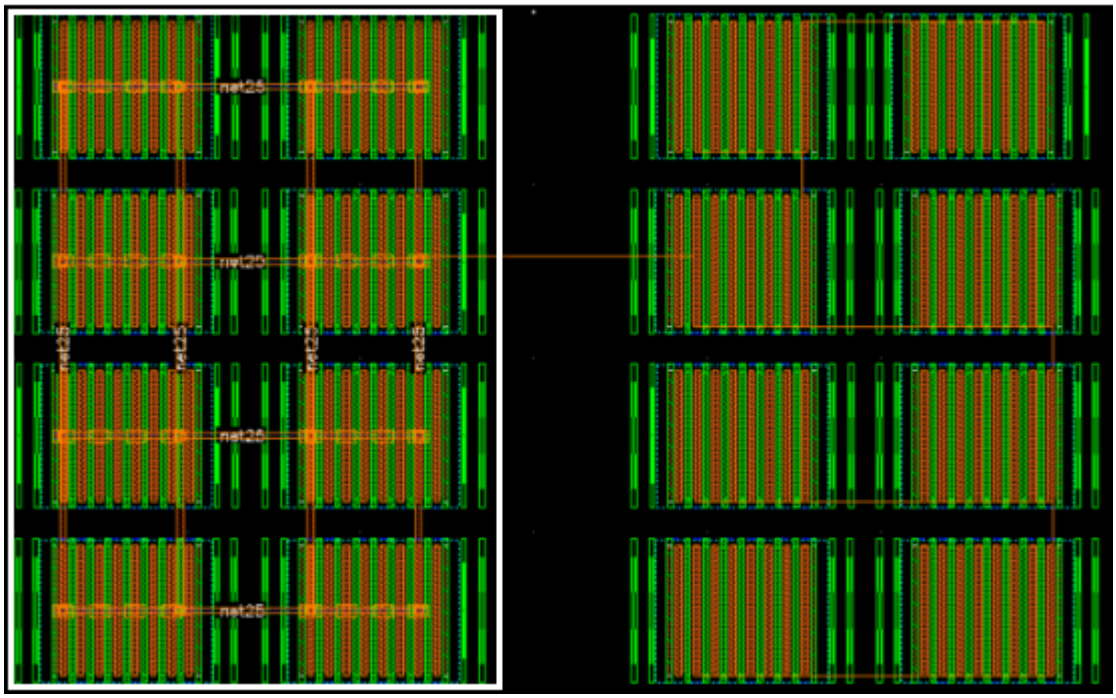
For area-based routing, use the *Select Area and Route* command to perform Tree routing in a specified area.

1. Open a design in the layout window.
2. In the Navigator Assistant, click *net25* to select the net. The instance pins on the selected net are highlighted.
3. In the Wire Assistant, select the design style as *Automatic* from the *Design Style* drop-down list.
4. Select *Tree Route* as the routing flow from the *Route Flow* drop-down list.
5. Ensure that *Generate Trunks* is selected in the *Routing Steps* group box.
6. Select *No Mesh* from the *Generate Trunks* drop-down list.
7. In the Navigator Assistant, right-click the selected net.
8. From the context-sensitive menu, click the *Select Area and Route with Wire Assistant Overrrides* option.
9. With `net25` still selected in the Navigator assistant, use the mouse to select two points to digitize a box around the area of the `net25` pins.

The following figure shows the result for the routed area. Only the area that was selected is routed.



The following figure shows the result for the route area when *Auto Mes*h is selected.



A

Forms Reference

This chapter describes the forms in the Virtuoso Space-based Router.

- Power Routing Form
  - Power Routing - Pad Ring tab
  - Power Routing - Core Ring tab
  - Power Routing - Block Ring tab
  - Power Routing - Stripes tab
  - Power Routing - Cell Rows tab
  - Power Routing - Pin To Trunk tab
  - Power Routing - Vias tab
  - Power Routing - Tie Shield tab
- Power Routing Options Form
  - Power Routing Options - Pad Ring tab
  - Power Routing Options - Core Ring tab
  - Power Routing Options - Block Ring tab
  - Power Routing Options - Stripes tab
  - Power Routing Options - Cell Rows tab
  - Power Routing Options - Pin To Trunk tab
  - Power Routing Options - Vias tab
  - Power Routing Options - Setup tab
- Power Routing Scheme Manager
  - Power Routing Scheme Manager - File tab

- Power Routing Scheme Manager - Compare tab
- Power Routing Schemes Compare Result

■ VSR Preset Forms

- VSR Save Preset Form
- VSR Delete Preset Form

Power Routing Form

The Power Routing form has tabs enabling you to set up options for each power routing operation.

Name - the scheme name for the power routing operation.

Edit - enables you to view or change the scheme definition.

Manager- displays the Power Routing Scheme Manager form.

Route - executes power routing with the options in the scheme.

Power Routing - Pad Ring tab

Pad Ring routing adds a ring between pad instances on the periphery of the design.

Related Topics

Using the Pad Ring Form

Pad Ring Environment Variables

Power Routing Options - Pad Ring tab

Power Routing - Core Ring tab

Core Ring routing adds a ring around the core of a design that is surrounded by pad instances.

Nets Ordering

Selected Nets - adds the select nets to the Nets Ordering section.

Related Topics

Using the Core Ring Form

Core Ring Environment Variables

Power Routing Options - Core Ring tab

Power Routing - Block Ring tab

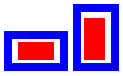
Block Ring routing adds rings around block instances.

Nets Ordering - order the nets for the core rings.

Selected Nets - adds the selected nets to the Nets Ordering section.

Blocks

Style - block ring style.



Adds separate rings around each block.

Adds a shared ring around all blocks.

Related Topics

Using the Block Ring Form

Block Ring Environment Variables

Power Routing Options - Block Ring tab

Power Routing - Stripes tab

Stripes routing adds net stripes at regular intervals.

Nets Ordering - order the nets for the stripes.

Selected Nets - adds the selected nets to the Nets Ordering section.

Region - the region in which to add the stripes.

Entire Cell View - uses the current place and route boundary (prBoundary) for the design. If no routing regions are given, and core rings exist, the core rings become the bounds for the stripes. If no routing regions are given, and core rings do not exist, stripes will cover the entire design. More than one routing region can be given.

View Area Only - uses the current zoom area for routing. If the zoom area is outside the prBoundary, a warning is issued.

A Selected Boundary - uses a selected boundary for routing. The boundary can be either a cluster boundary or an area boundary. However, you can select only one boundary at a time for routing. If you select more than one boundary, a warning is issued. If the selected boundary is outside the prBoundary, a warning is issued.

Custom Area - allows you to specify the region by doing one of the following.

Enter the values using the spin boxes for the lower-left (*xLo*, *yLo*) and the upper-right (*xHi*, *yHi*) coordinates for the region.

Click *Create*. Then draw the region in the workspace by dragging a rectangle from a lower-left coordinate to an upper-right coordinate. The coordinate positions are automatically entered in the form.

Related Topics

Using the Stripes Form

Stripes Environment Variables

Power Routing Options - Stripes tab

Power Routing - Cell Rows tab

Cell Row routing adds straps along aligned pins of standard cells.

Region - area in which to add the standard cell straps.

Entire Cell View - use the current place and route boundary (prBoundary) for the design.

If no routing regions are given, and core rings exist, the core rings become the bounds for the cell rows. If no routing regions are given, and core rings do not exist, cell rows will cover the entire design. More than one routing region can be given.

View Area Only - use the current zoom area for routing. If the zoom area is outside the prBoundary, a warning is issued.

Selected Boundary - use a selected boundary for routing. The boundary can be either a cluster boundary or an area boundary. However, you can select only one boundary at a time for routing. If you select more than one boundary, a warning is issued.

If the selected boundary is outside the prBoundary, a warning is issued.

Custom Area - allows you to specify the region by doing one of the following.

Enter the values using the spin boxes for the lower-left (*xLo*, *yLo*) and the upper-right (*xHi*, *yHi*) coordinates for the region.

Create - enables you to draw the region in the workspace by dragging a rectangle from a lower-left coordinate to an upper-right coordinate. The coordinate positions are automatically entered in the form.

Related Topics

Using the Cell Rows Form

Cell Row Environment Variables

Power Routing Options - Cell Rows tab

Power Routing - Pin To Trunk tab

Pin-To-Trunk routing allows you to add connections from the power pins of pad or macro instances to existing rings and rails. A macro instance is defined as any instance whose area is greater then 90 percent of the entire cell view area.

Related Topics

Using the Pin To Trunk Form

Pin To Trunk Environment Variables

Power Routing Options - Pin To Trunk tab

Power Routing - Vias tab

Vias routing adds vias for interlayer connection at intersections of rings, rails, stripes, straps, and cell rows, or between instance pins and stripes.

Via Locations - inter-layer connections can take place at one of the following.

Intersection of All Rings, Stripes, and Cell Rows - adds vias at the intersections of all rings, stripes and cell rows.

Selected Instances - adds vias between stripes and the pins of selected block instances.

Pin Layer - determines the layer to which the via should be inserted.

Related Topics

Using the Vias Form

Vias Environment Variables

Power Routing Options - Vias tab

Power Routing - Tie Shield tab

Ties shield wires to shield nets in the design.

Tie Frequency- distance between the ties.

Shield - the distance between the shield ties.

Coax - the distance between the coax shield ties.

Note: The number to the right of the spin boxes is the largest *minSpacing* value in the design. If the color of the number is red, the current setting in the spin box is smaller than the largest *minSpacing* constraint value. You are then likely to violate the *minSpacing* on layers with larger minimums. Best practice is to set a value greater than or equal to this number, in which case the number is black.

Route - executes the tie shield operation.

Related Topics

Using the Tie Shield Form

Tie Shield Environment Variables

Power Routing Options Form

The Power Routing Options form has tabs enabling you to set up options for changing the scheme definition of each power routing operation.

Name - the scheme name you want to define.

Manager- displays the Power Routing Scheme Manager form.

Rename - renames the scheme to the name entered in Name field.

Copy - copies the scheme to a new name. The default name is the original scheme name with _n, where n is a number, automatically appended.

Delete - deletes the current scheme. You cannot delete the `Default` scheme.

Power Routing Options - Pad Ring tab

The Pad Ring tab of the Power Routing Options form lets you specify pin layers and pin types to which to connect pad rings for the scheme.

Specified Pin Layers - the pin layers you want to connect.

Deselecting the check box routes pad rings on all layers in the list.
Environment variable: routePadRingAllLayers

Selecting the check box routes pad rings on the layers you highlight in the list box.
Environment variable: routePadRingLayers

Pin Types - pin types to which you want the pad rings to connect.

Rail Pins - pins that traverse a pad.
Environment variable: routePadRingRailPins

Edge Pins - pins on the edge of the pads.
Environment variable: routePadRingEdgePins

Related Topics

Using the Pad Ring Form

Pad Ring Environment Variables

Power Routing - Pad Ring tab

Power Routing Options - Core Ring tab

The Core Ring section of the *Power Routing Options* form lets you specify the core ring location and configuration for the scheme.

Ring Location

Centered Between Core and Pads - creates a ring between the core area of the design and the surrounding pad instances.
Environment variable: prouteCoreRingRingAtCenter

Relative To - route the core ring relative to one of the following.
Environment variable: prouteCoreRingRelativeTo

Core (outside) places the core ring relative to the outside the core of the design.
Environment variable: prouteCoreRingCoreClearance

I/O Pads (inside) places the core ring relative to the inside of the pads.
Environment variable: prouteCoreRingPadClearance

Specified Region (inside) places the core ring inside the region.
Environment variable: prouteCoreRingInAreaClearance

Specified Region (outside) places the core ring outside the region.
Environment variable: prouteCoreRingOutAreaClearance

Coordinates - allows you to specify the region by either using the spin boxes for the lower-left (*xLo*, *yLo*) and the upper-right (*xHi*, *yHi*) coordinates for the region or dragging a rectangle around the region.
Environment variables: prouteCoreRingRTXLo, prouteCoreRingRTYLo, prouteCoreRingRTXHi, prouteCoreRingRTYHi

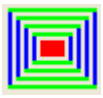
Clearance - the clearance between the core ring and the boundary.

Routing Configuration

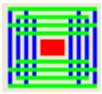
Horizontal Routing Layer - the horizontal routing layer for the core ring.
Environment variable: prouteCoreRingHorizLayer

Vertical Routing Layer - the vertical routing layer for the core ring.
Environment variable: prouteCoreRingVertLayer

Routing Style for multiple core rings.
Environment variable: prouteBlockRingLattice



Concentric core rings.



Latticed core rings. Duplicate wire segments extend to form a lattice.

Ring Clearance - the minimum spacing between the power/ground nets in the core ring.
Environment variable: prouteBlockRingNetClearance

Ring Width - the total width for each power/ground net.
Environment variable: prouteBlockRingNetWidth

Related Topics

Using the Core Ring Form

Core Ring Environment Variables

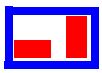
Power Routing - Core Ring tab

Power Routing Options - Block Ring tab

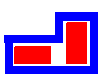
The Block Ring section of the Power Routing Options form lets you specify the ring and routing configuration for block rings.

Ring Configuration

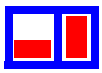
Routing Style for Single Ring Around Multiple Blocks
Environment variables: prouteBlockRingContour, prouteBlockRingChannels



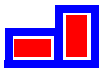
Adds block rings in a rectangular shape around the group of blocks.



Adds block rings contoured around the group of blocks.



Adds block rings in a rectangular shape around the group of blocks with rails in the channels between blocks.



Adds block rings contoured around the group of blocks with rails in the channels between blocks.

Block Clearance - clearance of the block rings to the prBoundary of the block.
Environment variable: prouteBlockRingBlockClearance

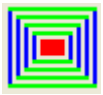
Routing Configuration

Horizontal Routing Layer - for the block ring.
Environment variable: prouteBlockRingHorizLayer

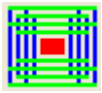
Vertical Routing Layer - for the block ring.
Environment variable: prouteBlockRingVertLayer

Routing Style - the style for multiple block rings.
Environment variable: prouteBlockRingLattice

The options are:



Concentric core rings.



Latticed core rings. Duplicate wire segments extend to form a lattice.

Ring Clearance - the minimum spacing between the power/ground nets in the ring. If the specified Ring Clearance value is greater than the indicated label, the label color is black; otherwise, the color is red.
Environment variable: prouteBlockRingNetClearance

Ring Width - the total width for each power/ground net. If the specified Ring Clearance value is greater than the indicated label, the label color is black; otherwise, the color is red.
Environment variable: prouteBlockRingNetWidth

Related Topics

Using the Block Ring Form

Block Ring Environment Variables

Power Routing - Block Ring tab

Power Routing Options - Stripes tab

The Stripes section of the Power Routing Options form lets you specify the placement and configuration of the horizontal and vertical stripes.

Horizontal Stripes - choose to create horizontal stripes.
Environment variable: prouteStripesHorizDir

> **Step Increment** - value to separate the horizontal stripes.
> Environment variable: prouteStripesYStep
>
> **Routing Layers** - horizontal stripes are routed on the highlighted layers.
> Environment variable: prouteStripesHorizLayers

Vertical Stripes - choose to create vertical stripes.
Environment variable: prouteStripesVertDir

> **Step Increment** - value to separate the vertical stripes.
> Environment variable: prouteStripesXStep
>
> **Routing Layers** - vertical stripes are routed on the highlighted layers.
> Environment variable: prouteStripesVertLayers

Stripe Configuration

> **Pin Clearance** - the clearance required between signal pins and stripes.
> Environment variable: prouteStripesPinClearance
>
> **Net Clearance -** the minimum spacing required between the power/ground nets within a stripe.
> Environment variable: prouteStripesNetClearance
>
> **Net Width** - the total width for each routed net.
> Environment variable: prouteStripesNetWidth
>
> **Minimum Stripe Length** - minimum length for the stripe.
> Environment variable: prouteStripesMinLength

Relative Start Location

Offset from either the **Design Boundary**, the design **Origin**, or the **Routing Area** (the *Routing Regions* given in the Power Router Stripes form).
Environment variable: prouteStripesOffsetFrom

Left - the X location of the first vertical stripe.
Environment variable: prouteStripesLeftOffset

Bottom - the Y location of the first horizontal stripe.
Environment variable: prouteStripesBottomOffset

Relative To - stripes are measured relative to the **Edge of Stripe** (left edge for vertical stripes and bottom edge for horizontal stripes), or the **Center Line of Stripe**.
Environment variable: prouteStripesCenterLine

Related Topics

Using the Stripes Form

Stripes Environment Variables

Power Routing - Stripes tab

Power Routing Options - Cell Rows tab

The Cell Rows section of the Power Routing Options form lets you specify routing layers, the cell row strap boundary, and whether the cell row straps should be extended to the nearest power rail.

Specified Routing Layers - the routing layers you want to connect.

Deselecting the check box routes pad rings on all layers in the list.

Selecting the check box routes pad rings on the layers you highlight in the list box.
Environment variable: prouteCellRowLayers

Boundary Rule

End of Row - routes cell row straps to the end of the defined rows.
Environment variable: prouteCellRowRowEnd

Last Cell in Row - routes cell row straps to the last cell in the row.

Extend Straps to Rails - extends the cell row straps to the nearest power rail.
Environment variable: prouteCellRowExtend

Related Topics

Using the Cell Rows Form

Cell Row Environment Variables

Power Routing - Cell Rows tab

Power Routing Options - Pin To Trunk tab

The Pin To Trunk section of the Power Routing Options form lets you specify pin layers, trunk layers and connection rules.

Specified Pin Layers - the pin layers you want to connect.

Deselecting the check box routes pad rings on all layers in the list.
Environment variable: routePinToTrunkAllLayers

Selecting the check box routes pad rings on the layers you highlight in the list box.
Environment variable: routePinToTrunkLayers

Specified Trunk Layer

Deselecting the check box connects all trunk layers.
Environment variable: routePinToTrunkSpecTLayer

Selecting the check box restricts pin-to-trunk connections to the trunk layer you choose in the pull down layer list.
Environment variable: routePinToTrunkTrunkLayer

Connection Rules

Minimum Trunk Width - specifies the minimum trunk width for a trunk to be connected to a pin.
Environment variable: routePinToTrunkMinTrunkWidth

Minimum Wire Width - specifies the minimum wire width for a wire to connect from a pin to a trunk.
Environment variable: routePinToTrunkMinWireWidth

Note: If you *Specified* a *Trunk Layer*, the numbers to the right of the *Minimum Trunk Width* and *Minimum Wire Width* spin boxes are the value of the *minWidth* constraint of the selected layer.

If you did not specify a trunk layer, the numbers to the right of the spin boxes are the largest *minWidth* of all the Specified Pin Layers. If you then set a *Minimum Wire Width*

value that is smaller than the number to the right of the spin box, you are likely to violate the *minWidth* on layers with smaller minimums. Best practice is to set a value greater than or equal to this number, in which case the number is black.

Maximum Wire Width - specifies the maximum wire width for a wire to connect from a pin to a trunk.
Environment variable: routePinToTrunkMaxWireWidth

Pin-to-Trunk uses the largest wire width that does not exceed the Maximum Wire Width or the width of the pin.

Note: If you *Specified* a *Trunk Layer*, the number to the right of the *Maximum Trunk Width* spin box is the value of the *maxWidth* constraint of the selected layer.

If you did not specify a trunk layer, the number to the right of the spin box is the smallest *maxWidth* of all the Specified Pin Layers. If you then set a *Maximum Wire Width* that is larger than this number, you are likely to violate the *maxWidth* on layers with smaller maximums. Best practice is to set a value less than or equal to this number, in which case the number is black.

Related Topics

Using the Pin To Trunk Form

Pin To Trunk Environment Variables

Power Routing - Pin To Trunk tab

Power Routing Options - Vias tab

The Vias section of the Power Routing Options form lets you specify layers and cut array types.

Layers - that you want to connect.

Layer Range - deselecting the check box inserts vias to connect all layers.

Selecting the check box inserts vias to connect a range of metal layers.
Environment variable: prouteViaInsertionLayerRange

Minimum and Maximum - pull down menus enable you to choose the bottom and top layers, respectively, to connect.
Environment variables: prouteViaInsertionMinLayer, prouteViaInsertionMaxLayer

Cut Array - the type of cut array.

Square - specifies a square cut array for each via, with an equal number of rows and the number of columns. However, the cut array might not be square in size, depending on spacing rules. If the `minAdjacentViaSpacing` rule is set, then the spacing of the cuts can be different in the X and Y directions, producing a cut array that is rectangular in dimension.

Rectangular - specifies that the maximum number of rows and columns for the cuts be used for the available space.

Specified Array Size - inserts vias with cut arrays of a specific dimension. If the cut array with these dimensions cannot fit within a via area, no via is created for that metal intersection.

Environment variable: prouteViaInsertionCutArray

Rows and Columns - for the specified array size.
Environment variables: prouteViaInsertionRows, prouteViaInsertionColumns

Related Topics

Using the Vias Form

Vias Environment Variables

Power Routing - Vias tab

Power Routing Options - Setup tab

Determines the packages available in the scheme.

Scheme Packages - the packages included in the named scheme.

Pad Ring

Core Ring

Block Ring

Stripes

Cell Rows

Pin To Trunk

Vias

Related Topics

Setting the Packages for the Scheme

Power Routing Scheme Manager

The Power Routing Scheme Manager enables you to load, save, and compare schemes.

Power Routing Scheme Manager - File tab

Name - the name of the file containing the schemes in the Name field.

Browse - alternatively, you can use this button to select the file containing the schemes.

Load - loads all the schemes in the file into the power routing scheme database

Save - saves the selected schemes to the given file name. If you save schemes to an existing file, the original contents of the file are removed and replaced by the newly saved schemes.

Power Routing Scheme Manager - Compare tab

Schemes - the schemes you select to compare from the drop down boxes. Click to compare specific packages within the schemes.

Compare - displays the Power Routing Compare Result form.

Power Routing Schemes Compare Result

The first column lists the package options that have different values in the two packages. The Total number of Differences for the two packages also displays.

The second column lists the option data for the indicated scheme.

The third column lists the option data for the indicated scheme, so you can compare the two schemes side by side.

Related Topics

Managing Schemes

VSR Preset Forms

This section describes the field descriptions of the following VSR Preset forms:

- VSR Save Preset Form
- VSR Delete Preset Form

VSR Save Preset Form

- **Label in Toolbar**

  Lets you specify a label for the preset file. You can also select a preset file from the drop-down list. When a preset file is selected, all other fields of the VSR Save Preset form are automatically populated with the keywords information available in the selected preset file. This is a mandatory field.

- **File Name**

  Lets you specify a name for the preset file to save all override constraints and environment variable values to a single file. This is also a mandatory field.

- **Directory**

  Lets you select the location where you want to save the preset file. You can select one of the following locations:

  - Current Virtuoso Invoking directory (`./.cadence/dfII/ia/presets`)
  - $HOME (`~/.cadence/dfII/ia/presets`)
  - $CDS_PROJECT (`$CDS_PROJECT/dfII/ia/presets`)

- **Create Toolbar Icon**

  Adds an icon on the Virtuoso Space-based Router Preset toolbar when the C*reate Toolbar Icon* is selected. When deselected, the preset is added as a menu item in the *VSR Load Preset* drop-down menu.

- **Optional Settings**

  Hides or displays all the optional settings in the VSR Save Preset form. These optional settings are not mandatory and can be left blank.

- **Icon File Name**

Uses the icon from the specified file name and associates it with the preset file. The associated icon is then displayed on the VSR Preset toolbar. This option is relevant only when the *Create Toolbar Icon* check box is selected.

- **Icon Text**

  Lets you specify the text that gets displayed on the toolbar icon. If icon text is not provided, the first five characters of the *Label in Toolbar* field are used as the text that gets displayed on the created icon.

  Note: The *Icon Text* field has significance only when the *Create Toolbar Icon* check box is selected and the *Icon File Name* is left blank. However, if *Create Toolbar Icon* is deselected, the text specified in the *Icon Text* field is listed as a menu option in the *VSR Load Preset* drop-down list.

- **Tooltip**

  Lets you specify the text that appears as a tooltip when the mouse pointer is placed on the preset icon on the toolbar.

- **Save**

  Saves the current settings to a preset file for future use.

- **Clear**

  Clears all the fields in the VSR Save Preset form.

Related Topics

The *Routing Scripts command also keep up with the values specified in the Override Constraints section of the Wire Assistant. For example, in the Wire Assistant, if you have specified the Bottom layer as* `M1`, *Top layer as* `M2`, *and have specified the Min Num Cuts value to* `2`, *and then run the customized Tcl script using Route – Routing Scripts, the routing layers and* `minNumCuts` *values are overriden.*

VSR Workspace

Preset File

VSR Delete Preset Form

- **Preset Label**

Lists all the presets that are currently loaded. The presets listed in the *Preset Label* drop-down list correspond to the values that you specify for the preset files in the *Label in Toolbar* field in the VSR Save Preset form.

- **File**

  Displays the filename of the selected preset.

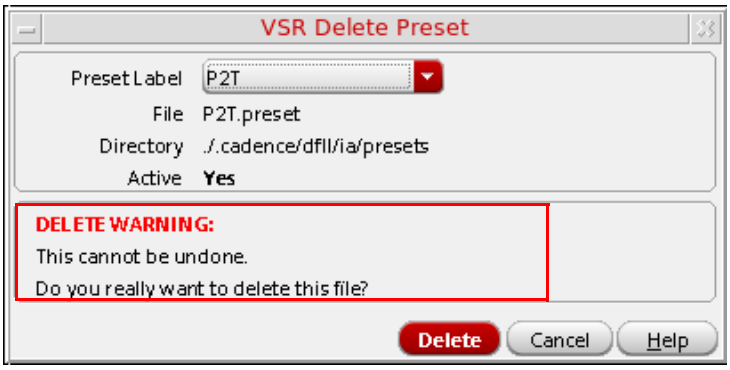
- **Directory**

  Displays the path to the location where the selected preset is saved.

- **Active**

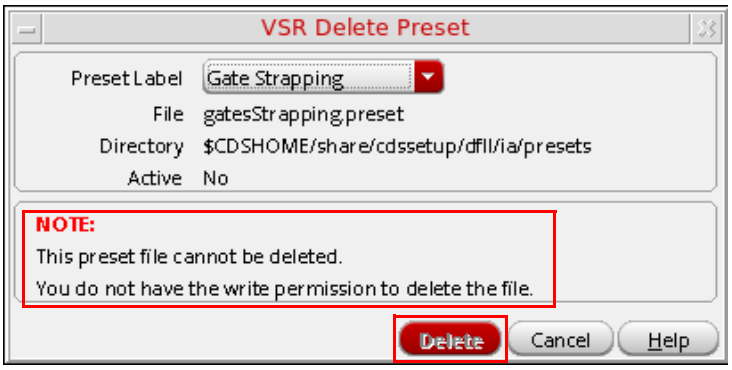
  Displays `Yes` if the selected preset file is the one that was loaded last; otherwise, displays `No`.

  Note: You can only delete a preset file for which the value in the *Active* field is `Yes`. A warning message is displayed stating that a deleted preset file cannot be restored in the current Virtuoso session.



However, if you do not have the appropriate permissions to delete the selected preset file, the value in the *Active* field is shown as `No`. Also, a message is displayed stating that

the preset file cannot be deleted and the *Delete* button is disabled, as shown in the following figure.



Related Topics

Deleting a Preset File

VSR Workspace

Preset File

B

Environment Variables

The system looks for the system-defined default settings in the following file:

install_dir/etc/tools/ia/.cdsenv
install_dir/etc/tools/layout/.cdsenv
install_dir/etc/tools/rte/.cdsenv
install_dir/etc/tools/p2t/.cdsenv

The following sections describe the environment variables that are used in the routing flow and to verify the routing connections.

- Automatic Router Environment Variables
- Power Routing Environment Variables
- Pin to Trunk Environment Variables

Automatic Router Environment Variables

The following sections describe the environment variables used in automatic routing and wire assistant.

- ia
  - blockTreatmentTreatAs
  - busBitTableRows
  - forceGlobalPlanning
  - globalAllNets
  - lockPreRoutes
  - presetDefaultFile
  - presetLoadMode
  - presetResetMode
  - vsrFlow
  - widthSpaceTableRows
- layout
  - adjustViatoWireEdge
  - checkRoutabilityCreateExcludeSet
  - checkRoutabilityMarkersLimit
  - checkRoutabilityMissingCoverObstruction
  - checkRoutabilityMissingPRBoundary
  - checkRoutabilityNotOnPRBoundaryEdge
  - checkRoutabilityPinOnGrid
  - checkRoutabilityPinCenterOnGrid
  - enableMaximizeCuts
  - enforceGapSpacingOnDiffPairVias
  - gapFillPurpose
  - removePrerouteDangles

- routeNetAtPinWidth
- routeNetAtPinWidthStyle
- routeSelectedNets
- routeWithLowEffort
- setTaperPinWidthNets
- trimBridgePurpose
- useDoubleCutVias
- useDoubleCutVias
- useWAOverrides
- viaMinNumCuts
- viaWECutColumns
- viaWECutRows
- viaWECutSpecified
- weBusBitShieldingGap
- weBusBitShieldingNetName
- weBusBitShieldingOption
- weBusBitShieldingWidth

■ rte

- checkRoutabilityBlockage
- checkRoutabilityMinWidth
- checkRoutabilityVia
- checkRoutabilityVia
- coverObsMasterTooltipLimit
- coverObstUnhiliteOnFormClosed
- deleteRoutingKeepFigGroupShapes
- deleteRoutingKeepPower
- extractedPinStyle

- fixErrorsFixMinAreaAtPins
- fixErrorsErrorTypesMinArea
- fixErrorsErrorTypesMinEnclArea
- fixErrorsErrorTypesMinEdge
- fixErrorsErrorTypesMinWidth
- fixErrorsErrorTypesRGrid
- fixErrorsErrorTypesNumCut
- fixErrorsErrorTypesSameMaskSpacing
- genRoutingStyle
- optimizeRouteReduceVias
- routeSearchAndRepair
- routingScriptingDir

ia

blockTreatmentTreatAs

```
ia blockTreatmentTreatAs cyclic { "Automatic" | "Real Metal" | "Minimum Width
     Shape" }
```

Description

Ensures that all blockages have proper spacing properties and that the blockages model the metal on the edge of the block properly.

- `Automatic` determines the blockage mode to be used according to the design style.
- `Real Metal` treats blockages as metal and the spacing rules are applied based on blockage dimensions.
- `Minimum Width Shape` treats blockages as metal with minimum width, regardless of the actual width of the blockage.

Default is `Automatic`.

GUI Equivalent

Command	*Route – Design Setup – Blockage Treatment*
Field	*Default Blockage Model*

Examples

```
envGetVal("ia" "blockTreatmentTreatAs")
envSetVal("ia" "blockTreatmentTreatAs" 'cyclic "Automatic")
envSetVal("ia" "blockTreatmentTreatAs" 'cyclic "Real Metal")
envSetVal("ia" "blockTreatmentTreatAs" 'cyclic "Minimum Width Shape")
```

Related Topics

Automatic Router Environment Variables

Blockage Treatment

busBitTableRows

```
ia busBitTableRows int integer_number
```

Description

The value specified for the environment variable is used to control the number of rows that are displayed in the bus bit table. By default, seven rows are displayed in the bus bit table.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Bus*
Field	*Bus Bit table*

Examples

```
envGetVal("ia" "busBitTableRows")
envSetVal("ia" "busBitTableRows" 'int 10)
```

Related Topics

Automatic Router Environment Variables

Bus

forceGlobalPlanning

```
ia forceGlobalPlanning boolean { t | nil }
```

Description

When set to `t`, allows forcing of global routing while in *Device* design style. Default is `nil`.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Automatic – Auto Route*
	Route – Design Setup – Auto Route
Form Field	*Force Global Planning*

Examples

```
envGetVal("ia" "forceGlobalPlanning")
envSetVal("ia" "forceGlobalPlanning" 'boolean t)
envSetVal("ia" "forceGlobalPlanning" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Optimization

Automatic Section

globalAllNets

```
ia globalAllNets boolean { t | nil }
```

Description

When set to `t`, forces the global router to use all nets even when routing a selected set of nets. Default is `nil`.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Automatic – Auto Route*
	Route – Design Setup – Auto Route
Form Field	*Global Route All Nets*

Examples

```
envGetVal("ia" "globalAllNets")
envSetVal("ia" "globalAllNets" 'boolean t)
envSetVal("ia" "globalAllNets" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Optimization

Automatic Section

globalIncludePreroutes

```
ia globalIncludePreroutes boolean { t | nil }
```

Description

Enables ECO flow routing with using existing pre-routes. This setting lets the router to use the existing floating pre-routes on a net to route the opens on the net.

Default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("ia" "globalAllNets")
envSetVal("ia" "globalAllNets" 'boolean t)
envSetVal("ia" "globalAllNets" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Optimization

Automatic Section

lockPreRoutes

```
ia lockPreRoutes boolean { t | nil }
```

Description

Allows for not locking the pre-routes. This is useful for ECO routes. Default is `t`.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Automatic – Auto Route*
	Route – Design Setup – Auto Route
Form Field	*Preserve Pre-routes*

Examples

```
envGetVal("ia" "lockPreRoutes")
envSetVal("ia" "lockPreRoutes" 'boolean t)
envSetVal("ia" "lockPreRoutes" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Optimization

Automatic Section

presetDefaultFile

```
ia presetDefaultFile string "file_names"
```

Description

Specifies the name of the preset file that is automatically loaded when Virtuoso starts. You need to specify the location of the preset file. The location of the preset file is one of following directories:

- `./.cadence/dfII/ia/presets`
- `<$HOME>/.cadence/dfII/ia/presets`
- `<$CDS_PROJECT>/dfII/ia/presets`

Example:

```
ia  presetDefaultFile string ".cadence/dfII/ia/preset/myDefault.preset"
```

GUI Equivalent

None

Examples

```
envGetVal("ia" "presetDefaultFile")
envSetVal("ia" "presetDefaultFile" 'string "preset")
```

Related Topics

Automatic Router Environment Variables

VSR Load Preset

VSR Workspace

presetLoadMode

```
ia presetLoadMode cyclic { "setting" | "checking" }
```

Description

Controls the loading of the preset mode. This environment variable can take two values, `setting` and `checking`. Default is `checking`.

- `setting` skips any preset entry that has an error and processes and loads only the entries that are error free.
- `checking` processes the entries only if all the entries in the preset file are error free. If even one entry has an error, none of the entries are processed, even if all other entries are error free.

GUI Equivalent

Command	*Window – Toolbar – Virtuoso Space-based Router*
	Window – Assistant – Wire Assistant
Field	*VSR Load Preset icon*

Examples

```
envGetVal("ia" "presetLoadMode")
envSetVal("ia" "presetLoadMode" 'cyclic "setting")
envSetVal("ia" "presetLoadMode" 'cyclic "checking")
```

Related Topics

Automatic Router Environment Variables

VSR Load Preset

VSR Workspace

presetResetMode

```
ia presetResetMode cyclic { "all" | "constraintOverrides" | "options" }
```

Description

Controls the default behavior of reset mode in the Wire Assistant. You can override the setting in the `.cdsenv` file in the home directory.

- `all` resets all the override values in the *Override Constraints* and *Automatic* sections of the Wire Assistant.
- `constraintOverrides` resets the override values in the *Override Constraints* section in the Wire Assistant.
- `options` resets the override values of the VSR options.

Default is `all`.

GUI Equivalent

Command	*Window – Toolbar – Virtuoso Space-based Router*
	Window – Assistant – Wire Assistant
Form Field	*VSR Reset Options* icon

Examples

```
envGetVal("ia" "presetResetMode")
envSetVal("ia" "presetResetMode" 'cyclic "all")
envSetVal("ia" "presetResetMode" 'cyclic "constraintOverrides")
envSetVal("ia" "presetResetMode" 'cyclic "options")
```

Related Topics

Automatic Router Environment Variables

VSR Reset Options

VSR Workspace

vsrFlow

```
ia vsrFlow string "flow_name"
```

Description

Lets you specify the flow command in the `vsrFinal.il` file. During routing, the values returned by `envGetVal("ia" "vsrFlow")` are: *Congestion Map*, *Auto Route*, *Delete Routing*, *Optimize Route*, *Fix Errors*, *Tie Shield*, *Finish Route*, *Check Routability*, or *Common Utils*. The return value can then be used in the SKILL code in the `vsrFinal.il` file to run the code specifically for the flow that is being entered. By default, it is null.

GUI Equivalent

None

Examples

```
envGetVal("ia" "vsrFlow")
envSetVal ("ia" "vsrFlow" 'string "Congestion")
envSetVal("ia" "vsrFlow" 'string "Fix Errors")
envSetVal("ia" "vsrFlow" 'string "Auto Route")
```

Related Topics

Automatic Router Environment Variables

widthSpaceTableRows

```
ia widthSpaceTableRows int integer_number
```

Description

The value specified for the environment variable is used to control the number of rows that are displayed in the width spacing table of the Wire Assistant form. By default, three rows are displayed in the width spacing table. If the value specified in this variable is greater than the total poly and metal layers that are there in the technology file, then the Wire Assistant form uses the number of poly and metal layers in the technology file.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Net*
Field	*Width Spacing table*

Examples

```
envGetVal("ia" "widthSpaceTableRows")
envSetVal("ia" "widthSpaceTableRows" 'int 100)
```

Related Topics

Automatic Router Environment Variables

Net

layout

adjustViatoWireEdge

```
layout adjustViatoWireEdge boolean { t | nil }
```

Description

Directs the router to move vias such that the edge of the metal layers in the vias overlap/ align with the edges of the wires. This can reduce the number of small notches around vias.

GUI Equivalent

Command	*Route – Design Setup – Optimization*
	Window – Assitant – Wire Assistant – Automatic – Optimization
Field	*Align Via to Wire Edge*

Examples

```
envGetVal("layout" "adjustViatoWireEdge")
envSetVal("layout" "adjustViatoWireEdge" 'boolean t)
envSetVal("layout" "adjustViatoWireEdge" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Optimization

checkRoutabilityCreateExcludeSet

```
layout checkRoutabilityCreateExcludeSet boolean { t | nil }
```

Description

Enables you to exclude certain nets from routing. This can help the router skip any net that cannot be routed successfully; thereby, increasing the performance and reducing the routing time.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Routability Check
	Verify – Design
Form Field	*Exclude Types*

Examples

```
envGetVal("layout" "checkRoutabilityCreateExcludeSet")
envSetVal("layout" "checkRoutabilityCreateExcludeSet" 'boolean t)
envSetVal("layout" "checkRoutabilityCreateExcludeSet" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

Routability Checking

checkRoutabilityMarkersLimit

```
layout checkRoutabilityMarkersLimit int integer_number
```

Description

Enables you to specify the maximum number of violations that can be reported in the Annotation Browser. Default is `1000`.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Routability Check
	Verify – Design
Form Field	*Markers Limit per Type*

Examples

```
envGetVal("layout" "checkRoutabilityMarkersLimit")
envSetVal("layout" "checkRoutabilityMarkersLimit" 'int 100)
```

Related Topics

Automatic Router Environment Variables

Routability Check

Routability Checking

checkRoutabilityMissingCoverObstruction

```
layout checkRoutabilityMissingCoverObstruction boolean { t | nil }
```

Description

Enables you to check for instances with missing blockage objects and cover obstructions. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Routability Check
	Verify – Design
Form Field	*Missing Cover Obstruction*

Examples

```
envGetVal("layout" "checkRoutabilityMissingCoverObstruction")
envSetVal("layout" "checkRoutabilityMissingCoverObstruction" 'boolean t)
envSetVal("layout" "checkRoutabilityMissingCoverObstruction" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

Routability Checking

checkRoutabilityMissingPRBoundary

```
layout checkRoutabilityMissingPRBoundary boolean { t | nil }
```

Description

Enables you to check for instances with missing place and route boundary. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Routability Check
	Verify – Design
Form Field	*Missing prBoundary*

Examples

```
envGetVal("layout" "checkRoutabilityMissingPRBoundary")
envSetVal("layout" "checkRoutabilityMissingPRBoundary" 'boolean t)
envSetVal("layout" "checkRoutabilityMissingPRBoundary" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

Routability Checking

checkRoutabilityNotOnPRBoundaryEdge

```
layout checkRoutabilityNotOnPRBoundaryEdge boolean { t | nil }
```

Description

Checks for pins that are not on the edge of the place and route boundary. The default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assistant – Wire Assistant – Automatic – Routability Check
	Verify – Design – Routability
Form Field	*Not on prBoundary Edge/Pin not on prBoundary Edge*

Examples

```
envGetVal("layout" "checkRoutabilityNotOnPRBoundaryEdge")
envSetVal("layout" "checkRoutabilityNotOnPRBoundaryEdge" 'boolean t)
```

Related Topics

Automatic Router Environment Variables

Routability Check

Routability Checking

checkRoutabilityPinOnGrid

```
layout checkRoutabilityPinOnGrid boolean { t | nil }
```

Description

When set to `t`, checks for pins that are off manufacturing grid. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Routability Check
Field	*On Grid – Edge Only*

Examples

```
envGetVal("layout" "checkRoutabilityPinOnGrid")
envSetVal("layout" "checkRoutabilityPinOnGrid" 'boolean t)
envSetVal("layout" "checkRoutabilityPinOnGrid" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

Routability Checking

checkRoutabilityPinCenterOnGrid

```
layout checkRoutabilityPinCenterOnGrid boolean { t | nil }
```

Description

When set to `t`, checks that the center of the pin is off manufacturing grid. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Routability Check
Field	*On Grid – Edge and Center*

Examples

```
envGetVal("layout" "checkRoutabilityPinCenterOnGrid")
envSetVal("layout" "checkRoutabilityPinCenterOnGrid" 'boolean t)
envSetVal("layout" "checkRoutabilityPinCenterOnGrid" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

Routability Checking

enableMaximizeCuts

```
layout enableMaximizeCuts boolean { t | nil }
```

Description

Lets you maximize the number of cuts in the layer overlap area. To avoid maximizing the number of cuts during routing, set this variable to `nil` before you begin routing. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Optimization
	Window – Assitant – Wire Assistant – Net – Via Configuration
Form Field	*Maximize Cuts*

Examples

```
envGetVal("layout" "enableMaximizeCuts")
envSetVal("layout" "enableMaximizeCuts" 'boolean t)
envSetVal("layout" "enableMaximizeCuts" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Via Configuration Form

Optimization

enforceGapSpacingOnDiffPairVias

```
layout enforceGapSpacingOnDiffPairVias boolean { t | nil }
```

Description

Lets you force the application of gap spacing between consecutive vias of the differential pair. Default is `t`.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Net – Via Configuration*
Field	*Enforce Gap Spacing on Differential Pair Vias*

Examples

```
envGetVal("layout" "enforceGapSpacingOnDiffPairVias")
envSetVal("layout" "enforceGapSpacingOnDiffPairVias" 'boolean t)
envSetVal("layout" "enforceGapSpacingOnDiffPairVias" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Via Configuration Form

gapFillPurpose

```
layout gapFillPurpose string "purpose_name"
```

Description

Specifies the purpose of all modified or newly created `SADPFillPatch` shapes when synchronized back to OpenAccess. The default purpose is `drawing`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "gapFillPurpose")
envSetVal("layout" "gapFillPurpose" 'string "drawing")
```

Related Topics

Automatic Router Environment Variables

removePrerouteDangles

```
layout removePrerouteDangles boolean { t | nil }
```

Description

Removes any pre-existing dangles connected to the “from” point and “to” point of the current command. This check box is `OFF` by default.

Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup– Optimization*
	Window – Assistant – Wire Assistant – Automatic – Auto Route
Field	*Remove Pre-Route Dangles*

Examples

```
envGetVal("layout" "removePrerouteDangles")
envSetVal("layout" "removePrerouteDangles" 'boolean t)
envSetVal("layout" "removePrerouteDangles" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Auto Route

Optimization

routeNetAtPinWidth

```
layout routeNetAtPinWidth boolean { t | nil }
```

Description

When set to `TRUE`, the *Net Width to* check box is selected by default.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Net tab*
Field	*Net Width to* check box

Examples

```
envGetVal("layout" "routeNetAtPinWidth")
envSetVal("layout" "routeNetAtPinWidth" 'boolean t)
envSetVal("layout" "routeNetAtPinWidth" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Net

routeNetAtPinWidthStyle

```
layout routeNetAtPinWidthStyle cyclic { "max_pin" | "min_pin" | "avg_pin" }
```

Description

Uses the specified pin width per net to route the non-taper section of the net.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Net tab*
Field	*Net Width to*

Examples

```
envGetVal("layout" "routeNetAtPinWidthStyle")
envSetVal("layout" "routeNetAtPinWidthStyle" 'cyclic "max_pin")
envSetVal("layout" "routeNetAtPinWidthStyle" 'cyclic "min_pin")
envSetVal("layout" "routeNetAtPinWidthStyle" 'cyclic "avg_pin")
```

Related Topics

Automatic Router Environment Variables

Net

routeSelectedNets

```
layout routeSelectedNets boolean { t | nil }
```

Description

Starts automatic routing. Depending on whether the all nets or selected nets are selected, routing is performed on all nets in the current design or only on selected nets. Default is `nil`.

GUI Equivalent

Command	*Window – Assistant – Wire Assistant – Automatic – Auto Route*
Field	*Route Net – Selected*

Examples

```
envGetVal("layout" "routeSelectedNets")
envSetVal("layout" "routeSelectedNets" 'boolean t)
envSetVal("layout" "routeSelectedNets" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Auto Route

routeWithLowEffort

```
layout routeWithLowEffort boolean { t | nil }
```

Description

Enables the detail router to exit early when the average reduction ratios for opens and errors do not converge in consecutive passes. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Auto Route*
	Window – Assitant – Wire Assistant – Automatic – Auto Route
Field	*Run With Low Effort*

Examples

```
envGetVal("layout" "routeWithLowEffort")
envSetVal("layout" "routeWithLowEffort" 'boolean t)
envSetVal("layout" "routeWithLowEffort" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Optimization

setTaperPinWidthNets

```
rte setTaperPinWidthNets boolean { t | nil }
```

Description

Enables the router to create pathSegs connecting to top-level pins or instPins using width matching the target pin. Default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("rte" "setTaperPinWidthNets")
envSetVal("rte" "setTaperPinWidthNets" 'boolean t)
envSetVal("rte" "setTaperPinWidthNets" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

trimBridgePurpose

```
layout trimBridgePurpose string "purpose_name"
```

Description

Specifies the purpose of all modified or newly created `bridgeMetal` shapes when synchronized back to OpenAccess. The default purpose is `drawing`.

GUI Equivalent

None

Examples

```
envGetVal("layout" "trimBridgePurpose")
envSetVal("layout" "trimBridgePurpose" 'string "drawing")
```

Related Topics

Automatic Router Environment Variables

useDoubleCutVias

```
layout useDoubleCutVias boolean { t | nil }
```

Description

Enables Automatic Routing, *Net – Route*, and the assisted routing commands to place double cut vias if the space allows. Default is `nil`.

GUI Equivalent

Command	*Window – Assistant – Wire Assistant – Net – Via Configuration*
	Route – Design Setup – Optimization
Field	*Attempt to Use Double Cut Vias*

Examples

```
envGetVal("layout" "useDoubleCutVias")
envSetVal("layout" "useDoubleCutVias" 'boolean t)
envSetVal("layout" "useDoubleCutVias" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Via Configuration Form

Optimization

useWAOverrides

```
layout useWAOverrides boolean { t | nil }
```

Description

When selected, the overrides made in the Wire Assistant are applicable for automatic routing. Default is `t`.

GUI Equivalent

Command	*Window – Assistant – Wire Assistant*
Field	*Override Constraints*

Examples

```
envGetVal("layout" "useWAOverrides")
envSetVal("layout" "useWAOverrides" 'boolean t)
envSetVal("layout" "useWAOverrides" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Override Constraints

viaMinNumCuts

```
layout viaMinNumCuts int integer_number
```

Description

The value specified for the environment variable is used to seed to `numCuts` widget of the WireAssistant in XL. This happens when VLS XL is started.

The changes of the Wire Assistant `numCuts` widget does not affect the environment variable and vice versa. Also, changing the value of the environment variable does not affect the Wire Assistant widget once VLS XL is started.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Net tab*
Field	*Min Num Cuts*

Examples

```
envGetVal("layout" "viaMinNumCuts")
envSetVal("layout" "viaMinNumCuts" 'int 100)
```

Related Topics

Automatic Router Environment Variables

Net

viaWECutColumns

`layout viaWECutColumns int` *`integer_number`*

Description

Specifies the number of columns of via cuts that can override the `minNumCut` rule with a more explicit configuration for creating and editing wires.

Note: The *Rows* and *Columns* field values are used by interactive routing commands, such as *Create Wire* and *Create Bus*. The combination of values must still be satisfied, however, can exceed the specified minimum number of cuts.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Net tab*
Field	*Col*

Examples

```
envGetVal("layout" "viaWECutColumns")
envSetVal("layout" "viaWECutColumns" 'int 1)
```

Related Topics

Automatic Router Environment Variables

Net

viaWECutRows

```
layout viaWECutRows int integer_number
```

Description

Specifies the number of rows of via cuts that can override the `minNumCut` rule with a more explicit configuration for creating and editing wires.

Note: The *Rows* and *Columns* field values are used by interactive routing commands, such as *Create Wire* and *Create Bus*. The combination of values must still be satisfied, however, can exceed the specified minimum number of cuts.

GUI Equivalent

Command	*Window – Assistant – Wire Assistant – Net tab*
Field	*Row*

Examples

```
envGetVal("layout" "viaWECutRows")
envSetVal("layout" "viaWECutRows" 'int 1)
```

Related Topics

Automatic Router Environment Variables

Net

viaWECutSpecified

```
layout viaWECutSpecified boolean { t | nil }
```

Description

Lets you toggle between the minimum number of cuts field and number of cuts by rows and columns field and then allows you to specify the number of cuts in *Row* and *Col* fields, respectively. Default is `nil`.

Note: The *Rows* and *Columns* field values are used by interactive routing commands, such as *Create Wire* and *Create Bus*. The combination of values must still be satisfied, however, can exceed the specified minimum number of cuts.

GUI Equivalent

Command	*Window – Assistant – Wire Assistant – Net tab*
Field	Button  next to *Min Num Cuts*

Examples

```
envGetVal("layout" "viaWECutSpecified")
envSetVal("layout" "viaWECutSpecified" 'boolean t)
envSetVal("layout" "viaWECutSpecified" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Net

weBusBitShieldingGap

```
layout weBusBitShieldingGap float float_number
```

Description

Specifies the spacing value for the bus bit. Default is `-1.0`.

GUI Equivalent

Command	*Window – Assistant – Wire Assistant – Bus* tab
Field	*Gap*

Examples

```
envGetVal("layout" "weBusBitShieldingGap")
envSetVal("layout" "weBusBitShieldingGap" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Bus

weBusBitShieldingNetName

```
layout weBusBitShieldingNetName string "shielding_net_name"
```

Description

Specifies the shielding net ID. Default is “ “.

GUI Equivalent

Command	*Window – Assistant – Wire Assistant – Bus* tab
Field	*Net*

Examples

```
envGetVal("layout" "weBusBitShieldingNetName")
envSetVal("layout" "weBusBitShieldingNetName" 'string "netA")
```

Related Topics

Automatic Router Environment Variables

Bus

weBusBitShieldingOption

```
layout weBusBitShieldingOption cyclic { "None" | "Default Lookup" | "Around Bus" |
     "Every Bit" }
```

Description

Specifies the shielding net mode. The three shielding modes that are supported are: `None`, `Default Lookup`, `Around Bus`, `Every Bit`. Default is `Default Lookup`.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Bus* tab
Field	*Add Shielding*

Examples

```
envGetVal("layout" "weBusBitShieldingOption")
envSetVal("layout" "weBusBitShieldingOption" 'cyclic "None")
envSetVal("layout" "weBusBitShieldingOption" 'cyclic "Around Bus")
envSetVal("layout" "weBusBitShieldingOption" 'cyclic "Every Bit")
```

Related Topics

Automatic Router Environment Variables

Bus

weBusBitShieldingWidth

```
layout weBusBitShieldingWidth float float_number
```

Description

Specifies the shielding width of the bus bit. Default is `-1.0`.

GUI Equivalent

Command	*Window – Assitant – Wire Assistant – Bus* t
Field	*Width*

Examples

```
envGetVal("layout" "weBusBitShieldingWidth")
envSetVal("layout" "weBusBitShieldingWidth" 'float 10.0)
```

Related Topics

Automatic Router Environment Variables

Bus

rte

checkRoutabilityBlockage

```
rte checkRoutabilityBlockage boolean { t | nil }
```

Description

Checks for the pins blocked by top-level blockage or blockage of another instance.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Routability Check
Field	*Pin Checks – Accessibility*

Examples

```
envGetVal("rte" "checkRoutabilityBlockage")
envSetVal("rte" "checkRoutabilityBlockage" 'boolean t)
envSetVal("rte" "checkRoutabilityBlockage" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

checkRoutabilityMinSpace

```
rte checkRoutabilityMinSpace boolean { t | nil }
```

Description

Directs the router to check for minimum pin spacing violations.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assitant – Wire Assistant – Automatic – Routability Check
Field	*Pin Checks – Spacing*

Examples

```
envGetVal("rte" "checkRoutabilityMinSpace")
envSetVal("rte" "checkRoutabilityMinSpace" 'boolean t)
envSetVal("rte" "checkRoutabilityMinSpace" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

checkRoutabilityMinWidth

```
rte checkRoutabilityMinWidth boolean { t | nil }
```

Description

Directs the router to check for minimum pin width violations.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assistant – Wire Assistant – Automatic – Routability Check
Field	*Pin Checks – Width*

Examples

```
envGetVal("rte" "checkRoutabilityMinWidth")
envSetVal("rte" "checkRoutabilityMinWidth" 'boolean t)
envSetVal("rte" "checkRoutabilityMinWidth" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

checkRoutabilityVia

```
rte checkRoutabilityVia boolean { t | nil }
```

Description

Directs the router to check for via escapes.

GUI Equivalent

Command	*Route – Design Setup – Routability Check*
	Window – Assistant – Wire Assistant – Automatic – Routability Check
Field	*Pin Checks – Accessibility*

Examples

```
envGetVal("rte" "checkRoutabilityVia")
envSetVal("rte" "checkRoutabilityVia" 'boolean t)
envSetVal("rte" "checkRoutabilityVia" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Routability Check

coverObsMasterTooltipLimit

```
rte coverObsMasterTooltipLimit int integer_number
```

Description

When the number of masters in the layout exceeds the limit specified using the `coverObsMasterTooltipLimit` environment variable, the Cover Obstructions Manager form does not provide the list of master's instance tooltip. This environment variable has been added to improve the performance when working with large layout designs. Default is `20`.

GUI Equivalent

None

Examples

```
envGetVal("rte" "coverObsMasterTooltipLimit")
envSetVal("rte" "coverObsMasterTooltipLimit" 'int 100)
```

Related Topics

Automatic Router Environment Variables

Managing Cover Obstructions While Routing

coverObstUnhiliteOnFormClosed

```
rte coverObstUnhiliteOnFormClosed boolean { t | nil }
```

Description

When set to `t`, it automatically unhighlights all cover obstructions when the Cover Obstruction Manager form is closed. When set to `nil`, the cover obstructions highlight color continues to display in the layout window even after the Cover Obstruction Manager form is closed. Default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("rte" "coverObstUnhiliteOnFormClosed")
envSetVal("rte" "coverObstUnhiliteOnFormClosed" 'boolean t)
envSetVal("rte" "coverObstUnhiliteOnFormClosed" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Managing Cover Obstructions While Routing

deleteRoutingKeepFigGroupShapes

```
rte deleteRoutingKeepFigGroupShapes boolean { t | nil }
```

Description

Directs the router to retain the fig group shapes when deleting the routing topology. The default value is `t`. When set to `nil`, the routing within fig groups is deleted.

GUI Equivalent

Command	*Route – Design Setup – Delete Routing*
	Window – Assistant – Wire Assistant – Automatic – Delete Routing
Field	*Keep Fig Group Shapes*

Examples

```
envGetVal("rte" "deleteRoutingKeepFigGroupShapes")
envSetVal("rte" "deleteRoutingKeepFigGroupShapes" 'boolean t)
envSetVal("rte" "deleteRoutingKeepFigGroupShapes" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Deleting Routing

deleteRoutingKeepPower

```
rte deleteRoutingKeepPower boolean { t | nil }
```

Description

Directs the router to keep the routed power nets when deleting the routing topology.

GUI Equivalent

Command	*Route – Design Setup – Delete Routing*
	Window – Assistant – Wire Assistant – Automatic – Delete Routing
Field	*Keep Power Nets*

Examples

```
envGetVal("rte" "deleteRoutingKeepPower")
envSetVal("rte" "deleteRoutingKeepPower" 'boolean t)
envSetVal("rte" "deleteRoutingKeepPower" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Deleting Routing

extractedPinStyle

```
rte extractedPinStyle cyclic { "Labeled Shapes Only" | "Connected Shapes on Same
     Layer" | "Whole Net on Routing Layers" }
```

Description

Lets you control which shapes should be converted into a pin shape. The shapes that are labeled or connected are extracted as pin shapes.

- `Labeled Shapes Only` only the shapes that are marked by text or property to become pin shapes. This option is selected by default.
- `Connected Shapes On Same Layer` only the marked shapes and the shapes that are recursively connected to become pin shapes.
- `Whole Net on Routing Layers` all the shapes in the net to become pin shapes.

Default is `Labeled Shapes Only`.

GUI Equivalent

Command	*Route – Design Setup*
Field	*Extracted Pin Style*

Examples

```
envGetVal("rte" "extractedPinStyle")
envSetVal("rte" "extractedPinStyle" 'cyclic "Labeled Shapes Only")
envSetVal("rte" "extractedPinStyle" 'cyclic "Connected Shapes on Same Layer")
envSetVal("rte" "extractedPinStyle" 'cyclic "Whole Net on Routing Layers")
)
```

Related Topics

Automatic Router Environment Variables

Extraction

fixErrorsErrorTypesExtension

```
rte fixErrorsErrorTypesExtension boolean { t | nil }
```

Description

Directs the router to fix all extension violations based on the extension constraints defined in the design , such as `minOppExtension` and `minExtensionEdge` constraints. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assistant – Wire Assistant – Automatic – Fix Violations
Field	*Extension*

Examples

```
envGetVal("rte" "fixErrorsErrorTypesExtension")
envSetVal("rte" "fixErrorsErrorTypesExtension" 'boolean t)
envSetVal("rte" "fixErrorsErrorTypesExtension" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

fixErrorsFixMinAreaAtPins

```
rte fixErrorsFixMinAreaAtPins boolean { t | nil }
```

Description

Directs the router to fix the minimum area violations. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assistant – Wire Assistant – Automatic – Fix Violations
Field	*Minimum Spacing*

Examples

```
envGetVal("rte" "fixErrorsFixMinAreaAtPins")
envSetVal("rte" "fixErrorsFixMinAreaAtPins" 'boolean t)
envSetVal("rte" "fixErrorsFixMinAreaAtPins" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

fixErrorsErrorTypesMinArea

```
rte fixErrorsErrorTypesMinArea boolean { t | nil }
```

Description

Directs the router to fix the `minArea` rule violations. Default is `t`

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assistant – Wire Assistant – Automatic – Fix Violations
Field	*Minimum Area*

Examples

```
envGetVal("rte" "fixErrorsErrorTypesMinArea")
envSetVal("rte" "fixErrorsErrorTypesMinArea" 'boolean t)
envSetVal("rte" "fixErrorsErrorTypesMinArea" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

fixErrorsErrorTypesMinEnclArea

```
rte fixErrorsErrorTypesMinEnclArea boolean { t | nil }
```

Description

Directs the router to fix the `minEnclosedArea` rule violations. Default is `t`

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assistant – Wire Assistant – Automatic – Fix Violations
Field	*Minimum Enclosed Area*

Examples

```
envGetVal("rte" "fixErrorsErrorTypesMinEnclArea")
envSetVal("rte" "fixErrorsErrorTypesMinEnclArea" 'boolean t)
envSetVal("rte" "fixErrorsErrorTypesMinEnclArea" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

fixErrorsErrorTypesMinEdge

```
rte fixErrorsErrorTypesMinEdge boolean { t | nil }
```

Description

Directs the router to fix the `minEdgeLength` rule violations. Default is `t`

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assistant – Wire Assistant – Automatic – Fix Violations
Field	*Minimum Edge*

Examples

```
envGetVal("rte" "fixErrorsErrorTypesMinEdge")
envSetVal("rte" "fixErrorsErrorTypesMinEdge" 'boolean t)
envSetVal("rte" "fixErrorsErrorTypesMinEdge" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

fixErrorsErrorTypesMinWidth

```
rte fixErrorsErrorTypesMinWidth boolean { t | nil }
```

Description

Directs the router to fix the `minWidth` rule violations. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assitant – Wire Assistant – Automatic – Fix Violations
Field	*Minimum Width*

Examples

```
envGetVal("rte" "fixErrorsErrorTypesMinWidth")
envSetVal("rte" "fixErrorsErrorTypesMinWidth" 'boolean t)
envSetVal("rte" "fixErrorsErrorTypesMinWidth" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

fixErrorsErrorTypesRGrid

```
rte fixErrorsErrorTypesRGrid boolean { t | nil }
```

Description

Directs the router to fix the routing grid violations. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assitant – Wire Assistant – Automatic – Fix Violations
Field	*Routing Grid*

Examples

```
envGetVal("rte" "fixErrorsErrorTypesRGrid")
envSetVal("rte" "fixErrorsErrorTypesRGrid" 'boolean t)
envSetVal("rte" "fixErrorsErrorTypesRGrid" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

fixErrorsErrorTypesNumCut

```
rte fixErrorsErrorTypesNumCut boolean { t | nil }
```

Description

Directs the router to fix the `minNumCut` rule violations. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assitant – Wire Assistant – Automatic – Fix Violations
Field	*Minimum Number of Cuts*

Examples

```
envGetVal("rte" "fixErrorsErrorTypesNumCut")
envSetVal("rte" "fixErrorsErrorTypesNumCut" 'boolean t)
envSetVal("rte" "fixErrorsErrorTypesNumCut" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

fixErrorsErrorTypesSameMaskSpacing

```
rte fixErrorsErrorTypesSameMaskSpacing boolean { t | nil }
```

Description

When set to `t`, attempts to fix same mask spacing spacing and cut class spacing violations. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Fix Violations*
	Window – Assitant – Wire Assistant – Automatic – Fix Violations
Field	*Same Mask Spacing*

Examples

```
envGetVal("rte" "fixErrorsErrorTypesSameMaskSpacing")
envSetVal("rte" "fixErrorsErrorTypesSameMaskSpacing" 'boolean t)
envSetVal("rte" "fixErrorsErrorTypesSameMaskSpacing" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Fix Violations

genRoutingStyle

```
rte genRoutingStyle cyclic { "Automatic" | "Device" | "ASIC" | "Chip Assembly" }
```

Description

Specifes the routing style for the design to be routed. The routing styles that are supported are: `Device Level`, `ASIC`, and `Chip Assembly`.

- `Automatic` is determined based on the size of the layout and the composition of macro instances. This evaluation happens when the design is first processed by the routing commands. The design style is retained until explicitly re-evaluated using the Refresh button to the right of the design style field.
- `Device` used when you need to route device-level and smaller designs that do not require congestion planning.
- `ASIC` used when you need to route custom digital, standard cell, and ASIC designs that require congestion planning.
- `Chip Assembly` used when you need to route chip assembly and larger designs that contain large macros and require congestion planning.

Default is `Automatic`.

GUI Equivalent

Command	*Route – Design Setup*
	Window – Assitant – Wire Assistant – Automatic
Field	*Design Style*

Examples

```
envGetVal("rte" "genRoutingStyle")
envSetVal("rte" "genRoutingStyle" 'cyclic "Automatic")
envSetVal("rte" "genRoutingStyle" 'cyclic "Device")
envSetVal("rte" "genRoutingStyle" 'cyclic "ASIC")
envSetVal("rte" "genRoutingStyle" 'cyclic "Chip Assembly")
```

Related Topics

Automatic Router Environment Variables

Design Style

optimizeRouteReduceVias

```
rte optimizeRouteReduceVias boolean { t | nil }
```

Description

Directs the router to reduce the number of vias by increasing the number of small same layer jogs.

GUI Equivalent

Command	*Route – Design Setup – Optimization*
	Window – Assitant – Wire Assistant – Automatic – Optimization
Field	*Reduce Jogs Through Vias*

Examples

```
envGetVal("rte" "optimizeRouteReduceVias")
envSetVal("rte" "optimizeRouteReduceVias" 'boolean t)
envSetVal("rte" "optimizeRouteReduceVias" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Optimization

routeSearchAndRepair

```
rte routeSearchAndRepair boolean { t | nil }
```

Description

Directs the router to search for and attempt to fix same net and different net spacing violations. The default value is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Optimization*
	Window – Assitant – Wire Assistant – Automatic – Optimization
Field	*Minimum Spacing*

Examples

```
envGetVal("rte" "routeSearchAndRepair")
envSetVal("rte" "routeSearchAndRepair" 'boolean t)
envSetVal("rte" "routeSearchAndRepair" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Optimization

routingScriptingDir

```
rte routingScriptingDir string "directoryName"
```

Description

Sets the default RDE directories that store the routing scripts. Once this environment variable is set, the available directories appear in the *Select a Routing Script* form. For more information, refer to Routing Scripts.

The default is `.:rde:.cadence/dfII/rde:~/rde:$(INSTALLDIR)/samples/rde`, where directory is separated by a colon(:).

GUI Equivalent

Command	*Route – Routing Script – Browse*
Field	*Directories*

Examples

```
envGetVal("rte" "routingScriptingDir")
envSetVal("rte" "routingScriptingDir" 'string "dir1")
```

Related Topics

Automatic Router Environment Variables

Routing Scripts

Power Routing Environment Variables

The following sections describe the environment variables used in power routing.

- General Environment Variables
  - prouteSelectScheme
- Pad Ring Environment Variables
  - proutePadRingAllLayers
  - proutePadRingEdgePins
  - proutePadRingLayers
  - proutePadRingNets
  - proutePadRingRailPins
- Block Ring Environment Variables
  - powerRouteSingleRing
  - prouteBlockRingBlockClearance
  - prouteBlockRingChannels
  - prouteBlockRingContour
  - prouteBlockRingHorizLayer
  - prouteBlockRingLattice
  - prouteBlockRingNetClearance
  - prouteBlockRingNets
  - prouteBlockRingNetWidth
  - prouteBlockRingVertLayer
- Core Ring Environment Variables
  - prouteCoreRingRingAtCenter
  - prouteCoreRingCoreClearance
  - prouteCoreRingHorizLayer
  - prouteCoreRingInAreaClearance

- prouteTieShieldShieldTieFreq

General Environment Variables

prouteSelectScheme

```
rte prouteSelectScheme string "scheme_name"
```

Description

Defines the scheme name for the power routing operation.

GUI Equivalent

Command	*Route – Power Routing*
Field	*Scheme – Name*

Examples

```
envGetVal("rte" "proutePadRingAllLayers")
envSetVal("rte" "proutePadRingAllLayers" 'string "schemeA")
```

Related Topics

Automatic Router Environment Variables

Using the Power Routing Form

Power Routing Form

Pad Ring Environment Variables

proutePadRingAllLayers

```
rte proutePadRingAllLayers boolean { t | nil }
```

Description

Routes pad rings on all routing layers. The default value is `t`.

GUI Equivalent

Command	*Route – Power Routing – Pad Ring* tab
Field	*Specified Pin Layers* (unselected)

Examples

```
envGetVal("rte" "proutePadRingAllLayers")
envSetVal("rte" "proutePadRingAllLayers" 'boolean t)
envSetVal("rte" "proutePadRingAllLayers" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding a Pad Ring

Using the Pad Ring Form

Changing the Pad Ring Scheme Definition

rtePowerRoutePadRing

proutePadRingEdgePins

```
rte proutePadRingEdgePins boolean { t | nil }
```

Description

Routes to the pin edge. Values are `t` or `nil`. The default value is `nil`.

GUI Equivalent

Command	*Route – Power Routing – Pad Ring* tab
Field	*Edge Pins*

Examples

```
envGetVal("rte" "proutePadRingEdgePins")
envSetVal("rte" "proutePadRingEdgePins" 'boolean t)
envSetVal("rte" "proutePadRingEdgePins" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding a Pad Ring

Using the Pad Ring Form

Changing the Pad Ring Scheme Definition

rtePowerRoutePadRing

proutePadRingLayers

```
rte proutePadRingLayers string "layer_names"
```

Description

The layers to use to connect pad pins. Value is a string with one or more layer name separated by space character. If the string is empty, the router uses the first routable layer in the design.
The `?allRouteLayers` argument must be set to `nil` to use this argument.

GUI Equivalent

Command	*Route – Power Routing – Pad Ring* tab
Field	*Specified Pin Layers* (selected)

Examples

```
envGetVal("rte" "proutePadRingLayers")
```

```
envSetVal("rte" "proutePadRingLayers" 'string "Metal11 Metal12 Metal13 Metal14")
```

Related Topics

Automatic Router Environment Variables

Adding a Pad Ring

Using the Pad Ring Form

Changing the Pad Ring Scheme Definition

rtePowerRoutePadRing

routePadRingNets

```
rte routePadRingNets string "net_name"
```

Description

Declares power and ground nets for pad ring power route.

GUI Equivalent

None

Examples

```
envGetVal("rte" "routePadRingNets")
envSetVal("rte" "routePadRingNets" 'string "net1 net2")
```

Related Topics

Automatic Router Environment Variables

Adding a Pad Ring

Using the Pad Ring Form

Changing the Pad Ring Scheme Definition

rtePowerRoutePadRing

routePadRingRailPins

```
rte routePadRingRailPins boolean { t | nil }
```

Description

Routes to rail pins on the pads. Values are `t` or `nil`. The default value is `t`.

GUI Equivalent

Command	*Route – Power Routing – Pad Ring* tab
Field	*Rail Pins*

Examples

```
envGetVal("rte" "routePadRingRailPins")
envSetVal("rte" "routePadRingRailPins" 'boolean t)
envSetVal("rte" "routePadRingRailPins" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding a Pad Ring

Using the Pad Ring Form

Changing the Pad Ring Scheme Definition

rtePowerRoutePadRing

Block Ring Environment Variables

powerRouteSingleRing

```
rte powerRouteSingleRing boolean { t | nil }
```

Description

Specifies that the block rings are executed individually on each block. The default value is `nil`.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Blocks – Style*

Examples

```
envGetVal("rte" "powerRouteSingleRing")
envSetVal("rte" "powerRouteSingleRing" 'boolean t)
envSetVal("rte" "powerRouteSingleRing" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

Power Routing - Block Ring tab

rtePowerRouteBlockRing

prouteBlockRingBlockClearance

```
rte prouteBlockRingBlockClearance float float_number
```

Description

Separates the block rings and prBoundary of the block instance by this value. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Block Clearance*

Examples

```
envGetVal("rte" "prouteBlockRingBlockClearance")
envSetVal("rte" "prouteBlockRingBlockClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

prouteBlockRingChannels

```
rte prouteBlockRingChannels boolean { t | nil }
```

Description

Adds rails in the channels between block instances. Values are `t` or `nil`. The default value is `nil`.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Routing Style/Channel*

Examples

```
envGetVal("rte" "prouteBlockRingChannels")
envSetVal("rte" "prouteBlockRingChannels" 'boolean t)
envSetVal("rte" "prouteBlockRingChannels" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

prouteBlockRingContour

```
rte prouteBlockRingContour boolean { t | nil }
```

Description

Causes power nets to follow the contour of block instances. Values are `t` or `nil`. The default value is `nil`.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Routing Style/Contour*

Examples

```
envGetVal("rte" "prouteBlockRingContour")
envSetVal("rte" "prouteBlockRingContour" 'boolean t)
envSetVal("rte" "prouteBlockRingContour" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

prouteBlockRingHorizLayer

```
rte prouteBlockRingHorizLayer string "layer_names"
```

Description

The horizontal metal layer to use to route the core ring. Value is a layer string.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Horizontal Routing Layer*

Examples

```
envGetVal("rte" "prouteBlockRingHorizLayer")
envSetVal("rte" "prouteBlockRingHorizLayer" 'string "Metal1 Metal3 Metal5")
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

prouteBlockRingLattice

```
rte prouteBlockRingLattice boolean { t | nil }
```

Description

Extends duplicated net segments to or from a lattice. Values are `t` or `nil`. If set to `nil`, the segments are concentric. The default value is `nil`.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Routing Style/Lattice Style*

Examples

```
envGetVal("rte" "prouteBlockRingLattice")
envSetVal("rte" "prouteBlockRingLattice" 'boolean t)
envSetVal("rte" "prouteBlockRingLattice" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

prouteBlockRingNetClearance

```
rte prouteBlockRingNetClearance float float_number
```

Description

The minimum clearance between any two power nets. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Ring Clearance*

Examples

```
envGetVal("rte" "prouteBlockRingNetClearance")
envSetVal("rte" "prouteBlockRingNetClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

prouteBlockRingNets

```
rte prouteBlockRingNets string "net_name"
```

Description

Declares power and ground nets for block ring power route.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Net Ordering*

Examples

```
envGetVal("rte" "prouteBlockRingNets")
envSetVal("rte" "prouteBlockRingNets" 'string "net2 net3")
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

prouteBlockRingNetWidth

```
rte prouteBlockRingNetWidth float float_number
```

Description

The minimum width between any two power nets. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Ring Width*

Examples

```
envGetVal("rte" "prouteBlockRingNetWidth")
envSetVal("rte" "prouteBlockRingNetWidth" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

prouteBlockRingVertLayer

```
rte prouteBlockRingVertLayer string "layer_names"
```

Description

The vertical metal layer to use to route the core ring. Value is a layer string.

GUI Equivalent

Command	*Route – Power Routing – Block Ring* tab
Field	*Vertical Routing Layer*

Examples

```
envGetVal("rte" "prouteBlockRingVertLayer")
envSetVal("rte" "prouteBlockRingVertLayer" 'string "Metal2 Metal4 Metal6")
```

Related Topics

Automatic Router Environment Variables

Adding Block Rings

Using the Block Ring Form

Changing the Block Ring Scheme Definition

rtePowerRouteBlockRing

Core Ring Environment Variables

prouteCoreRingRingAtCenter

```
rte prouteCoreRingRingAtCenter boolean { t | nil }
```

Description

Routes a ring between the core area of the design and the surrounding pad instances. Values are `t` or `nil`. If `nil`, the ring is routed according to the setting of the `?relativeTo` argument. The default value is `t`.

GUI Equivalent

Command	*Route – Power Routing – Edit – Core Ring* tab
Field	*Centered between Core and Pads*

Examples

```
envGetVal("rte" "prouteCoreRingRingAtCenter")
envSetVal("rte" "prouteCoreRingRingAtCenter" 'boolean t)
envSetVal("rte" "prouteCoreRingRingAtCenter" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingCoreClearance

```
rte prouteCoreRingCoreClearance float float_number
```

Description

Use when `?relativeTo` is set to `'outsideCore`. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Clearance, Core (Outside)*

Examples

```
envGetVal("rte" "prouteCoreRingCoreClearance")
envSetVal("rte" "prouteCoreRingCoreClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingHorizLayer

```
rte prouteCoreRingHorizLayer string "layer_names"
```

Description

The horizontal metal layer to be used to route the core ring. Value is a layer string.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Horizontal Routing Layer*

Examples

```
envGetVal("rte" "prouteCoreRingHorizLayer")
envSetVal("rte" "prouteCoreRingHorizLayer" 'string "Metal1 Metal3 Metal5")
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingInAreaClearance

```
rte prouteCoreRingInAreaClearance float float_number
```

Description

Use when `?relativeTo` is set to `'insideRegion`. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Clearance, Specific Region (inside)*

Examples

```
envGetVal("rte" "prouteCoreRingInAreaClearance")
envSetVal("rte" "prouteCoreRingInAreaClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingLattice

```
rte prouteCoreRingLattice boolean { t | nil }
```

Description

Extends duplicate net segments to or from a lattice. Values are `t` or `nil`. If set to `nil`, the segments are concentric. The default value is `nil`/`Concentric Rings`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Routing Style*

Examples

```
envGetVal("rte" "prouteCoreRingLattice")
envSetVal("rte" "prouteCoreRingLattice" 'boolean t)
envSetVal("rte" "prouteCoreRingLattice" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingNetClearance

```
rte prouteCoreRingNetClearance float float_number
```

Description

The minimum clearance between any two power nets. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Ring Clearance*

Examples

```
envGetVal("rte" "prouteCoreRingNetClearance")
envSetVal("rte" "prouteCoreRingNetClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingNets

```
rte prouteCoreRingNets string "net_name"
```

Description

Declares power and ground nets for core ring power route.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Net Ordering*

None

Examples

```
envGetVal("rte" "prouteCoreRingNets")
envSetVal("rte" "prouteCoreRingNets" 'string "netA netB")
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingNetWidth

```
rte prouteCoreRingNetWidth float float_number
```

Description

The minimum width between any two power nets. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Ring Width*

Examples

```
envGetVal("rte" "prouteCoreRingNetWidth")
envSetVal("rte" "prouteCoreRingNetWidth" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingOutAreaClearance

```
rte prouteCoreRingOutAreaClearance float float_number
```

Description

Use when `?relativeTo` is set to `'outsideRegion`. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Clearance, Specific Region (outside)*

Examples

```
envGetVal("rte" "prouteCoreRingOutAreaClearance")
envSetVal("rte" "prouteCoreRingOutAreaClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingPadClearance

```
rte prouteCoreRingPadClearance float float_number
```

Description

Use when `?relativeTo` is set to `'insideIOPads`. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Clearance, I/O Pads (inside)*

Examples

```
envGetVal("rte" "prouteCoreRingPadClearance")
envSetVal("rte" "prouteCoreRingPadClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingRelativeTo

```
rte prouteCoreRingRelativeTo cyclic { "Core (outside)" | "I/O Pads (inside)" |
     "Specific Region (inside)" | "Specific Region (outside)" }
```

Description

The location of the ring is relative to the specified location. The values are the following:

- `'insideIOPads`
- `'outsideCore`
- `'insideRegion`
- `'outsideRegion`

With `insideRegion` or `outsideRegion`, also specify the region using the `?areaX/YLo` and `?areaX/YHi` arguments.

You must also specify a positive, nonzero clearance value using one of the following four variables.

- prouteCoreRingRTXLo
- prouteCoreRingRTXLo
- prouteCoreRingInAreaClearance
- prouteCoreRingRTXLo

The default value is `Core (outside)`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Relative To*

Examples

```
envGetVal("rte" "prouteCoreRingRelativeTo")
envSetVal("rte" "prouteCoreRingRelativeTo" 'cyclic "Core (outside)")
envSetVal("rte" "prouteCoreRingRelativeTo" 'cyclic "I/O Pads (inside)")
envSetVal("rte" "prouteCoreRingRelativeTo" 'cyclic "Specific Region (inside)")
envSetVal("rte" "prouteCoreRingRelativeTo" 'cyclic "Specific Region (outside)")
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingRTXHi

```
rte prouteCoreRingRTXHi float float_number
```

Description

Use to specify the region if `?relativeTo` is set to either `'insideRegion` or `'outsideRegion`. Value is a floating number that must be within the prBoundary of the design. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Coordinates XHi*

Examples

```
envGetVal("rte" "prouteCoreRingRTXHi")
envSetVal("rte" "prouteCoreRingRTXHi" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingRTXLo

```
rte prouteCoreRingRTXLo float float_number
```

Description

Use to specify the region if `?relativeTo` is set to either `'insideRegion` or `'outsideRegion`. Value is a floating number that must be within the prBoundary of the design. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Coordinates XLo*

Examples

```
envGetVal("rte" "prouteCoreRingRTXLo")
envSetVal("rte" "prouteCoreRingRTXLo" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingRTYHi

```
rte prouteCoreRingRTYHi float float_number
```

Description

Use to specify the region if `?relativeTo` is set to either `'insideRegion` or `'outsideRegion`. Value is a floating number that must be within the prBoundary of the design. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Coordinates YHi*

Examples

```
envGetVal("rte" "prouteCoreRingRTYHi")
envSetVal("rte" "prouteCoreRingRTYHi" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingRTYLo

```
rte prouteCoreRingRTYLo float float_number
```

Description

Use to specify the region if `?relativeTo` is set to either `'insideRegion` or `'outsideRegion`. Value is a floating number that must be within the prBoundary of the design. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Coordinates YLo*

Examples

```
envGetVal("rte" "prouteCoreRingRTYLo")
envSetVal("rte" "prouteCoreRingRTYLo" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

prouteCoreRingVertLayer

```
rte prouteCoreRingVertLayer string "layer_names"
```

Description

The vertical metal layer to be used to route the core ring. Value is a layer string.

GUI Equivalent

Command	*Route – Power Routing – Core Ring* tab
Field	*Vertical Routing Layer*

Examples

```
envGetVal("rte" "prouteCoreRingVertLayer")
envSetVal("rte" "prouteCoreRingVertLayer" 'string "Metal2 Metal4 Metal6")
```

Related Topics

Automatic Router Environment Variables

Adding Core Rings

Using the Core Ring Form

Changing the Core Ring Scheme Definition

rtePowerRouteCoreRing

Cell Row Environment Variables

prouteCellRowExtend

```
rte prouteCellRowExtend boolean { t | nil }
```

Description

Routes the cell rows to the power rail. If more than one rail exists, routes to the farthest rail. Values are `t` or `nil`. The default value is `t`.

GUI Equivalent

Command	*Route – Power Routing – Cell Rows* tab
Field	*Extend Straps to Rails*

Examples

```
envGetVal("rte" "prouteCellRowExtend")
envSetVal("rte" "prouteCellRowExtend" 'boolean t)
envSetVal("rte" "prouteCellRowExtend" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Standard Cell Row Straps

Using the Cell Rows Form

Changing the Cell Rows Scheme Definition

rtePowerRouteCellRow

prouteCellRowLayers

```
rte prouteCellRowLayers string "layer_names"
```

Description

The layers to use to route the cell rows. Value is a list of layer strings. Separate layers with spaces. If the string is empty, the router uses the first routable layer in the design.
The `?allRouteLayers` argument must be set to `nil` to use this argument.

GUI Equivalent

Command	*Route – Power Routing – Cell Rows* tab
Field	*Specified Routing Layers* (selected)

Examples

```
envGetVal("rte" "prouteCellRowLayers")
envSetVal("rte" "prouteCellRowLayers" 'string "Metal1 Metal2 Metal3 Metal4")
```

Related Topics

Automatic Router Environment Variables

Adding Standard Cell Row Straps

Using the Cell Rows Form

Changing the Cell Rows Scheme Definition

rtePowerRouteCellRow

prouteCellRowNets

```
rte prouteCellRowNets string "net_name"
```

Description

Declares power and ground nets for cell row power route.

GUI Equivalent

None

Examples

```
envGetVal("rte" "prouteCellRowNets")
envSetVal("rte" "prouteCellRowNets" 'string "netB netC")
```

Related Topics

Automatic Router Environment Variables

Adding Standard Cell Row Straps

Using the Cell Rows Form

Changing the Cell Rows Scheme Definition

rtePowerRouteCellRow

prouteCellRowRowEnd

```
rte prouteCellRowRowEnd boolean { t | nil }
```

Description

Routes standard cells to the end of defined rows. Values are `t` or `nil`. If `nil`, standard cells are routed to the last cell in the row. The default value is `nil`.

GUI Equivalent

Command	*Route – Power Routing – Cell Rows* tab
Field	*Boundary Rule – End of Row*

Examples

```
envGetVal("rte" "prouteCellRowRowEnd")
envSetVal("rte" "prouteCellRowRowEnd" 'boolean t)
envSetVal("rte" "prouteCellRowRowEnd" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Standard Cell Row Straps

Using the Cell Rows Form

Changing the Cell Rows Scheme Definition

rtePowerRouteCellRow

Pin To Trunk Environment Variables

routePinToTrunkAllLayers

```
rte routePinToTrunkAllLayers boolean { t | nil }
```

Description

Uses all layers for pin to trunk routing. Values are `t` or `nil`. If `nil`, the router uses the layers specified by the `?pinLayers` argument. The default value is `t`.

GUI Equivalent

Command	*Route – Power Routing – Pin To Trunk tab*
Field	*Specified Pin Layers (unselected)*

Examples

```
envGetVal("rte" "routePinToTrunkAllLayers")
envSetVal("rte" "routePinToTrunkAllLayers" 'boolean t)
envSetVal("rte" "routePinToTrunkAllLayers" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Connecting Pin-to-Trunk

Using the Pin To Trunk Form

Changing the Pin-To-Trunk Scheme Definition

rtePowerRoutePinToTrunk

proutePinToTrunkLayers

```
rte proutePinToTrunkLayers string "layer_names"
```

Description

The layers to use for pin to trunk routing. Value is a list of layer strings. Separate layers with spaces. If the string is empty, the router uses the first routable layer in the design.

The `?allPinLayers` argument must be set to `nil` to use this argument.

GUI Equivalent

Command	*Route – Power Routing – Pin To Trunk* tab
Field	*Specified Pin Layers* (selected)

Examples

```
envGetVal("rte" "proutePinToTrunkLayers")
envSetVal("rte" "proutePinToTrunkLayers" 'string "Metal1 Metal2 Metal3 Metal4")
```

Related Topics

Automatic Router Environment Variables

Connecting Pin-to-Trunk

Using the Pin To Trunk Form

Changing the Pin-To-Trunk Scheme Definition

rtePowerRoutePinToTrunk

proutePinToTrunkMaxWireWidth

```
rte proutePinToTrunkMaxWireWidth float float_number
```

Description

The maximum wire width for pin to trunk connection. Value is a positive, nonzero, floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Pin To Trunk* tab
Field	Maximum Wire Width

Examples

```
envGetVal("rte" "proutePinToTrunkMaxWireWidth")
envSetVal("rte" "proutePinToTrunkMaxWireWidth" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Connecting Pin-to-Trunk

Using the Pin To Trunk Form

Changing the Pin-To-Trunk Scheme Definition

rtePowerRoutePinToTrunk

routePinToTrunkMinTrunkWidth

```
rte routePinToTrunkMinTrunkWidth float float_number
```

Description

The minimum width of a target trunk for pin to trunk connection. Value is a positive, nonzero, floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Pin To Trunk* tab
Field	*Minimum Trunk Width*

Examples

```
envGetVal("rte" "routePinToTrunkMinTrunkWidth")
envSetVal("rte" "routePinToTrunkMinTrunkWidth" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Connecting Pin-to-Trunk

Using the Pin To Trunk Form

Changing the Pin-To-Trunk Scheme Definition

rtePowerRoutePinToTrunk

proutePinToTrunkMinWireWidth

```
rte proutePinToTrunkMinWireWidth float float_number
```

Description

The minimum wire width for pin to trunk connection. Value is a positive, nonzero, floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Pin To Trunk* tab
Field	*Minimum Wire Width*

Examples

```
envGetVal("rte" "proutePinToTrunkMinWireWidth")
envSetVal("rte" "proutePinToTrunkMinWireWidth" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Connecting Pin-to-Trunk

Using the Pin To Trunk Form

Changing the Pin-To-Trunk Scheme Definition

rtePowerRoutePinToTrunk

proutePinToTrunkNets

```
rte proutePinToTrunkNets string "net_name"
```

Description

Declares power and ground nets for pin to trunk power route.

GUI Equivalent

None

Examples

```
envGetVal("rte" "proutePinToTrunkNets")
envSetVal("rte" "proutePinToTrunkNets" 'string "netC netD")
```

Related Topics

Automatic Router Environment Variables

Connecting Pin-to-Trunk

Using the Pin To Trunk Form

Changing the Pin-To-Trunk Scheme Definition

rtePowerRoutePinToTrunk

proutePinToTrunkSpecTLayer

```
rte proutePinToTrunkSpecTLayer boolean { t | nil }
```

Description

Uses the trunk layer for routing . Values are `t` or `nil`. If set to `t`, `?trunkLayer` must contain a valid routing layer. The default value is `nil`.

GUI Equivalent

Command	*Route – Power Routing – Pin To Trunk* tab
Field	*Specified Trunk Layer* (unselected)

Examples

```
envGetVal("rte" "proutePinToTrunkSpecTLayer")
envSetVal("rte" "proutePinToTrunkSpecTLayer" 'boolean t)
envSetVal("rte" "proutePinToTrunkSpecTLayer" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Connecting Pin-to-Trunk

Using the Pin To Trunk Form

Changing the Pin-To-Trunk Scheme Definition

rtePowerRoutePinToTrunk

proutePinToTrunkTrunkLayer

```
rte proutePinToTrunkTrunkLayer string "layer_names"
```

Description

Uses this layer as the trunk layer. Value is a layer string.

GUI Equivalent

Command	*Route – Power Routing – Pin To Trunk* tab
Field	*Specified Trunk Layer* (selected)

Examples

```
envGetVal("rte" "proutePinToTrunkTrunkLayer")
envSetVal("rte" "proutePinToTrunkTrunkLayer" 'string "Metal1 Metal2 Metal3
Metal4")
```

Related Topics

Automatic Router Environment Variables

Connecting Pin-to-Trunk

Using the Pin To Trunk Form

Changing the Pin-To-Trunk Scheme Definition

rtePowerRoutePinToTrunk

Stripes Environment Variables

prouteStripesBottomOffset

```
rte prouteStripesBottomOffset float float_number
```

Description

When `?offsetFrom` is set to `'designBoundary`, use this value as the y location for the first horizontal stripe relative to the bottom of the design area.

When `?offsetFrom` is set to `'origin`, use this value as the x location for the first vertical stripe relative to the design origin.

When `?offsetFrom` is set to `'routingArea`, this value has no affect.

The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Bottom*

Examples

```
envGetVal("rte" "prouteStripesBottomOffset")
envSetVal("rte" "prouteStripesBottomOffset" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesCenterLine

```
rte prouteStripesCenterLine boolean { t | nil }
```

Description

Uses the centerline of the stripes for offset measurement. Value is `t` or `nil`. If `nil`, the offset is measured from the left edge for the first vertical strip and from the bottom edge for the first horizontal stripe.

The default value is `nil (Edge of Stripe).`

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Relative To*

Examples

```
envGetVal("rte" "prouteStripesCenterLine")
envSetVal("rte" "prouteStripesCenterLine" 'boolean t)
envSetVal("rte" "prouteStripesCenterLine" 'boolean nil)
```

Related Topics

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesHorizDir

```
rte prouteStripesHorizDir boolean { t | nil }
```

Description

Creates horizontal stripes. Values are `t` or `nil`. The default value is `t`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Horizontal Stripes*

Examples

```
envGetVal("rte" "prouteStripesHorizDir")
envSetVal("rte" "prouteStripesHorizDir" 'boolean t)
envSetVal("rte" "prouteStripesHorizDir" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesHorizLayers

```
rte prouteStripesHorizLayers string "layer_names"
```

Description

Use the specified horizontal layers to route the horizontal stripes. `?horiStripes` must be set to `t` for this argument to take affect. Value is a list of horizontal metal layers. Separate layers with spaces. If the string is empty, the router uses the first routable layer in the design.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Routing Layers*

Examples

```
envGetVal("rte" "prouteStripesHorizLayers")
envSetVal("rte" "prouteStripesHorizLayers" 'string "Metal1 Metal2 Metal3 Metal4")
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesLeftOffset

```
rte prouteStripesLeftOffset float float_number
```

Description

When `?offsetFrom` is set to `'designBoundary`, use this value as the `x` location for the first vertical stripe relative to the left of the design area.

When `?offsetFrom` is set to `'origin`, use this value as the `y` location for the first horizontal stripe relative to the design origin.

When `?offsetFrom` is set to `'routingArea`, this value has no affect.

The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Left*

Examples

```
envGetVal("rte" "prouteStripesLeftOffset")
envSetVal("rte" "prouteStripesLeftOffset" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesMinLength

```
rte prouteStripesMinLength float float_number
```

Description

Minimum stripe length. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Minimum Stripe Length*

Examples

```
envGetVal("rte" "prouteStripesMinLength")
envSetVal("rte" "prouteStripesMinLength" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesNetClearance

```
rte prouteStripesNetClearance float float_number
```

Description

Uses this clearance between stripes. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Net Clearance*

Examples

```
envGetVal("rte" "prouteStripesNetClearance")
envSetVal("rte" "prouteStripesNetClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesNets

```
rte prouteStripesNets string "net_name"
```

Description

Declares power and ground nets for stripes power route.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Nets Ordering*

Examples

```
envGetVal("rte" "prouteStripesNets")
envSetVal("rte" "prouteStripesNets" 'string "netB netC")
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesNetWidth

```
rte prouteStripesNetWidth float float_number
```

Description

Stripe width. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Net Width*

Examples

```
envGetVal("rte" "prouteStripesNetWidth")
envSetVal("rte" "prouteStripesNetWidth" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesOffsetFrom

```
rte prouteStripesOffsetFrom cyclic { "Design Boundary" | "Origin" | "Routing Area"
     }
```

Description

Stripes starting location. Values are the following:

- `'designBoundary`
- `'origin`
- `'routingArea`

The default value is `'designBoundary`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Offset From*

Examples

```
envGetVal("rte" "prouteStripesOffsetFrom")
envSetVal("rte" "prouteStripesOffsetFrom" 'cyclic "designBoundary")
envSetVal("rte" "prouteStripesOffsetFrom" 'cyclic "origin")
envSetVal("rte" "prouteStripesOffsetFrom" 'cyclic "routingArea")
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesPinClearance

```
rte prouteStripesPinClearance float float_number
```

Description

Uses this clearance between signal pins and power nets. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Pin Clearance*

Examples

```
envGetVal("rte" "prouteStripesPinClearance")
envSetVal("rte" "prouteStripesPinClearance" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesVertDir

```
rte prouteStripesVertDir boolean { t | nil }
```

Description

Creates vertical stripes. Values are `t` or `nil`. The default value is `t`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Vertical Stripes*

Examples

```
envGetVal("rte" "prouteStripesVertDir")
envSetVal("rte" "prouteStripesVertDir" 'boolean t)
envSetVal("rte" "prouteStripesVertDir" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesVertLayers

```
rte prouteStripesVertLayers string "layer_names"
```

Description

Uses the specified vertical layers to route the vertical stripes. `?vertStripes` must be set to `t` for this argument to take affect. Value is a list of vertical metal layers. Separate layers with spaces. If the string is empty, the router uses the first routable layer in the design.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Routing Layers*

Examples

```
envGetVal("rte" "prouteStripesVertLayers")
envSetVal("rte" "prouteStripesVertLayers" 'string "Metal1 Metal2 Metal3 Metal4")
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesXStep

```
rte prouteStripesXStep float float_number
```

Description

When `?vertStripes` is set to `t`, the router uses this value to separate the vertical stripes. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Step Increment*

Examples

```
envGetVal("rte" "prouteStripesXStep")
envSetVal("rte" "prouteStripesXStep" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

prouteStripesYStep

```
rte prouteStripesYStep float float_number
```

Description

When `?horiStripes` is set to `t`, the router uses this value to separate the horizontal stripes. Value is a positive, nonzero floating number. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Stripes* tab
Field	*Step Increment*

Examples

```
envGetVal("rte" "prouteStripesYStep")
envSetVal("rte" "prouteStripesYStep" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Adding Stripes

Using the Stripes Form

Changing the Stripes Scheme Definition

rtePowerRouteStripes

Vias Environment Variables

prouteViaInsertionColumns

`rte prouteViaInsertionColumns int` *`integer_number`*

Description

The number of cut columns. Value is an integer. The `?cutArrayRule` argument must be set to `'arraySize`, for this value to be in effect. The default value is `4`.

GUI Equivalent

Command	*Route – Power Routing – Vias* tab
Field	*Columns*

Examples

```
envGetVal("rte" "prouteViaInsertionColumns")
envSetVal("rte" "prouteViaInsertionColumns" 'int 10)
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

prouteViaInsertionCutArray

```
rte prouteViaInsertionCutArray cyclic { "square" | "rectangular" | "arraySize" }
```

Description

The cut via configuration shape. Possible values are:

- `'square`
- `'rectangular`
- `'arraySize`

If you specify `'arraySize`, you must also enter values for the `?cutArrayRows` and `?cutArrayColumns` arguments.

The default value is `Rectangular`.

GUI Equivalent

Command	*Route – Power Routing – Vias* tab
Field	*Cut Array*

Examples

```
envGetVal("rte" "prouteViaInsertionCutArray")
envSetVal("rte" "prouteViaInsertionCutArray" 'cyclic "Square")
envSetVal("rte" "prouteViaInsertionCutArray" 'cyclic "Rectangular")
envSetVal("rte" "prouteViaInsertionCutArray" 'cyclic "Specified Array Size:")
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

prouteViaInsertionLayerRange

```
rte prouteViaInsertionLayerRange boolean { t | nil }
```

Description

Uses the layer range specified by the `?minLayer` and `?maxLayer` arguments. Values are `t` or `nil`. If `nil`, the router uses the top and bottom layers of the design as the min and max layers of the layer range. The default value is `nil`.

GUI Equivalent

Command	*Route – Power Routing – Vias* tab
Field	*Layer Range*

Examples

```
envGetVal("rte" "prouteViaInsertionLayerRange")
envSetVal("rte" "prouteViaInsertionLayerRange" 'boolean t)
envSetVal("rte" "prouteViaInsertionLayerRange" 'boolean nil)
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

prouteViaInsertionLayers

```
rte prouteViaInsertionLayers string "layer_names"
```

Description

Defines the layer to which the via should be inserted.

GUI Equivalent

Command	*Route – Power Routing – Vias* tab
Field	*Via Location – Pin Location*

Examples

```
envGetVal("rte" "prouteViaInsertionLayers")
```

```
envSetVal("rte" "prouteViaInsertionLayers" 'string "Metal11 Metal12 Metal13")
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Power Routing - Vias tab

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

prouteViaInsertionLocation

```
rte prouteViaInsertionLocation cyclic { "Intersection of All Rings, Stripes, and
     Cell Rows" | "Selected Instances" }
```

"Gate" | "Source and Drain" | "All" }

Description

Lets you select the location at which inter-layer connections can take place and vias can be added. The possible values are:

- `Intersection of All Rings, Stripes, and Cell Rows` - adds vias at the intersections of all rings, stripes, and cell rows.
- `Selected Instances` - adds vias between stripes and the pins of selected block instances.

GUI Equivalent

Command	*Route – Power Routing – Vias* tab
Field	*Via Location*

Examples

```
envGetVal("rte" "prouteViaInsertionLocation")
envSetVal("rte" "prouteViaInsertionLocation" 'cyclic "Intersection of All Rings,
Stripes, and Cell Rows")
envSetVal("rte" "prouteViaInsertionLocation" 'cyclic "Selected Instances")
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Power Routing - Vias tab

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

prouteViaInsertionMaxLayer

```
rte prouteViaInsertionMaxLayer string "layer_names"
```

Description

Uses this layer as the highest metal layer for connection. The value is a layer string.

GUI Equivalent

Command	*Route – Power Routing – Vias* tab
Field	*Maximum*

Examples

```
envGetVal("rte" "prouteViaInsertionMaxLayer")
envSetVal("rte" "prouteViaInsertionMaxLayer" 'string "Metal1 Metal2 Metal3
Metal4")
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

prouteViaInsertionMinLayer

```
rte prouteViaInsertionMinLayer string "layer_names"
```

Description

Uses this layer as the lowest metal layer for connection. The value is a layer string.

GUI Equivalent

Command	*Route – Power Routing – Vias* tab
Field	*Minimum*

Examples

```
envGetVal("rte" "prouteViaInsertionMinLayer")
envSetVal("rte" "prouteViaInsertionMinLayer" 'string "Metal1 Metal2 Metal3
Metal4")
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

prouteViaInsertionNets

```
rte prouteViaInsertionNets string "net_name"
```

Description

Declares power and ground nets for via insertion.

GUI Equivalent

None

Examples

```
envGetVal("rte" "prouteViaInsertionNets")
envSetVal("rte" "prouteViaInsertionNets" 'string "netC netE")
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

prouteViaInsertionRows

```
rte prouteViaInsertionRows int integer_number
```

Description

The number of cut rows. Value is an integer. The `?cutArrayRule` argument must be set to `'arraySize`, for this value to be in effect. The default value is `4`.

GUI Equivalent

Command	*Route – Power Routing – Vias* tab
Field	*Rows*

Examples

```
envGetVal("rte" "prouteViaInsertionRows")
envSetVal("rte" "prouteViaInsertionRows" 'int 10)
```

Related Topics

Automatic Router Environment Variables

Inserting Vias

Using the Vias Form

Changing the Vias Scheme Definition

rtePowerRouteViaInsertion

Tie Shield Environment Variables

prouteTieShieldCoaxTieFreq

```
rte prouteTieShieldCoaxTieFreq float float_number
```

Description

Specifies the maximum distance between ties that must be inserted to tie tandem shield wires and parallel shield wires for coaxial shielding. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Tie Shield* tab
Field	*Tie Frequency – Coax*

Examples

```
envGetVal("rte" "prouteTieShieldCoaxTieFreq")
envSetVal("rte" "prouteTieShieldCoaxTieFreq" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Tie Shielding

Using the Tie Shield Form

rtePowerRouteTieShield

prouteTieShieldShieldTieFreq

```
rte prouteTieShieldShieldTieFreq float float_number
```

Description

Specifies the maximum distance between ties that must be inserted to tie the new shield wires to their respective shield nets. The default value is `0`.

GUI Equivalent

Command	*Route – Power Routing – Tie Shield* tab
Field	*Tie Frequency – Shield*

Examples

```
envGetVal("rte" "prouteTieShieldShieldTieFreq")
envSetVal("rte" "prouteTieShieldShieldTieFreq" 'float 1.0)
```

Related Topics

Automatic Router Environment Variables

Tie Shielding

Using the Tie Shield Form

rtePowerRouteTieShield

Pin to Trunk Environment Variables

The following sections describe the environment variables used in pin to trunk routing.

- Pin to Trunk Section
  - autoComposeTrunk
  - connectPinType
  - highLightTwigType
  - includeNonOrthogonalTwigs
  - orderTrunks
  - pinToTrunk
  - routingAreaLLx
  - routingAreaLLy
  - routingAreaURx
  - routingAreaURy
  - routingScope
  - selectEastTrunk
  - selectNorthTrunk
  - selectOrthoPinsMode
  - selectOverInstTrunk
  - selectSouthTrunk
  - selectTrunkMode
  - selectTrunkRangeValue
  - selectTrunkWithinRange
  - selectWestTrunk
  - trunkExtending
  - trunkTapering
  - trunkToTrunk

- nonGateTwigWidthMode
- mergeTapsInCloseProximity
- nonGateTwigLayerMode
- nonGateTwigWidthMode
- pinStrapLayer
- pinStrapLocationPreference
- pinStrapMaxPinCount
- pinStrapMaxPinDistance
- pinStrapping
- pinStrapPinLayer
- pinStrapWidth
- strapGateSourceDrainPins
- tapCoverPinPercent
- tapLowerViasCoverPinPercent
- trunkOverDeviceViaCoverMode
- trunkOverDeviceViaCoverPercent
- twigLengthMatching
- verGateTwigLayer
- verGateTwigWidth
- verNonGateTwigLayer
- verNonGateTwigWidth
- viaCoverPinWidth

■ Trunk to Trunk Section

- connectTrunkType

Pin to Trunk Section

autoComposeTrunk

```
p2t autoComposeTrunk boolean { t | nil }
```

Description

Converts shapes of selected nets into trunk objects. The trunk objects can then be used to perform Pin to Trunk routing. Shapes that are not used in Pin to Trunk routing are not converted into trunks. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Compose Trunks*

Examples

```
envGetVal("p2t" "autoComposeTrunk")
envSetVal("p2t" "autoComposeTrunk" 'boolean t)
envSetVal("p2t" "autoComposeTrunk" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Composing Trunks

connectPinType

```
p2t connectPinType cyclic { "Gate" | "Source and Drain" | "All" }
```

Description

Lets you select the type of pin that should be used for Pin to Trunk routing. The possible values are:

- `Gate`: connects the selected trunk to the gate pins.
- `Source and Drain`: connects the trunk to the source and drain pins.
- `All`: connects the trunk to all the existing pins.

Default is `All`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	Pin Type

Examples

```
envGetVal("p2t" "connectPinType")
envSetVal("p2t" "connectPinType" 'cyclic "Gate")
envSetVal("p2t" "connectPinType" 'cyclic "Source and Drain")
envSetVal("p2t" "connectPinType" 'cyclic "All")
```

Related Topics

Pin to Trunk Environment Variables

Pin Selection

highLightTwigType

```
p2t highLightTwigType cyclic { "Orthogonal" | "Non Orthogonal" | "All" }
```

Description

Controls twig highlighting in the `Highlight Trunks` command. The possible values are:

- `Orthogonal`: highlights only orthogonal twigs.
- `Non Orthogonal`: highlights only non-orthogonal twigs.
- `All`: highlights both orthogonal and non-orthogonal twigs.

Default is `Orthogonal`.

GUI Equivalent

Command	*Navigator – Net – Highlight Trunks*
Field	None

Examples

```
envGetVal("p2t" "highLightTwigType")
envSetVal("p2t" "highLightTwigType" 'cyclic "Orthogonal")
envSetVal("p2t" "highLightTwigType" 'cyclic "Non Orthogonal")
envSetVal("p2t" "highLightTwigType" 'cyclic "All")
```

Related Topics

Pin to Trunk Environment Variables

Highlighting Trunks

includeNonOrthogonalTwigs

```
p2t includeNonOrthogonalTwigs boolean { t | nil }
```

Description

When set to `t`, pins that are not orthogonally accessible to a trunk are selected by Pin to Trunk routing. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Include Non-orthogonal Twigs*

Examples

```
envGetVal("p2t" "includeNonOrthogonalTwigs")
envSetVal("p2t" "includeNonOrthogonalTwigs" 'boolean t)
envSetVal("p2t" "includeNonOrthogonalTwigs" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Include Non-orthogonal Twigs

orderTrunks

```
p2t orderTrunks boolean { t | nil }
```

Description

Optimizes the total routing length by re-ordering trunks not connected to a pin or another trunk. The trunks with more open twigs on the top are placed closer to the top end while the trunks with more open twigs on the bottom are placed closer to the bottom end. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	Trunk Ordering

Examples

```
envGetVal("p2t" "orderTrunks")
envSetVal("p2t" "orderTrunks" 'boolean t)
envSetVal("p2t" "orderTrunks" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Ordering Trunks

pinToTrunk

```
p2t pinToTrunk boolean { t | nil }
```

Description

Ensures that routing is completed between the pins orthogonal to the trunks. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Pin to Trunk Routing*

Examples

```
envGetVal("p2t" "pinToTrunk")
envSetVal("p2t" "pinToTrunk" 'boolean t)
envSetVal("p2t" "pinToTrunk" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

routingAreaLLx

```
p2t routingAreaLLx float float_number
```

Description

Specifies the lower-left X-coordinate of the area in which the partially or fully enclosed trunks and pins will be considered for routing. Default is `0.0`. Value must be a floating point number.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Specified Area (LLx)*

Examples

```
envGetVal("p2t" "routingAreaLLx")
envSetVal("p2t" "routingAreaLLx" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Range or Area

Specified Area

routingAreaLLy

```
p2t routingAreaLLy float float_number
```

Description

Specifies the lower-left Y-coordinate of the area in which the partially or fully enclosed trunks and pins will be considered for routing. Default is `0.0`. Value must be a floating point number.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Specified Area (LLy)*

Examples

```
envGetVal("p2t" "routingAreaLLy")
envSetVal("p2t" "routingAreaLLy" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Range or Area

Specified Area

routingAreaURx

```
p2t routingAreaURx float float_number
```

Description

Specifies the upper-right X-coordinate of the area in which the partially or fully enclosed trunks and pins will be considered for routing. Default is `0.0`. Value must be a floating point number.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Specified Area (URx)*

Examples

```
envGetVal("p2t" "routingAreaURx")
envSetVal("p2t" "routingAreaURx" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Range or Area

Specified Area

routingAreaURy

```
p2t routingAreaURy float float_number
```

Description

Specifies the upper-right Y-coordinate of the area in which the partially or fully enclosed trunks and pins will be considered for routing. Default is `0.0`. Value must be a floating point number.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Specified Area (URy)*

Examples

```
envGetVal("p2t" "routingAreaURy")
envSetVal("p2t" "routingAreaURy" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Range or Area

Specified Area

routingScope

```
p2t routingScope cyclic { "Current Editable Cellview" | "Current View Area" |
     "Specified Area" }
```

Description

Specifies the options to control the scope and region of Pin to Trunk routing. In addition, it allows you to specify the trunks that are to be used for Pin to Trunk routing and Pin to Trunk relation. This environment variable can take three values.

- `Current Editable Cellview`: all trunks and pins in the editable cellview are considered for routing.
- `Current View Area`: trunks and pins partially or fully enclosed in the current viewable area are considered for routing.
- `Specified Area`: specifies the area in which the trunks and pins partially or fully enclosed are considered for routing.

Default is `Current Editable Cellview`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Scope*

Examples

```
envGetVal("p2t" "routingScope")
envSetVal("p2t" "routingScope" 'cyclic "Current Editable Cellview")
envSetVal("p2t" "routingScope" 'cyclic "Current View Area")
envSetVal("p2t" "routingScope" 'cyclic "Specified Area")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Routing Scope

Specifying Range or Area

selectEastTrunk

```
p2t selectEastTrunk boolean { t | nil }
```

Description

Selects the trunks on the East side of a pin for routing. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Search from Pin Side (East)*

Examples

```
envGetVal("p2t" "selectEastTrunk")
envSetVal("p2t" "selectEastTrunk" 'boolean t)
envSetVal("p2t" "selectEastTrunk" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Search from Pin Side

selectNorthTrunk

```
p2t selectNorthTrunk boolean { t | nil }
```

Description

Selects the trunks on the North side of a pin for routing. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Search from Pin Side (North)*

Examples

```
envGetVal("p2t" "selectNorthTrunk")
envSetVal("p2t" "selectNorthTrunk" 'boolean t)
envSetVal("p2t" "selectNorthTrunk" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Search from Pin Side

selectOrthoPinsMode

```
p2t selectOrthoPinsMode cyclic { "Auto" | "Channel" | "Space" | "All" }
```

Description

Determines the mode that should be used to route the orthogonal pins to trunks. The possible values are:

- `Auto`: automatically determines the mode of pin selection for orthogonal pins.
- `Channel`: selects the pins orthogonally accessible to a trunk in the same channel.
- `Space`: selects the pins orthogonally accessible to a trunk through one or more channels.
- `All`: selects all the pins that are orthogonal to a trunk.

Default is `Auto`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Pin-Trunk Relation (Mode)*

Examples

```
envGetVal("p2t" "selectOrthoPinsMode")
envSetVal("p2t" "selectOrthoPinsMode" 'cyclic "Auto")
envSetVal("p2t" "selectOrthoPinsMode" 'cyclic "Channel")
envSetVal("p2t" "selectOrthoPinsMode" 'cyclic "Space")
envSetVal("p2t" "selectOrthoPinsMode" 'cyclic "All")
```

Related Topics

Pin to Trunk Environment Variables

Mode

selectOverInstTrunk

```
p2t selectOverInstTrunk boolean { t | nil }
```

Description

Selects the trunks that are over an instance for routing. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Search from Pin Side (Over)*

Examples

```
envGetVal("p2t" "selectOverInstTrunk")
envSetVal("p2t" "selectOverInstTrunk" 'boolean t)
envSetVal("p2t" "selectOverInstTrunk" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Search from Pin Side

selectSouthTrunk

```
p2t selectSouthTrunk boolean { t | nil }
```

Description

Selects the trunks on the South side of a pin for routing. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Search from Pin Side (South)*

Examples

```
envGetVal("p2t" "selectSouthTrunk")
envSetVal("p2t" "selectSouthTrunk" 'boolean t)
envSetVal("p2t" "selectSouthTrunk" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Search from Pin Side

selectTrunkMode

```
p2t selectTrunkMode cyclic { "Closest or Selected" | "As Many As Possible" }
```

Description

Lets you to select the number and the trunks that are to be used by the Pin to Trunk routing engine. The two possible values are:

- `Closest or Selected`: selects the closest trunk that is orthogonally accessible from the pin for routing. This means that a pin can be connected to only one trunk.
- `As Many As Possible`: all the trunks that are orthogonally accessible from a pin are selected for routing. This means that a pin can be connected to more than one trunks.

Default is `Closest or Selected`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Selection*

Examples

```
envGetVal("p2t" "selectTrunkMode")
envSetVal("p2t" "selectTrunkMode" 'cyclic "Closest or Selected")
envSetVal("p2t" "selectTrunkMode" 'cyclic "As Many As Possible")
```

Related Topics

Pin to Trunk Environment Variables

Trunk Selection

selectTrunkRangeValue

```
p2t selectTrunkRangeValue float float_number
```

Description

Specifies the value of the range from the edge of a channel in which trunks should be selected for routing. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*In Range*

Examples

```
envGetVal("p2t" "selectTrunkRangeValue")
envSetVal("p2t" "selectTrunkRangeValue" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

In Range

selectTrunkWithinRange

```
p2t selectTrunkWithinRange boolean { t | nil }
```

Description

Selects the trunks that are within a specified range from the edge of the channel where a pin can be accessed. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*In Range*

Examples

```
envGetVal("p2t" "selectTrunkWithinRange")
envSetVal("p2t" "selectTrunkWithinRange" 'boolean t)
envSetVal("p2t" "selectTrunkWithinRange" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

In Range

selectWestTrunk

```
p2t selectWestTrunk boolean { t | nil }
```

Description

Selects the trunks on the West side of a pin for routing. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Search from Pin Side (West)*

Examples

```
envGetVal("p2t" "selectWestTrunk")
envSetVal("p2t" "selectWestTrunk" 'boolean t)
envSetVal("p2t" "selectWestTrunk" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Search from Pin Side

trunkExtending

```
p2t trunkExtending boolean { t | nil }
```

Description

Extends the selected trunk while routing. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Trunk Extending*

Examples

```
envGetVal("p2t" "trunkExtending")
envSetVal("p2t" "trunkExtending" 'boolean t)
envSetVal("p2t" "trunkExtending" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Extending Trunks

Extend Trunk

trunkTapering

```
p2t trunkTapering boolean { t | nil }
```

Description

When set to `t`, enables trunk tapering. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Trunk Tapering*

Examples

```
envGetVal("p2t" "trunkTapering")
envSetVal("p2t" "trunkTapering" 'boolean t)
envSetVal("p2t" "trunkTapering" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Tapering Trunks

Taper Trunk Width

trunkToTrunk

```
p2t trunkToTrunk boolean { t | nil }
```

Description

Connects floating trunks and add vias at the intersection of the trunks. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Trunk to Trunk Routing*

Examples

```
envGetVal("p2t" "trunkToTrunk")
envSetVal("p2t" "trunkToTrunk" 'boolean t)
envSetVal("p2t" "trunkToTrunk" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Trunk to Trunk Routing

trunkTrimming

```
p2t trunkTrimming boolean { t | nil }
```

Description

Trims both ends of a trunk while routing. Default is `t`.

Note: Trunks with no twigs are not trimmed.

GUI Equivalent

Command	*Route – Design Setup – Pin to Trunk*
Field	*Trunk Trimming*

Examples

```
envGetVal("p2t" "trunkTrimming")
envSetVal("p2t" "trunkTrimming" 'boolean t)
envSetVal("p2t" "trunkTrimming" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Trimming Trunks

Trim Trunk

Trunk Section

extendTrunkDirMode

```
p2t extendTrunkDirMode cyclic { "One" | "Both" }
```

Description

Specifies the direction in which you want to extend the trunk. This environment variable can take two values.

- `One`: extends the trunk in a single direction.
- `Both`: extends the trunk in both directions up to the last pin.

Default is `Both`.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Extend Trunk Direction*

Examples

```
envGetVal("p2t" "extendTrunkDirMode")
envSetVal("p2t" "extendTrunkDirMode" 'cyclic "One")
envSetVal("p2t" "extendTrunkDirMode" 'cyclic "Both")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

extendTrunkEndForOneDirMode

```
p2t extendTrunkEndForOneDirMode cyclic { "Left/Lower" | "Right/Upper" }
```

Description

When set, you can extend the trunk either in the *Right/Upper* or *Left/Lower* direction. By default, the trunk is extended in the *Right/Upper* direction.

- `Right/Upper`: extends the right end of a horizontal trunk or upper end of a vertical trunk.
- `Left/Lower`: extends the left end of a horizontal trunk or lower end of a vertical trunk.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*One Direction for Auto Route*

Examples

```
envGetVal("p2t" "extendTrunkEndForOneDirMode")
envSetVal("p2t" "extendTrunkEndForOneDirMode" 'cyclic "Left/Lower")
envSetVal("p2t" "extendTrunkEndForOneDirMode" 'cyclic "Right/Upper")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

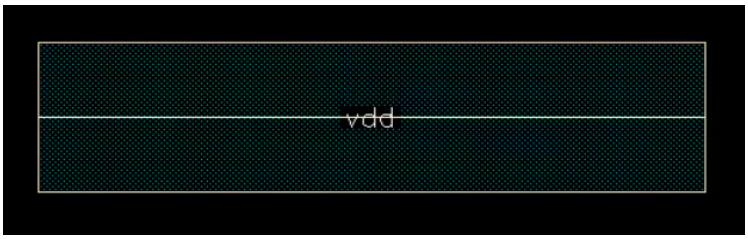
highlightTrunkHaloType

```
p2t highlightTrunkHaloType cyclic { "none" | "plain" | "glow" | "fadeout"}
```

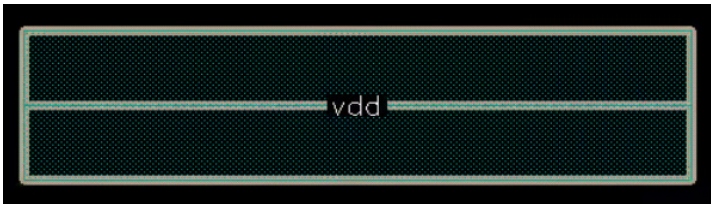
Description

Lets you specify the halo effect. The halo is added to provide a highlight around a trunk. The halo effects can be one of these:

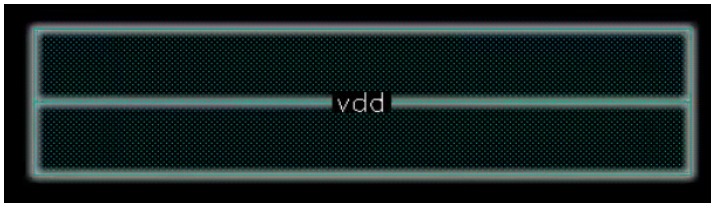
- `none`: does not add a halo effect around a trunk.



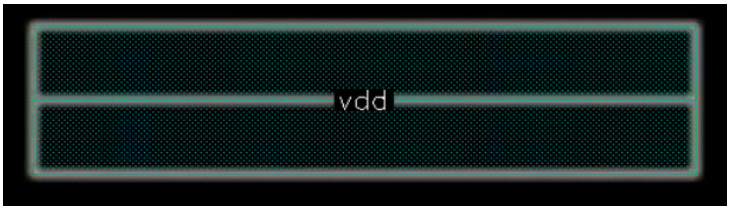
- `plain`: adds a plain halo surrounding a trunk. This is the default value.



- `glow`: adds a strong glowing halo surrounding a trunk.



- `fadeout`: has the same display effect as `glow`.



GUI Equivalent

None

Examples

```
envGetVal("p2t" "highlightTrunkHaloType")
envSetVal("p2t" "highlightTrunkHaloType" 'cyclic "none")
envSetVal("p2t" "highlightTrunkHaloType" 'cyclic "plain")
envSetVal("p2t" "highlightTrunkHaloType" 'cyclic "glow")
envSetVal("p2t" "highlightTrunkHaloType" 'cyclic "fadeout")
```

Related Topics

Pin to Trunk Environment Variables

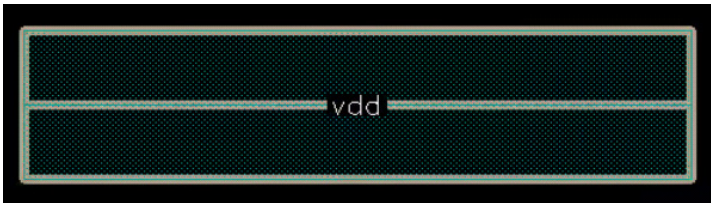
highlightTrunkHaloWidth

```
p2t highlightTrunkHaloWidth cyclic { "thin" | "normal" | "thick" }
```

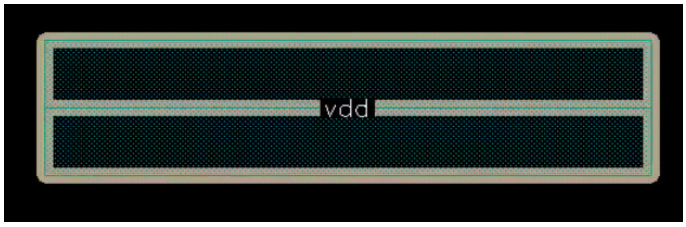
Description

Lets you specify the thickness of the halo that is added around a trunk. The default value is `thin`. This variable is effective only if `highLightTrunkHaloType` is set to a value other than `none`.

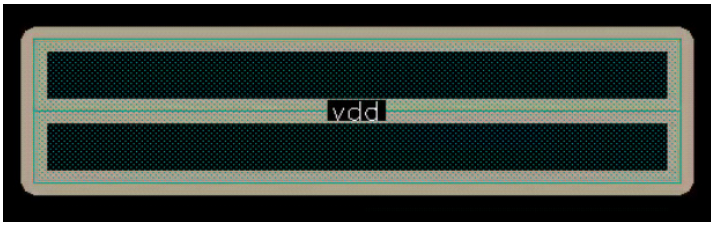
- `thin`: adds a halo with a width thinner than normal.



- `normal`: adds a halo with a width specified in the technology file.



- `thick`: adds a halo with a width thicker than normal.



GUI Equivalent

None

Examples

```
envGetVal("p2t" "highlightTrunkHaloWidth")
```

```
envSetVal("p2t" "highlightTrunkHaloWidth" 'cyclic "thin")
envSetVal("p2t" "highlightTrunkHaloWidth" 'cyclic "normal")
envSetVal("p2t" "highlightTrunkHaloWidth" 'cyclic "thick")
```

Related Topics

Pin to Trunk Environment Variables

trimTrunkMode

```
p2t trimTrunkMode cyclic { "Both Ends" | "Left/Lower End" | "Right/Upper End" }
```

Description

Specifies how to trim the trunks during Pin to Trunk routing. This environment variable can take three values.

- `Both Ends`: trims both ends of the trunk.
- `Left/Lower End`: trims the left or lower ends of the trunk.
- `Right/Upper End`: trims the right or upper ends of the trunk.

Default is `Both Ends`.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Trim Trunk Mode*

Examples

```
envGetVal("p2t" "trimTrunkMode")
envSetVal("p2t" "trimTrunkMode" 'cyclic "Both Ends")
envSetVal("p2t" "trimTrunkMode" 'cyclic "Left/Lower End")
envSetVal("p2t" "trimTrunkMode" 'cyclic "Right/Upper End")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

trunkSetup

```
p2t trunkSetup boolean { t | nil }
```

Description

When set to `t`, enables you to setup layer and width to be used for trunk generation. This means that router can select the layer and width based on other criteria. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Creation – Use*

Examples

```
envGetVal("p2t" "trunkSetup")
envSetVal("p2t" "trunkSetup" 'boolean t)
envSetVal("p2t" "trunkSetup" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Generation Options (ICADVM18.1 EXL Only)

trunkSetupHorTrunkLayer

```
p2t trunkSetupHorTrunkLayer string "layer_names"
```

Description

Specifies the horizontal trunk layer to create horizontal trunks. By default, no layer is set up.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Layer (Horizontal Trunk)*

Examples

```
envGetVal("p2t" "trunkSetupHorTrunkLayer")
envSetVal("p2t" "trunkSetupHorTrunkLayer" 'string "Metal1 Metal2 Metal3 Poly")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Generation Options (ICADVM18.1 EXL Only)

trunkSetupHorTrunkWidth

```
p2t trunkSetupHorTrunkWidth float float_number
```

Description

Specifies the width of the horizontal trunk to be created. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Width (Horizontal Trunk)*

Examples

```
envGetVal("p2t" "trunkSetupHorTrunkWidth")
envSetVal("p2t" "trunkSetupHorTrunkWidth" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Generation Options (ICADVM18.1 EXL Only)

trunkSetupVerTrunkLayer

```
p2t trunkSetupVerTrunkLayer string "layer_names"
```

Description

Specifies the vertical trunk layer to create vertical trunks. By default, no layer is set up.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Layer (Vertical Trunk)*

Examples

```
envGetVal("p2t" "trunkSetupVerTrunkLayer")
envSetVal("p2t" "trunkSetupVerTrunkLayer" 'string "Metal1 Metal2 Metal3 Poly")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Generation Options (ICADVM18.1 EXL Only)

trunkSetupVerTrunkWidth

```
p2t trunkSetupVerTrunkWidth float float_number
```

Description

Specifies the width of the vertical trunk to be created. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Width (Vertical Trunk)*

Examples

```
envGetVal("p2t" "trunkSetupVerTrunkWidth")
envSetVal("p2t" "trunkSetupVerTrunkWidth" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Generation Options (ICADVM18.1 EXL Only)

trunkTaperEdgeStyle

```
p2t trunkTaperEdgeStyle cyclic { "Orthogonal" | "Diagonal" | "Any Angle" }
```

Description

Specifies the edge style when the trunk width is tapered. This environment variable can take three values.

- `Orthogonal`: the edge of the trunk connecting two steps is perpendicular to the steps.
- `Diagonal`: edge of the trunk connecting two steps is slanted at 45 degrees from previous step to the next step.
- `Any Angle`: the stair-step appearance reduces to form a smooth-slanting edge throughout the tapered trunks.

Default is `Orthogonal`.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Taper Edge Style*

Examples

```
envGetVal("p2t" "trunkTaperEdgeStyle")
envSetVal("p2t" "trunkTaperEdgeStyle" 'cyclic "Orthogonal")
envSetVal("p2t" "trunkTaperEdgeStyle" 'cyclic "Diagonal")
envSetVal("p2t" "trunkTaperEdgeStyle" 'cyclic "Any Angle")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

trunkTaperFirstIntervalLength

```
p2t trunkTaperFirstIntervalLength float float_number
```

Description

Specifies the length of first tapered trunk segment. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*First Trunk Segment Length*

Examples

```
envGetVal("p2t" "trunkTaperFirstIntervalLength")
envSetVal("p2t" "trunkTaperFirstIntervalLength" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

trunkTaperIntervalMode

```
p2t trunkTaperIntervalMode cyclic { "Auto" | "Custom" }
```

Description

Specifies where trunk tapering happens. This environment variable can take two values.

- `Auto`: trunk tapering happens at each twig connection.
- `Custom`: trunk tapering happens at the interval specified by you for the `trunkTaperFirstIntervalLength` and `trunkTaperMiddleIntervalLength` environment variables.

Default is `Auto`.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	Interval Mode

Examples

```
envGetVal("p2t" "trunkTaperIntervalMode")
envSetVal("p2t" "trunkTaperIntervalMode" 'cyclic "Auto")
envSetVal("p2t" "trunkTaperIntervalMode" 'cyclic "Custom")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

trunkTaperMiddleIntervalLength

```
p2t trunkTaperMiddleIntervalLength float float_number
```

Description

Specifies the length of all tapered trunk segments excluding the first and the last trunk segment. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Middle Trunk Segment Length*

Examples

```
envGetVal("p2t" "trunkTaperMiddleIntervalLength")
envSetVal("p2t" "trunkTaperMiddleIntervalLength" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

trunkTaperReductionMode

```
p2t trunkTaperReductionMode cyclic { "Auto" | "Specify Percentage" | "Specify
     Value" }
```

Description

Enables you to specify the extent of width reduction on each side of a tapered trunk segment. This environment variable can take three values.

- `Auto`: reduces the width by a value equal to the width of the previous outgoing twig segment.
- `Specify Percentage`: reduces the width of the tapered trunk segment by the percentage specified by the `trunkTaperReductionPercent` environment variable.
- `Specify Value`: reduces the width of the tapered trunk segment by the value specified by the `trunkTaperReductionValue` environment variable.

Default is `Auto`.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Trunk Width Reduction Per Side*

Examples

```
envGetVal("p2t" "trunkTaperReductionMode")
envSetVal("p2t" "trunkTaperReductionMode" 'cyclic "Auto")
envSetVal("p2t" "trunkTaperReductionMode" 'cyclic "Specify Percentage")
envSetVal("p2t" "trunkTaperReductionMode" 'cyclic "Specify Value")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

trunkTaperReductionPercent

```
p2t trunkTaperReductionPercent float float_number
```

Description

Specifies the percentage by which the width of the tapered trunk segment is to be reduced. The reduction per tapered side is a percentage of the original trunk width. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Trunk Width Reduction Per Side (Specify Percentage)*

Examples

```
envGetVal("p2t" "trunkTaperReductionPercent")
envSetVal("p2t" "trunkTaperReductionPercent" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

trunkTaperReductionValue

```
p2t trunkTaperReductionValue float float_number
```

Description

Specifies a value by which the width of the tapered trunk segment is to be reduced. The reduction per tapered side is according to the specified value. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Trunk Width Reduction Per Side (Specify Value)*

Examples

```
envGetVal("p2t" "trunkTaperReductionValue")
envSetVal("p2t" "trunkTaperReductionValue" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

trunkTaperSideMode

```
p2t trunkTaperSideMode cyclic { "Both Sides" | "Left/Lower Side" | "Right/Upper
     Side" }
```

Description

Lets you reduce the width of a trunk from a particular side or from both sides. By default, the trunk is tapered from both sides. This environment variable can take three values.

- `Both Sides`: tapers both sides of the trunk,
- `Left/Lower Side`: tapers the left or lower side of the trunk.
- `Right/Upper Side`: tapers the right or upper side of the trunk.

GUI Equivalent

Command	*Route – Design Setup – Trunk*
Field	*Tapering Side*

Examples

```
envGetVal("p2t" "trunkTaperSideMode")
envSetVal("p2t" "trunkTaperSideMode" 'cyclic "Both Sides")
envSetVal("p2t" "trunkTaperSideMode" 'cyclic "Left/Lower Side")
envSetVal("p2t" "trunkTaperSideMode" 'cyclic "Right/Upper Side")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk Modification Options

Twig Section

connectMultiPinShapes

```
p2t connectMultiPinShapes boolean { t | nil }
```

Description

When set to `t`, the router connects to all the simple shapes and sub-shapes of all the pins of a terminal. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Connect All Pin Shapes to Trunks*

Examples

```
envGetVal("p2t" "connectMultiPinShapes")
envSetVal("p2t" "connectMultiPinShapes" 'boolean t)
envSetVal("p2t" "connectMultiPinShapes" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying General Twig Options

gateTwigLayerMode

```
p2t gateTwigLayerMode cyclic { "Use Layout Context" | "Prefer Routing Direction" |
     "Prefer Pin Layer" | "Specified" }
```

Description

Controls selecting a layer for generating gate twigs. This environment variable can take four values.

- `Use Layout Context`: selects a layer to insert a gate twig from a set of valid routing layers.
- `Prefer Routing Direction`: considers the preferred routing direction of the layer while creating a twig.
- `Prefer Pin Layer`: selects a layer matching the layer of a pin shape to which the twig is to be connected.
- `Specified`: specifies the twig layers to be used by the router for creating horizontal and vertical twigs.

Default is `Use Layout Context`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Gate Twig Layer*

Examples

```
envGetVal("p2t" "gateTwigLayerMode")
envSetVal("p2t" "gateTwigLayerMode" 'cyclic "Use Layout Context")
envSetVal("p2t" "gateTwigLayerMode" 'cyclic "Prefer Routing Direction")
envSetVal("p2t" "gateTwigLayerMode" 'cyclic "Prefer Pin Layer")
envSetVal("p2t" "gateTwigLayerMode" 'cyclic "Specified")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

gateTwigWidthMode

```
p2t gateTwigWidthMode cyclic { "Use Net Constraints" | "Use Pin Width" | "Specified"
     }
```

Description

Controls the width that the router uses to route gate twigs. This environment variable can take three values.

- `Use Net Constraints`: determines the twig width using the width constraints specified for the net.
- `Use Pin Width`: considers the width of the gate twigs equal to the width of the pin.
- `Specified`: specifies the value for the twig width.

Default is `Use Pin Width`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Gate Twig Width*

Examples

```
envGetVal("p2t" "gateTwigWidthMode")
envSetVal("p2t" "gateTwigWidthMode" 'cyclic "Use Net Constraints")
envSetVal("p2t" "gateTwigWidthMode" 'cyclic "Use Pin Width")
envSetVal("p2t" "gateTwigWidthMode" 'cyclic "Specified")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

horGateTwigLayer

```
p2t horGateTwigLayer string "layer_names"
```

Description

Specifies the layer for creating horizontal gate twigs. By default, no layer is set up. It allows the router to select the most appropriate layer.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Horizontal Twig Layer*

Examples

```
envGetVal("p2t" "horGateTwigLayer")
envSetVal("p2t" "horGateTwigLayer" 'string "Metal1 Metal2 Metal3 Poly")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

horGateTwigWidth

```
p2t horGateTwigWidth float float_number
```

Description

Specifies the width value for creating horizontal gate twigs. Default is `0.0`. This means that router determines the width based on other criteria.

Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Horizontal Twig Width*

Examples

```
envGetVal("p2t" "horGateTwigWidth")
envSetVal("p2t" "horGateTwigWidth" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

horNonGateTwigLayer

```
p2t horNonGateTwigLayer string "layer_names"
```

Description

Specifies the layer for creating horizontal non-gate twigs. By default, no layer is set up. It allows the router to select the most appropriate layer.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Horizontal Twig Layer*

Examples

```
envGetVal("p2t" "horNonGateTwigLayer")
envSetVal("p2t" "horNonGateTwigLayer" 'string "Metal1 Metal2 Metal3 Poly")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

horNonGateTwigWidth

```
p2t horNonGateTwigWidth float float_number
```

Description

Specifies the width value for creating horizontal non-gate twigs. Default is `0.0`. This means that router determines the width based on other criteria.

Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Horizontal Twig Width*

Examples

```
envGetVal("p2t" "horNonGateTwigWidth")
envSetVal("p2t" "horNonGateTwigWidth" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

mergeTapsInCloseProximity

```
p2t mergeTapsInCloseProximity boolean { t | nil }
```

Description

Merges the twigs that are close to each other. This means that the router merges adjacent parallel twigs if the distance between them is less than the minimum spacing. When set to nil, each twig is independently connected to the trunk. Default is `t`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Merge Taps in Close Proximity*

Examples

```
envGetVal("p2t" "mergeTapsInCloseProximity")
envSetVal("p2t" "mergeTapsInCloseProximity" 'boolean t)
envSetVal("p2t" "mergeTapsInCloseProximity" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying General Twig Options

nonGateTwigLayerMode

```
p2t nonGateTwigLayerMode cyclic { "Use Layout Context" | "Prefer Routing Direction"
     | "Prefer Pin Layer" | "Specified" }
```

Description

Controls selecting a layer for generating non-gate twigs. This environment variable can take four values.

- `Use Layout Context`: selects a layer to insert a non-gate twig from a set of valid routing layers.
- `Prefer Routing Direction`: considers the preferred routing direction of the layer while creating a non-gate twig.
- `Prefer Pin Layer`: selects a layer matching the layer of a pin shape to which the non-gate twig is to be connected.
- `Specified`: specifies the twig layers to be used by the router for creating horizontal and vertical non-gate twigs.

Default is `Use Layout Context`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Non-Gate Twig Layer*

Examples

```
envGetVal("p2t" "nonGateTwigLayerMode")
envSetVal("p2t" "nonGateTwigLayerMode" 'cyclic "Use Layout Context")
envSetVal("p2t" "nonGateTwigLayerMode" 'cyclic "Prefer Routing Direction")
envSetVal("p2t" "nonGateTwigLayerMode" 'cyclic "Prefer Pin Layer")
envSetVal("p2t" "nonGateTwigLayerMode" 'cyclic "Specified")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

nonGateTwigWidthMode

```
p2t nonGateTwigWidthMode cyclic { "Use Net Constraints" | "Use Pin Width" |
     "Specified" }
```

Description

Controls the width that the router uses to route non-gate twigs. This environment variable can take three values.

- `Use Net Constraints`: determines the non-gate twig width using the width constraints specified for the net.
- `Use Pin Width`: considers the width of the gate twigs equal to the width of the pin.
- `Specified`: specifies the value for the twig width.

Default is `Use Pin Width`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Non-Gate Twig Width*

Examples

```
envGetVal("p2t" "nonGateTwigWidthMode")
envSetVal("p2t" "nonGateTwigWidthMode" 'cyclic "Use Net Constraints")
envSetVal("p2t" "nonGateTwigWidthMode" 'cyclic "Use Pin Width")
envSetVal("p2t" "nonGateTwigWidthMode" 'cyclic "Specified")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

pinStrapLayer

```
p2t pinStrapLayer string "layer_names"
```

Description

Specifies the layer on which strapping should take place during Pin to Trunk routing. By default, no layer is set up. The router picks the layer closest to the pin and whose preferred routing direction matches the direction of the strap.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Strap Layer*

Examples

```
envGetVal("p2t" "pinStrapLayer")
envSetVal("p2t" "pinStrapLayer" 'string "Metal1 Metal2 Metal3 Poly")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Pin Strapping Options

pinStrapLocationPreference

```
p2t pinStrapLocationPreference cyclic { "Closer To Pin" | "Closer To Trunk" }
```

Description

Enables you to select the location where the strap should be created. This environment variable can take two values.

- `Closer to Pin`: creates the strap closer to the pin than the trunk.
- `Closer to Trunk`: creates the strap closer to the trunk than the pin.

Default is `Closer To Pin.`

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Strap Location*

Examples

```
envGetVal("p2t" "pinStrapLocationPreference")
envSetVal("p2t" "pinStrapLocationPreference" 'cyclic "Closer To Pin")
envSetVal("p2t" "pinStrapLocationPreference" 'cyclic "Closer To Trunk")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Pin Strapping Options

pinStrapMaxPinCount

```
p2t pinStrapMaxPinCount int integer_number
```

Description

Specifies the maximum number of pins that can be strapped during Pin to Trunk routing. Default is `2`. Value should be a non-negative integer.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Maximum Pin Count*

Examples

```
envGetVal("p2t" "pinStrapMaxPinCount")
envSetVal("p2t" "pinStrapMaxPinCount" 'int 100)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Pin Strapping Options

pinStrapMaxPinDistance

```
p2t pinStrapMaxPinDistance float float_number
```

Description

Specifies the maximum edge-to-edge distance between two adjacent pins to be strapped. Default is `10` micron. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Maximum Pin Distance*

Examples

```
envGetVal("p2t" "pinStrapMaxPinDistance")
envSetVal("p2t" "pinStrapMaxPinDistance" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Pin Strapping Options

pinStrapping

```
p2t pinStrapping boolean { t | nil }
```

Description

When set to `t`, straps the pins on different instances. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Strap Pins on Different Instances*

Examples

```
envGetVal("p2t" "pinStrapping")
envSetVal("p2t" "pinStrapping" 'boolean t)
envSetVal("p2t" "pinStrapping" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Pin Strapping Options

pinStrapPinLayer

```
p2t pinStrapPinLayer string "layer_names"
```

Description

Specifies the pin layer used to select pins to be strapped together. By default, no layer is set up.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Pin Layer*

Examples

```
envGetVal("p2t" "pinStrapPinLayer")
envSetVal("p2t" "pinStrapPinLayer" 'string "Metal1 Metal2 Metal3 Poly")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Pin Strapping Options

pinStrapWidth

```
p2t pinStrapWidth float float_number
```

Description

Specifies the width of the strap that is created during Pin to Trunk routing. Default is `0.0`, which implies that strap width will match the minimum width of the strap layer. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Strap Width*

Examples

```
envGetVal("p2t" "pinStrapWidth")
envSetVal("p2t" "pinStrapWidth" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Pin Strapping Options

strapGateSourceDrainPins

```
p2t strapGateSourceDrainPins boolean { t | nil }
```

Description

When set to `t`, straps the gate, source, and drain pins of the same instance and same net together before making a connection to the trunk. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Strap Pins on Same Device-Level Instance*

Examples

```
envGetVal("p2t" "strapGateSourceDrainPins")
envSetVal("p2t" "strapGateSourceDrainPins" 'boolean t)
envSetVal("p2t" "strapGateSourceDrainPins" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Pin Strapping Options

tapCoverPinPercent

```
p2t tapCoverPinPercent int integer_number
```

Description

Specifies the percentage of a pin shape to be covered by a twig when a trunk is not over a device. Default is `100`. Value must be a positive integer.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Trunk Offset from Device (Tap) Topmost Vias*

Examples

```
envGetVal("p2t" "tapCoverPinPercent")
envSetVal("p2t" "tapCoverPinPercent" 'int 100)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Via over Pin Coverage Options

tapLowerViasCoverPinPercent

```
p2t tapLowerViasCoverPinPercent int integer_number
```

Description

Specifies the percentage of a pin shape to be covered by the other layers of the tapping vias when the trunk is on a layer that is at least two layers above or below the pin layer. Default is `100`. Value must be a positive integer.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Other Vias*

Examples

```
envGetVal("p2t" "tapLowerViasCoverPinPercent")
envSetVal("p2t" "tapLowerViasCoverPinPercent" 'int 100)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Via over Pin Coverage Options

trunkOverDeviceViaCoverMode

```
p2t trunkOverDeviceViaCoverMode cyclic { "Match Trunk" | "Match Pin" | "Match Pin
     Except Top" }
```

Description

Controls the percentage of source and drain pin coverage with via stacks, when the trunk is entirely over the device. This environment variable can take three values.

- `Match Trunk`: covers the width of the trunk.
- `Match Pin`: covers the width of the pin.
- `Match Pin Except Top`: covers the width of the trunk on the top via layer and covers the width of the pin on the lower via layers.

Default is `Match Trunk`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Trunk Over Device Via (Mode)*

Examples

```
envGetVal("p2t" "trunkOverDeviceViaCoverMode")
envSetVal("p2t" "trunkOverDeviceViaCoverMode" 'cyclic "Match Trunk")
envSetVal("p2t" "trunkOverDeviceViaCoverMode" 'cyclic "Match Pin")
envSetVal("p2t" "trunkOverDeviceViaCoverMode" 'cyclic "Match Pin Except Top")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Via over Pin Coverage Options

trunkOverDeviceViaCoverPercent

```
p2t trunkOverDeviceViaCoverPercent int integer_number
```

Description

Specifies the percentage of a trunk or a pin to be covered with vias when the trunk is entirely over devices. Default is `100`. Value should be a non-negative integer.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Percentage Pin Covered by Vias*

Examples

```
envGetVal("p2t" "trunkOverDeviceViaCoverPercent")
envSetVal("p2t" "trunkOverDeviceViaCoverPercent" 'int 100)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Via over Pin Coverage Options

twigLengthMatching

```
p2t twigLengthMatching boolean { t | nil }
```

Description

When set to `t`, the shorter twigs are extended so that all twigs on the selected nets have the same length. The twig length is matched to the length of the largest twig irrespective of how far the trunk is placed from the pin. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Match Twig Length on Selected Nets*

Examples

```
envGetVal("p2t" "twigLengthMatching")
envSetVal("p2t" "twigLengthMatching" 'boolean t)
envSetVal("p2t" "twigLengthMatching" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying General Twig Options

verGateTwigLayer

```
p2t verGateTwigLayer string "layer_names"
```

Description

Specifies the layer for creating vertical gate twigs. By default, no layer is set up.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Vertical Twig Layer*

Examples

```
envGetVal("p2t" "verGateTwigLayer")
envSetVal("p2t" "verGateTwigLayer" 'string "Metal1 Metal2 Metal3 Poly")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

verGateTwigWidth

```
p2t verGateTwigWidth float float_number
```

Description

Specifies the width value for creating vertical gate twigs. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Vertical Twig Width*

Examples

```
envGetVal("p2t" "verGateTwigWidth")
envSetVal("p2t" "verGateTwigWidth" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

verNonGateTwigLayer

```
p2t verNonGateTwigLayer string "layer_names"
```

Description

Specifies the layer for creating vertical non-gate twigs. By default, no layer is set up.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Vertical Twig Layer*

Examples

```
envGetVal("p2t" "verNonGateTwigLayer")
envSetVal("p2t" "verNonGateTwigLayer" 'string "Metal1 Metal2 Metal3 Poly")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

verNonGateTwigWidth

```
p2t verNonGateTwigWidth float float_number
```

Description

Specifies the width value for creating vertical non-gate twigs. Default is `0.0`. Value must be a positive floating point number.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Vertical Twig Width*

Examples

```
envGetVal("p2t" "verNonGateTwigWidth")
envSetVal("p2t" "verNonGateTwigWidth" 'float 1.0)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying Gate Twig and Non-Gate Twig Options

viaCoverPinWidth

```
p2t viaCoverPinWidth boolean { t | nil }
```

Description

When set to `t`, covers the entire width of a pin with vias. Otherwise, covers pins using the width of the routed wire. Default is `nil`.

GUI Equivalent

Command	*Route – Design Setup – Twig*
Field	*Via Covers Width of Pin*

Examples

```
envGetVal("p2t" "viaCoverPinWidth")
envSetVal("p2t" "viaCoverPinWidth" 'boolean t)
envSetVal("p2t" "viaCoverPinWidth" 'boolean nil)
```

Related Topics

Pin to Trunk Environment Variables

Specifying Twig Options

Specifying General Twig Options

Specifying Gate Twig and Non-Gate Twig Options

Trunk to Trunk Section

connectTrunkType

```
p2t connectTrunkType cyclic { "All Trunks" | "Crossed Trunks Only" }
```

Description

Specifies the connection for the types of trunks. This environment variable can take two values.

- `All Trunks`: connects all crossed and floating trunks.
- `Crossed Trunks Only`: connects only crossed trunks.

Default is `All Trunks`.

GUI Equivalent

Command	*Route – Design Setup – Trunk to Trunk Connection*
Field	*Mode*

Examples

```
envGetVal("p2t" "connectTrunkType")
envSetVal("p2t" "connectTrunkType" 'cyclic "All Trunks")
envSetVal("p2t" "connectTrunkType" 'cyclic "Crossed Trunks Only")
```

Related Topics

Pin to Trunk Environment Variables

Specifying Trunk to Trunk Connections

C

Wire Assistant

The Wire Assistant is a dockable assistant that provides a quick, easy, and single point of access to the commonly used options for all routing purposes: automatic routing, interactive routing, assisted routing, and post-route optimization. The Wire Assistant is available in Layout Suite XL and higher tiers.

The Wire Assistant also provides single-point access to some of the constraint values to be used by the interactive routing for creating and editing wires. Using the constraint look-up feature, the Wire Assistant provides the values to be used and a way to create transient constraints to override some of the values in the fields. You can override values for constraints such as minimum width, minimum spacing, minimum number of cuts, valid layers, and valid vias. You can override the constraint values in the Wire Assistant if the values you specify are:

- Greater than the constraint value in the looked-up constraint group for the applicable minimum rules.
- Within the layer/via range in the looked-up constraint group for *Routing*, *Pin Escape* layers, and *Valid Vias*. See the Override Constraints section of the Wire Assistant.

For more information about constraint group look-up precedence, see Constraint Group Lookup Precedence.

In addition, while routing a net from the *Navigator* assistant, you can choose to use the Wire Assistant to override values. To do this:

1. In the *Navigator* assistant, right-click a net.

   The *Net* context-sensitive menu displays.

2. Choose the *Route With WA Override* command.

   Note: This command supports Specialty Routing.

Unlike the *Process Rule Editor*, the constraints set up by using the Wire Assistant are not persistent, with an exception. The spacing constraint set up by using the Wire Assistant is persistent on the resulting wiring, not net. In addition, the constraints set up by using the Wire Assistant do not impact the *Route – Automatic Routing* and the *Batch Checker*. In the Wire Assistant, overrides created by Automatic Routing are supported. These overrides are

created when Automatic routing is done through the *Automatic* section of Wire Assistant, or from the *Navigator* context-sensitive menu.

This appendix covers the following topics:

Displaying and Hiding the Wire Assistant

To display the Wire assistant, do one of the following:

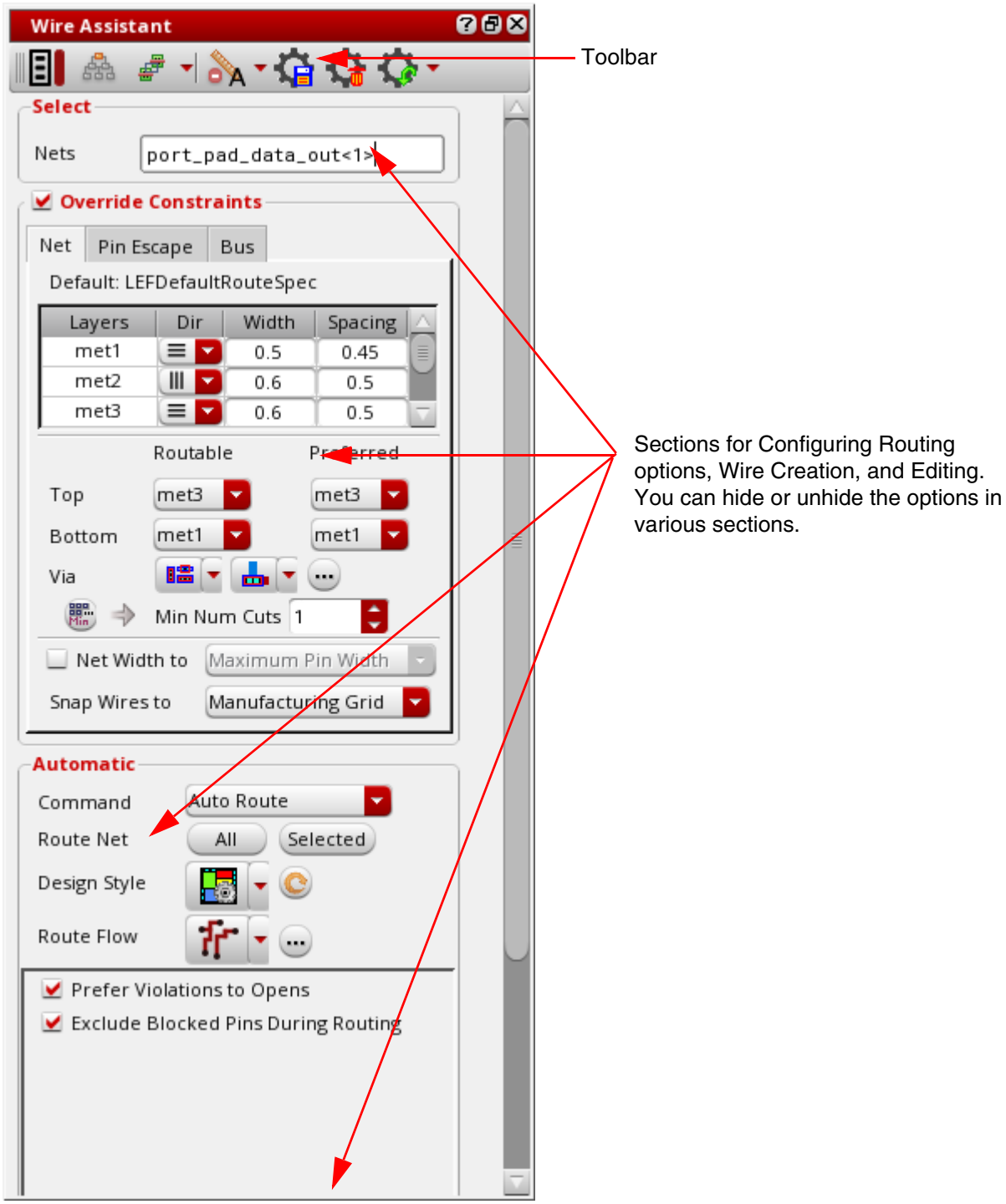
- Choose *Window – Assistants – Wire Assistant*.
- Right-click anywhere in the layout window menu bar and choose *Assistants – Wire Assistant*.

Once selected, Wire Assistant is added as a docked assistant pane within the current layout window. By default, Wire Assistant is positioned at the lower-right area of the session window.

To hide the Wire assistant, do one of the following:

- Click the *Hide* button *(X)* in the Wire Assistant title bar.
- Right-click anywhere in the layout window menu bar and choose *Assistants – Wire Assistant.*

Wire Assistant Graphical User Interface



The Wire Assistant comprises of two components, the toolbar and the different sections for configuring routing settings and configuring settings while creating and editing a wire or a bus. You can add or remove these sections from the Wire Assistant. You can also configure the fields and options to appear in each section, as required. This section covers the following:

- Wire Assistant Title Bar Buttons
- Wire Assistant Toolbar
- Wire Assistant Sections
- Wire Assistant Forms

Wire Assistant Title Bar Buttons

The Wire Assistant title bar has the following buttons:

- *Help*

  Displays information about how to use the Wire Assistant.

- *Float/Dock*

  Docs or undocks the Wire Assistant.

  Note: You can also drag the Wire Assistant to dock or undock it.

- *Hide*

  Hides the Wire Assistant.

Wire Assistant Toolbar

The Wire Assistant toolbar contains the following buttons:

- *Virtuoso Space-based Router and Pin to Trunk Toolbars - On/Off*  icon is similar to the *Wire Assistant Form* icon on the Virtuoso Space-based Router toolbar. This icon controls the display of the *Virtuoso Space-based Router and the Pin To Trunk* toolbars. When the icon is clicked, the toolbars are displayed in the layout window. Otherwise, the toolbars are hidden. By default, the *Virtuoso Space-based Router and Pin To Trunk Toolbar*s icon is turned off.

- *Configure Wire Assistant Form Visibility*  : Lets you view the Wire Assistant Visibility Form.
- *Seed Attributes From a Constraint Group*  : Lets you select a constraint group to use for seeding the values in the Override Constraints section – the width and spacing table, the valid routing layers and vias, the valid taper layers, vias, taper halo, and minimum number of cuts. The constraint groups defined in your technology file appear in the drop-down list.

  Note: The *Seed Attributes From a Constraint Group* is a quick way to fill out the override constraint values in the Override Constraints section. It is not an option to change the default constraint group. To change the default constraint group, use the Layout Editor Options form.
- Reset Override Constraints And VSR Options
- VSR Save Preset
- VSR Delete Preset
- VSR Load Preset

For more information on the VSR Preset toolbar icons, see Working with VSR Presets.

Wire Assistant Sections

The Wire Assistant provides access to various options under the sections mentioned here. The Wire Assistant options that are available in other GUIs or context-sensitive menus, have been indicated. For others, it can be assumed that they are available exclusively in the Wire Assistant.

Note: Options enabled or disabled or parameter values changed using the Wire Assistant are updated in the corresponding GUIs. Similarly, for GUI options that also exist in the Wire Assistant, any updates in the GUI gets reflected in the Wire Assistant settings.

- Select Section
- Override Constraints
- Automatic Section
- Interactive Section

By default, the Wire Assistant displays the *Select*, *Override Constraints*, and *Automatic* sections. The *Automatic* and *Interactive* sections are mutually exclusive to each other and only one of them is expanded at any particular time. The *Automatic* section is expanded by default when the Wire Assistant is first launched, or after escaping from one of the interactive

commands. The *Interactive* section is enabled in the Wire Assistant when you start the *Create Bus, Create Wire, Point to Point*, and *Guided Routing* commands.

Select Section

The following option is available in the *Select* section of the Wire Assistant.

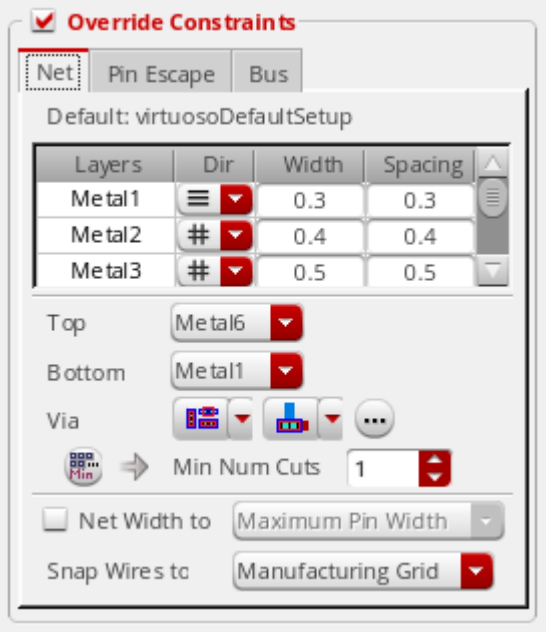
Nets

Lets you specify one or more net names for the current wire. The created wire is added to the specified net. A new net is created if the one specified does not already exist. A bus name can also be provided in the *Net* field. For example, `net1<0:15>`.

The net names in the *Navigator* assistant and the Wire Assistant are synchronized automatically. This means that if there are selected net in the *Navigator* assistant, they are automatically imported to the *Nets* field of the Wire Assistant.

Note: The license token usage is based on the number of nets that are to be routed and not the absolute number of nets in the design.

Override Constraints



The *Override Constraints* section is displayed in the Wire Assistant by default. This section consists of the following three tabs:

- Net
- Pin Escape
- Bus

When the *Override Constraints* check box is selected, the automatic and interactive routing commands, such as Finish and P2P routing, considers the overriding values of width, spacing, and via overrides into the Wire Assistant and accordingly routes the design. However, when the *Override Constraints* check box is deselected, the automatic and interactive routing commands do not consider the overriding values of width, spacing, and via overrides into the Wire Assistant and the routing is performed based on the default values.

Net

The constraint group used to seed the width and spacing table is mentioned above it. The *Default* constraint group is derived from the *Wire* setting in the *Layout Editor Options* form. For more information about the form, see *Wire Editing* in the Virtuoso Layout Suite documentation.

You can change the constraint group from which to seed the table by selecting a value from the *Seed Attributes From a Constraint Group* list on the *Wire Assistant* toolbar. If you select a different value from the *Seed Attributes From a Constraint Group* list, the *Default* label changes to *Seeded* and the constraint group name is changed to the one selected. For example, `Seeded: LefDefaultRouteSpec`. When you override a value in the *Override Constraints* section, the constraint group name is updated to default (`Default: VirtuosoDefaultSetup`), which is the same as the one selected in the *Default Wire Constraint Group field* in the *Layout Editor Options* form.

By using weWADefaultSeedCGName, you can seed a default constraint group on the Wire Assistant.

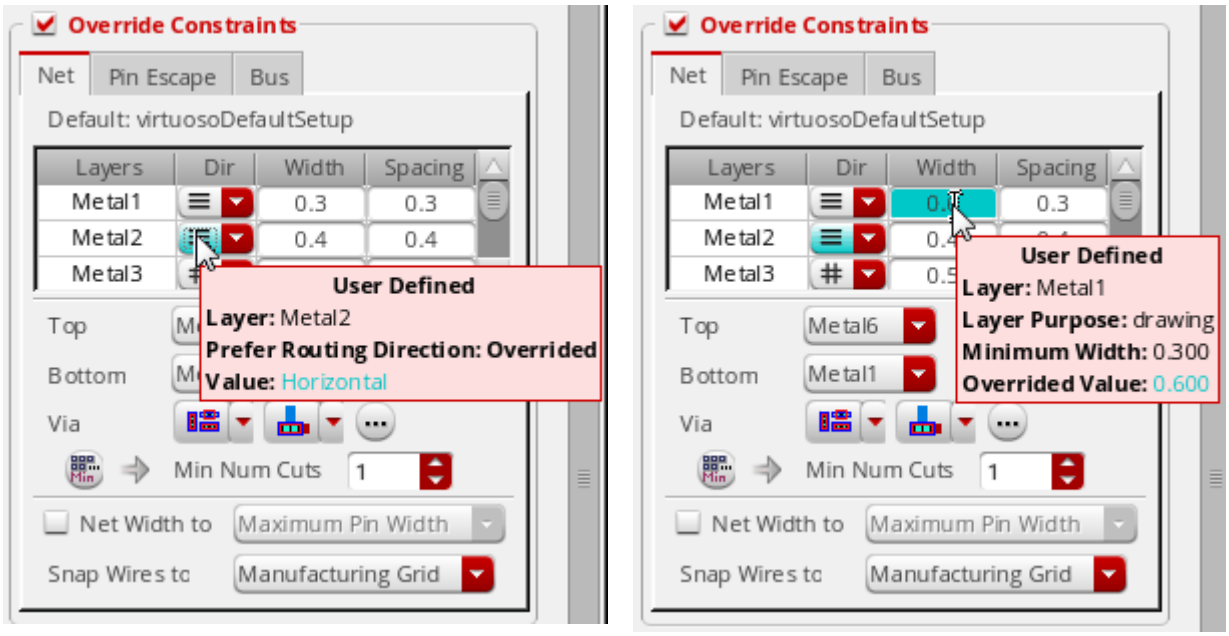
Some of the features of the *Net* tab include:

- The constraint group selected from the *Seed Attributes From a Constraint Group* list does not change the application default constraint group setting in the *Wire* section of the *Layout Editor Options* form.
- Specifying routing layers only in the *Net* tab does not guarantee that the router uses only those layers. This is because the *Pin Escape* tab in the *Overrides Constraint* section

also has option of selecting the valid layer. The router can connect to the pin using the layers in the *Pin Escape* tab. However, if you want to limit all routing to a particular set of layers, layers should be specified both in the *Net* tab as well as the *Pin Escape* tab.

- While overriding the constraint values in the Width and Spacing table, ensure that the values specified for the *Width*, *Spacing*, and *Min Num Cuts* are greater than or equal to the values for the corresponding minimum rules in the looked-up constraint group. In addition, the valid routing layers and valid vias should be a subset of the looked-up constraint group for *Routing* and *Pin Escape* layers and *Valid Vias*. Ensure that the override values do not create artwork that will immediately violate an existing looked-up constraint value.
- Modification of the values in the *Override Constraints* section does not change the content of the constraint group in the OpenAccess database.
- If you override any value in the *Override Constraints* section, the change is indicated by colored fields in the table and by color bars next to the other updated fields. The figures below illustrate some examples where the constraint values are overridden in the table and for other fields.

  You can place the pointer on the colored fields in the table, or on a color bar to view the override information in a tooltip, as shown in the figures below.



Width and Spacing Table allows you to override the preferred routing direction, the width, and the spacing for the layers to be used while creating wires. You can override the current preferred routing direction for each layer by selecting a different value from the *Dir* list. The routing directions supported by the Wire Assistant include *Horizontal*

, *Vertical* , and *None* . Any update to the routing direction for a layer in the *Layers* assistant is automatically updated in the *Dir* column for that layer in the Wire Assistant. However, when you update the routing direction for a layer in the Wire Assistant, the Layers assistant is not updated. This is because the Wire Assistant does not update the constraints or attributes in the technology file. The updated attribute is saved temporarily.

You can click in the *Width* and *Spacing* fields to edit them. If a net has been specified in the *Net Name* field, then the net width and spacing overrides, if any, are visible in the table. If you change any width value, then the *Use Width* option in the Create Wire section automatically sets to *Last Specified*, if it is not done already.

The number of digits after the decimal point allowed in both *Width and Spacing* table and the *Bus Bit* table is depended upon the *Manufacturing Grid* resolution in the technology file. This means that if the manufacturing grid is `0.0005`, then both the tables allow you to specify four digits only after the decimal point. Similarly, when the manufacturing grid is `0.00025`, then only five digits can be specified after the decimal point.

If you select a non-drawing, voltage-aware layer purpose to create a wire, the layer purpose is represented in the Layers column in the `<layer:purpose_abbreviation>` format, as in `metal1:dr4`.

Note: The wire editor allows the creation of bus wires even if the terminals or instance terminals distance is less than the override spacing, regardless of the current DRD checking mode. After creating the bus, you should run the *Batch Checker* on the result. The *Batch Checker* will flag error on the pins or instPins that do not have enough spacing for the override spacing value. For more information about the *Batch Checker*, see Batch Checking.

- Top Layer
  - **Routable**

    Enables you to select the top valid routing layer limits and overrides the `validLayers` setting in the selected constraint group. The specified layer range usually intersects the *Pin Escape* layer range.

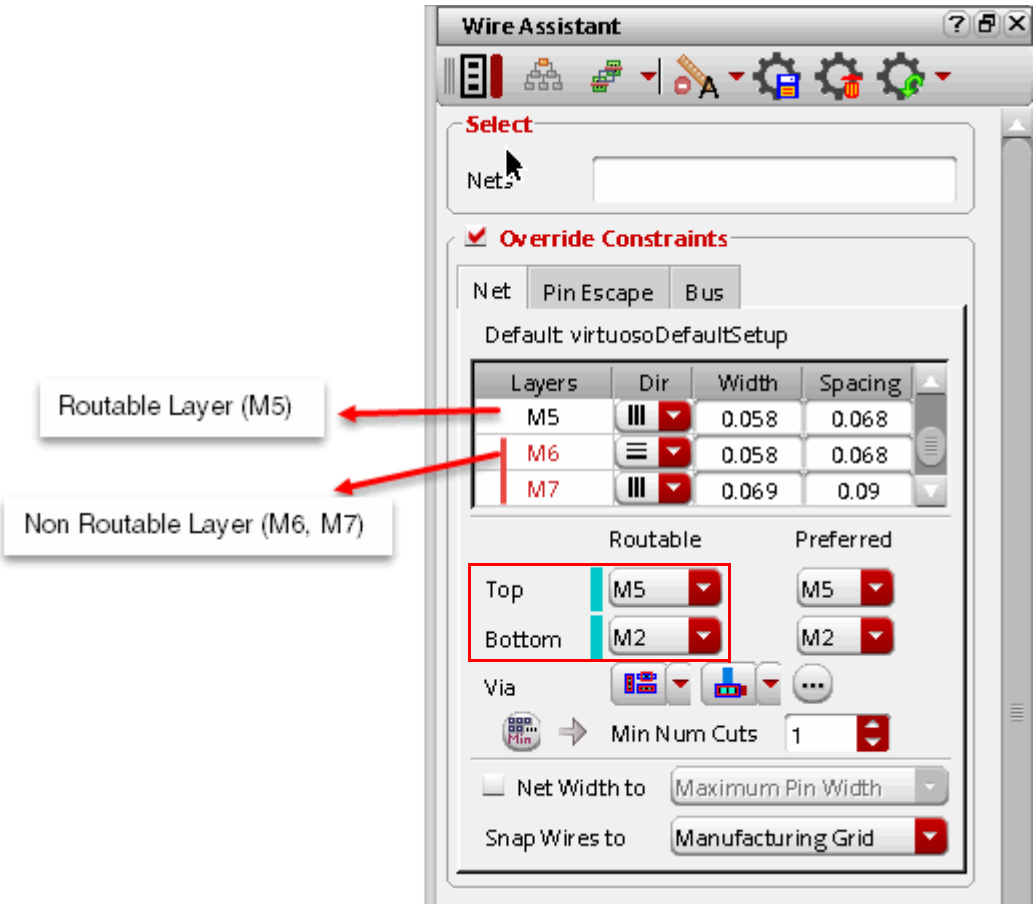
    Note: The override layer range restricts error fixing and optimization to be performed only on a subset of layers.

    When the top layer is selected from the *Routable* drop-down list, the layer name in the *Layers* column of the width and spacing table displays different font color for routable and non routable layers.

Font Color/Style	Layer Type

Red and Normal	Non routable layers
Black and Normal	Routable layers

The following figure shows only the specified *Routable* layer range.

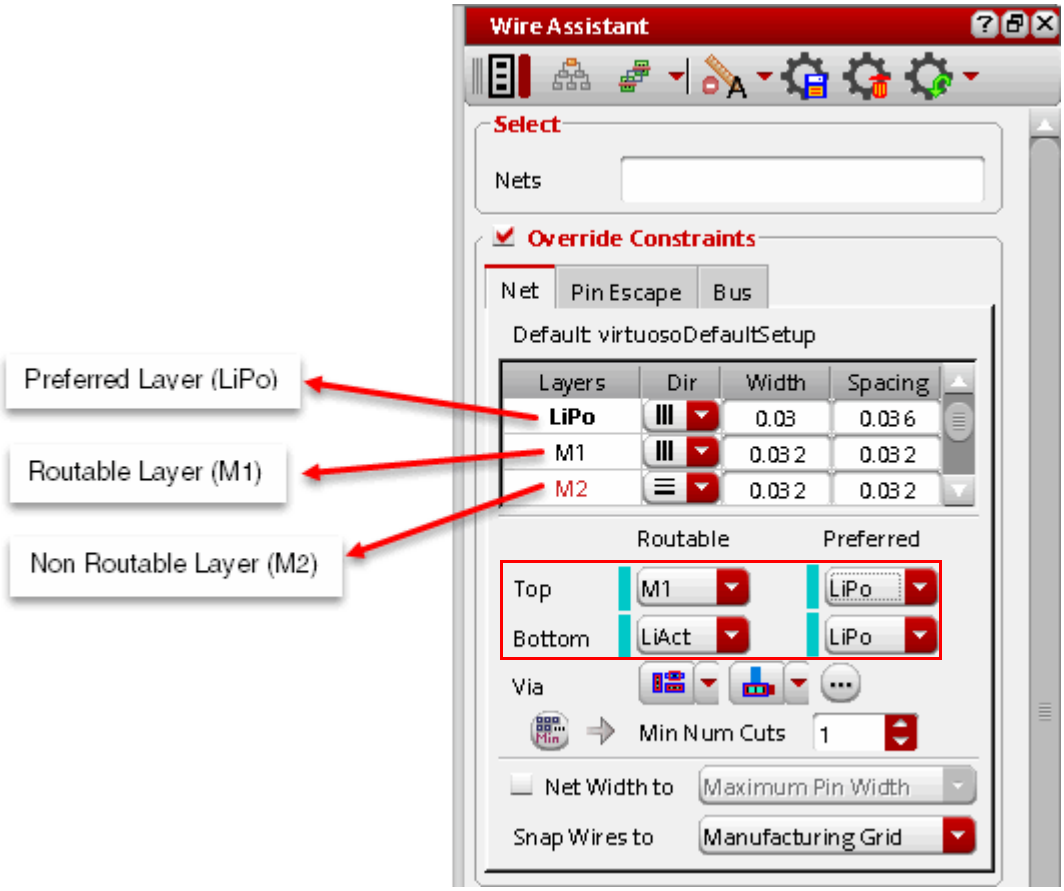


- **Preferred**

  Enables you to select the top preferred layer for further limiting the routing layers. The router can route the maximum possible wires on the selected preferred layers. When the router is unable to route on the preferred layers, the layers outside the *Preferred* layer range but within the *Routable* layer range are used for routing.

  Note: If there are top and bottom *Routable* layers, then, by default, the top and bottom *Preferred* layer range is the same as the *Routable* layer range.

When the top layer is selected from the *Preferred* drop-down list, the preferred layers appear bold faced in the *Layers* column of the width and spacing tables, as shown in the following figure.



- Bottom Layer
  - **Routable**

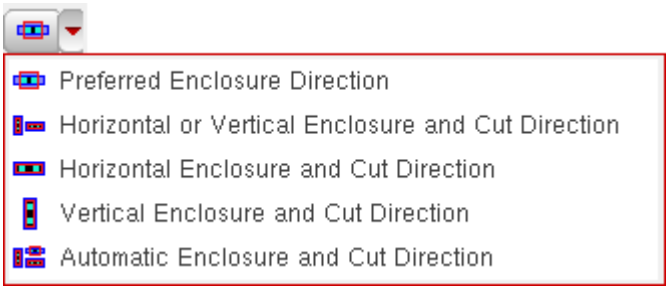
    Enables you to select the bottom valid routing layer limits and overrides the `validLayers` setting in the selected constraint group. For more information, see Top Layer.
  - **Preferred**

    Enables you to select the bottom preferred layer for further limiting the routing layers. For more information, see Top Layer.
- Via

  Consists of three common via controls. Using these via controls, you can specify the common settings for all vias in a layout design.

❑ Enclosure and Cut Direction

This is the first drop-down list next to the *Via* field. You can specify the enclosure and cut direction settings only to vias that are not fully enclosed. From the *Enclosure and cut Direction* drop- down list, you can select one of the following options.



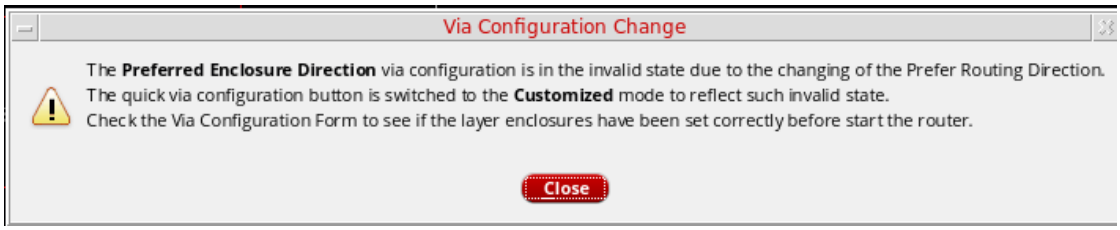
Note: A fully enclosed via is always preferred to a non-fully enclosed via.

❍ Preferred Enclosure Direction

When selected, the metal layers are crossed. The via enclosures created on metal layers follow the preferred routing direction and no cut direction is implied.

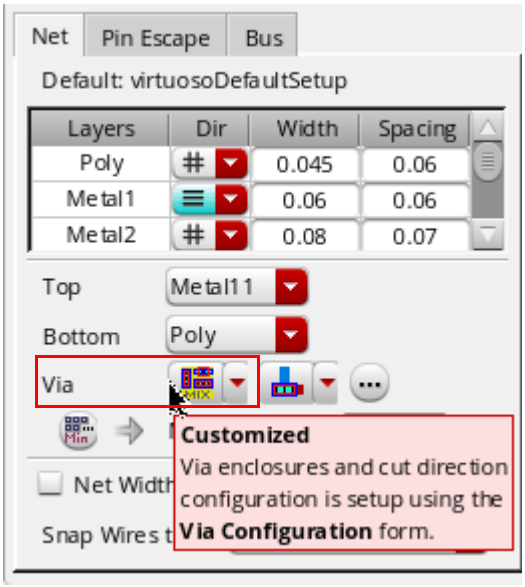
If you want to set both routing direction and via configuration to *Preferred Enclosure Direction*, it is recommended to set the routing direction from the *Dir* list of the Width and Spacing table before specifying the via configuration to *Preferred Enclosure Direction*. This is because the *Preferred Enclosure Direction* via configuration is based on the current routing direction of each layer.

However, if you have set the via configuration to *Preferred Enclosure Direction* and then change the routing direction of any layer in the width and spacing table, then the warning message is displayed, as shown in the following figure.



The *Preferred Enclosure Direction* via configuration is in the invalid state due to the change in the routing direction. To reflect the invalid state, the via

configuration and the ellipses button next to the *Via* field changes to the *Customized* mode, as shown in the following figure.



Therefore, check the Via Configuration form to see if the layer enclosures have been appropriately specified before you start routing.

- **Horizontal or Vertical Enclosure and Cut Direction**

  When selected, the metal layers are inline. The via enclosures created on metal layers and the cut direction are either horizontal or vertical.

- **Horizontal Enclosure and Cut Direction**

  When selected, the metal layers are inline. The via enclosures created on metal layers and the cut direction are horizontal.

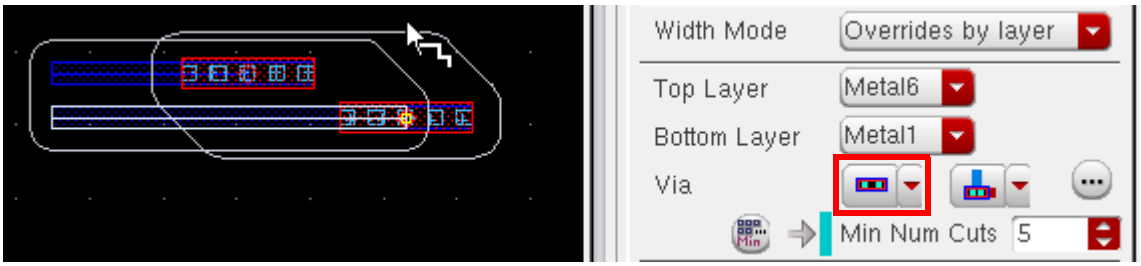
- **Vertical Enclosure and Cut Direction**

  When selected, the metal layers are inline. The via enclosures created on metal layers and the cut direction are vertical.

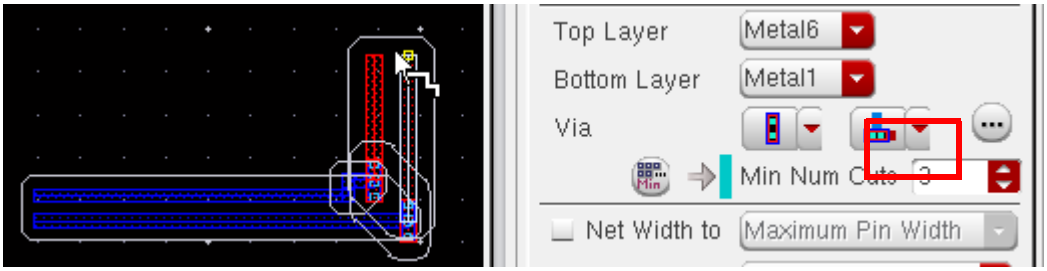
- **Automatic Enclosure and Cut Direction**

  When selected, the metal layers are either crossed or inline. Depending on how the via is created on the metal layers, the via enclosures and the cut direction are either horizontal or vertical. For example, When Metal3-Metal4 inline is ON, M3 enclosure direction being horizontal and M4 enclosure direction being vertical, there is a conflict. When router finds a conflict in enclosure direction on Metal layers, router chooses any via with cut direction horizontal-horizontal, vertical-vertical, or cross.

The following figure shows an example of how a via is placed when the *Horizontal Enclosure and Cut Direction* and *Vertical Enclosure and Cut Direction* options are selected.



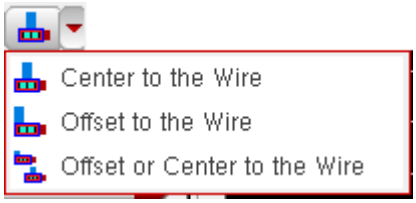
Via placement when Horizontal Enclosure and Cut Direction option is selected



Via placement when Vertical Enclosure and Cut Direction option is selected

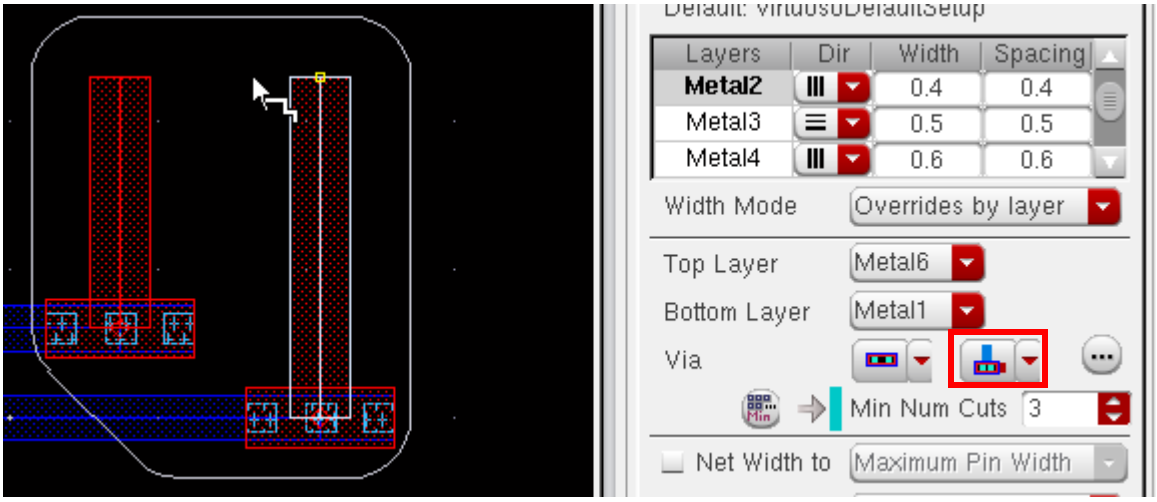
- **Edge Alignment**

  This is a common via control used to specify the via alignment to the wire.



  - **Center to the Wire**

When selected, the origin of the via created on a wire is on the center of the wire. The following figure shows an example where a via created on a wire is aligned to the center to the wire.



- **Offset to the Wire**

  When selected, the via created on a wire is aligned to the edges of the wire.

- **Offset or Center to the Wire**

  When selected, the via created on a wire is either aligned to the edge of the wire or center of the wire. In this mode, the offset vias are preferred by the router but the center is used.

- **Ellipses button** 

  When clicked, displays the Via Configuration Form. Using the Via Configuration Form, you can configure the settings for each individual via that exists in the layout design.

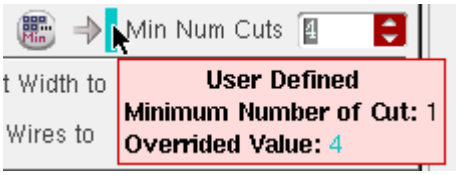
- **Number of Via Cuts**

  The minimum number of cuts may be overridden by specifying either a *Min Num Cuts* value or the number of cuts in *Row* and *Col* fields, respectively. By default, the *Min Num Cuts* field is displayed. To toggle between the minimum number of cuts field and number of cuts by rows and columns field, click the  or  button respectively.

  - **Min Num Cuts**

    By specifying the via cuts as minimum number of cuts, you can override the `minNumCut` rule value for creating and editing wires. If the override value is less

than either the technology rule or the current constraint group on the net or on the design, then the value specified in the Wire Assistant is reset to the minimum value. The figure below displays the override information in a tooltip for the minimum number of cuts. You can place the pointer on the colored field or on a color bar to view the tooltip information.



Note: If you want to insert double cut vias where ever possible, which is considered as a 'soft' constraint, select the *Attempt to Use Double Cut Vias* option from the Via Options form. This option allows pin escape and routing with double cut vias. However, if you want to insert double cut vias everywhere, which is considered as a 'hard' constraint, then use either a `minNumCut` constraint that could be added from the constraint manager or the `Min Num Cuts` override setting from the Wire Assistant.

- **Rows and Columns**

  By specifying the via cuts as the number of rows or the number of columns of via cuts you can override the `minNumCut` rule with a more explicit configuration for creating and editing wires. The increase and decrease in the number of via row and column is dependent on the minimum number of via cuts.

  ```
  number of via rows * number of via columns >= minimum number of via cuts.
  ```

  This condition should always be satisfied. In case the condition is not satisfied, `minNumCut` violation is created by the interactive routing commands, such as *Create Wire*. To avoid the `minNumCut` violation, the Wire Assistant makes adjustment to the number of via rows and columns whenever it is required to satisfy the condition. For example, if minimum number of via cut is set to 9, then the row and column automatically adjusts to `1` and `9`. The Wire Assistant makes adjustment on number of via rows and columns to satisfy the condition. Else, no adjustments are made.

  When the `minNumCut` value is increased, the column and row value is automatically adjusted to meet the constraint. However, if `minNumCut` value is reduced, the existing larger values for column and row still satisfy the rule and therefore no automatic adjustment is made to decrease the values. However, the number of rows and columns can be manually reduced to satisfy the `minNumCut` constraint. When either *Columns* or *Rows* is updated to a very small number that does not satisfy the existing `minNumCuts` setting, the

opposite (for example, *Rows* or *Columns*, respectively), is changed to a higher number in order to satisfy the `minNumCuts` rule.

Note: *Row* and *Col* fields are only supported by the interactive routing commands, such as *Create Wire* and *Create Bus*. Therefore, these fields are disabled when the Wire Assistant is in an auto-routing mode.

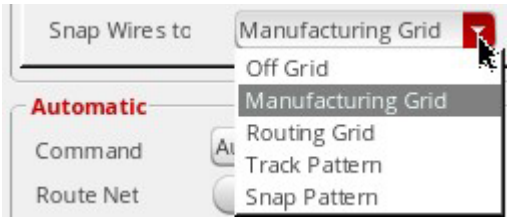
■ Net Width To

When the *Net Width to* checkbox is selected, the *Net Width to* drop-down is enabled. You can specify the pin width per net to route the non-taper section of the net. The net width is determined by the value of this field. You can set the net width to one of the following values.

- **Maximum Pin Width** uses the largest pin width.
- **Average Pin Width** uses the average pin width.
- **Minimum Pin Width** uses the smallest pin width.

■ Snap Wires To

Controls the snapping of wires to a grid.



- **Off Grid**

  Specify this option when the pins are off the grid.

- **Manufacturing Grid**

  If selected, the wires are snapped to the *Manufacturing Grid* when the routing grid and track pattern are not defined. This is also the default option.

- **Routing Grid**

If selected, the segments other than the ones directly connecting to the off-grid pins are snapped to the routing grid.

- **Track Pattern**

  Controls the cursor snapping to track the grid and forces routing on the grid where pins are on grid. The track patterns are saved on the design and so each design can have its own set of non-uniform track patterns.

- **Snap Pattern**

  If selected, enables the snapping of wires to the nearest snap pattern track.

  The *Snap Pattern* does not restrict the routing layer. It only uses track patterns and WSPs that exists in the design. For example, it is possible that in some designs only lower layers have WSPs while upper layers do not. In such cases, routing is performed on all routing layers. The top and bottom layer range is used, which maps to a valid routing layers constraint, to control the routing layers. In case, you do not want to route a particular layer then either remove it from your valid routing layers constraint or adjust the bottom or the top routing layer.

  When either *Snap Pattern* or *Track Pattern* is selected from the *Snap Wires To* drop-down list, the Auto Route flow considers the patterns and track patterns in the design. The WSPs and track patterns can co-exist in a design, However, if there is a conflict between a WSP and a track pattern, WSP takes precedence and track pattern is ignored. Also, warning messages appear in some specific cases.

  - A warning is issued if WSP and track pattern co-exist on the same layer. In this case, routing proceeds further until it is completed.
  - A warning is issued if WSP and track pattern is missing in all layers. In this case, routing stops.

Note: The wire editor follows the union of both the *Top* and *Bottom layers* in the *Net* and *Pin Escape* tabs in the *Override Constraints* section of the Wire Assistant. For example, if the *Top* and *Bottom* layers in the *Net* tab are set to `Metal4` and `Metal1`, respectively, and the *Top* and *Bottom* layers in the *Pin Escape* tab are set to `Metal5` and `Metal2`, respectively, the routing interval assumed by the wire editor is between `Metal5` and `Metal1`.

Pin Escape

The *Pin Escape* tab appears in the *Override Constraints* section of the Wire Assistant. This tab enables you to select the *Top* and *Bottom* layer limits and overrides the `validLayers` setting in the `virtuosoDefaultTaper` constraint group. If a `virtuosoDefaultTaper` constraint group does not already exist, it is created based on the specified *Layers*, *Vias*, and *Halo* values, in the wire default constraint group.

The following options are available in the *Pin Escape* tab of the *Override Constraints* section.

- **Mode**

  There are three modes:

  - **Allow Taper in Halo**

    Enables tapering only when needed within a halo area that either defaults to 10 tracks or is the value specified in the *Halo Value* field. This option is selected by default.

  - **Use Net Rules Only**

    Enables pin tapering to avoid `minEdge` violations.

    The router tries to satisfy all the net constraints, In order to maintain the net constraints, the router ensures that no violations are created and specifically the `minStepEdgeLength` constraint is not violated. Therefore, when Pin Escape mode

is set to *Use Net Rules Only*, it tapers wire to pin Width till first via in order to avoid `minStepEdgeLength` constraints violations.

Note: The automatic pin tapering for avoiding `minEdge` violation is still in effect even when you select the *Use Net Rules Only* option.

- **Use Pin Width to First Via**

  Sets the `taperToFirstVia` constraint to true for the created taper specification. This results in tapering the width from the pin to the first layer change for the specified net.

- **Halo**

  Allows you to specify the positive float value to override the `taperHalo` value of the `virtuosoDefaultTaper` constraint group. The default value is `-1`.

- **Top**

  Enables you to select the top layer limits and overrides the `validLayers` setting in the `virtuosoDefaultTaper` constraint group.

- **Bottom**

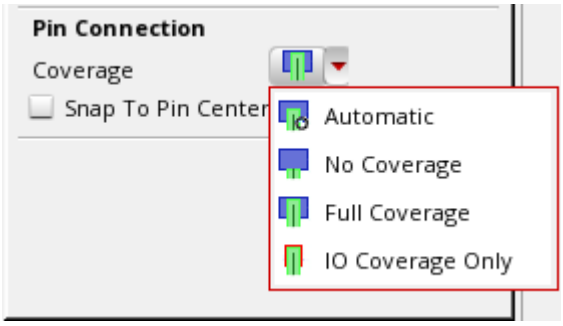
  Enables you to select the bottom layer limits and overrides the `validLayers` setting in the `virtuosoDefaultTaper` constraint group.

- **Via**

  When clicked, displays the *Pin Escape* tab of the Via Configuration form. You can use the options in the *Pin Escape* tab to select the via to be used when tapering.

Pin Connection

- **Coverage**



  - **Automatic**

When wire extension is not truncated then router chooses extension so that based on the pin shape, the extension never goes beyond the pin.

- **No Coverage**

  When the wire extension just touches the pin and does not overlap the pin.

- **Full Coverage**

  The wire completely overlaps the pin along the direction of the wire.

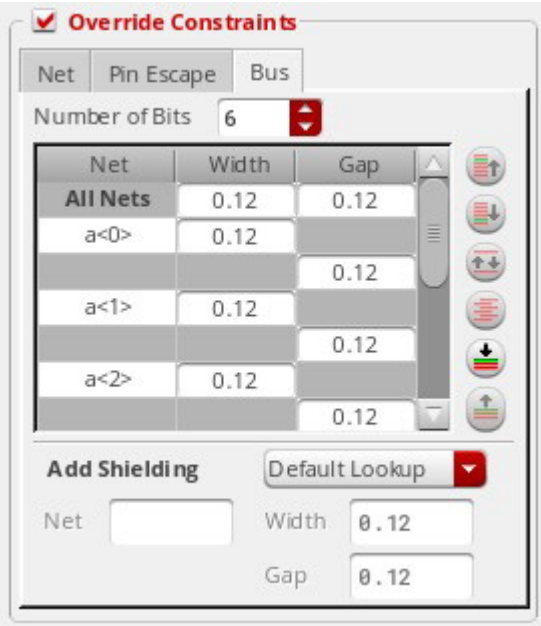
- **IO Coverage Only**

  The wire completely overlaps only the IO pins along the direction of the wire. However, it leaves connections to the instance pins unchanged.

- **Snap to Pin Center** in the Automatic coverage mode snaps wires to the center of rectangular pins or pins that are created as polygons and have rectangular shapes with half extend. *Snap to Pin Center*, in any other coverage mode, results in the coverage mode taking precedence. This option applies to wires created using *Create Wire*, *Point to Point*, and *Finish Wire* commands and is available in the Create Wire Form and Create Bus Form.

  Note: The *Adjust Pin Coverage* option in the *Optimize* section of the Wire Assistant considers all the pin connection coverage modes.

Bus



Using the *Bus* tab in the *Override Constraints* section, you can change the bus bits configuration and enable the Add Shielding net feature. You can perform the following tasks using the options in the *Bus* tab.

- specify the width and spacing of individual bus bit
- rearrange the ordering of bus bit
- add parallel shielding even if specified nets do not have the shielding constraint.interleave bits of different buses

The options in the *Bus* tab are enabled when you start the *Bus* command from the *Create – Wiring* menu and the *Bus* option is selected in the Wire Assistant Visibility form. This section is enabled as long as the *Create Bus* command is running.

The options available in the *Bus* tab are as follows:

- **Number of Bits** enables you to specify the number of wires in an unassigned bus. The default value is `2`.
- **Bus Bit table** is automatically populated with the nets that are selected in Navigator Assistant.

  Note: You can have duplicated nets in the Bus Bit table.

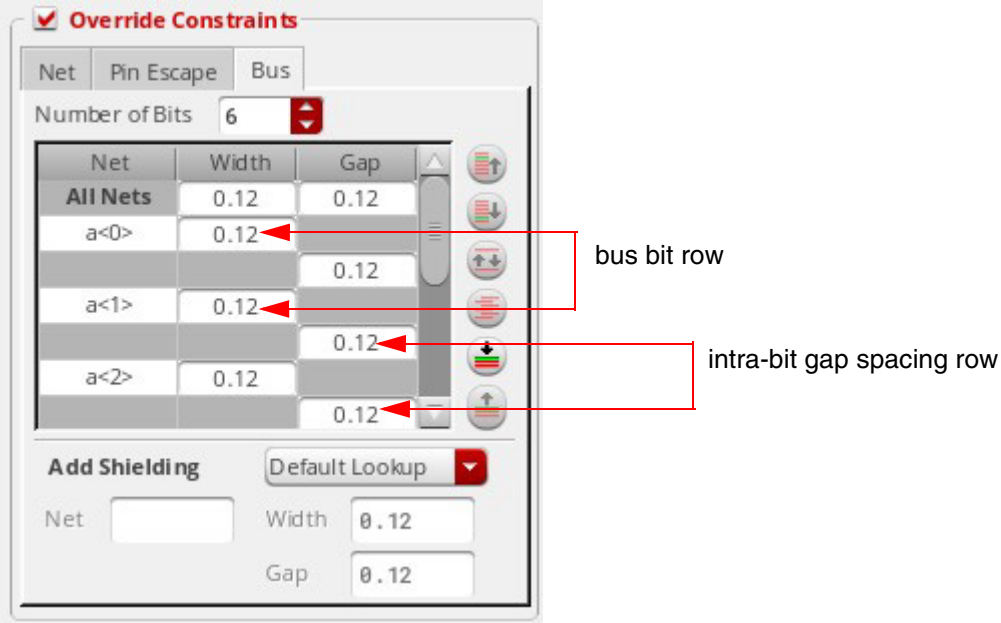
  - The first row in the Bus Bit table is called *All Nets*. Using this row, you can override the width and spacing values for all the bus bits in the form. A warning message is displayed in the CIW if the value specified is less than the default look-up value.
  - The rows in the Bus Bit table are the bus bit row and the intra-bit gap spacing row.

    The bus bit row has two columns.

    - **Net** is the name of the net for the bus bit. The name of bus bit can be a single net name or the bit-range names. For more information, refer to Specifying Net Names for Bus Bit.
    - **Width** is the minimum width of the bus bit. The default value of the bus bit width is the constraint look-up minimum width.

    The intra-bit gap spacing row has one column.

- **Gap** is the minimum spacing between two adjacent bus bits. The default value of the intra bit spacing is the constraint look-up minimum spacing.



Add Shielding

The *Add Shielding* mode enables you to add shielding net whether or not bus bit nets have shielding constraint assigned on the OpenAccess database. If the bus bit nets do not exist on the OpenAccess database, the *Create Bus* command automatically assigns the shielding constraint to those non-existing bus bit nets after the bus is created. The four *Add Shielding Net modes* are as follows

- **None** no shielding net is added during bus creation if the bus bit nets has the shielding constraint on the database. When the *Add Shielding Net* is set to *None*, the three shielding fields *Net*, *Width*, and *Gap* are disabled.
- **Default Lookup** if the bus bit nets already have shielding constraint on the OpenAccess database, the *Create Bus* command automatically uses the existing constraint. If the bus bit nets does not have shielding constraint on the OpenAccess database, then the create bus does not create any shielding wires.
- **Around Bus** the shielding net is added on the periphery of the bus.
- **Every Bit** the shielding net is added between the bus bits.

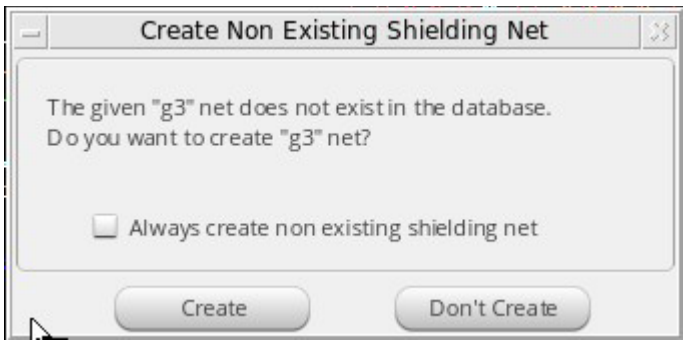
  Note: When the mode is set to either *Around Bus* or *Every Bit*, specify the shielding net, shielding net width, and shielding gap in the *Net*, *Width*, and *Gap* fields, respectively.

Environment Variable: weBusBitShieldingOption

Regardless of the *Add Shielding* mode that is set, if the bit net already has shielding constraint assigned, the *Create Bus* command does not override the existing shielding constraints.

- **Net**

  Specifies a non-existing net name. When a non-existing net name is specified in the *Net* field, a dialog box, as shown below, appears confirming whether or not you want to create the shielding net with the specified name in the OpenAccess database.



  To create a shielding net with the specified name, click *Create*. The net with the specified name is created in the memory when the user digitizes the first point. The shielded net is saved to the OpenAccess database once the bus is created. If the *Create Bus* command is revoked, then the net is not saved in the OpenAccess database.

  If you do not want to create a non-existing shielding net, click *Don't Create*. In this case the shielding net is not created and a message is displayed in the CIW stating that the specified net does not exist in the OpenAccess database and asks you to specify a net that exists in the OpenAccess database.

  In order to automatically create a new shielding net, select the *Always create non existing shielding net* checkbox. and then click *Create*. Next time when you specify a non-existing shielding net in the *Net* field, the *Create Non Existing Shielding Net* dialog box is not displayed and the non-existing shielding net is created in the memory automatically.

  Environment Variable: weBusBitShieldingNetName

- **Width**,

  The default value is -1. The value specified in the *Width* field should not be less than the constraint minimum width value.

  Environment variable: weBusBitShieldingWidth

- **Gap**

  The default value is -1. The value specified in the *Gap* field should not be less than the constraint minimum spacing value.

  Environment variable: weBusBitShieldingGap

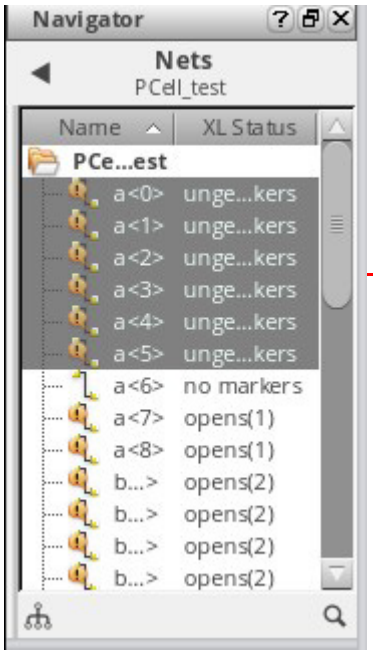
Important

The value specified in both *Width* and *Gap* fields should not be less than the constraint minimum width and spacing values, respectively. If the values of those fields are not valid, the form automatically reverts to its previous valid value.

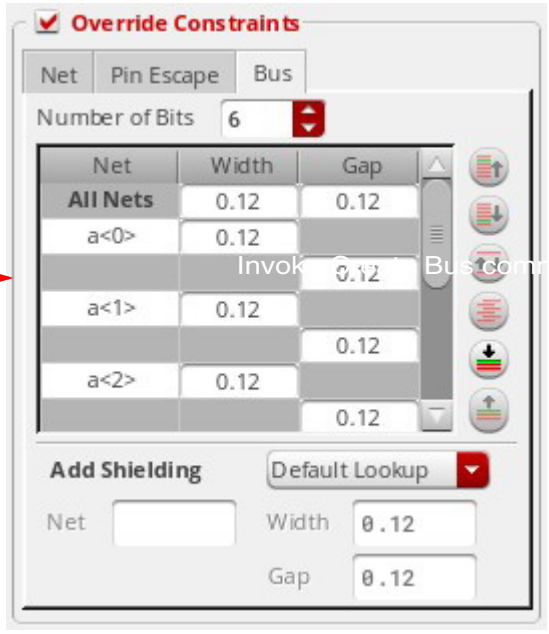
The features of the *Bus* tab include:

- Synchronizing from the Navigator Assistant

  The nets selected in the Navigator Assistant are automatically synchronized and imported to the *Select* section of the Wire Assistant. The imported nets are also displayed in the Bus Bit table when the Create Bus command is started from the *Create – Wiring* menu, and the *Bus* option is selected in the Wire Assistant Visibility form. *This* helps you create buses in free space. The following figure illustrates the selected nets in Navigator Assistant and the bus bit table in the *Create Bus* section of Wire Assistant.



Selected nets in Navigator Assistant



Selected Nets are automatically updated in the Bus Bit table when Create Bus command is invoked

Note: The corresponding bus-bit width and intra-bit gap spacing cells display the default

constraint look-up values.

- Non-shielded Bus Bit Overrides

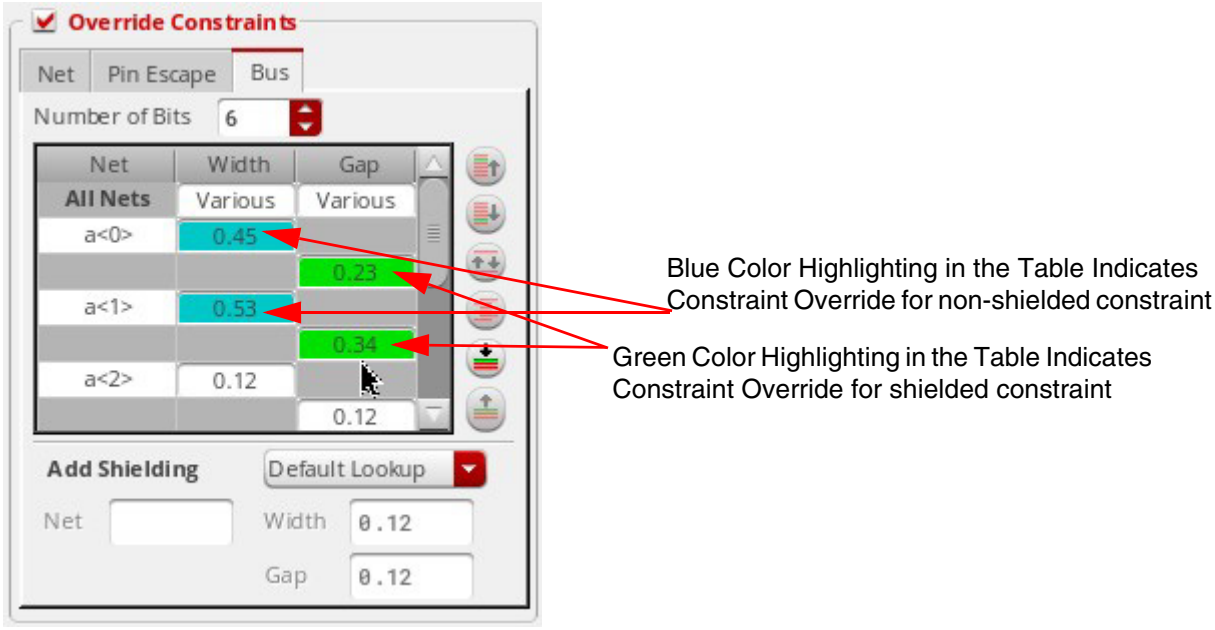
  While overriding the constraint values in the Width and Spacing table, ensure that the values specified for *Width* and *Spacing* are greater than or equal to the values for the corresponding minimum rules in the looked-up constraint group. Ensure that the override values do not create artwork that will immediately violate an existing looked-up constraint value.

- Sync from the *Wire Assistant* and current editing layer

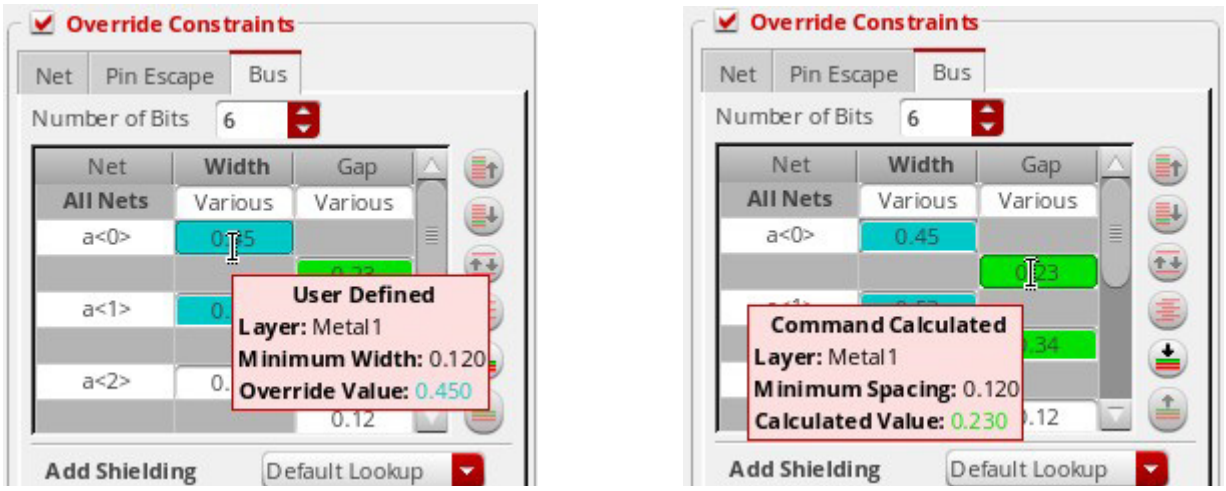
  Whenever there is a change in the current editing layer or a change in the Wire Assistant override values of the current editing layer, the width and spacing value in the bus bit table are automatically updated as per the new changed value. The width and spacing cell background and tooltip are also updated accordingly.

- Background Coloring and Tooltip

  Similar to the Width and Spacing table in the *Bus* tab of the *Override Constraints* section, if you override any value, the change is indicated by colored fields in the table. The figures below illustrate some examples where the width and spacing values are overridden in the table.



You can place the pointer on the colored fields in the table to view the override information in a tooltip, as shown in the figure below.



- Validation of the bus bit table

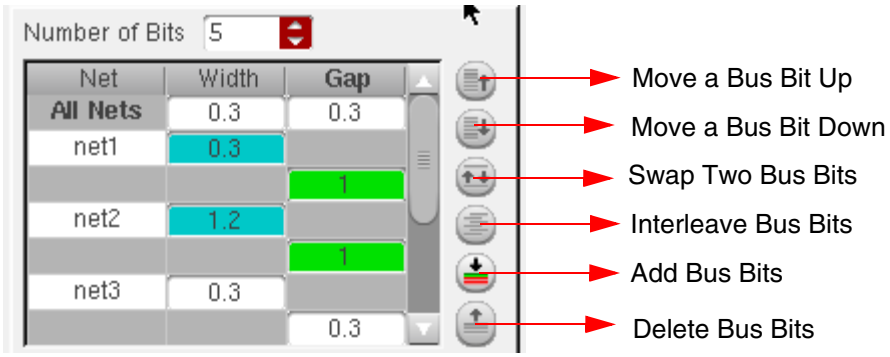
Important

The bus bit table is valid only if the following conditions are satisfied:

- The bus is started from the space. This means that the bus does not start from any IO ports, pre-existing wires, or the instance pins.
- The bus is started before the second clicking point.
- The bus bit configuration cannot be changed once the second point has been digitized. However, the bit width and intra bit spacing values can be changed at any time while creating the bus.

Using the Buttons in the Bus Tab

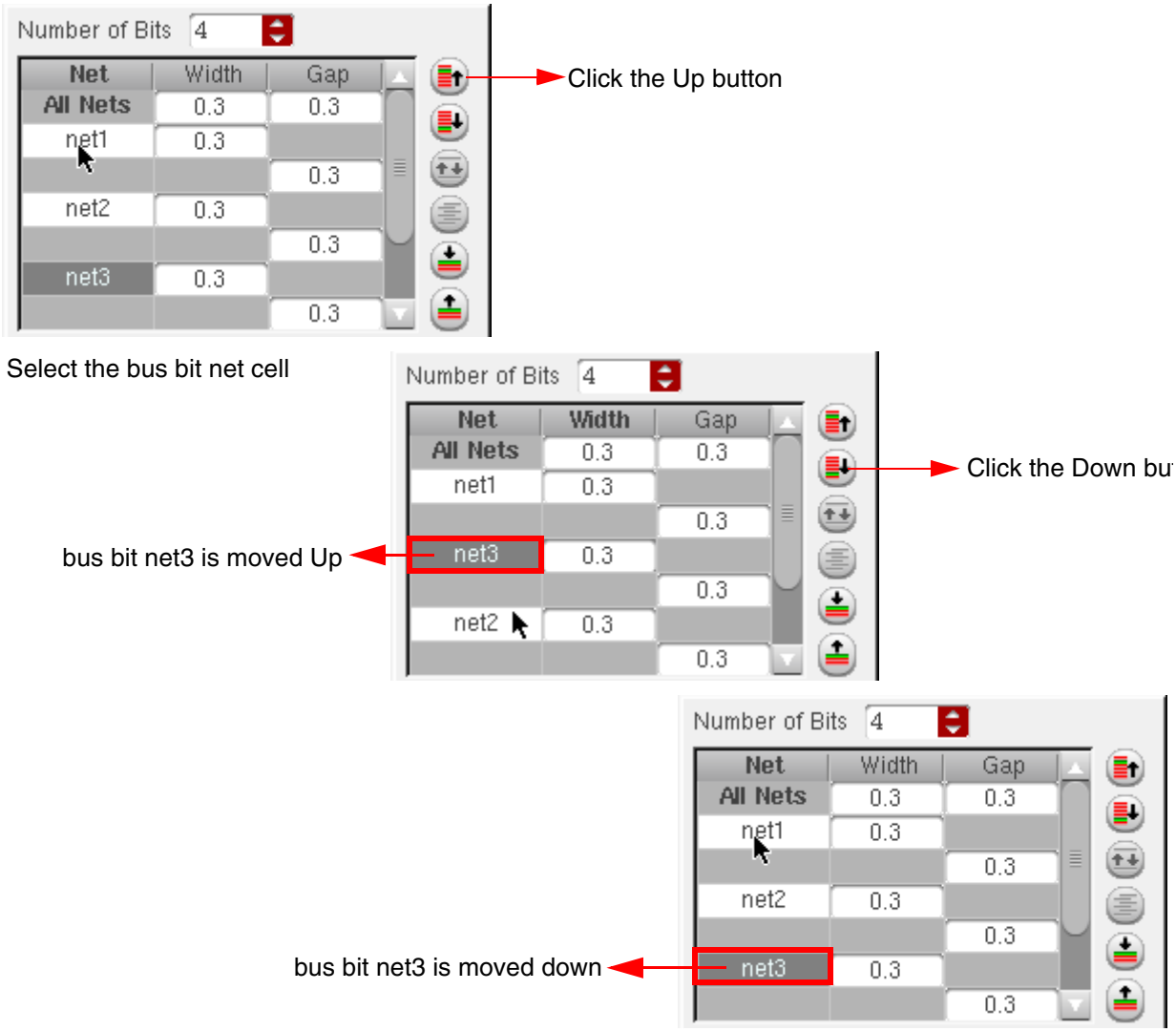
The following figure displays the various buttons available in the *Bus* tab of the *Overrides Constraint* section.



- *Moving a Bus Bit Up and Down*

  The *Move a Bus Bit Up* and *Move a Bus Bit Down* buttons in the *Bus* tab are used to move the bus bit up or down in the Bus Bit table. To move the bus bit up or down, select the bus bit Net cell and then click the *Move a Bus Bit Up* or *Move a Bus Bit Down* button. When the bus bit is moved, the width value associated with it is moved along with the bus bit. However, the two intra-bit gap spacing values above or below the moved bus bit are not moved. The moved bus bit net remains selected after the move operation is

completed. The following figure shows an example of moving the bus bit `net3` up and down.

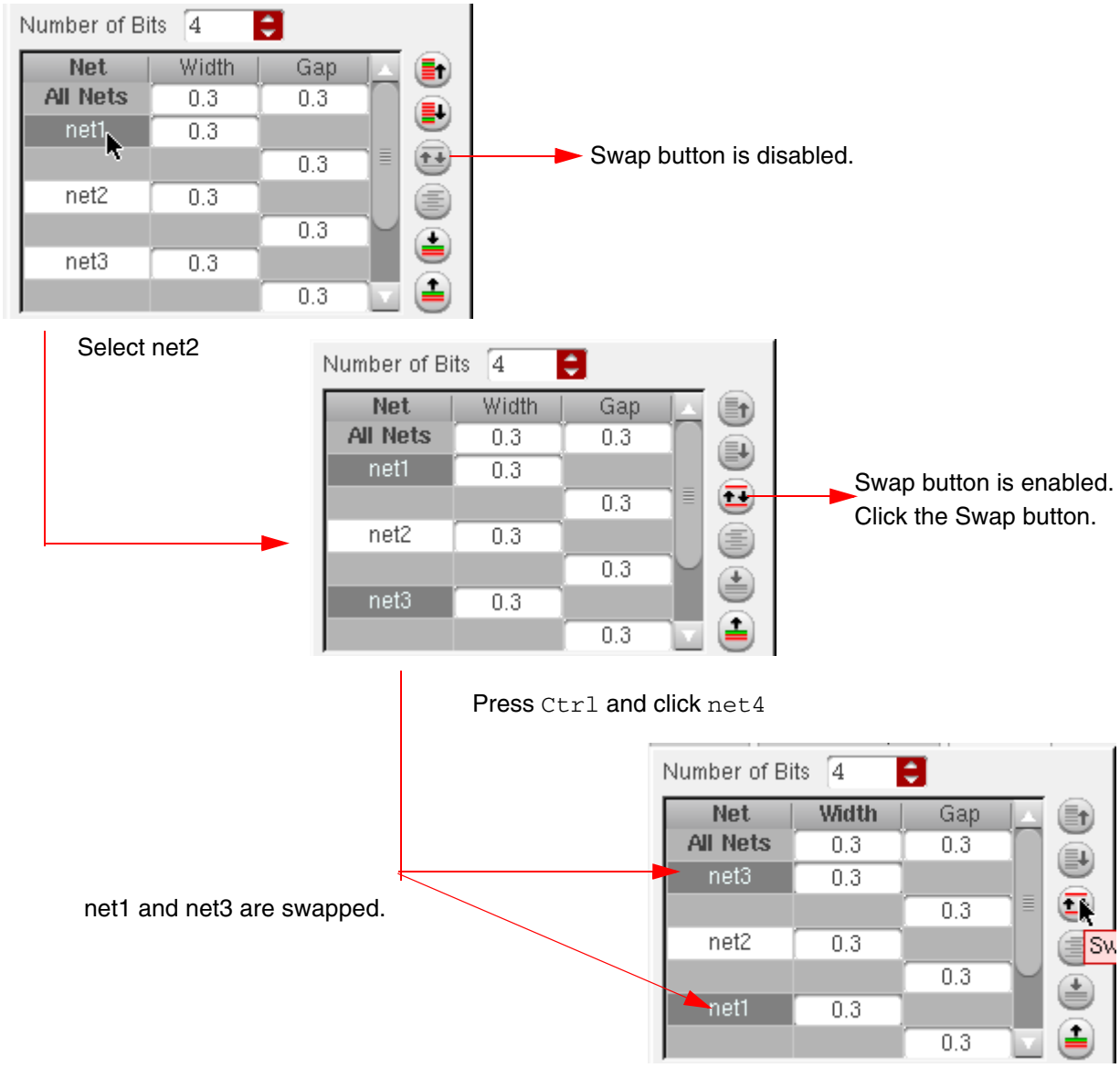


Note: The *Move a Bus Bit Up* and *Move a Bus Bit Down* buttons are enabled only when there is a single bus bit selected in the bus bit table. The two buttons are disabled if no bus bit is selected.

- *Swapping Two Bus Bits*

  The *Swap Two Bus Bits* button in the *Bus* tab is used to swap two bus bits. Select a bus bit Net from the bus bit table. To swap bus bits, you need to select two bus bit Nets. To select the second bus bit, hold the `Ctrl` key and click the second bus bit Net cell in the bus bit table. Now, click the *Swap Two Bus Bits* button. The two selected bus bit nets are swapped with each other. When the two bus bit nets are swapped, their

associated width is also swapped together. However, the intra-bit gap spacing values above and below the two bus bit nets are not swapped. Also, the swapped bus bits remain selected after the swap operation is complete. The following figure shows an example of swapping the bus bit `net1` and `net3`.



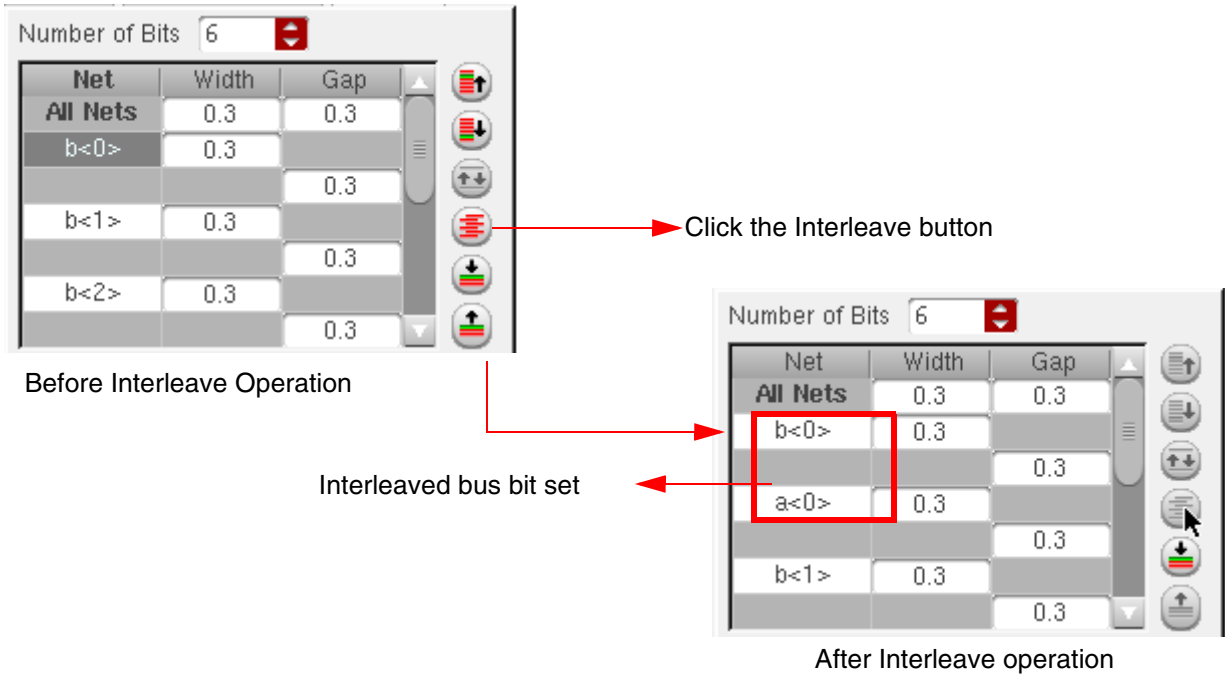
Note: The *Swap Two Bus Bits* button is enabled when there are just two selected bus bit Net cells.

- *Interleaving Bus Bits*

  The *Interleave Bus Bits* button in the *Bus* tab interleaves two buses of the same length. The *Interleave Bus Bits* button is enabled only if every bus bit has the same number of

bits and the bus bits are not mixed with other nets in between. Else, the button is disabled.

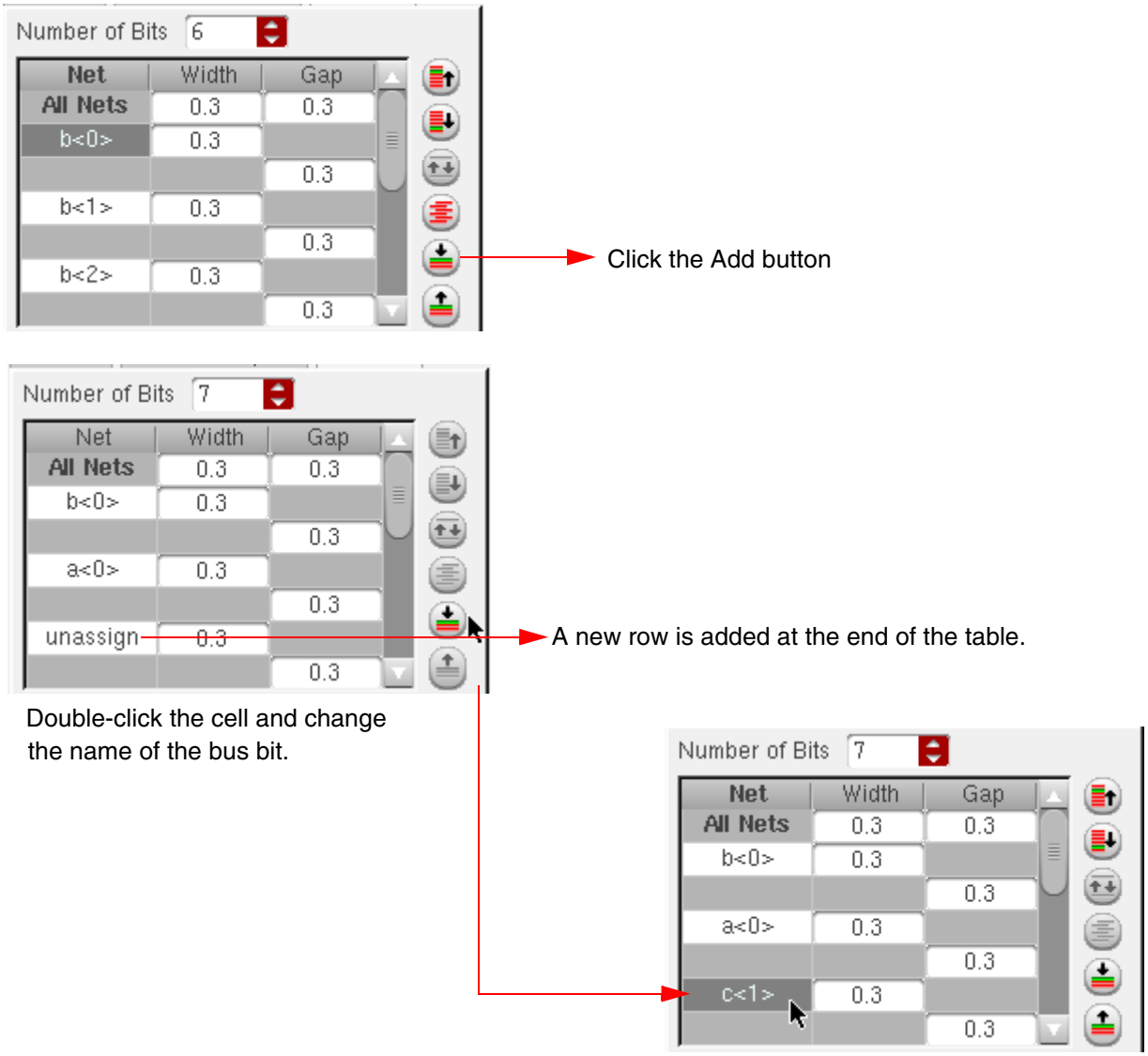
To interleave the bus bits, click the *Interleave Bus Bits* button. It then takes the first half and the second half set of bus bits and interleaves them together. For example, a<0>,a<1>,a<2>,b<0>, b<1>, b<2> is interleaved into “a<0> <b0>, a<1> b<1>,and a<2> b<2>, respectively. The following figure illustrates the given example. The intra-bit gap spacing values are not interleaved, similar to the move and swap operation. However, all the selected bits are deselected after the interleave operation is completed.



■ *Adding Bus Bits*

The *Add Bus Bits* button is used to add a new bus bit. When you click the *Add Bus Bits* button, a new bus bit row is added with the unassign net in the bus bit table. The width and spacing cells contains the default constraint look-up values. If a bus bit is selected in the bus bit table, the new bus bit is added after the selected bus bit; else, the new bus

bit is added at the end of the bus bit table. The following figure shows an example of adding a new bus bit c<1>.

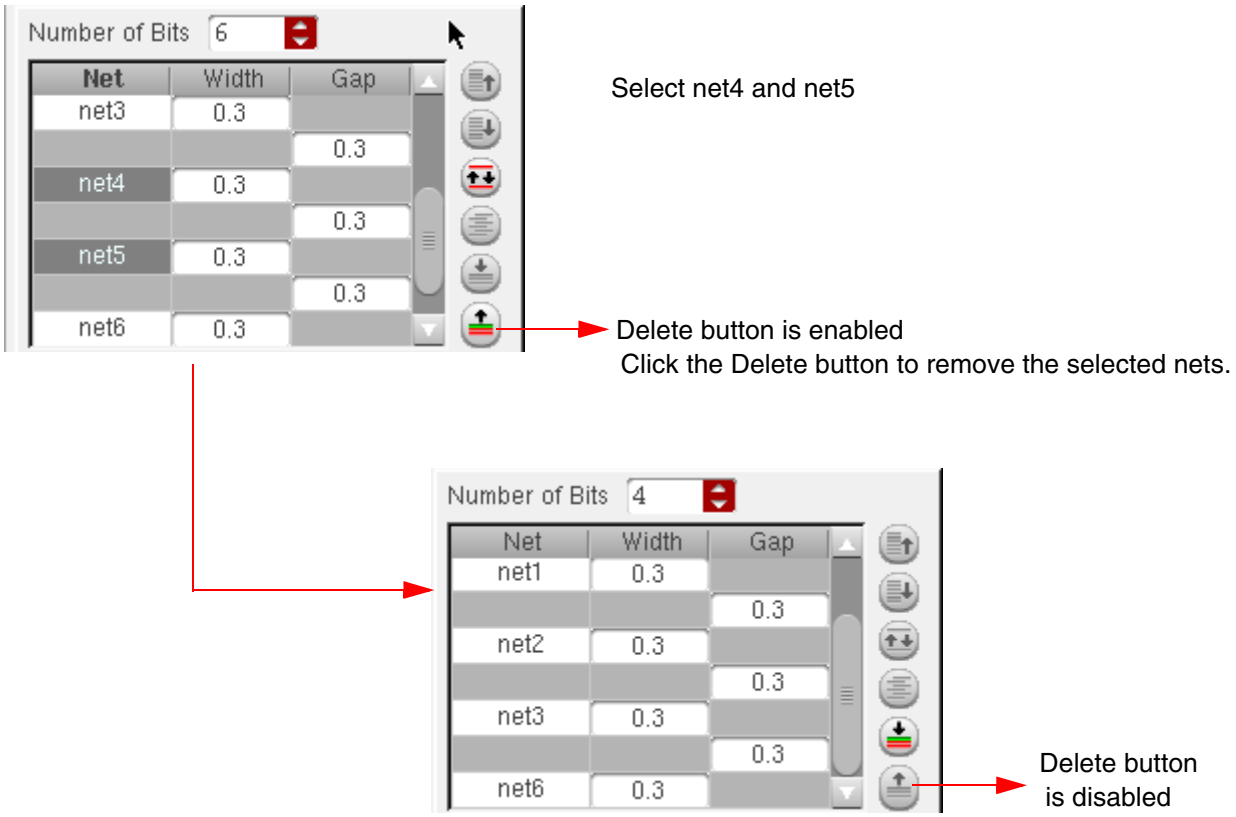


Note: The *Add Bus Bits* button is disabled if there are more than one selected bus bit nets in the table.

While adding three new bus bits when you specify the net name for the first bus bit and press `Enter` or `Tab` to move to the next unassigned bus bit, the unassign bus bits are not removed and a message is displayed in CIW at the first digitized point.

- *Deleting Bus Bits*

  The *Delete Bus Bits* button is used to delete a bus bit. To delete a bus bit or multiple bus bits, select the bus bits that you want to delete and then click the *Delete Bus Bits* button. The selected bus bits are deleted from the table. The intra-bit gap spacing row above the selected bus bit is also deleted. The *Delete Bus Bits* button is disabled after the delete operation is completed. The following figure shows an example of deleting net4 and net5 from the bus bit table.



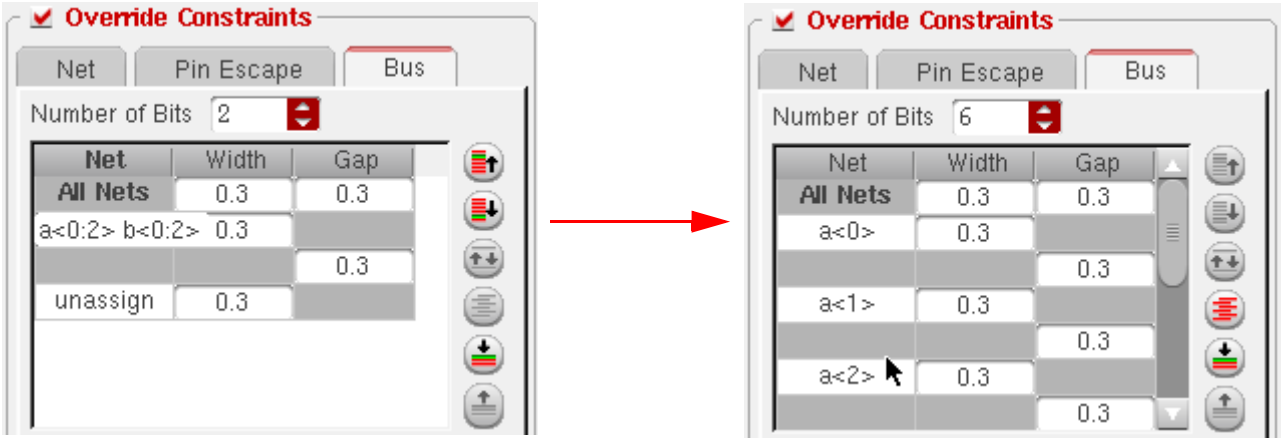
Note: The *Delete* button is disabled if there is no bus bit selected in the bus bit table.

Specifying Net Names for Bus Bit

After the *Create Bus* command is invoked, the *Bus* section is enabled in the *Override Constraints* section of the Wire Assistant. If there are no nets selected in the Navigator window, the bus bit adds two empty bus bit rows with one intra-bit gap spacing row in between. You can then specify a name to the bus bit.

The *Net* columns of the bus bit rows displays `unassign`. The `unassign` is the net name and signifies that you are about to create `unassign` net. You can specify a name to the `unassign` net. The name of the net can be specified in the *Net* column of the bus bit row in

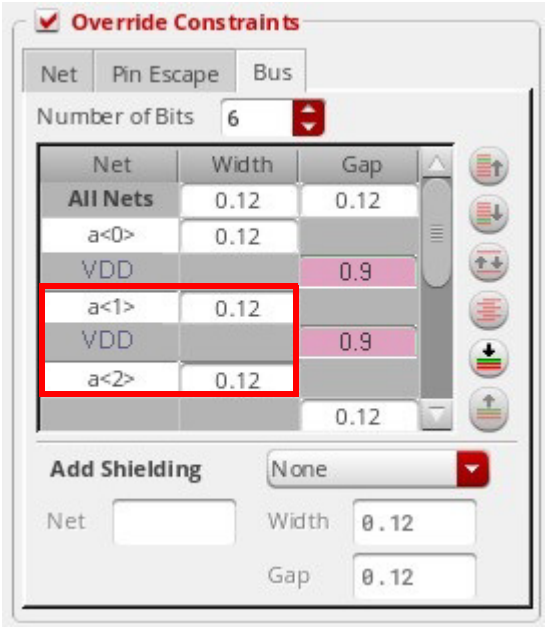
the *Bus* tab. To specify a net name, double-click the *Net* column. The name of a bus bit can be a single net name or the bit-range names. For example, specify `a<0:2> b<0:2>` in the *Net* column and press the `Enter` key, The bus bit table expands this bus bit syntax and automatically replaces the current bus bit rows with six bus bit rows and five intra-bit gap spacing rows `a<0>`, `a<1>`, `a<2>`, `b<0>`, `b<1>`, `b<2>`, as shown in the figure below.



Existing Shielding Constraint on Bus Bit Net

When a bus bit has a pre-existing shield constraint on the database, the bus bit automatically calculates the intra-bit shielding space value, regardless of the *Add Shielding* mode. The following figure shows an example where net1 already has shielding constraint with shielding net `VDD`, 0.3 as shielding width, and 0.3 as shielding space. The *Add Shielding* mode is specified as *None*. The bus bit table still calculates the intra-bit shielding space as 0.9. The shielding net *VDD* is added between net1 and net2. Also, the background color of the intra-

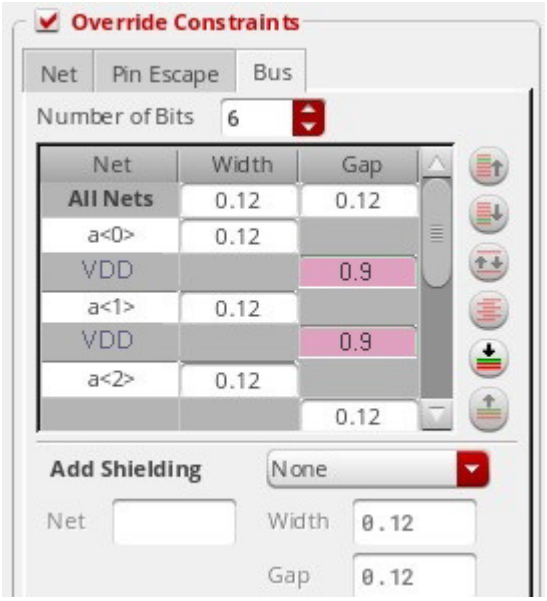
bit gap spacing cell has changed to purple with an appropriate tooltip as shown in the figure below.



Changing the Add Shielding Net Mode

Let us change the *Add Shielding* mode and add a shielding net to a bus bit net. To do this, perform the following steps:

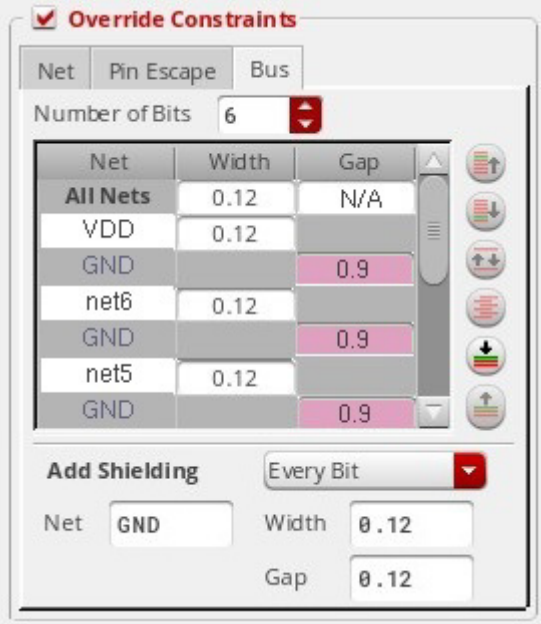
1. Select the *Every Bit* mode from the *Add Shielding* drop-down list.



Select Every Bit mode

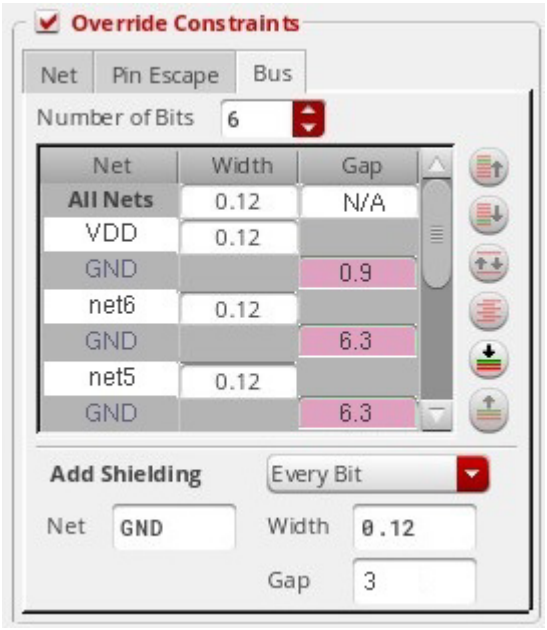
2. Specify the name of the shielding net as `GND` in the *Net* field. The *Width* and *Gap* fields are automatically filled up with the minimum constraint look-up values. After specifying the net name, the intra-bit shielding space value is calculated automatically and the bus bit table is updated with the new value.

   The *Net* column in the bus bit table now displays `GND` because bus bit net `net5` is shielded by net `VDD` and bus bit net `net6` is shielded by net `GND`.

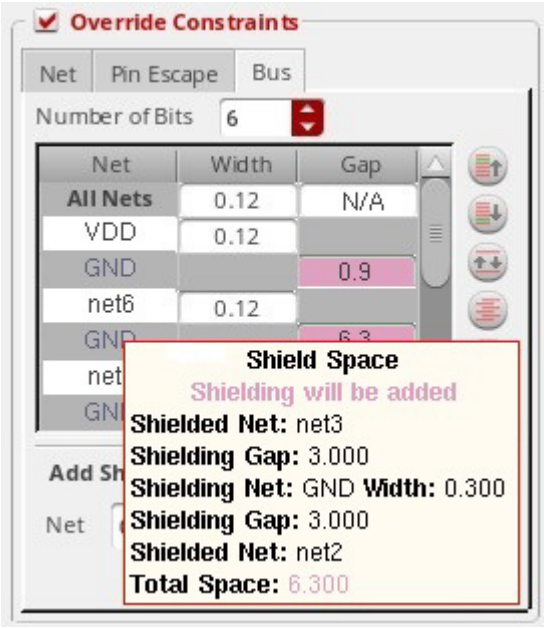


Specify Shielding Net as GND

3. Specify `3.0` as the new gap value in the *Gap* field. After entering the new value, the intra-bit shielding space is recalculated and the tooltip is also updated accordingly. This is shown in the following figure.

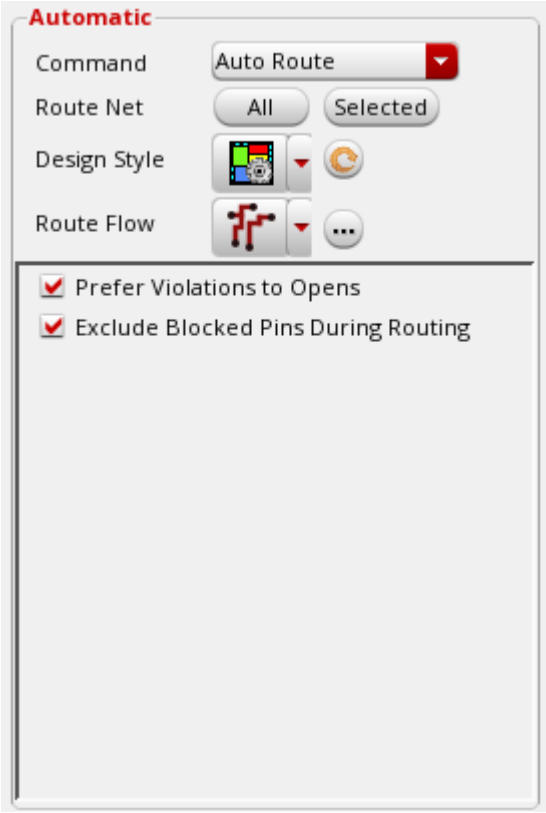


Specify the gap as 3.0



Recalculated intra-bit shielding space value along with the updated tooltip

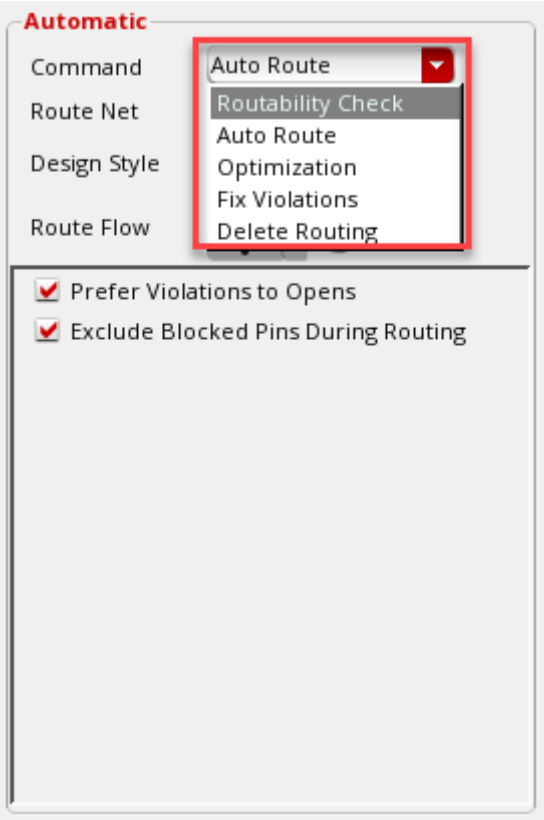
Automatic Section



The Automatic section is expanded by default when Wire Assistant is first launched, or after escaping from one of the interactive commands. The *Automatic* section allows you to specify the options for automatic routing, DRC fixing violations, and post-routing optimization.

■ Command

Click *Command*. The *Command* drop-down list displays the options that let you quickly navigate to other automatic routing flow sections in the Wire Assistant, as shown in the following figure.



You can quickly navigate to the respective routing section in the Wire Assistant by clicking the command. The following routing commands are available in the *Command* drop-down list.

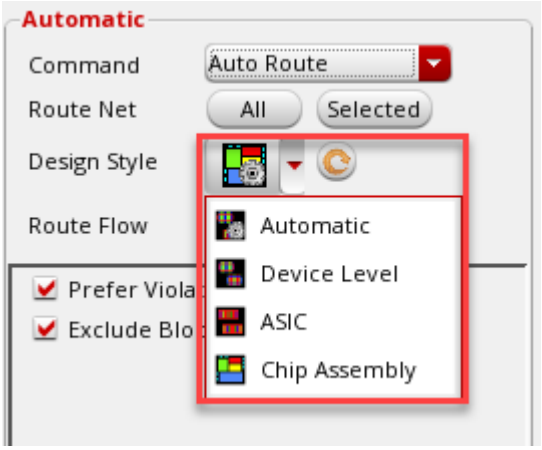
- Routability Check
- Auto Route
- Optimization
- Fixing Violations
- Delete Routing

Route Net

This field is updated depending on the option selected in the *Command* field. The details related to this field is explained in the respective routing flow sections.

Design Style

The Design Style drop-down list consists of the four design styles: *Automatic*, *Device Level*, *ASIC*, and *Chip Assembly*, as shown in the following figure. For more information, refer to Design Style.



Note: The *Design Style* field is disabled when the routing flow command is selected as *Optimization*, *Fix Violations*, or *Delete Routing* from the *Command* drop-down list.

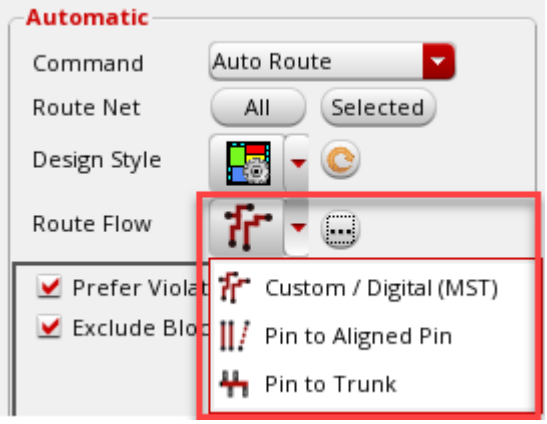
- **Refresh button** 

  When clicked, the *Refresh* button next to the *Design Style* field prompts arecalculation of the automatic design style that has been determined. The refresh button is only enabled when the design style is selected as *Automatic*.

- **Route Flow**

  The drop-down list consists of three routing topology styles: *Custom / Digital (MST), Pin to Aligned Pin*, and *Pin To Trunk*. You can select the routing style to be used for the type of design you are routing. By default, the routing topology style that was specified last appears to be selected in the drop-down list. This field is only enabled when

the *Auto Route* command is selected from the *Command* drop-down list. For other commands in the *Command* drop-down list, the *Route Flow* field is disabled.



- **Custom / Digital (MST)**

  This is the default routing style.

- **Pin to Aligned Pin**

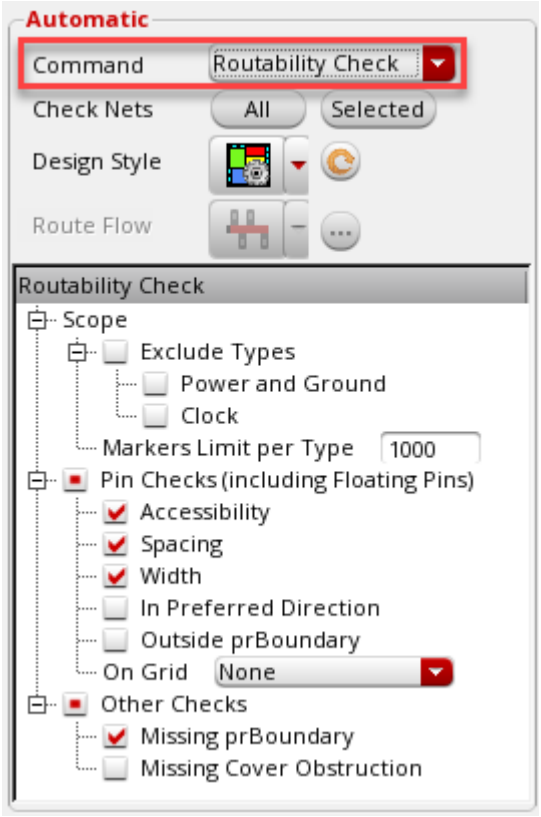
- **Pin to Trunk**

For more information on the routing flow and the related options, refer to Specifying Routing Flow Options.

- **Ellipses button** 

When clicked, displays the route flow along with the options in the Virtuoso Space-based Router Options form, based on the selection in the *Route Flow* field in the Wire Assistant. For example, if *Custom / Digital (MST)* is selected in the *Route Flow* field in the Wire Assistant, the *Custom / Digital (MST)* pane along with its options is displayed in the Virtuoso Space-based Router Options form.

Routability Check



The *Routability Check* feature lets you perform a set of routing checks to determine the potential routing issues early in the cycle. The routing issues can prevent the router from achieving optimal results.

Using the *Routability Check* option available in the *Auto* section of the Wire Assistant, you can check the width, spacing, and accessibility of pins, and other issues that may affect routing. Because this feature checks the data against foundry rules, user constraints, and Wire Assistant overrides, it is important to set up the routability check options in the Wire Assistant before running Routability Check.

Routability checks are designed primarily for the *Chip Assembly* routing mode. Therefore, if you set the routing mode to *Chip Assembly* in the Wire Assistant, the whole suite of routing checks is available. However, if you set the routing mode to *ASIC* or *Device Style* mode, only the pin accessibility checks are available.

To perform routability checks on a design, choose the *Routability Check* option from the *Command* drop-down list. The routability checks are displayed in the Wire Assistant. After you have selected the routability checks to be run on the design, click the *All* or *Selected*

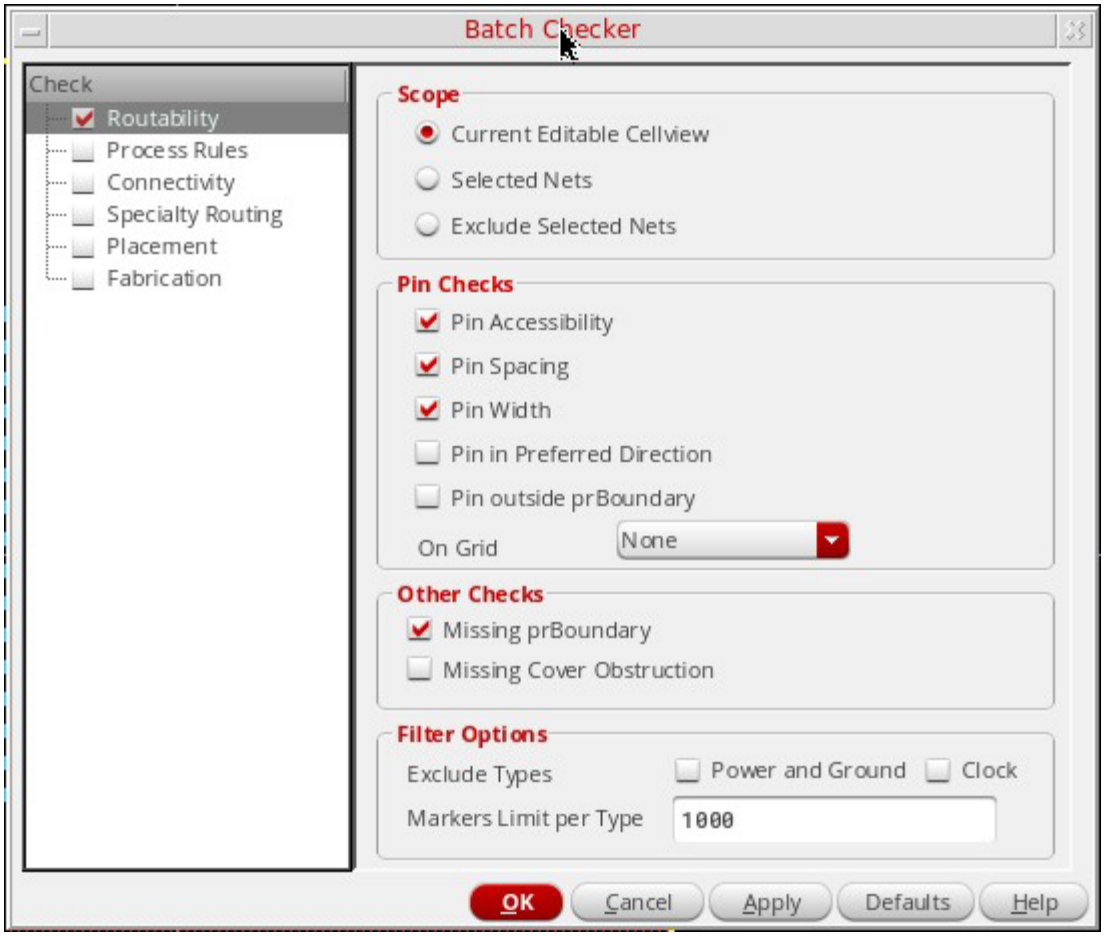
button to start routability checks. When the checks are complete, a summary report appears in CIW and the results are displayed on the *Routing* tab of the Annotation Browser.

From Wire Assistant, you can run the routability checks either on all nets or on selected nets. *Check Nets.*

- **All:** Routability checks are performed for all the nets available in the design.
- **Selected:** Routability checks are performed only for the nets selected in the design. The nets are selected using the Navigator assistant.

For more information on the routablility check options, see Routability Check.

The routability checks are also available in the Batch Checker form. This allows you to run routability checks along with various other checks. To open the Batch Checker form, choose *Verify – Design*.

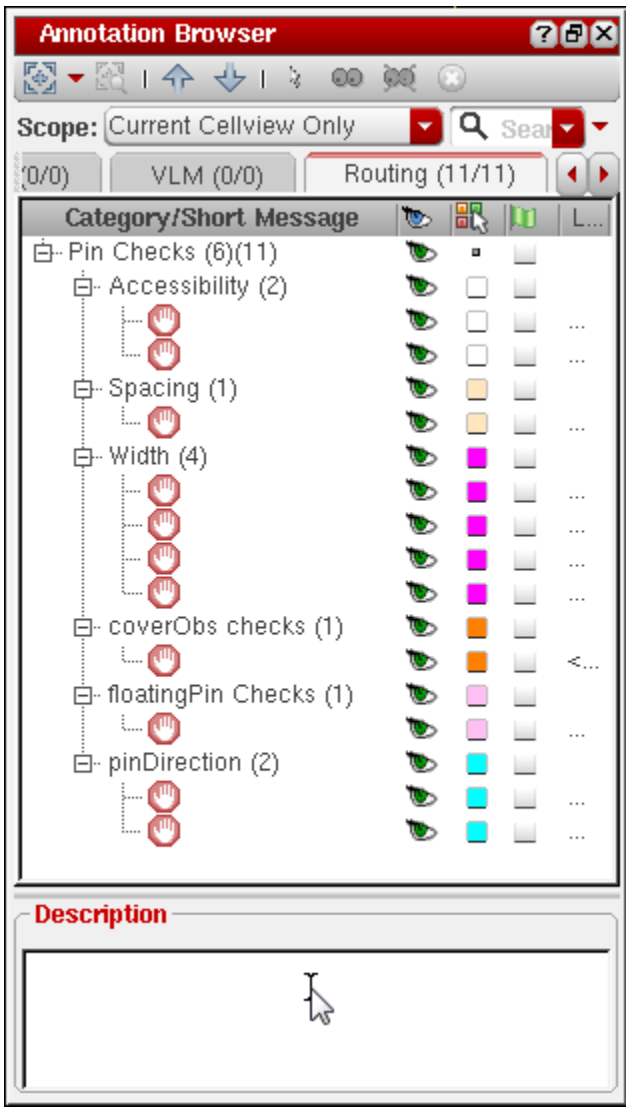


Select the required routability checks and click *All* to perform the routability checks on all the nets in the design. If you want to run the routability checks on only the selected nets, click *Selected*.

The routability checker generates a routability report, which summarizes the routability issues that are found in the design. This report is displayed in CIW, as shown in the following figure.

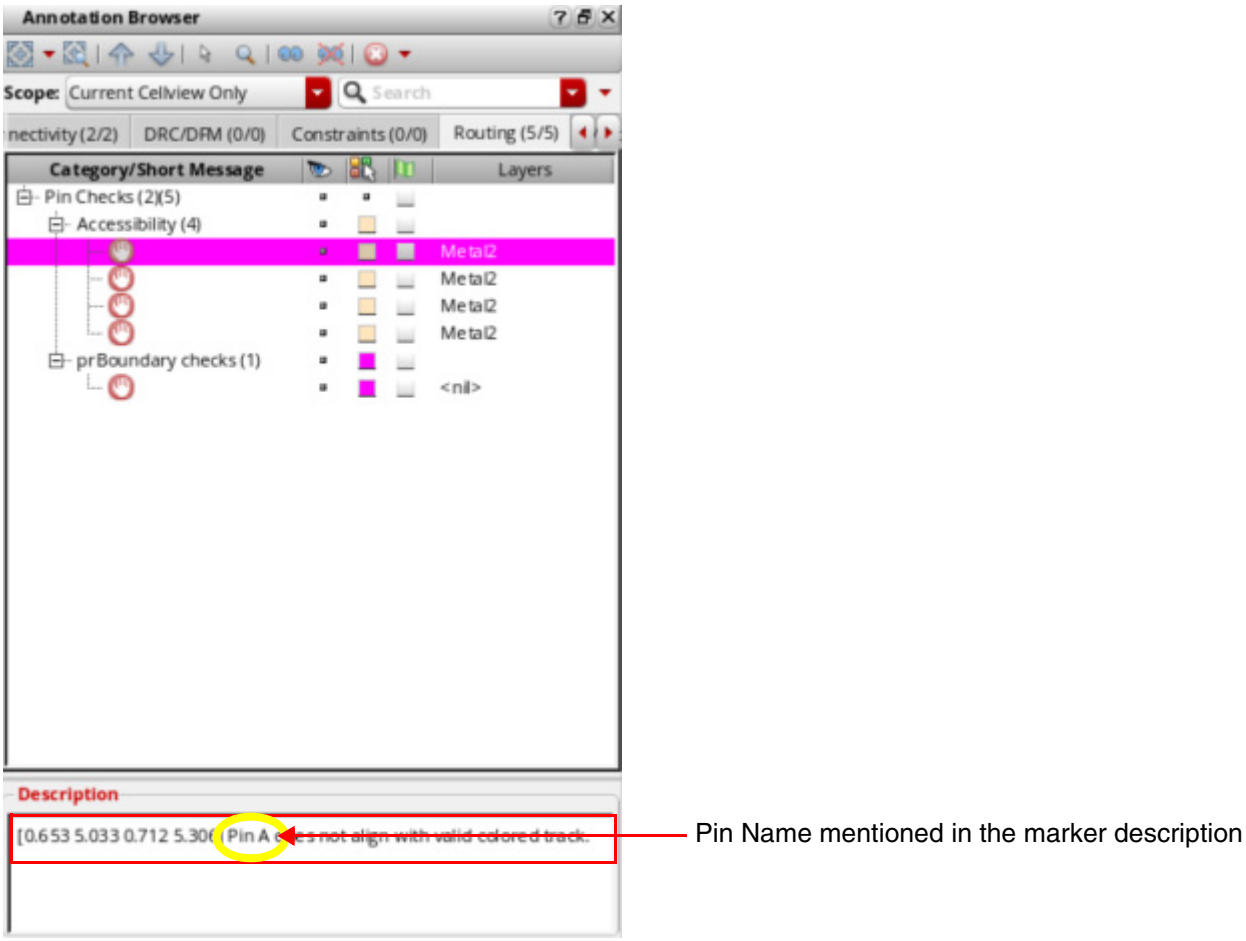
```
************************************************************
*                                                          *
*    Begin Check Routability Flow                          *
*                                                          *
************************************************************
Running: check_routability -gen_markers -clear_annotations -no_diffusion_shape_check true -
no_grid_check true -get_unreachable_pin_set sel:2dbe42b0 -error_limit 1000 -exclude_type { power
ground clock }
Performing routability checks...
Begin adding guides to complete connectivity...
    Connectivity updated on 3 net(s), 2 guide(s) created
End adding guides to complete connectivity 0.0s (kernel), 0.0s (user), 0.0s (elapsed), 1125.0MB vm,
1125.0MB peak vm
+-----------------------------------------------------+
| Routability Report Summary:                         |
+-----------------------------------------------------+
|          0 Min spacing violation(s)                 |
|          2 Min width violation(s)                   |
|          0 Access violation(s)                      |
+-----------------------------------------------------+
|          4 pin(s) found                             |
|          4 pin(s) checked                           |
|          0 pin(s) already connected (skipped)       |
|          0 pin(s) not used (skipped)                |
+-----------------------------------------------------+
Begin _delete_guides_...
Processed 3 nets in 0.00 seconds CPU time (0.00 nets/second).  Deleted guides from 2 net(s)
End _delete_guides_ 0.0s (kernel), 0.0s (user), 0.0s (elapsed), 1125.0MB vm, 1125.0MB peak vm
Routability checks done.
REPORTING_STATS | Elapsed Time: 0.0 Secs | Mem: "Current: 1125.00" Mb | Cmd: check_routability -
gen_markers -clear_annotations -no_diffusion_shape_check true -no_grid_check true -
get_unreachable_pin_set sel:2dbe42b0 -error_limit 1000 -exclude_type { power ground clock }
************************************************************
*                                                          *
*    End Check Routability Flow                            *
*                                                          *
************************************************************
```

The error markers are displayed on the *Routability* tab of the Annotation Browser. The following figure shows violations for some pin checks, floating pin checks, and cover obstruction checks generated during a routability check.

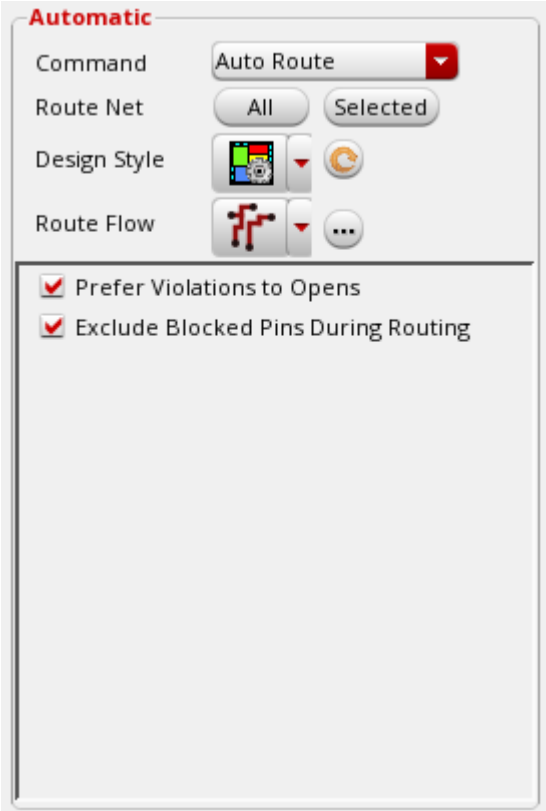


Note: Issues such as DRC violations should be addressed before attempting to route the design.

In the *Description* section of the Annotation Browser, the description of the pin checks mentions the name of pin that is not on the grid or on WSP, as shown in the following figure.



Auto Route



When the *Auto Route* option is selected from the *Command* drop-down list, you can specify the following Automatic routing options.

- **Route Net**
  - **All** routes all the nets that are available in the layout design. The `tieHigh`, `tieLow`, and `clock` nets are only routed when the nets are explicitly selected in the Navigator Assistant and then the *Selected* command is run. The nets with signal type `power` and `ground` nets at the top-level connected to power and ground nets at level-1 are not routed by the Minimum Spanning Tree routing. The power routing or pin to trunk routing should be used for such nets.
  - **Selected** routes only the selected nets in the design. Also, the `tieHigh` and `tieLow` nets are routed when explicitly selected

When the routing of the nets is successful, a summary report appears in CIW as shown in the following figure.

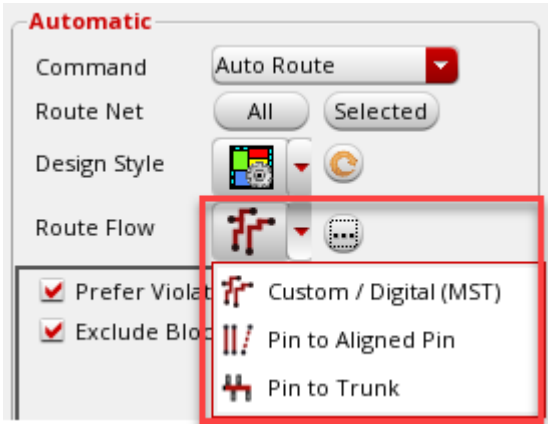
```
*                                                                      *
************************************************************************
(VCA-103012): Skipping the Routability Check step because no changes have been made since the last Routability Check was run in the
current session.
************************************************************************
*                                                                      *
*                      Begin Route Summary                             *
*                                                                      *
************************************************************************
Design Style was automatically determined to be deviceLevel
Number of nets in design: 14
Number of nets attempted: 7
Number of nets routed with no opens/shorts: 2
Number of nets excluded: 0
Number of terminals excluded: 0
Number of opens on attempted nets: 0
Number of shorts on attempted nets: 5 (pre-existing: 0, new: 5)
************************************************************************
*                                                                      *
*                      End Route Summary                               *
*                                                                      *
************************************************************************
```

■ Design Style

The design style determines which steps in the Wire Assistant flow will run and also affects the heuristics of global and detail routes. For more information, see Design Style.

■ Route Flow

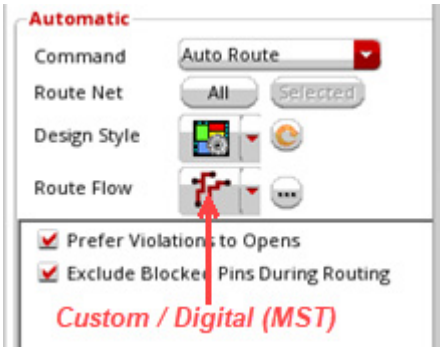
Depending on the routing flow selected from the *Route Flow* drop-down list, the options that are relevant and useful for the selected routing flow are displayed in the section.



- *Custom/ Digital (MST)*

The *Custom / Digital (MST)* is the default routing style. The options that are displayed when this routing flow is selected are the following:

- *Wrong Way Tax*
- *Wrong Way Tax Value*
- *Remove Pre-Route Dangles*
- *Run With Low Effort*
- *Prefer Violations to Opens*
- *Exclude Blocked Pins During Routing*
- *Force Global Planning*
- *Global Route All Nets*
- *Preserve Pre-Routes*



Note: These options are displayed only when they are selected in the Wire Assistant Visibility form. By default, only the *Prefer Violations to Opens* and *Exclude Blocked Pins During Routing* options are displayed.

- **Pin to Aligned Pin**

The options that are displayed when this routing flow is selected from the *Route Flow* drop-down list are as follows:

- *Wrong Way Tax*
- *Wrong Way Tax Value*
- *Remove Pre-Route Dangles*
- *Run With Low Effort*
- *Prefer Violations to Opens*

- *Exclude Blocked Pins During Routing*
- *Preserve Pre-Routes*

Note: These options are displayed only when they are selected in the Wire Assistant Visibility form. By default, only the *Prefer Violations to Opens* and *Exclude Blocked Pins During Routing* options are displayed.

- **Pin to Trunk**

The options that are displayed when this routing flow is selected from the *Route Flow* drop-down list are as follows:

- *Generate Trunks*
- Extend Trunks
- *Route Twigs*
- *Trim Trunks*
- *Trunk to Trunk*
- Mode All Trunks, Crossed Trunks Only, and Trunk Mesh Routing

Clicking the ellipses button next to the field displays the Trunk Mesh Routing Configuration Form (ICADVM18.1 EXL Only).

- *Prefer Violations to Opens*

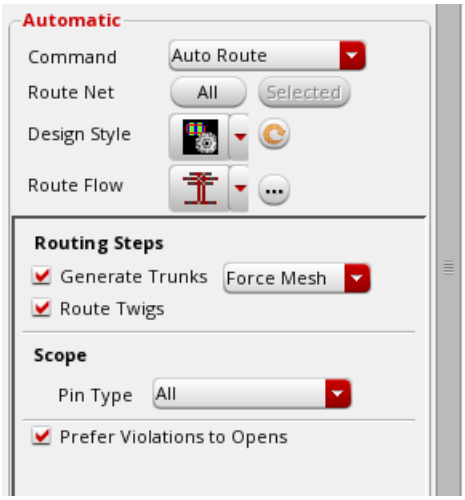
For more information on the route flows, refer to Specifying Routing Flow Options

- **Tree Route (ICADVM 18.1 Only)**

  The options that are displayed when this routing flow is selected from the *Route Flow* drop-down list are as follows:

  - *Generate Trunks - No Mesh, Auto Mesh, and Force Mesh*
  - *Route Twigs*
  - *Scope*
  - *Pin Type - All, Gate, Source And Drain*

- *Prefer Violations to Opens*



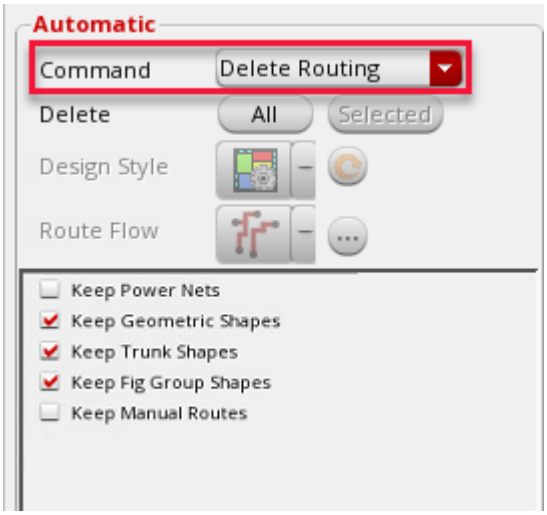
For more information on Tree Routing, see Specifying Routing Flow Options.

Track based Routing in Auto Route Flow

You can now do routing on WSPs or track patterns using the *Auto Route* command in the Wire Assistant. To enable track based routing, the following conditions must be satisfied.

- Before starting the Auto Route routing command, select the *Track Pattern* option from the *Snap Wire To* drop-down list if track pattern exists in the design. You can select the Snap Pattern option if the WSPs exists in the design.
- Each routing layer that has a track pattern or WSP must have a defined routing direction, which is either horizontal or vertical but not both.
- Center of pins must be on a track grid.

Delete Routing



When the *Delete Routing* option is selected from the *Command* drop-down list, you can specify the following delete rouitng options.

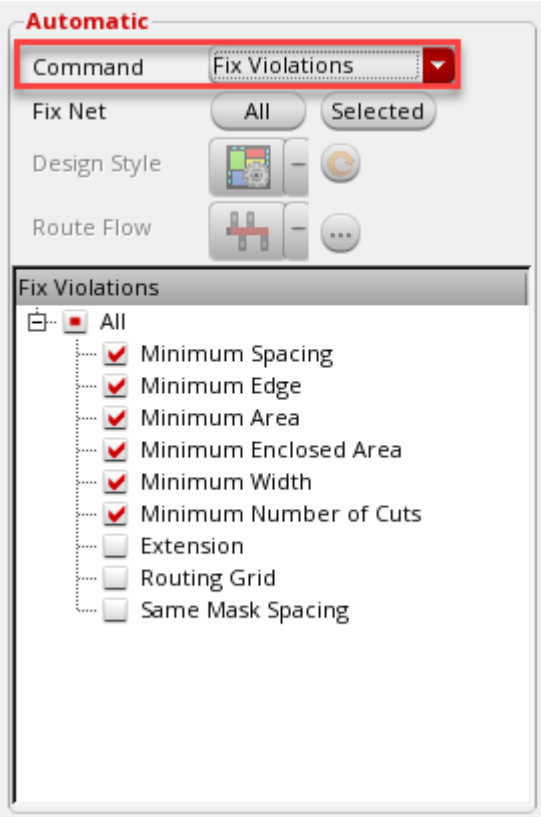
- **Delete**
  - **All** deletes all the routing that is available in the layout design.
  - **Selected** deletes only the selected routing in the layout design.

The *Design Style* and *Route Flow* fields are disabled when the *Delete Routing* command is selected from the *Command* drop-down list.

- **Keep Power Nets**
- **Keep Geometric Shapes**
- **Keep Trunk Shapes**
- **Keep Fig Group Shapes**
- **Keep Manual Routes**

For more information on the delete routing options, refer to Deleting Routing.

Fixing Violations



When the *Fix Violations* option is selected from the *Command* drop-down list, you can specify the following DRC fixing violation options.

- **Fix Net** fixes net violations
  - **All** fixes violations for all the nets that are available in the layout design.
  - **Selected** fixes violations of only the selected nets in the layout design.

The *Design Style* and *Route Flow* fields are disabled when the *Delete Routing* command is selected from the *Command* drop-down list.

Fix Violations

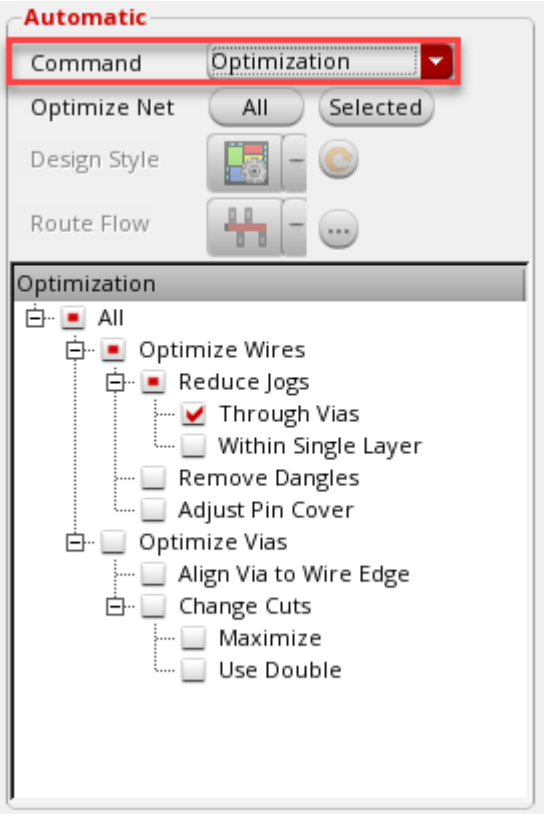
- All

  When selected, fix all the violations
  - Minimum Spacing
  - Minimum Edge

- Minimum Area
- Minimum Enclosed Area
- Minimum Width
- Minimum Number of Cuts
- Extension
- Routing Grid
- (ICADVM 18.1 Only) Same Mask Spacing

For more information on the above options, see Fix Violations.

Optimization



When the *Optimization* option is selected from the *Command* drop-down list, you can specify the following post-routing optimization options.

- **Optimize Net** optimizes existing routing

- **All** optimizes all nets that are available in the layout design.
- **Selected** optimizes only the selected nets in the layout design.

The *Design Style* and *Route Flow* fields are disabled when the *Delete Routing* command is selected from the *Command* drop-down list.

- **All** selecting the *All* checkbox allows you to easily select or deselect all options in the *Optimization* section.
- Optimize Wires
  - Reduce Jogs
    - Through Vias
    - Within Single Layer
  - Remove Dangles
  - Adjust Pin Cover
- Optimize Vias
  - Align Via to Wire Edge
  - Change Cuts
    - Maximize
    - Use Double

For more information on the routing optimization options, refer to Optimization.

Interactive Section

This section is enabled in the *Wire Assistant* when you start the *Create Bus, Create Wire, Point to Point, and Guided Routing* commands. In the *Interactive* section, you can specify the Create Wire, Create Bus, Point to Point, Guided routing, Finish Wire and Finish Entire Net options.

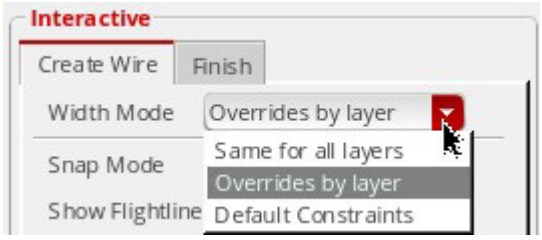
- Create Wire
- Create Bus
- Finish
- Point to Point
- Wire Assistant Forms

Create Wire

The *Create Wire* tab is displayed in the *Interactive* section of the *Wire Assistant* when you start the *Create – Wiring – Wire* command, and *Create Wire* is selected in the *Wire Assistant Visibility* form. This tab remains available in the *Wire Assistant* as long as the *Create Wire* command is active. The following options are available in the *Create Wire* tab of the Wire Assistant. The same options are also available in the Create Wire form.

- **Width Mode**

  Using the *Width Mode* drop-down list, you can specify one of the following options.



  - Same for all layers
  - Overrides by layer
  - Default Constraints

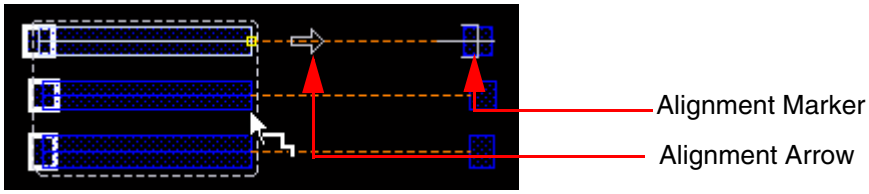
  By default, the *Overrides by layer* option is selected.

- **Snap Mode** controls how the cursor snaps when you create a wire. This option is available in the Create Bus Form and Create Wire Form. From the Snap Mode drop-down list, you can select one of the following snap modes while creating a wire.

- diagonal
- orthogonal
- L90XFirst
- L90YFirst

- **Show Flightline** controls the display of flightlines during *Wire*, *Bus*, and *Guided Routing* commands. They are displayed by default.
  - **on** displays the flightlines. This is the default setting.
  - **off** hides the flightlines. You can gain performance improvements by using this setting in the case of large buses.

    Note: The *Finish Wire* and *Finish Entire Net* commands display flightlines even if the *Flightlines* setting in the *Wire Assistant* is set to *off*.
- **Show Alignment Markers**, if selected, dynamically displays an alignment arrow and an alignment marker. The arrow points to the closest object on the active flightline. While creating a bus, the alignment marker appears when any of the bus bits align with their target pins.

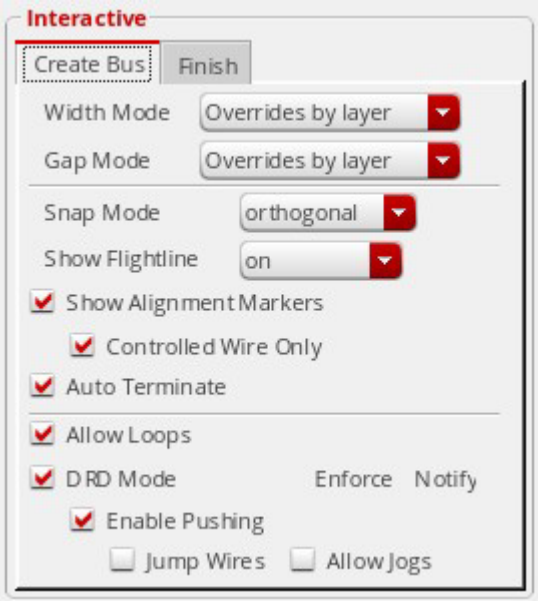


  This option is available in the Layout Editor Options Form.
  - **Controlled Wire Only** displays an alignment arrow and an alignment marker only if the control wire is aligned with its target pin. For more information about control wires, see Using the Control Wire. This checkbox is selected by default.

    Environment Variable: weShowCtrlWireAlignMarkerOnly
- **Auto Terminate** completes the *Wire* command as soon as a point is digitized on a flightline target object (pin, existing wire, or via). When creating a bus, *Auto Terminate* snaps all wires in the bus to the target pins. This checkbox is selected by default. It is also available in Create Wire Form.
- **Allow Loops** allows the creation of loops when creating or editing wires. This option is available in the Layout Editor Options Form.
- **DRD Mode** enables you to specify the DRD editing mode. If you select this checkbox, both *Enforce* and *Notify* modes are switched on. With *Enforce*, the currently running

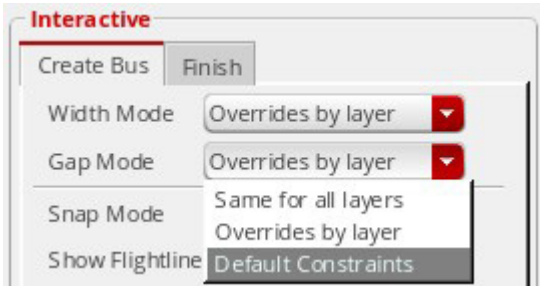
command fails and a message is generated when there are DRD violations. With *Notify*, a message is generated when there are DRD violations but you can continue with the edit operation.

Create Bus



The *Create Bus* tab is displayed in the *Interactive* section of the *Wire Assistant* when you start the *Create – Wiring – Bus* command, and *Create Bus* is selected in the *Wire Assistant Visibility* form. This tab remains available in the *Wire Assistant* as long as the *Create Bus* command is running. The following options are available in the *Create Bus* tab of the Wire Assistant. The same options are also available in the Create Bus form.

- Gap Mode



  - Same for all layers
  - Overrides by layer

- Default Constraints

  By default, the *Overrides by layer* option is selected.

See Create Wire for the following Options

- Width Mode
- Snap Mode
- Show Flightline

  Controls the display of flightlines. The *on* and *off* option in the *Show Flightline* field is the same as in the *Create Wire* command. The addiitonal option *minimum* is available in the *Show Flightline* field when the *Create Bus* command is enabled.

  - **minimum** displays only three flightlines per bus - one for the central control wire and the other two for the extreme side bus bits (left and right or top and bottom bus bits). This setting applies only to bus mode. You can gain performance improvements by using this setting in the case of large buses.

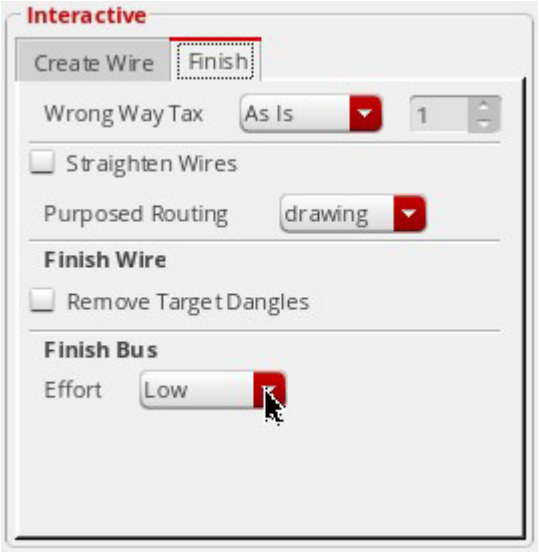
- Show Alignment Markers
  - Controlled Wire Only

    Displays an alignment arrow and an alignment marker only if the control wire is aligned with its target pin. For more information about control wires, see Using the Control Wire. This option is selected by default.

    Environment Variable: weShowCtrlWireAlignMarkerOnly

- Auto Terminate
- Allow Loops
- DRD Mode
  - Enable Pushing
    - Jump Wires
    - Allow Jogs

Finish



The Finish tab is by default displayed in the *Interactive* section of Wire Assistant if *Finish* is selected in the *Wire Assistant Visibility* form. The following options are available in the *Finish* tab of the Wire Assistant.

- **Wrong Way Tax** controls whether or not wrong way routing is permitted.
  - **As Is** uses the default setting as initialized by the router. This setting can allow certain wrong way routing if required. This is selected by default and maps to the tax value `1`.
  - **Allow** permits wrong way routing freely. This maps to the tax value `0`.
  - **None** does not allow wrong way routing. The wrong way tax is `100`.
  - **Custom** lets you control the extent of wrong way routing you want to permit. You can select this option and specify a value in the *Wrong Way Tax Value* field. The default value is 1. You can increase the value to impose a higher restriction on wrong way routing.

  Environment variable: allLayersWrongWay

- **Wrong Way Tax Value** indicates the extent to which wrong way routing can be created. This cyclic field is placed next to the *Wrong Way Tax* combo box. This field is editable only when *Wrong Way Tax* is set to *Custom*. You can specify a value between `0` to `100`, `0` being the level of maximum freedom, and `100` being the level of maximum restriction for creating wrong way routing.

  Environment variable: allLayersWrongWayCustomValue

- **Straighten Wires** runs optimization that smoothens wires by removing unnecessary jogs where possible. By default, this option is deselected. When this option is selected, it performs the critic pass after Finish Wire, Guided Routing, Point to Point Routing, and Auto Route flow. For the Finish Wire command, the critic pass is run from the last clicked point of the target pin. For Point to Point Routing, the critic pass is run on every consecutive clicked point. However, if the Point to Point mode is the Shortest Connection, another critic pass is run from the first clicked point to the last point.
- **Purposed Routing** enables you to select the non-drawing, voltage-aware and drawing purposes if the voltage-related purposes are defined in the `techPurposes` section of your technology file.

Finish Wire

- **Remove Target Dangles**, if selected, the *Finish Wire* command removes any pre-existing dangles on the target segment. This prevents creation of dangles between the last segment of the *Create Wire* command and the first segment of the *Finish Wire* command. This checkbox is off by default.

Finish Bus

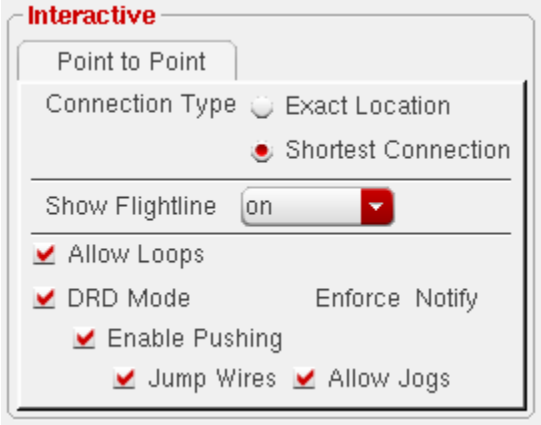
- **Effort**

  Controls the algorithms and the time that is used to complete the *Finish* command for a bus.

  Environment variable: `finishEffort`

  - **Low** performs the simple routing the fastest. However, only routes the designs in which there are no obstructions between the current wire and the finished target. This is the default option.
  - **Medium** performs the simple routing. However, if the routing fails, the bits are routed individually. This is a slower routing option but might route more cases.
  - **High** performs the simple routing. However, if the routing fails, the full router engine runs. This is a slowest routing option but will route most of the cases.

Point to Point



The *Point to Point* tab is displayed in the *Interactive* section of the *Wire Assistant* when you start the *Create – Wiring – Point to Point* command and *Point to Point* is selected in the *Wire Assistant Visibility* form. This tab remains available in the *Wire Assistant* as long as the *Point to Point* command is running. The following options are available in the *Point to Point* tab of the Wire Assistant. The same options are also available in the Point to Point Form.

- **Connection Type**
  - **Exact Location** uses the exact digitized points to create routing between two given points.
  - **Shortest Connection** uses the shortest route for connecting points and may also use pre-existing wires for creating the route.

See Create Wire for the following options.

- **Show Flightline**
- **Allow Loops**
- **DRD Mode**
  - **Enable Pushing**

Wire Assistant Forms

This section describes the following Wire Assistant forms:

- Wire Assistant Visibility Form

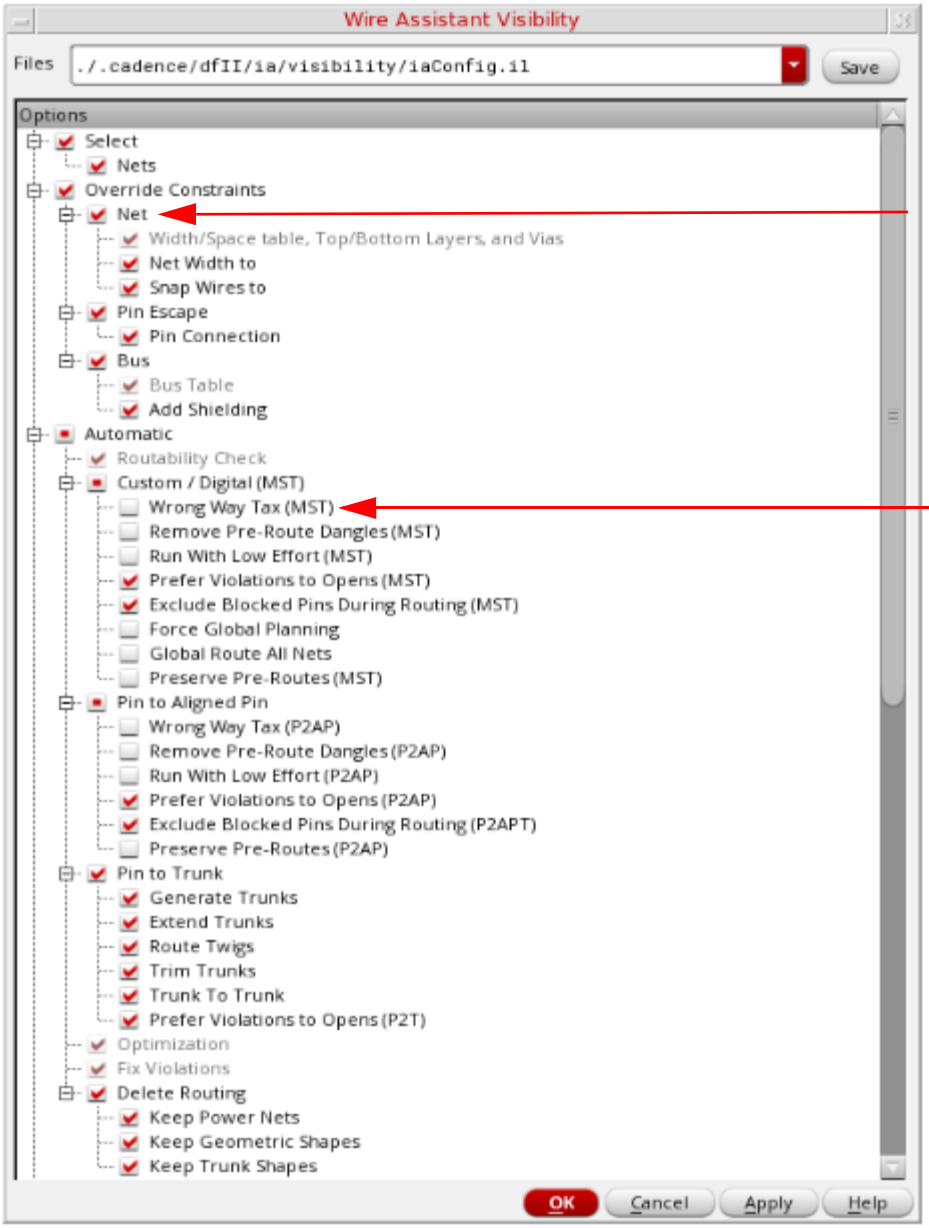
- Via Configuration Form
- Trunk Mesh Routing Configuration Form (ICADVM18.1 EXL Only)

Wire Assistant Visibility Form

You can open the Wire Assistant Visibility form using one of the following methods:

- Click *Configure Wire Assistant* drop-down arrow and then click the *Visibility* option from the *Configure Wire Assistant* drop-down list.

- Click the *Configure Wire Assistant*  button.



Use this form to configure sections and the options in each section that you want to display or hide in the *Wire Assistant* for a session. In this form, the top-level nodes (next to the - sign) represent titles of sections that appear in the *Wire Assistant*. The child nodes of these top-level nodes represent the fields and options that are available for the respective sections. Selecting the checkbox next to the top-level or child node displays the respective section and the field or option in a section, respectively, in the *Wire Assistant.*

The *Files* list displays all the possible locations where you can save the settings of the *Wire Assistant Options* form. By default, the settings are stored in the `iaConfig.il` file in the following directory:

`./.cadence/dfII/ia/visibility/iaConfig.il`

The following figure shows the contents of a sample `iaConfig.il` file, which indicates the visibility of fields and options in the *Wire Assistant*:

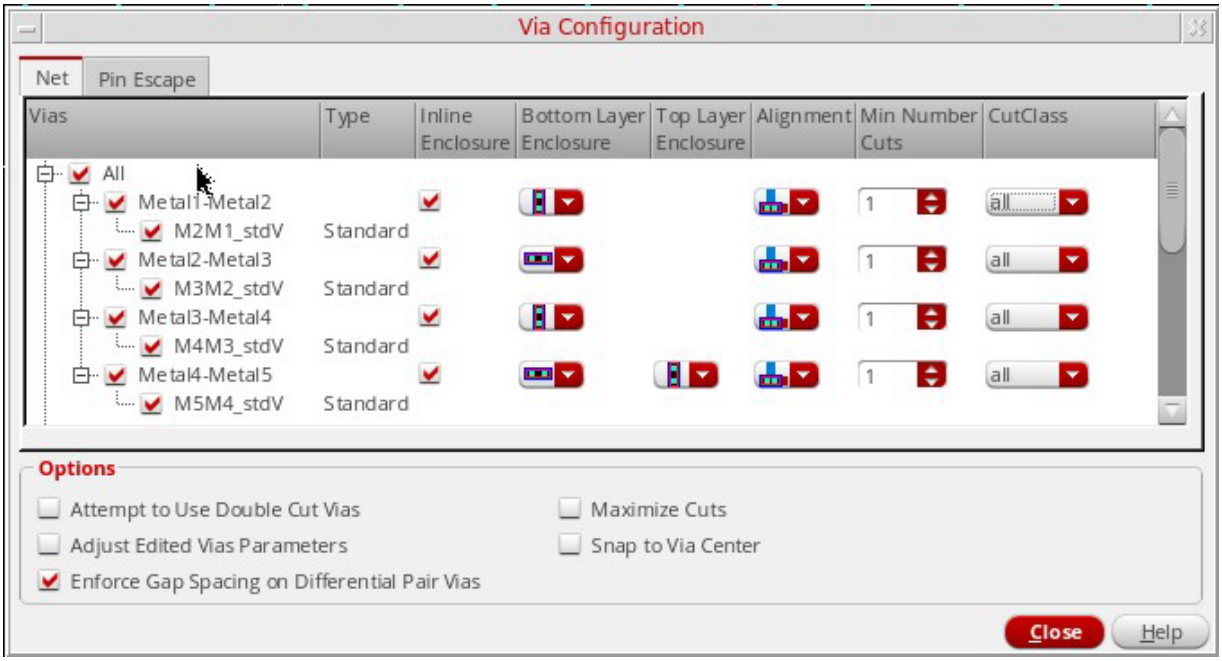
```
( "Connectivity"
    ( "[Connectivity]: Net Name" netName "" )
)
( "Override Constraints"
    ( "[Override Constraints]: Width/Space Table" widthSpacingTable
      (("poly1" "Horizontal" 0.6 1.0) ("metal1" "Horizontal" 0.8 1.0) ("metal2" "Verti
    )
    ( "[Override Constraints]: Minimum Number of Cuts" sbNumberCut 1 )
    ( "[Override Constraints]: Top Routing Layers" cfRoutingTopLayer "metal3" )
    ( "[Override Constraints]: Top Pin Escape Layers" cfTaperTopLayer "metal3" )
    ( "[Override Constraints]: Bottom Routing Layers" cfRoutingBottomLayer "poly1" )
    ( "[Override Constraints]: Bottom Pin Escape Layers" cfTaperBottomLayer "poly1" )
    ( "[Override Constraints]: Routing Vias" validViaButton "*" )
    ( "[Override Constraints]: Pin Escape Vias" taperValidViaButton "*" )
    ( "[Override Constraints]: Pin Escape Halo" taperHaloValue -1.0 )
)
( "Create Bus"
)
( "Shield"
)
( "Create Wire(s)"
)
( "Finish Net"
)
( "P2P Connection Type"
)
( "Guided Routing Envelope"
)
( "Tunnel"
)
( "Assisted Routing"
)
( "Auto Route"
    ( "[Auto Route]: Route Style" weNetRouteRoutingStyle "Device" )
    ( "[Auto Route]: Wrong Way Tax" weNetRouteWrongWayCost "As Is" )
    ( "[Auto Route]: Wrong Way Tax Value" weNetRouteWrongWayValue 1 )
    ( "[Auto Route]: Taper Pin Width" weNetRouteTaperToPinWidth nil )
    ( "[Auto Route]: Taper To Pin Width Style" weNetRouteTaperToPinWidthStyle "To Firs
    ( "[Auto Route]: Set Net Width" weNetRouteRouteNetAtPinWidth nil )
    ( "[Auto Route]: Set Net At Pin Width Style" weNetRouteRouteNetAtPinWidthStyle "Us
    ( "[Auto Route]: Attempt to Use Double Cut Vias" weNetRouteUseDoubleCutVias nil )
    ( "[Auto Route]: Run Post Route Refinement" weNetRouteRefinement nil )
    ( "[Auto Route]: Run with Low Effort" weNetRouteLowEffort t )
    ( "[Auto Route]: Use Override Constraints" weNetRouteUseOverrides t )
    ( "[Auto Route]: Run All Or Selected" weNetRouteWAAllOrSelected "All" )
)
( "Wiring"
    ( "[Wiring]: Snap To Pin Center" snapToPin nil )
    ( "[Wiring]: Snap To Via Center" snapToViaCenter nil )
    ( "[Wiring]: Automatic Via Alignment" autoViaAlignment nil )
    ( "[Wiring]: Automatic Length Control" autoLengthControl t )
)
```

Via Configuration Form

The Via Configuration form is displayed when you click the ellipses button  next to the Via field. The Via configuration form displays the *Net* tab and *Pin Escape* tab and a set of options that are common to both tabs.

- **Net tab**

  By default, the *Net* tabbed page is displayed. The *Net* tabbed page displays the valid standard and custom vias from the application default constraint group. Using the *Net* tabbed page in the Via Configuration form, you can configure the settings for each individual via.



The *Via Configuration* table on the *Net* tabbed page shows the configuration settings for each via in the layout design. The following table provides the description of various columns in the *Via Configuration* table.

Column Name	Description
Vias	The list in this column displays the valid viaDefs and valid via variants in a tree-like structure from the constraint group selected in the Wire Assistant.
Type	This is a non-editable column that displays the type of the viaDef or via variant.

Column Name	Description
Inline Enclosure	When selected, the vias whose enclosure is fully enclosed or inline with the wire are preferred. Selecting Inline Enclosure is a preference, When Inline Enclosure is deselected, router could still use an inline via if other vias do not give an appropriate solution.
Bottom Layer Enclosure	Allows you to specify the extension for the bottom layer of the layer pair. It is also used as the top layer for the layer pair above it. When the above and below layer enclosures are in the same direction and inline enclosure is specified, then the router prefers cut directions in the same direction as the direction of the top and bottom layer enclosures.
Top Layer Enclosure	Allows you to specify the extension for the top layer of the layer pair. This is only valid for the last entry in the Via Configuration form, which corresponds to the top layer of the highest layer purpose pair via in the list of valid vias. All other layer pairs use the bottom layer enclosure from the adjacent layer pair to determine its top layer enclosure direction.
Alignment	Sets the origin of a via with respect to the wire on which the via is created.
Min Number Cuts	By specifying the via cuts as minimum number of cuts, you can override the `minNumCut` rule value for creating and editing wires.

Note: Using the *All* check box in the *Via Configuration* table, you can select or deselect all vias at the same time.

- **Pin Escape tab**

  Displays the reduced options for vias used to escape from a pin.

- Options

  - **Attempt to Use Double Cut Vias**

    If you want to insert double cut vias wherever possible and not everywhere, select the *Attempt to Use Double Cut Vias* option. Using this option allows pin escape and routing with double cut vias. However, if you want to insert double cut vias everywhere, then use the `minNumCut` constraint added in constraint manager.

  - **Adjust Edited Vias Parameters**

When selected, enables the re-evaluation of via parameters. For example, with *Automatic Via Alignment* on, you stretch a wire in such a way that the via needs to be re-aligned, and the re-alignment is done if the *Adjust Edited Vias Parameters* checkbox is selected; if the *Adjust Edited Vias Parameters* checkbox is not selected, the via is not re-aligned and stays the way it was before stretching the wire.

- **Enforce Gap Space on Differential Pair Vias**

  When selected, uses the gap spacing for vias; otherwise, it uses `minSpacing` for vias. When gap spacing is used, the vias are shifted to maintain gap spacing.

- **Maximize Cuts**

  Enables a post processing step to attempt to pack as many cuts as the layer overlap area allows, based on the applicable cut spacing and layer enclosure rules. It does not control maximizing cuts in the router flow. It only controls vias during wire editing. Environment Variable: enableMaximizeCuts

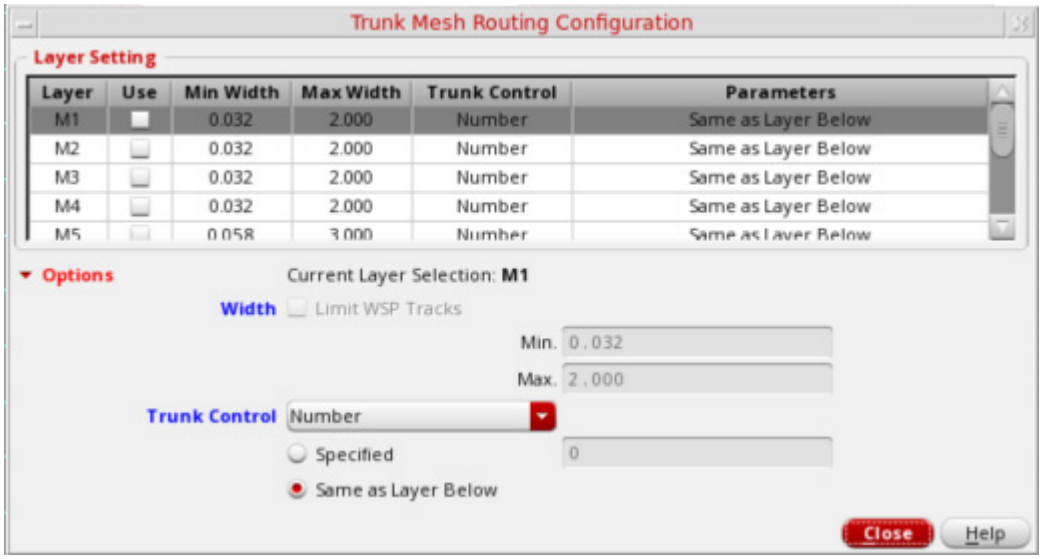
- **Snap to Via Center**

  Snaps wires to the center of vias. If this check box is selected and you click a via, the starting wire starts from the clicked point and two small segments are added to connect to the via origin. With this check box selected, tapping a via and a wire gives preference to the via. This check box is off by default.

  Environment variable: snapToViaCenter

Trunk Mesh Routing Configuration Form (ICADVM18.1 EXL Only)

The Trunk Mesh Routing Configuration form is displayed when you click the ellipses button  next to the *Mode* field.



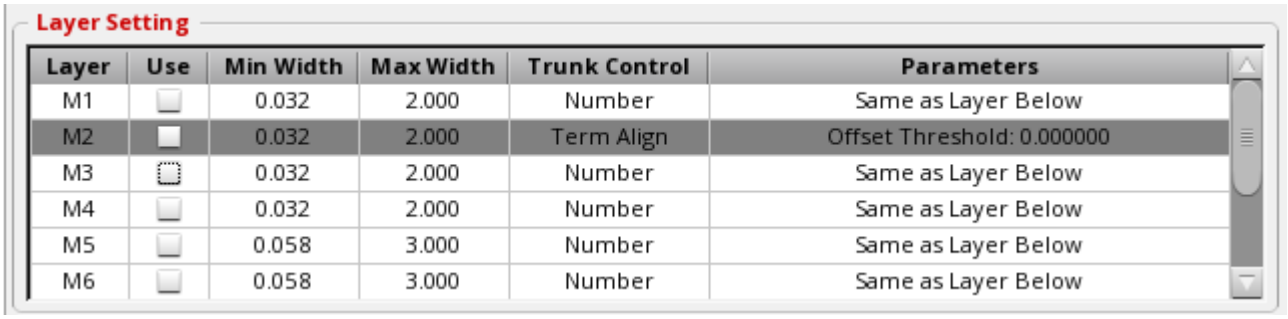
Note: The ellipses button is enabled only when the *Trunk Mesh Routing* option is selected from the *Mode* drop-down list.

The Trunk Mesh Routing Configuration form displays a table of layer settings and a set of options that you can use to configure trunk mesh routing.

- Layer Setting Table
- Options

Layer Setting Table

The *Layer Setting* table in the form shows the configuration settings for each layer in the design. The table is not used for updating the options. It is only for viewing the values.

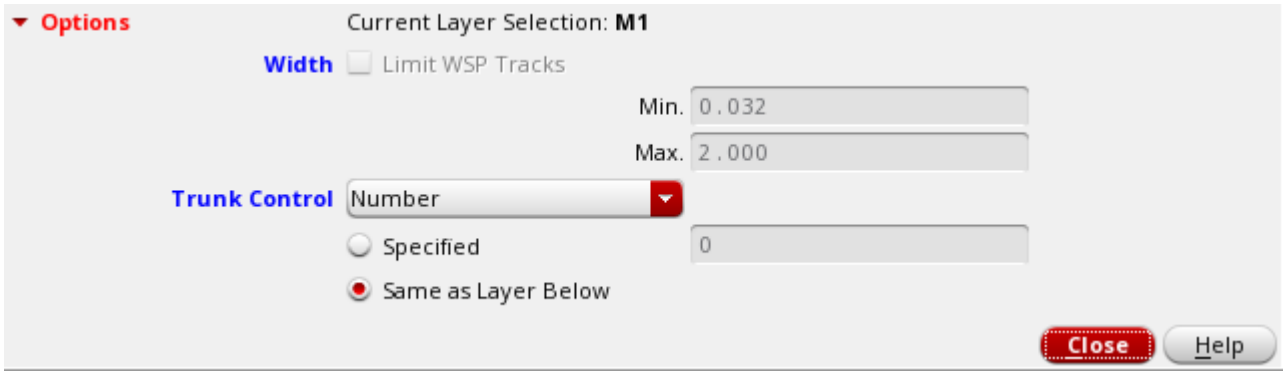


The following table provides the description of each column in the *Layer Setting* table.

Column Name	Description
Layer	Displays the layer name. You can select the row of a layer to update the options of that layer.
Use	Select the *Use* check box to select the layers that you want to use during mesh routing.
Min Width	Displays the minimum width to be used for the trunks created during mesh routing.
Max Width	Displays the maximum width to be used for the trunks created during mesh routing.
Trunk Control	Displays the option that have been specified for trunk creation. *Trunk Control* has three options: *Number*, *Term Align*, and *Spacing*.
Parameters	Displays the selected parameters depending on the *Trunk Control* option that is selected.

Options

To update the values of the parameters displayed in the *Layer Setting* table, you need to select a row in the table. You can then use the displayed options to update and set the values of the parameters for the selected row.



- *Current Layer Selection*

  Displays the layer name of the row selected in the *Layer Settings* table.

- *Width*

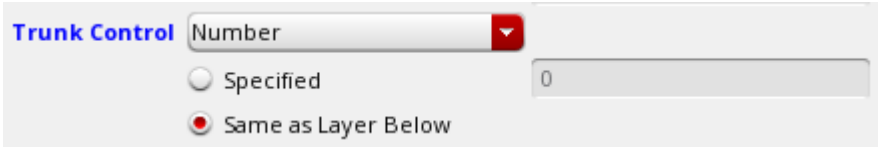
  Specify the minimum and maximum widths to limit the widths of the trunks created during mesh routing. The width is used when you want to limit the use of WSPs, if there are various widths defined for the WSPs in use.

- *Trunk Control*

  Specify the option to be used for trunk creation. You can specify one of the three options to control trunk creation: *Number*, *Term Align*, and *Spacing*.

  - *Number*

    Lets you specify the number of trunks to be created.

    - *Specified*

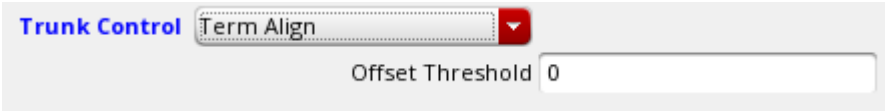
      When selected, lets you specify the exact number of trunks to be created on the selected layer.

    - *Same as Layer below*

When selected, lets you create the same number of trunks as the below layer.

- *Term Align*

  Lets you align the vertical trunks to source and drain pins of the devices.



  - *OffSet Threshold*

    In case, the WSPs in the design do not overlap the pins, you can specify the offset threshold so that the trunks are created within the specified threshold.

- *Spacing*

  Lets you specify the options to space out the trunks when mesh routing creates the trunks for a selected net in the design. By default, the minSpacing value is considered.

  - *Max. Number of Trunks*

    Specifies a number to limit the number of trunks to be created during mesh routing.

  - *Tracks*

    Specifies the number of tracks to space out the created trunks.

  - *Distances*

    Specifies the distance between the created trunks.

Related Topics

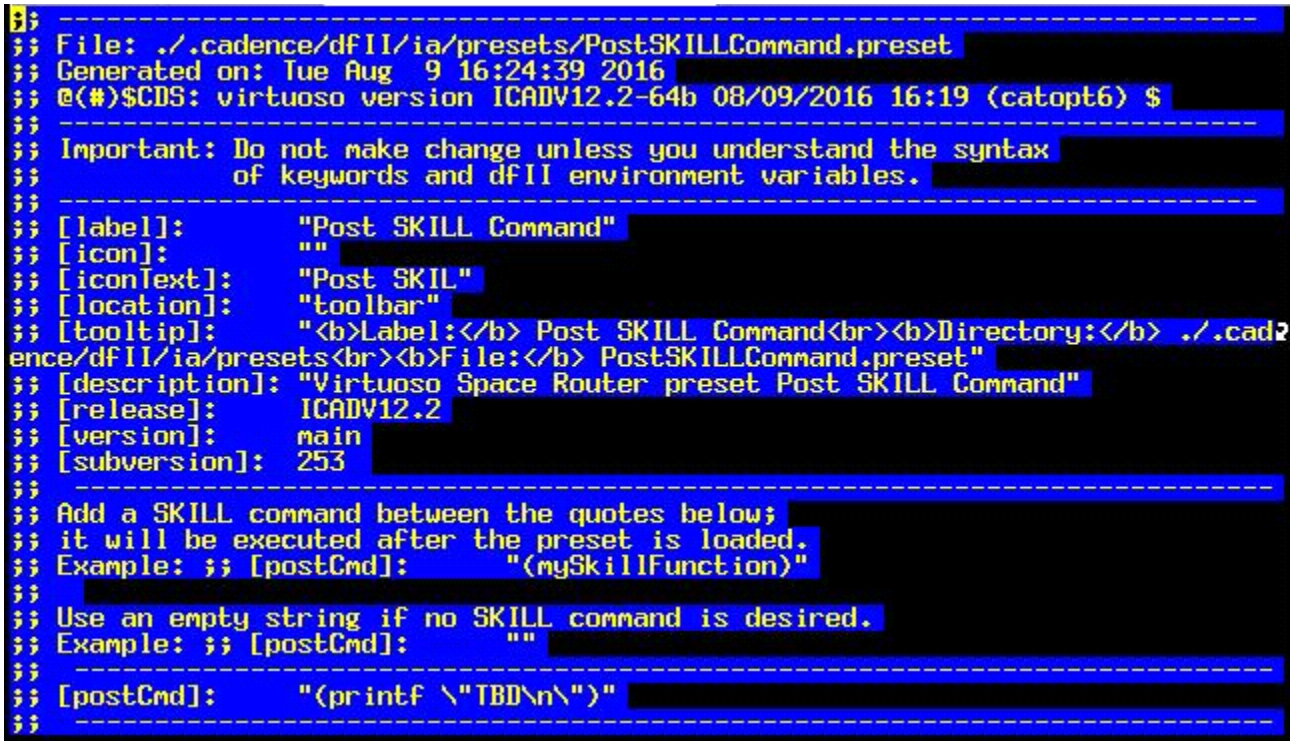
Using Trunk Mesh Routing in the Pin to Trunk Route Flow

D

Preset File

This chapter describes the information about the VSR Preset file.

A preset file helps you to change the DFII environment variables quickly and with ease. The format of the preset file is similar to the `.cdsenv` file in the `$HOME` directory. You can use a text editor to manually edit the preset file. The new VSR Preset file lets you save and restore the router-related environment variables not only for Pin to Trunk routing, but also for other automatic routing commands. The following figure shows the beginning section of a sample VSR Preset file.



A preset file begins with comments that provide information about the path, filename, time, and the Virtuoso version for which this file has been generated. The comment that has the second token begins with `"[" `, ends with `"]:"`, and the text enclosed within `"[" and "]:"`

constitutes a keyword. These keywords are mainly used by the VSR Preset feature to add the menu item or the toolbar icon for a particular preset file.

- [label]:

  Used to identify a particular preset file. The value specified for this keyword is used as a menu item in the *VSR Load Preset* drop-down menu of the VSR Preset toolbar.

  This label also helps to identify the preset file that has been loaded in the log file, that is, instead of using `iaVSRPreset_<number>` to identify the menu item, it now uses `iaVSRPreset_<[label]:>` as a menu item. For example:

  ```
  [label]: = "Optimization and Fix Violation"
  ```

  When this particular preset file is loaded, the VSR Preset generates `"iaVSRPreset_Optimization_and_Fix_Violation"` in the log file instead of `"iaVSRPreset_<number>"`. This helps you to identify the preset file that has been loaded when the log file is analyzed.

  The `label` keyword only accepts alpha-numeric value and white-space characters. Any other character, such as `'/'`, `'-'`, `'*'`, `'+'`, is interpreted as a mathematical operation by the SKILL function. If a label has any of the following characters: " `*`, `/`, `+`, `-`, `` ` ``, `~`, `!`, `@`, `#`, `$`, `%`, `^`, `&`, `(`, `)`, `=`, `[`, `]`, `{`, `}`, `|`, `\`, `/`, `;`, `:`, `<`, `>`, `.`, `?`, the character is interpreted as an underscore. Therefore, a label `M3+M2=M32` is interpreted as `M3_M2_M32`.

Important

The value specified for the `label` keyword must be unique.

- [icons]:

  Lets you specify the already existing icon file. It can also be an empty string. If VSR Preset can find the specified icon file and the `[location]:` keyword is defined as `toolbar`, then VSR Preset uses the specified icon file to create the icon and place it next to the *VSR Load Preset* icon. You can then click the icon to load the particular preset.

- [iconText]:

  Lets you specify the text for the icon if the `[icon]:` keyword is an empty string and the `[location]:` is defined as `toolbar`. If the `[location]:` is also an empty string, VSR Preset uses the value of the `[label]:` keyword to generate the text for the icon. If the value specified for the `[iconText]:` keyword is a single word, the VSR Preset specifies text of maximum five characters in one line. However, if the value of this keyword is more than a single word, then VSR Preset specifies the text in two lines with maximum of five characters on each line.

- [location]:

Defines where the preset is added. The location can be either the VSR Preset toolbar or the *VSR Load Preset* drop-down menu. If the location is defined as `toolbar`, the preset is added as an icon to the toolbar; and if the location is defined as `menu`, the preset is added to the *VSR Load Preset* drop-down menu.

- [tooltip]:

  Lets you specify the text that should get displayed as a tooltip when you place the mouse pointer on the preset icon.

- [description]:

  The value of this keyword is used when the preset file is loaded and the keyword value is displayed in CIW. The value of this keyword is automatically specified if the preset is saved using the VSR Save Preset form.

- [release]:

  Indicates the Virtuoso release for which the preset file is created.

- [version]:

  Indicates the Virtuoso version for which the preset file is created.

- [subversion]:

  Indicates the Virtuoso sub-version for which the preset file is created.

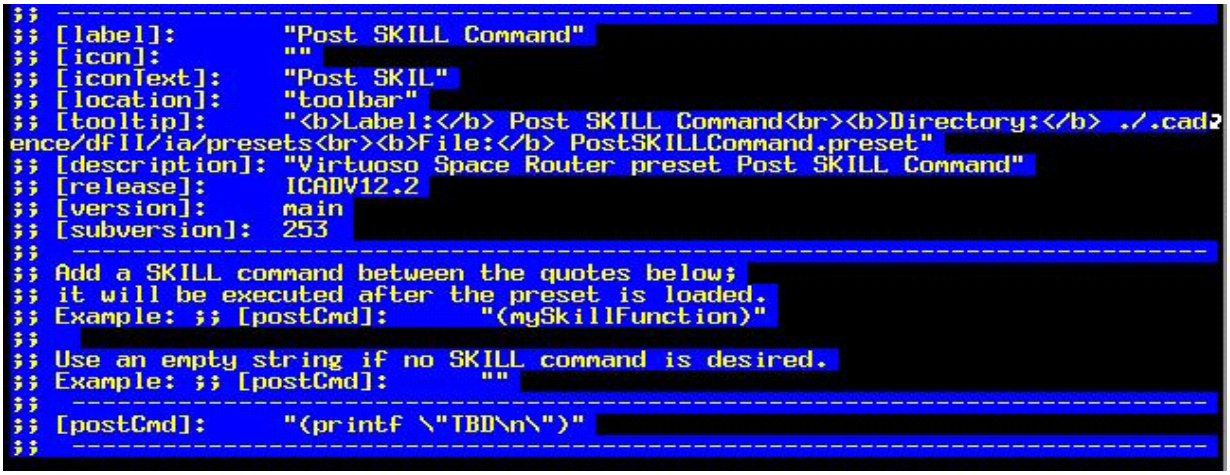
- [postCmd]:

  This is either an empty string or a SKILL command. The SKILL command is run after the VSR Preset executes all the entries on the preset file. You can manually edit this value.

  If a preset file has `[postCmd]:` keyword as a non-empty string, VSR Preset specifies the icon text with cyan as the text color. In addition, in the drop-down menu, an asterisk (`*`) appears to the right of the menu item. These indicators help you distinguish between the presets that have `[postCmd]:` and the presets that do not have `[postCmd]:`.

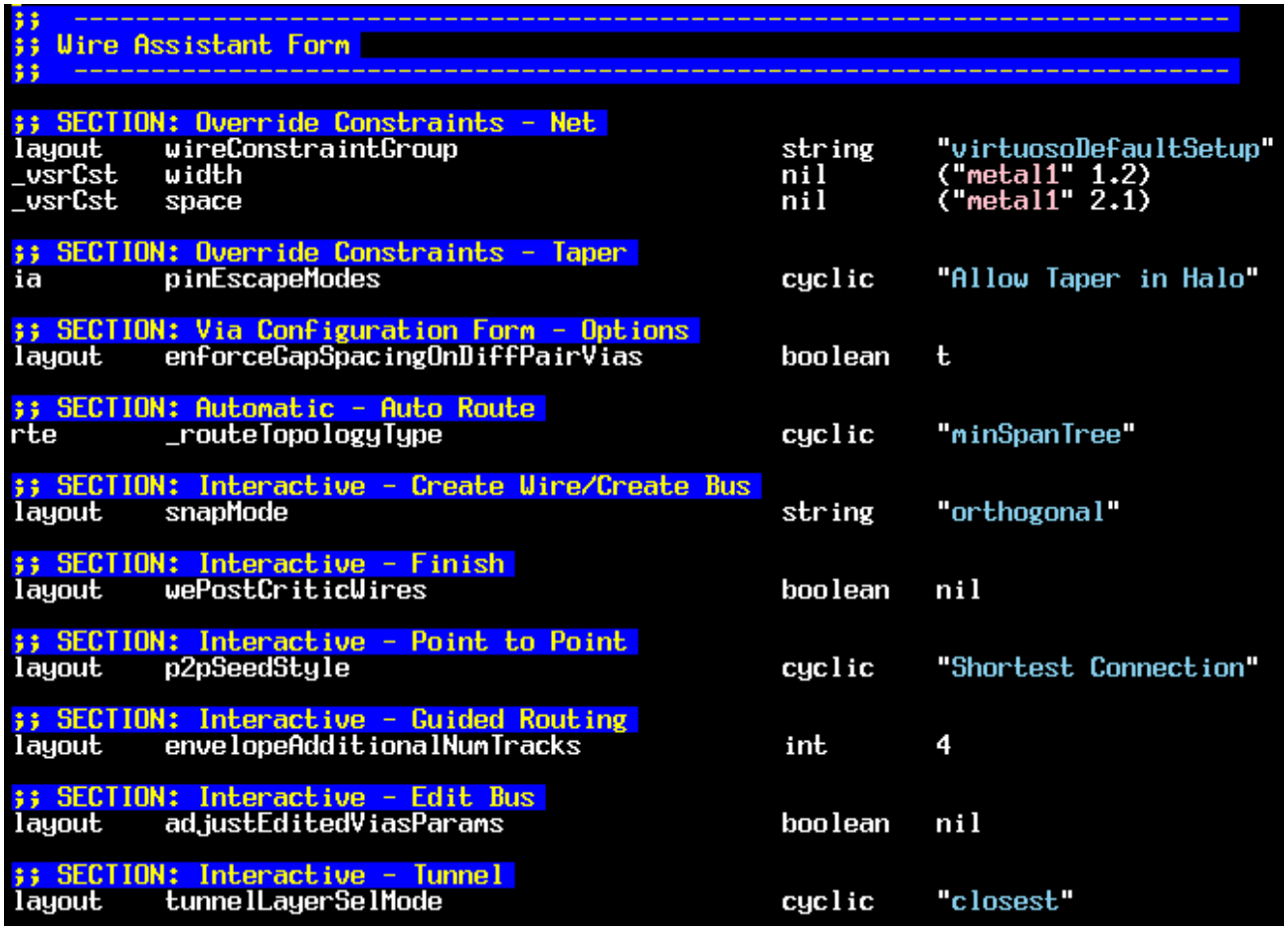
  If a preset file has the `[postCmd]:` keyword as an empty string, the value of this keyword is preserved when the preset file is overwritten by the settings specified in the VSR Save

Preset form. The following figure displays an example of the `[postCmd]:` keyword in a preset file.



The other sections of the preset file consist of the entries for constraint override values and environment variables in the Wire Assistant, Virtuoso Space-based Router Options form, and Via Configuration form. The middle section of the preset file is for both Wire Assistant and Via Configuration form. The entries in this section of the preset file are arranged based on how

the various sections of the Wire Assistant are arranged. The following figure shows a sample of the middle sections of the preset file.



```
;;  ------------------------------------------------------------------------
;; Wire Assistant Form
;;  ------------------------------------------------------------------------

;; SECTION: Override Constraints - Net
layout    wireConstraintGroup                    string    "virtuosoDefaultSetup"
_vsrCst   width                                  nil       ("metal1" 1.2)
_vsrCst   space                                  nil       ("metal1" 2.1)

;; SECTION: Override Constraints - Taper
ia        pinEscapeModes                         cyclic    "Allow Taper in Halo"

;; SECTION: Via Configuration Form - Options
layout    enforceGapSpacingOnDiffPairVias        boolean   t

;; SECTION: Automatic - Auto Route
rte       _routeTopologyType                     cyclic    "minSpanTree"

;; SECTION: Interactive - Create Wire/Create Bus
layout    snapMode                               string    "orthogonal"

;; SECTION: Interactive - Finish
layout    wePostCriticWires                      boolean   nil

;; SECTION: Interactive - Point to Point
layout    p2pSeedStyle                           cyclic    "Shortest Connection"

;; SECTION: Interactive - Guided Routing
layout    envelopeAdditionalNumTracks            int       4

;; SECTION: Interactive - Edit Bus
layout    adjustEditedViasParams                 boolean   nil

;; SECTION: Interactive - Tunnel
layout    tunnelLayerSelMode                     cyclic    "closest"
```

The last section of the preset file consists of the entries for the Virtuoso Space-based Router Options form. The entries in this section of the preset file are arranged based on how the

various sections in the form are arranged. The following figure shows a sample of the last section of the preset file.

E

Frequently Asked Questions

This FAQ contains the answers to customers' most frequently asked questions about constraints. To view the answer to any question, click the question in the list below.

Frequently Asked Questions

Question#1: In Virtuoso, which are the places where constraints can be set?

Answer:

- Constraint Setting

  There are several places where constraints can be set in Virtuoso.

  - Constraint Manager (CM)/Process Rule Editor (PRE)

    The technology or process constraints are set in the constraint manager's process rule editor. There are three options in the Process Rule Editor form.

    - The *Tech* option sets the process or foundry rules and other constraint groups stored in the technology library.
    - The *Design* and *Object* options setup user-defined constraint groups on the design level or Object level. The foundry constraint group is for process rules and user-defined constraint groups are constraints for design considerations.

  - Wire Assistant

    The Wire Assistant allows you to set temporary override constraints, which can now be saved. They operate on the net that is being interactively edited as an override to constraints assigned to that net or route (or the default constraint group) and are not

persistent on the edited net. You can save the constraints to a constraint group and assign them to a net in the Constraint Manager.

- RIDE/VSR GUI (Sequencer)/Routing Scripts

  Constraints can be set in TCL for foundry and design rules. In addition to the run script, they can also be set in the .riderc and the RIDE built-in process (rde.startup_script). If there are constraints that are supported in the Cadence technology file and you have write permission, they will be synchronized when RIDE is exited. Design and object level constraints will be stored in the view that is being edited, so you will have write permission.

For more information, refer to the Virtuoso documents.

- Constraint Usage

  Once the constraints are set up, there are several menus that use the specified constraints.

  - Options — Editor, Wire Editor Default constraint group

    This assigns a constraint to nets which do not have a constraint group assigned to the net.

  - Wire Assistant Seed Attributes From a Constraint Group

    This seeds the Wire assistant form values for the constraints in the form (width, spacing, etc) from the constraint values in the selected constraint group.

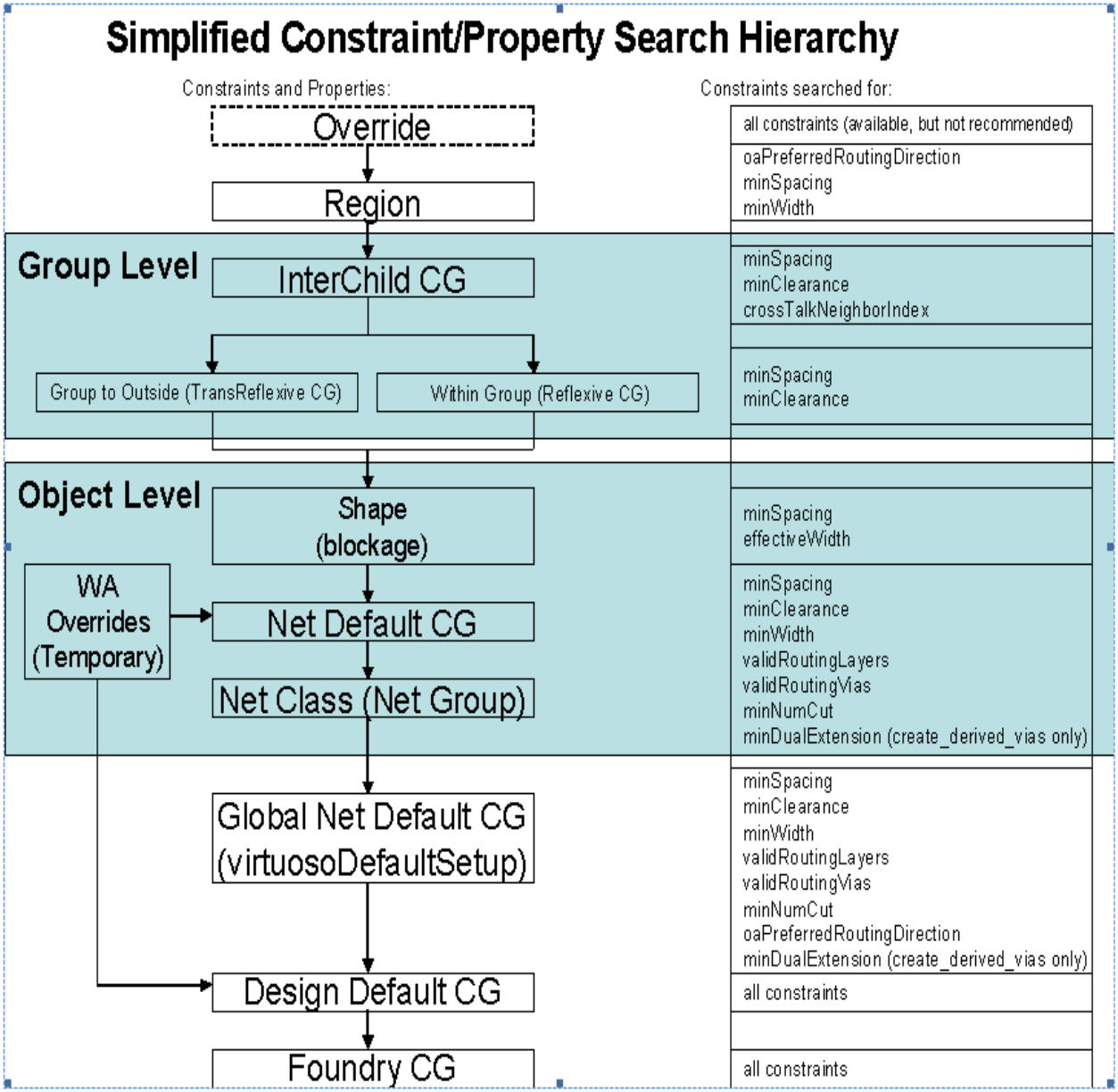
  - VSR GUI— Default Constraint Group

    This assigns a constraint to nets which do not have an explicit constraint assigned to them. It is initially given the same value that is in the Options — Editor form.

Question#2: What is the constraint lookup order and how does override work with automatic and interactive routing?

Answer:

The constraints search hierarchy determines the order and which constraints the router looks at. Additionally, the interactive router can assign overrides for the net which is currently being edited which overrides the value of the constraint for any constraints set on that net. The constraint search hierarchy diagram is as shown below.



The constraints have the highest priority from top to bottom. The "Constraints searched for:" column shows the constraints that are supported at each level of the constraint hierarchy. The Wire Assistant (WA) overrides to the left of the main flow stack are set on the net or routes for the net being edited.

Question#3: What are the possible reasons that the router may not connect to a pin?

Answer:

Possible reasons that the router may not connect to a pin are as follows:

- A layer conflict. For example, a poly pin but, poly is not in the validLayers constraint of the net, design, or application default constraint group.
- A spacing conflict. For example, the width + spacing of the net is larger than the distance between the target pin and a neighboring pin or blockage.